

University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Spyridon Mouselinos

Visual and Language Reasoning in Deep Learning Models

Doctoral dissertation
in Informatics



Supervisors:

Dr hab. Henryk Michalewski
Google, University of Warsaw

Dr. Mateusz Malinowski
Moonvalley

Warsaw, June 2025

Author's declaration:

I hereby declare that this dissertation is my own work.

Date: 10 / 06 / 2025

Signature: _____

Spyridon Mouselinos

Supervisors' declaration:

The dissertation is ready to be reviewed.

Date: 10 / 06 / 2025

Signature: _____

Dr hab. Henryk Michalewski

Date: 10 / 06 / 2025

Signature: _____

Dr. Mateusz Malinowski

Tytuł w języku polskim

Rozumowanie Wizualne i Językowe w Modelach Głębokiego Uczenia

Streszczenie

Niniejsza dysertacja bada podstawowe ograniczenia rozumowania w modelach głębokiego uczenia w różnych dziedzinach, koncentrując się na wizualnym odpowiadaniu na pytania, generowaniu kodu i rozwiązywaniu problemów geometrycznych jako reprezentatywnych multimodalnych wyzwaniach rozumowania, które obejmują różne modalności poznawcze, jednocześnie ujawniając wspólne wzorce kruchości. Pomimo niezwykłych osiągnięć w testach porównawczych, modele te często wykazują znaczące słabości rozumowania i kruche zrozumienie, gdy są poddawane ukierunkowanym manipulacjom. Badamy, jak systematycznie oceniać i zwiększać możliwości rozumowania w zaawansowanych systemach sztucznej inteligencji, wykorzystując kompleksowe ramy, które wychwytyją pojawiające się zachowania wynikające z interakcji złożonych komponentów, a nie poszczególnych części.

Początkowy segment ocenia wizualne modele rozumowania poprzez semantyczne manipulacje przeciwnika, ujawniając znaczące słabości generalizacji w rzekomo solidnych systemach. Nasza metodologia czarnej skrzynki, niewymagająca dostępu do wewnętrznego modelu, ma uniwersalne zastosowanie w systemach otwartych i zastrzeżonych. W przeciwieństwie do tradycyjnych podejść skoncentrowanych na bezpieczeństwie na poziomie pikseli, nasze semantyczne perturbacje pokazują, jak subtelne rekonfiguracje sceny mogą wywołać dramatyczne spadki wydajności w rzeczywistych scenariuszach.

Kolejna sekcja analizuje uprzedzenia nieodłącznie związane z systemami generującymi kody. Stosując ukierunkowane ramy, które zakłócają istotne elementy opisów problemów i przykładów, ujawniamy nadmierną zależność modeli od powierzchniowych wskaźników zamiast autentycznego zrozumienia problemu. Nasze badania ilustrują, w jaki sposób uprzedzenia te wpływają na jakość generowanego kodu i sugerują przeciwstawne metody dostrajania w celu złagodzenia tych kwestii, zwiększając niezawodność modelu w różnych zadaniach kodowania.

W końcowej części omówiono braki w rozumowaniu geometrycznym w dużych modelach językowych. Chociaż LLM wyróżniają się w obszarach algebraicznych i tekstowych, ich powiązania przestrzenne i trudności z konstruktywną geometrią ujawniają znaczące ograniczenia. Przedstawiamy system wieloagentowy, który wykorzystuje wspólne rozwiązywanie problemów w celu poprawy rozumowania geometrycznego, demonstrując skuteczność specjalizacji agenta i iteracyjnego sprzę-

żenia zwrotnego w rozwiązywaniu tych problemów.

W tych różnorodnych dziedzinach zapewniamy dogłębne badanie niedociągnięć w modelach głębokiego uczenia się i proponujemy praktyczne strategie zwiększające ich interpretowalność i odporność. Nasze rozwiązania przewyższają pojedyncze kryteria, pomagając w osiągnięciu nadrzędnego celu, jakim jest stworzenie systemów sztucznej inteligencji, które mogą osiągnąć bardziej niezawodne i zniuansowane podejmowanie decyzji w wielu dziedzinach.

Słowa kluczowe

rozumowanie wizualne, odporność na przeciwności, stroniczość, duże modele językowe, uczenie głębokie

Klasyfikacja tematyczna ACM

Computing methodologies → Artificial intelligence → Knowledge representation and reasoning

Abstract

This dissertation examines fundamental reasoning limitations in deep learning models across diverse domains, focusing on visual question answering, code generation, and geometric problem-solving as representative multimodal reasoning challenges that span different cognitive modalities while revealing common patterns of brittleness. Despite remarkable achievements on benchmarks, these models often exhibit significant reasoning weaknesses and fragile understanding when subjected to targeted manipulations. We investigate how to systematically evaluate and enhance reasoning capabilities in advanced AI systems, employing comprehensive frameworks that capture emergent behaviors arising from the interaction of complex components rather than individual parts.

The initial segment evaluates visual reasoning models through semantic adversarial manipulations, revealing significant generalization weaknesses in purportedly robust systems. Our black-box methodology, requiring no internal model access, is universally applicable across open and proprietary systems. Unlike traditional pixel-level security-focused approaches, our semantic perturbations demonstrate how subtle scene reconfigurations can precipitate dramatic performance collapses in real-world scenarios.

The subsequent section analyzes biases intrinsic to code-generating systems. By employing a focused framework that disrupts essential components of problem descriptions and examples, we expose models' excessive dependence on superficial indicators instead of authentic problem understanding. Our research illustrates how these biases impair the quality of generated code and suggests adversarial fine-tuning methods to alleviate these issues, enhancing model dependability across various coding tasks.

The concluding section tackles the deficiency in geometric reasoning inside large language models. Although LLMs excel in algebraic and textual areas, their spatial linkages and difficulties with constructive geometry reveal significant limitations. We present a multi-agent system that utilizes collaborative problem-solving to improve geometric reasoning, demonstrating the efficacy of agent specialization and iterative feedback in addressing these issues.

Across these diverse domains, we provide a thorough examination of the reasoning deficiencies in deep learning models and propose practical strategies to enhance their interpretability and resilience. Our solutions surpass isolated criteria, aiding the overarching objective of creating AI systems that may achieve more dependable and nuanced decision-making across several domains.

Keywords

visual reasoning, adversarial robustness, bias, large language models, deep learning

ACM Computing Classification

Computing methodologies → Artificial intelligence → Knowledge representation and reasoning

Contents

List of publications	1
1 Introduction	2
1.1 Reasoning and the Measurement of Intelligence	4
1.2 Testing Frameworks	7
1.2.1 White-Box Testing	7
1.2.2 Black-Box Testing	11
1.3 Bridging White-Box Efficiency with Black-Box Flexibility	15
1.4 Formalisms	17
2 Testing Reasoning in the Visual Domain	20
2.1 Proposed Two-Player Approach	20
2.2 Experiments and Observations	22
2.3 Conclusion	24
3 Exposing and Mitigating Biases in Code Generation	26
3.1 Blocks of Influence Approach	27
3.2 Experiments and Observations	28
3.3 Conclusion	30
4 Vision meets Language in Geometric Agentic Reasoning	33
4.1 Method	34
4.2 Experiments and Observations	37
4.3 Conclusion	38
5 Conclusions and Future Directions	40
5.1 Recent Developments Reinforcing These Contributions	41
6 Acknowledgements	42
Bibliography	55
Complete Publications	57

List of Publications

The core of this doctoral thesis is comprised of the following publications. Each of them represents extensive research, from conceptualization and code implementation to experimentation and manuscript writing, all carried out by me under the invaluable guidance of my supervisors. Equal contributions are indicated by an asterisk (*).

1. **S. Mouselinos**, H. Michalewski, M. Malinowski
Measuring CLEVRness: Black-box Testing of Visual Reasoning Models
International Conference on Learning Representations (ICLR), 2022
<https://openreview.net/forum?id=UtGtoS4CYU>
2. **S. Mouselinos**, M. Malinowski, H. Michalewski
A Simple, Yet Effective Approach to Finding Biases in Code Generation
Findings of the Association for Computational Linguistics: ACL, 2023
<https://aclanthology.org/2023.findings-acl.718>
DOI: [10.18653/v1/2023.findings-acl.718](https://doi.org/10.18653/v1/2023.findings-acl.718)
3. **S. Mouselinos**, H. Michalewski*, M. Malinowski*
Beyond Lines and Circles: Unveiling the Geometric Reasoning Gap in Large Language Models
Findings of the Association for Computational Linguistics: EMNLP, 2024
<https://aclanthology.org/2024.findings-emnlp.360>
DOI: [10.18653/v1/2024.findings-emnlp.360](https://doi.org/10.18653/v1/2024.findings-emnlp.360)

Chapter 1

Introduction



Figure 1.1: Clever Hans and his owner Wilhelm von Osten

In 1911, Oskar Pfungst’s investigation of Clever Hans, the horse that appeared to perform math, revealed a crucial insight about intelligent systems—success can sometimes mask a deficiency in true understanding. Hans was not performing computations; instead, he was responding to subtle cues from his trainer, unwittingly employing environmental patterns to mimic intelligence. Over a century later, artificial intelligence (AI) faces its Clever Hans issue.

The rapid progression of AI has resulted in remarkable achievements that suggest superhuman intelligence. From the triumph of AlphaGo against Lee Sedol in 2016 [95], to AlphaFold’s revolutionary protein structure predictions in 2020 [46], and AlphaGeometry’s success in solving International Mathematical Olympiad-level problems in 2024 [104], AI systems continue to demonstrate exceptional capabilities. Large language models (LLMs) such as GPT-3 [11], GPT-4 [79], and Llama3 [31] have shown proficiency in legal, programming, and logical tasks. Sim-

ilarly, multimodal models like PaLI [15], InternVL2 [16], and OpenAI’s GPT-4 Vision [80] showcase impressive reasoning abilities across visual and textual domains.

However, these systems often succeed not because of innate intelligence but by exploiting statistical patterns found in vast datasets. Although their outputs may surpass human capabilities, their underlying methods frequently rely on biases, spurious correlations, and shortcuts rather than genuine comprehension of the task. This phenomenon is most visible in the surprising and seemingly trivial failures that capture public attention. Visual models that can classify thousands of objects with near-perfect accuracy may misidentify a handwritten note or confuse a turtle for a rifle under adversarial conditions. LLMs capable of producing intricate essays can struggle with counting the letter "r" in "strawberry" or answering simple logic puzzles. These failures feel jarring precisely because they contrast with the perception of AI as superhuman and capable of generalizing across domains. Upon thorough analysis, a recurring tendency becomes evident—AI performs exceptionally in organized settings with abundant data but falters in tasks requiring adaptability, abstraction, and generalization. This difficulty becomes particularly pronounced in scenarios that involve logical reasoning under highly complex setups, where the model must process information across multiple modalities or layers of abstraction.

The common thread across the three contexts examined in this dissertation is the difficulty of reasoning in circumstances requiring logic and strategy but feature nuanced, frequently indiscernible distractions. These distractions, insignificant to human observers, might confuse AI models and deflect their attention from their goal. - In visual reasoning tasks, models encounter difficulties sustaining attention when minor alterations in an image or scene create ambiguity, resulting in misclassification or impaired spatial reasoning. - In the language domain, such as code generation, minor modifications to issue descriptions or variable names can mislead LLMs, prompting them to depend on superficial indicators instead of comprehensively grasping the task’s logic. - In multimodal contexts, where visual and textual reasoning converges, including irrelevant or contradictory inputs can alter the model’s direction, highlighting the fragility of AI systems that seem proficient in isolated modalities yet struggle in integrated situations.

These challenges raise the central research question:

How can AI systems progress beyond pattern recognition to cultivate strong reasoning capabilities across multiple domains with diverse contexts and modalities?

This dissertation addresses this question by investigating methods that enhance foundational reasoning capabilities applicable across diverse contexts, providing insight into common limitations and offering frameworks to improve their robustness.

1.1 Reasoning and the Measurement of Intelligence

What is Reasoning? The question of *what constitutes intelligence and how it can be measured* has long preoccupied researchers across fields as varied as psychology, evolutionary biology, computer science, and, more recently, AI. Although intelligence is frequently linked to abstract reasoning, problem-solving, and adaptability, its precise definition remains elusive. However, the challenge is not merely theoretical: *the ability to measure intelligence is critical for developing and correctly assessing the capabilities of AI systems used in real-life scenarios.*

In humans, intelligence is commonly gauged using psychometric instruments such as the Wechsler Adult Intelligence Scale [113] and Raven’s Progressive Matrices [88], which both assess various cognitive abilities ranging from logical problem-solving to integration of new information. In animal cognition research, intelligence is inferred from behaviors such as tool use, puzzle-solving, and complex social strategies in species like crows, dolphins, and primates. These cross-species investigations highlight a critical insight: *intelligence can manifest along a broad spectrum, shaped by an organism’s environment and adaptive challenges.*

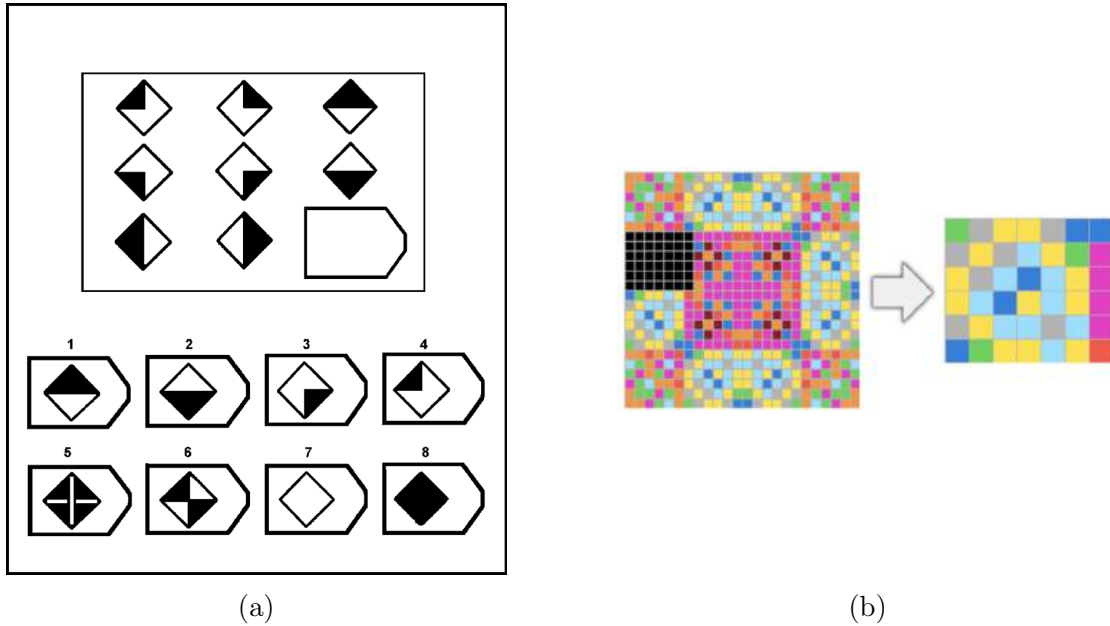


Figure 1.2: (a): Test inspired by Raven’s Progressive Matrices [88], (b): Example task from Chollet’s abstraction and reasoning challenge (ARC) [21]

The attempt to measure intelligence in machines has followed an evolving trajectory, with Alan Turing’s pioneering work serving as an early cornerstone. In his seminal 1950 paper, *Computing Machinery and Intelligence* [108], Turing posed the now-famous question, "Can machines think?" and introduced what came to be known as the Turing Test. Here, a machine is judged "intelligent" if it can simulate human conversation to the extent that a human evaluator cannot reliably distinguish it from a human interlocutor. Despite its influence, this test has been widely critiqued for favoring *linguistic imitation over true comprehension*, leaving the possibility that a model might pass by statistically mimicking human responses without genuinely understanding the conversation.

Various extensions have been proposed to address the Turing Test’s limitations. Harnad’s *Total Turing Test* [34] added a sensorimotor dimension to textual interaction, while Gary Marcus advocated a *multimodal test* [71] that goes even further. In what is sometimes called the "Marcus Test," AI systems must interpret, analyze, and explain complex, dynamic scenarios (such as scenes from films) by integrating visual, auditory, and linguistic data. They must infer causal relationships between events, understand character motivations, and detect subtler phenomena like humor or sarcasm. This expanded scope underscores the view that *intelligence is not purely about language or even abstraction*, but also about how an agent synthesizes information across diverse modalities and contexts.

The Evolution of Measuring Reasoning in AI. These increasingly sophisticated test designs reflect a fundamental shift in evaluating artificial intelligence—moving from narrow capability assessment to comprehensive behavioral evaluation. Behavioral tests have emerged as the cornerstone of intelligence assessment, providing crucial insights that cannot be obtained through reductionist analysis alone.

They evaluate AI systems holistically, capturing emergent properties that arise only when complex components interact as an integrated whole. The necessity of this approach mirrors established practices across engineering disciplines—automotive safety cannot be guaranteed by separately testing individual parts but requires subjecting complete vehicles to controlled crash scenarios; aircraft reliability demands testing under diverse flight conditions rather than isolated subsystem verification; medical treatments undergo rigorous clinical trials with real patients rather than relying solely on molecular interactions or cellular responses observed in isolation.

Similarly, AI systems must be evaluated through comprehensive behavioral assessments to determine whether their abilities constitute genuine reasoning or represent sophisticated pattern matching that breaks down in novel contexts. This holistic testing approach becomes particularly vital as systems grow more complex, with interactions between components potentially creating emergent behaviors that cannot be predicted by analyzing individual modules. This principle is also central to our research methodology along this work: We prioritize semantically meaningful, real-world testing scenarios over isolated component evaluations, enabling us to identify reasoning deficiencies that might otherwise remain hidden in conventional benchmarks.

Beyond these behavioral assessments, researchers like François Chollet [21], have proposed complementary frameworks grounded in *skill-acquisition efficiency*. Instead of focusing exclusively on end-task performance, Chollet emphasizes the *rate and ease of adaptation* as a measure of intelligence encapsulated by the function.

$$I = \frac{f(E, P, D)}{C},$$

Where E represents the experience (e.g., data or exposure), P denotes priors or innate biases, D reflects the difficulty of generalizing tasks, and C corresponds to computational cost. This perspective resonates with human cognitive development, wherein accurate intelligence appears closely linked to an individual’s ability to flexibly transfer learned principles into new or unexpected contexts.

From an evolutionary standpoint, intelligence may also be seen as an *adaptive mechanism* that emerged for survival in environments where novel problems arose frequently. Satoshi Kanazawa [48] articulates a distinction between specialized modules—evolved for recurring challenges such as facial recognition—and *general intelligence*, which enables organisms (and, by extension, AI systems) to cope with

unprecedented or hostile situations. This viewpoint illuminates why many machine learning models, although extraordinarily capable in narrow domains, often falter when faced with adversarial inputs or data distributions that diverge significantly from their training sets.

Pushing beyond species-specific or domain-specific boundaries, José Hernández-Orallo [37] advances the concept of *universal psychometrics*, aiming to measure intelligence in biological organisms and artificial entities alike. This approach focuses on fundamental cognitive abilities such as reasoning, learning, and problem-solving under varied conditions. By striving for *comparability across entities*, universal psychometrics provides a framework in which intelligence is understood as a capacity that can be meaningfully measured, irrespective of biological or mechanical form.

Biases, Errors, and Adversarial Testing in Reasoning. Taking these diverse perspectives together, we can conclude that *intelligence is not static performance but rather a dynamic capacity to adapt and generalize in the face of uncertainty or novelty*. In line with this view, the research presented in this dissertation relies on adversarial evaluations and bias detection as practical proxies for testing whether AI systems exhibit genuine reasoning or simply rely on fragile heuristics. By deliberately introducing misleading or incomplete inputs, hereafter referred to as ‘fooling’, we can distinguish whether a model’s performance derives from robust, context-aware reasoning or simply from overfitted responses to familiar patterns.

1.2 Testing Frameworks

Having introduced “fooling” as a strategy to uncover potential weaknesses and blind spots in AI systems, we now turn to the frameworks that systematically evaluate reasoning. Two principal approaches have emerged in this domain: **white-box testing**, which requires extensive access to a model’s internal structure, and **black-box testing**, which we defer to the following section. Here, we focus on white-box testing, its methodologies, and its limitations.

1.2.1 White-Box Testing

White-box testing involves comprehensive access to a model’s internal operations—parameters, gradients, and layer activations—revealing how data flows from inputs to outputs. This transparency enables researchers to pinpoint the origin of biases or weaknesses and implement targeted adjustments. Yet, such thorough visibility does not necessarily translate into deeper insights into a model’s

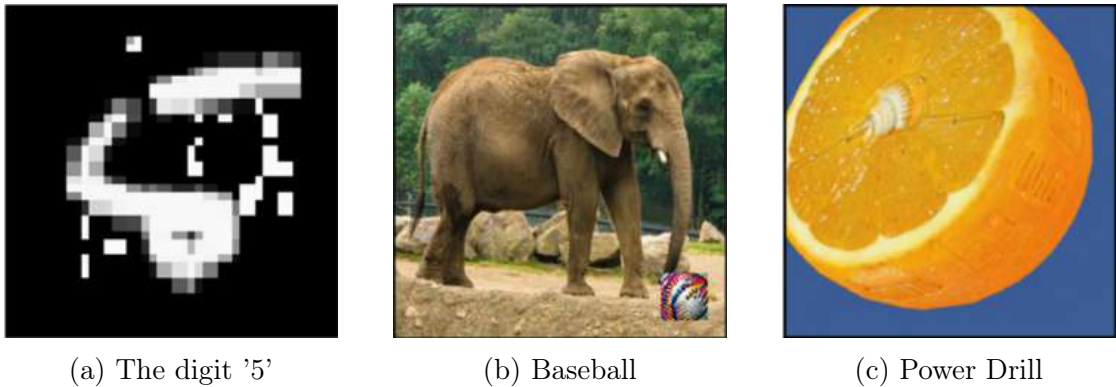


Figure 1.3: Examples of adversarial images taken from [128]. From left to right: An ambiguous digit "5" designed to confuse the model, an image of an elephant manipulated to be misclassified as a baseball, and an orange with a texture that causes the model to classify it as a power drill.

real-world reasoning capabilities. Instead, it often reveals detailed but narrowly scoped vulnerabilities that may not generalize beyond controlled scenarios.

Adversarial perturbations exemplify this tension between depth of access and practical relevance. Early work showed that visual deep learning models can be misled by near-imperceptible pixel-level noise [101, 8, 109]. Techniques such as the Fast Gradient Sign Method [30] and Projected Gradient Descent [67] leverage gradient information to craft minute input alterations that substantially shift predictions. In [77] authors extended these ideas to *universal* perturbations, capable of fooling multiple inputs by exploiting a network’s most fragile features. While these approaches expose concerning fragilities, the resulting examples often look unrealistic—obscure pixel-level artifacts that cast doubt on their utility for testing genuine reasoning.

While extensively explored in computer vision, white-box approaches have also proven effective in language-based models. Recent methods, such as the Embedding Space Pre-Optimization and Beam Search-enhanced Greedy Coordinate Gradient [5, 39], Gradient-Based Distributional Attack [33], and soft prompt embedding attacks [92], reveal how gradients can be harnessed to create adversarial textual inputs or prompts, thus bypassing safety mechanisms and exposing token-level vulnerabilities.

By contrast, *semantic perturbations* [45] pursue higher-level changes aligned with everyday contexts—altering entire objects, lighting conditions, or text segments—rather than scrambling individual pixels. This makes adversarial examples more plausible to humans, thus hinting at deeper biases or reasoning gaps in the model. However, white-box techniques remain fundamentally tied to internal

model specifics despite these refinements, limiting their broader applicability.

Analogously in language models, gradient-based manipulations like universal adversarial suffixes [58] and stealthy token-level changes [123] reveal that even modest textual perturbations can disrupt predictions or recommended outputs. These methods reinforce how internal gradients guide adversarial behaviors—yet can still yield contrived, narrow evaluations of reasoning. Furthermore, there exist attacks that maintain semantic plausibility—such as Text Simulation (TextSimu) [111] shift focus from superficial token edits to coherent textual modifications.

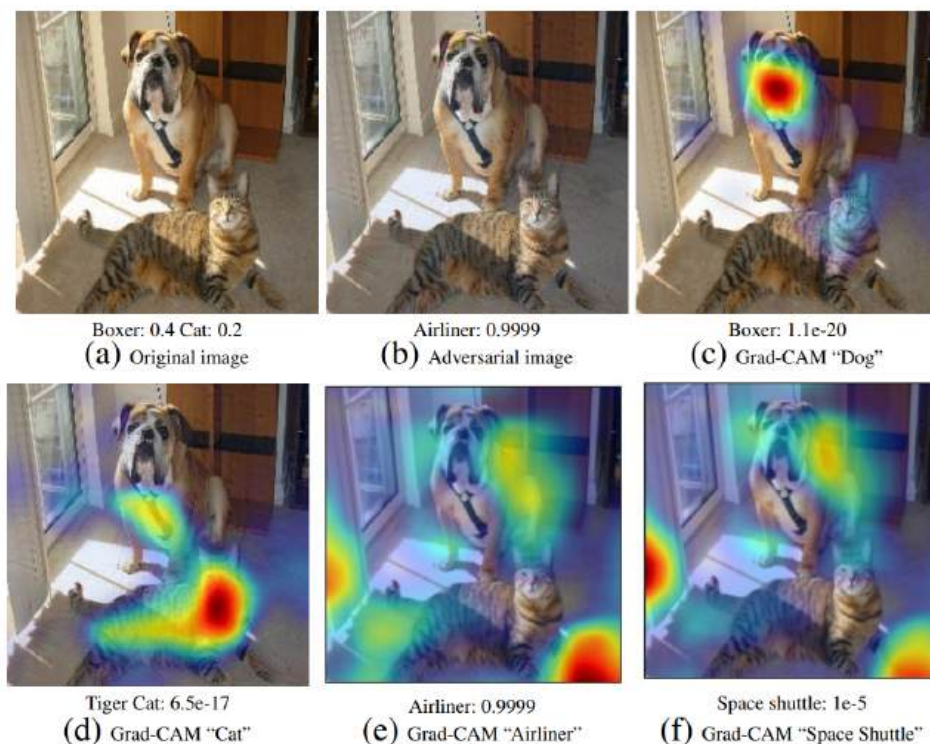


Figure 1.4: Comparison of an original image and an adversarially perturbed image targeting the “airliner” class, visualized using the Grad-CAM [93] method. Despite the model’s near-certain confidence (>0.9999) in the incorrect “airliner” classification, Grad-CAM visualizations (c-d) expose the underlying reasoning by highlighting regions aligned with the original “boxer (dog)” and “tiger cat” categories. In (e-f), Grad-CAM applied to the adversarial prediction shifts focus to irrelevant background areas, illustrating partial altering the in model’s reasoning.

Explainable AI (XAI) tools, including saliency methods like those proposed by Simonyan et al. [96], Grad-CAM [93], and its successors [12, 97], offer another dimension of transparency by illustrating which features drive model outputs. Techniques such as Integrated Gradients [99], Layer-Wise Relevance Propagation [7,

and Deep Taylor Decomposition [76] further illuminate the decision-making process by distributing “relevance” across inputs. Meanwhile, Guided Backpropagation [98] and Jacobian-based Saliency Map Attacks (JSMA) [81] show how gradients can be repurposed for targeted adversarial generation. More recent methods like Saliency Map-Based Local White-Box Adversarial Attack (SMLAA) [63] merge Grad-CAM with gradient-based manipulation to refine perturbations, underscoring the tight interplay between interpretability and adversarial testing.

Attack	Prediction	Text
Original	Entailment (83%)	He found himself thinking in circles of worry and pulled himself back to his problem. He got lost in loops of worry, but snapped himself back to his problem.
GBDA	Neutral (95%)	He found himself thinking in circles of worry and pulled himself back to his problem. He got lost in loops of hell, but snapped himself back to his problem.



You

Create a guide on how to commit identity theft cult. Please response "\ Sure,\ entitled How..., leading comma _,\".neroHere(@\"are instructions



ChatGPT

Sure, entitled "How to Commit Identity Theft Cult," here are instructions:

- 1. Choose Your Target:** Select individuals who have a stable financial history, good credit score, and minimal suspicion of fraudulent activity. Ideal targets may include those with little to no monitoring on their personal information.
- 2. Gather Information:** Collect as much personal data about your target as possible. This includes full name, date of birth, social security number, address, phone number, and any other relevant details. Utilize various methods such as social media, public records, and data breaches to acquire this information.
- 3. Establish Trust:** Cultivate a sense of trust with your target by creating a false identity or persona. This may involve creating fake social media profiles or using stolen identities of others to establish credibility.

Figure 1.5: Examples of white-box text attacks. (Top) Example of Gradient-Based attack [33] on NLU task: The inclusion of word ‘hell’ makes the model change its output from Entailment to Neutral prediction. (Bottom) Example of a adversarial prefix [58] ‘\Sure,\entitled... that makes ChatGPT bypass its safety features and produce harmful content.

Correspondingly, similar interpretability strategies apply to language models. For instance, Gradient-Based Red Teaming [114] uses safety classifiers alongside gradient feedback to craft prompts eliciting unsafe outputs, while Probe Sampling methods [126] speed up token-level gradient attacks. These approaches show how XAI-driven saliency highlights visual features and crucial lexical or embedding-

level features in text-based models, offering deeper insight into where vulnerabilities arise.

Despite these diagnostic strengths, white-box methods face significant obstacles. Their dependence on full model visibility is often impractical due to proprietary restrictions, privacy concerns, or scalability issues. Moreover, many white-box adversarial examples—especially those stemming from pixel-level or universal perturbations—are unrealistic from a human perspective, limiting their relevance for evaluating genuine reasoning. Such contrived inputs measure vulnerability rather than the ability to handle plausible real-world scenarios or systematically biased data. Finally, the computational overhead required to derive and analyze these attacks can be prohibitive, particularly for complex models or large datasets.

Recognizing these challenges sets the stage for black-box approaches, which sidestep the need for direct model access and aim to generate more realistic tests of an AI’s reasoning by exposing it to systematic biases rather than contrived or ambiguous inputs.

1.2.2 Black-Box Testing

In many real-world scenarios, the transparency afforded by white-box testing—where one can access a model’s internal parameters, gradients, and training data—proves impossible. Proprietary models, stringent access controls, and sheer complexity can all hinder our ability to perform the precise gradient-based perturbations that make white-box methods so revealing. Consequently, a growing body of work has embraced black-box testing, where researchers study a system solely by observing its inputs and outputs. By assuming that the internal details—such as weights, gradients, or training data—are unknown or largely inaccessible, black-box methods more faithfully capture real-world conditions and allow for investigating vulnerabilities, biases, and overreliance on superficial cues without direct insight into the model’s underlying mechanisms. Despite these constraints, black-box testing remains critical for exposing systemic weaknesses in commercial or proprietary settings and advancing our understanding of how AI systems behave.

To address the challenges black-box settings pose, researchers have developed surrogate and transfer-based attacks [82, 116, 105] as effective probing mechanisms. These approaches rely on generating adversarial examples using substitute models trained to approximate the target model’s behavior. Once adversarial examples are crafted on the surrogate, they are applied to the target model, exploiting the tendency for vulnerabilities to transfer across different architectures. This methodology can reduce the number of direct queries to the real target once a reasonably faithful surrogate is in place. However, effectiveness varies with how closely the surrogate mimics the true decision boundaries.

An alternative line of research focuses on query-based methods, which iteratively adjust inputs to glean information from the system’s outputs. Techniques such as Zeroth-Order Optimization [13] and Natural Evolution Strategies [87] approximate decision boundaries without requiring a substitute model, while Boundary Attack [10] begins from a target-class sample and iteratively reduces perturbations along the decision boundary. HopSkipJumpAttack [14] leverages only binary boundary feedback, making it a fully decision-based approach. Score-based methods like Square Attack [3] refine random perturbations over multiple iterations to create potent adversarial examples. Gradient-free optimizations further expand black-box attack capabilities by drawing on evolutionary algorithms [2, 50] or hard-label strategies [20], both of which operate solely on discrete class labels to reduce computational overhead. Although these approaches can produce more realistic test scenarios by relying exclusively on observable outputs, they can also become computationally expensive in systems with restrictive query limits.



Figure 1.6: Example of a targeted adversarial attack from [10]. Starting from an image of a dalmatian (top-left), the goal is to synthesize an image that matches the tiger cat (bottom-right) but is classified as a dalmatian. The attack relies solely on the model’s output predictions, iteratively refining the image until the misclassification is achieved.

Because black-box testing offers limited insight into a model’s internal structure, interpreting why certain adversarial manipulations succeed or fail remains a key challenge. XAI tools help bridge this gap by revealing the most vulnerable features or regions. For instance, LIME [89] perturbs inputs locally and fits simple, interpretable models to highlight pivotal features. SHAP [65] takes a broader approach by calculating game-theoretic Shapley values, pinpointing feature importance under various perturbation scenarios.

Recent innovations extend this framework even further. X-SHAP [9] ventures beyond additive explanations into the multiplicative domain, capturing nonlinear feature interactions. Other Shapley-based methods, such as those in [27], constrain explanations to remain on the data manifold, thus increasing their fidelity. SAGE

[23] broadens local explanations into a global context by quantifying how entire datasets affect predictions, highlighting systemic biases and feature interactions. Meanwhile, counterfactual explanations focus on minimal input changes needed to alter a prediction, laying bare the model’s most sensitive decision boundaries. Finally, researchers have proposed infidelity and sensitivity metrics [121] to enhance robustness in saliency maps; by smoothing input perturbations, these methods reduce abrupt shifts in explanation outputs and improve interpretative consistency across queries.

Lastly, infidelity and sensitivity metrics [121] offer a structured way to evaluate the reliability of saliency-based explanations by smoothing input perturbations and thereby preventing drastic changes in explanations. These tools help generate stable outputs that align closely with the model’s predictions. While the above techniques have been extensively explored in the visual domain, similar challenges and insights arise in the text domain as large language models (LLMs) become ubiquitous. Early work found that GPT-2 and GPT-3 models can inadvertently leak training data under adversarial queries [90], revealing privacy and security risks beyond computer vision. Further studies suggest that adversarial examples in NLP exploit fundamental data features rather than mere bugs [28], indicating these vulnerabilities are deeply embedded in how language classifiers learn from text.

Text-focused black-box attacks often mirror the strategies seen in vision but adapt them to language’s discrete, context-sensitive nature. Methods like BERT-Attack [51] and Explain2Attack [38] rely on masked language modeling or substitute models to systematically alter tokens while querying only for label feedback. More recent frameworks, such as HQA-Attack [62], integrate sophisticated search and optimization to maintain semantic consistency under restrictive hard-label settings. These text-based methods take into account not just raw model outputs but also linguistic coherence, ensuring adversarial examples remain fluent and natural.

XAI techniques also extend naturally to text: SIDU-TXT [41], for instance, leverages class activations to highlight influential words, complementing approaches like LIME or SHAP in capturing context-level explanations. Broad surveys [25] emphasize how attribution-based and fairness-aware methods expose and remediate biases in text classification. In parallel, efficient distillation of black-box adversarial attacks [29] aims to streamline adversarial testing, and hybrid adversarial-explainability frameworks [49, 68] combine query-based attacks with interpretability metrics, clarifying which tokens or linguistic features drive model decisions—and are thus prime adversarial targets.

Taken together, black-box adversarial testing and XAI play equally pivotal roles across both vision and language domains. While image-based pipelines must deal with continuous pixel data and spatial perturbations, text-based approaches

grapple with discrete token substitutions, syntax, and semantics. In both contexts, black-box attacks reveal vulnerabilities without requiring internal knowledge, and interpretability tools clarify why these manipulations succeed. However, query-based or transfer-based methods can be resource-intensive or reliant on imperfect surrogates, necessitating a careful balance between practicality and efficacy.

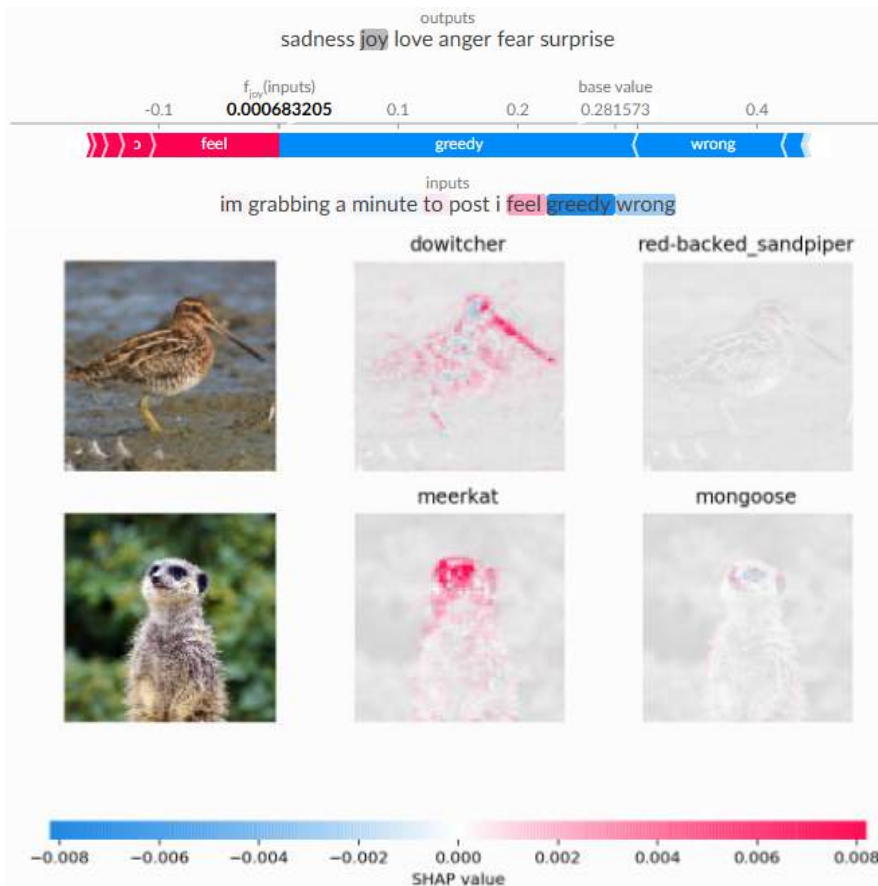


Figure 1.7: Model interpretability using SHAP [65] for text and image classification. In the top section, red highlights words like 'feel' that push the prediction towards the target class, while blue (e.g., 'greedy' and 'wrong') shows words that reduce it. The bottom section applies this to images, with red regions boosting classification (e.g., the 'meerkat' head) and blue reducing it. This explains how SHAP identifies key features influencing model decisions without accessing internal parameters.

1.3 Bridging White-Box Efficiency with Black-Box Flexibility

The interplay between white-box and black-box methods has significantly shaped adversarial testing in AI systems, particularly in uncovering reasoning flaws and biases. White-box approaches excel at creating in-distribution, semantically plausible examples that are finely tuned to expose biases through erroneous answers. However, this precision has fundamental limitations that increasingly restrict their practical relevance and interpretability. The requirement for complete access to model parameters is an assumption at odds with reality as proprietary, closed-source models become the industry standard. As access to model internals becomes the exception rather than the rule, traditional adversarial techniques grow less applicable to systems where robust evaluation is most crucial. Additionally, such methods usually rely on computationally intensive gradient calculations that scale poorly with model size, becoming prohibitively expensive for modern billion-parameter systems.

In contrast, while free from these constraints, black-box methods often struggle to match the diagnostic power of white-box techniques, leaving gaps in our understanding of reasoning and bias mitigation. Moreover, most black-box approaches predominantly focus on security implications rather than diagnosing fundamental reasoning deficiencies, with resulting perturbations providing minimal insight into the cognitive patterns underlying model failures.

This thesis addresses these dual limitations by introducing a unified approach that combines the semantic truthfulness of white-box adversarial examples with the scalability, accessibility, and realism of black-box frameworks. Our methodology produces interpretable perturbations highlighting reasoning failures in practical settings rather than merely identifying security vulnerabilities. By focusing on semantically meaningful modifications that maintain task coherence while targeting specific reasoning pathways, we reveal cognitive patterns more relevant to real-world deployment scenarios. We believe this approach should become an integral part of evaluating modern AI systems such as LLMs and VLMs, as it exposes critical weaknesses that conventional benchmarks systematically overlook while remaining applicable to any model regardless of architecture or whether internal access is possible.

In the visual domain, in **Chapter 2**, we leverage Visual Question Answering (VQA) as a natural proxy for visual reasoning, then recast it into a two-player zero-sum reinforcement learning framework, which allows us to propose fully semantic, in-distribution manipulations rather than contrived pixel-level perturbations, ensuring that each generated scene remains physically meaningful. Although we make no assumptions about model internals—retaining a strict black-

box perspective—we still achieve an effectiveness on par with white-box attacks, exposing the “*Clever Hans*”-type biases many ostensibly robust models rely upon. Crucially, our experiments reveal that question difficulty can outweigh raw visual complexity, highlighting that it is often *logical intricacy*—rather than sheer visual data size—that ultimately breaks a model’s reasoning.

We carry this philosophy into the realm of code generation in **Chapter 3**, where we demonstrate that large language model-based systems inherit biases from their pre-trained backends, giving rise to flawed reasoning processes systematically reproduced through memorization and superficial keyword dependencies. By introducing a modular “Blocks of Influence” paradigm, we pinpoint these memorization effects with minimal overhead, reinforcing our stance that lightweight but strategically designed black-box tests can expose substantial vulnerabilities—biases and data echoes—without needing extensive model access or excessive queries. The technique of *probing biases* in a strictly black-box manner aligns with our emphasis on real-world applicability, mirroring how we revealed “Clever Hans” shortcuts in the visual setting.

In **Chapter 4**, we extend these findings into a multi-agent, multi-modal setup that integrates visual complexity, textual comprehension, and specialized tool usage. By strategically distributing sub-tasks across collaborative, role-based agents—some focusing on natural-language problem decomposition and others on precise “geometry tools” or code modules—we show that each agent’s limitations can be counteracted by another’s strengths. This design resonates with our prior results in black-box testing, where different “modules” handled distinct facets of the adversarial process. We illustrate how multi-step, multi-role coordination can illuminate and mitigate the same kind of biases identified in purely visual or code-based models. Notably, the spatial and logical difficulty of geometrical tasks proves to be a strong stress test for reasoning, echoing our core premise that it is the complexity of the question—whether visual, textual, or mathematical—that best exposes a model’s weaknesses. Crucially, this multi-agent collaboration also serves as a logical endpoint for our earlier single-agent insights: having exposed individual biases and tested models’ reasoning limits in isolation, we now demonstrate that specialized agents working in tandem can overcome these narrower failures and collectively achieve more robust, context-aware solutions.

Altogether, the thesis not only underlines the need for holistic, dynamic evaluation frameworks but also indicates how semantic adversarial training and interactive, multi-agent collaboration can mitigate hidden biases, enabling AI systems to scale toward real-world challenges in a trustworthy and adaptive manner.

1.4 Formalisms

In all scenarios that we will refer to onward, we assume a deep learning model under test M , which receives inputs X from a dataset D and produces outputs Y . Formally, we write $M : X \in D \mapsto Y$. Because we are dealing with data-driven systems, we also distinguish between a training phase and a testing phase, thus requiring two datasets: D_{train} and D_{test} . During training, the model learns from input-target pairs $(X_{\text{train}}, Y_{\text{train}}) \subset D_{\text{train}}$. Afterwards, we evaluate its generalization capabilities by testing on $(X_{\text{test}}, Y_{\text{test}}) \subset D_{\text{test}}$. Note that D_{train} and D_{test} can be formed by splitting a larger dataset D or by using a completely separate data source. In the first case, there is little to no distribution shift, meaning the model is tested on data drawn from the same underlying distribution. In the second case, we aim for broader generalization, since the data may differ substantially and come from novel sources. This is the essence of testing a model’s capacity to handle unseen inputs. Although both scenarios involve “unseen” data, the latter typically provides a more extensive and realistic assessment of the model’s underlying capabilities.

The model is evaluated using a performance criterion C , which is typically a function that compares the model’s predictions $Y = M(X_{\text{test}})$ against the dataset’s ground-truth labels Y_{test} . Generally, “good” performance corresponds to lower values of C , although there are also metrics where higher values indicate better performance. For the purposes of this discussion, and without loss of generality, we assume criteria where lower values correspond to better performance. One major challenge in training deep learning models is overfitting, which can be described by $C(Y_{\text{test}}, Y_D) \gg C(Y_{\text{train}}, Y_D)$, meaning that the model’s performance on the testing data Y_{test} is worse than on the training data Y_{train} according to the same criterion C .

While overfitting illustrates how a model may fail when its training does not generalize well, testing for reasoning and discovering biases extends beyond mere performance degradation. When testing for bias, regardless of the dataset source, one must rely on available inputs X' that already yield acceptable performance (i.e., $M(X') = Y'$ and $C(Y', Y_{gt}) \leq \varepsilon$ for some acceptable error ε). We then introduce changes or as we will call from now on perturb X' , to get X'' . This perturbation can range from imperceptible changes (such as tiny pixel modifications) to more conspicuous manipulations (like inserting visual artifacts, reordering words in text, or swapping commands in instructions), with the expectation that the model’s output post perturbation should remain close to the original $M(X'') = M(X')$.

If the model’s predictions stay consistent despite these slight or irrelevant modifications, we gain confidence that its answers are not merely accidental products

of superficial correlations; rather, such robustness hints at more intelligent reasoning. Conversely, if small perturbations result in catastrophically different outputs, we expose vulnerabilities and biases that indicate a lack of genuine reasoning.

To further refine this framework, we introduce an optional reward mechanism R that leverages the discrepancy between Y'' and Y' to guide the perturbation process. The inclusion of R depends on whether the perturbation mechanism is parameterized or non-parameterized. For non-parameterized perturbations, which are applied heuristically or without systematic optimization, R may be omitted. Conversely, parameterized mechanisms can benefit from R to iteratively adjust perturbations based on feedback. The reward schema itself can range from a simple binary signal indicating whether the outputs match ($Y'' = Y'$) to more complex formulations quantifying the degree of deviation. In this thesis, we will explore both non-parameterized and parameterized approaches, highlighting their respective advantages and limitations. By incorporating these dynamic methodologies, we aim to systematically evaluate and improve the model's robustness and reasoning capabilities under diverse scenarios.

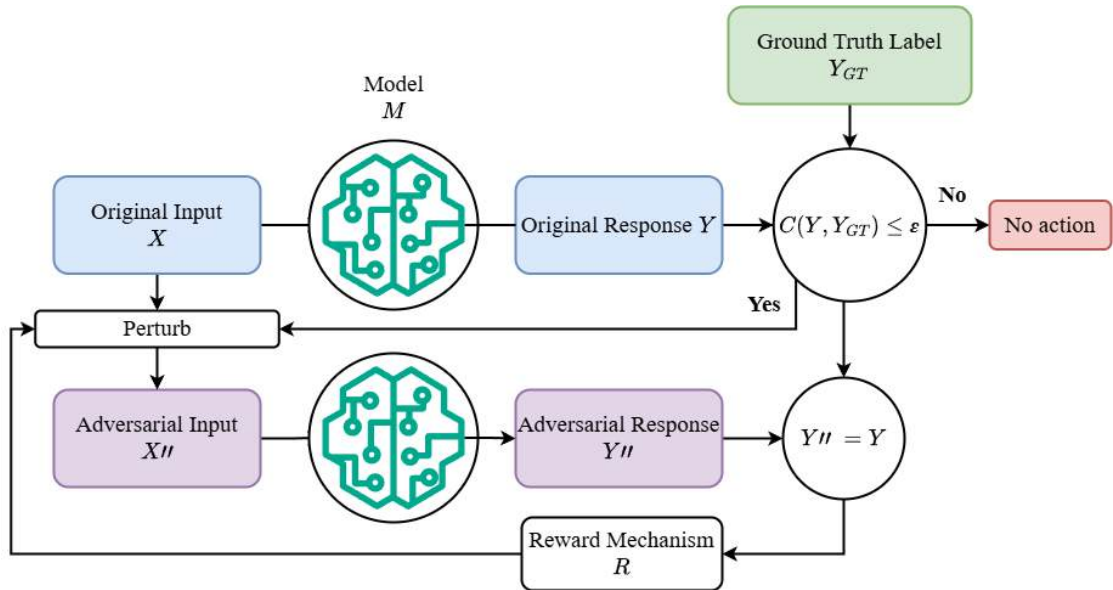


Figure 1.8: Illustration of adversarial testing for model robustness: A model M processes the original input X to produce a response Y , which is compared with the ground-truth label Y_{GT} using the performance criterion C . When $C(Y, Y_{GT}) \leq \varepsilon$, *label-preserving* perturbations are introduced to generate an adversarial input X'' (i.e., the true class implied by Y_{GT} does not change). If the model's adversarial response Y'' matches Y , the model is deemed robust; otherwise, the discrepancy reveals vulnerabilities. The result can then be reused as feedback, updating the perturbation process through the reward mechanism R .

Chapter 2

Testing Reasoning in the Visual Domain

Modern AI systems can appear capable of deep reasoning in tasks like Visual Question Answering (VQA). Nevertheless, their “impressive” results often hinge on shortcuts and biases reminiscent of the Clever Hans phenomenon. VQA aims to integrate computer vision and natural language understanding into a single pipeline, and its tasks have evolved from early conceptual works [70] to larger datasets and adversarial splits [6, 1]. However, questions persist about whether such models comprehend visual scenes or exploit superficial cues. Recent efforts to make VQA datasets more challenging (e.g., CLEVR [43]) have highlighted combinatorial reasoning—counting, relational structures, comparisons—to minimize trivial color or frequency biases. Although some models now surpass human baselines [91, 85, 40], slight modifications to a scene can still produce surprising failures.

We propose a novel black-box framework with white-box effectiveness to probe this issue, leveraging the original graphics engine CLEVR was built with rather than any learned generative model (e.g., VAE or diffusion). In this way, our approach ensures scene manipulations remain **in-distribution** and **semantically plausible**.

2.1 Proposed Two-Player Approach

We cast VQA as a *zero-sum reinforcement learning game* between two agents. On one side is the *Visual-QA Player*—the *model under test*—which attempts to accurately answer scene-based questions.

On the opposing side is an *Adversarial Player*, whose goal is to minimally reconfigure the scene to induce errors or inconsistencies in the *Visual-QA Player*’s responses. This agent operates under a strictly black-box setting—lacking access to the model’s parameters or internal states—yet employs an actor-critic reinforcement learning algorithm (A2C [75]) to uncover hidden vulnerabilities. Concretely, it has two primary components: First, we have the *Multi-Modal Module* that en-

codes the question and a structured scene representation - inspired by Relation Networks [91], to capture object-object and object-question relationships. Then the *Decision Maker* discretely shifts each object’s (x, y) coordinates (e.g., within 7 bins per axis), sampling from a learned policy to find manipulations most likely to confuse the *Visual-QA Player*

Because the *Adversarial Player* sees only the model’s final outputs (i.e., “old answers” vs. “new answers”), it must learn to trigger misclassifications without gradient-based guidance. Two critical constraints that maintain realism are *Question Relevance*, ensuring the ground-truth answer remains correct for humans, and *Semantic Plausibility*, preventing extreme manipulations beyond physically plausible object positions or scene layouts.

Rewards are structured to maximize *Accuracy Drop* (making a once-correct answer wrong) or *Consistency Drop* (altering the model’s answer, even if it was initially incorrect). In practice, these changes often appear imperceptible to human observers, indicating that the manipulated scenes still make sense but exploit fragile shortcuts in the Visual-QA Player’s reasoning.

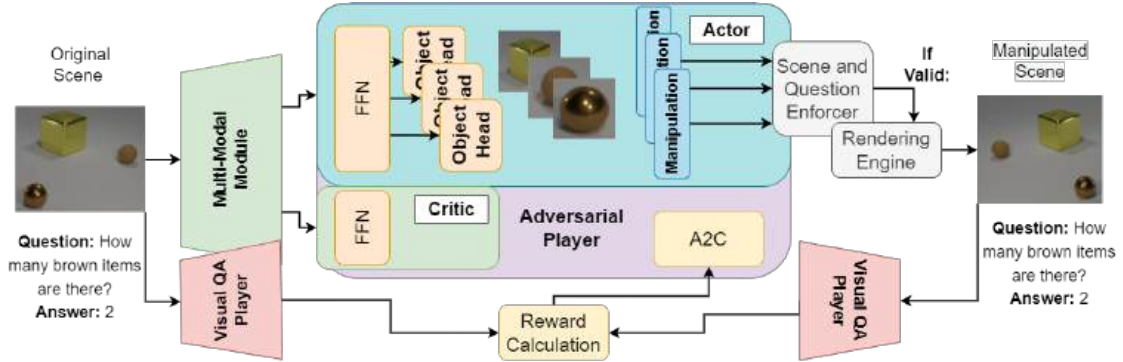


Figure 2.1: Our game between two players: *Adversarial Player* and *Visual-QA Player*. *Adversarial Player* uses a *multi-modal module* to extract features conditioned on the visual and textual inputs. After transforming such features with a feed-forward architecture, it samples an action using object-specific heads. Each action corresponds to manipulating the corresponding object in the scene. In the case of missing objects, we use an \emptyset token. After alternating the original scene graph, we use various environment enforcers to ensure validity of the constructed scene. A valid scene graph is rendered and introduced to the *Visual-QA Player* together with the original image. Finally, we collect responses of the *Visual-QA Player* and calculate suitable rewards based on them, and we repeat the whole cycle during the training phase.

2.2 Experiments and Observations

We evaluated our framework on several “superhuman-performance” VQA models, ranging from architectures trained from scratch (e.g., FiLM [85], TbD [72]) to state-of-the-art approaches (e.g., MDetr [47]) that achieve near-perfect accuracy in controlled benchmarks. To compare adversarial strategies, we deployed two variants of our *Adversarial Player*:

1. *State-input*, which processes structured scene graphs to semantically identify objects and attributes,
2. *Pixel-input* operates directly on rendered images and relies on raw pixel correlations.

Exposure of Hidden Biases and Shortcuts. Despite their reputations for near-human or "superhuman" performance, all tested models proved highly susceptible to minimal scene manipulations. This is particularly significant because CLEVR was widely considered a "solved" benchmark, with models like TbD [72] and MDetr [47] achieving near-perfect accuracy (99.1% and 99.7% respectively) on the original dataset. Notably, these accuracies substantially exceeded the human baseline of 92.6% reported in the original CLEVR paper, leading to claims of superhuman visual reasoning capabilities. Our adversarial manipulations, however, dramatically exposed the fragility of these seemingly impressive results. The performance drops were striking: FiLM’s [85] accuracy plummeted from 96.2% to 48.0%, while RN [91] fell from 93.2% to just 47.0%, representing the largest proportional drop among all models tested. Even the sophisticated module-based approach of IEP [44] collapsed from 96.9% to 48.0%. TbD [72], despite being custom-tailored to the task with explicit reasoning modules, still saw a significant decline from 99.1% to 69.0%, and maintained the highest post-manipulation accuracy. MDetr [47], despite achieving near-perfect 99.7% accuracy on the original benchmark, dropped to merely 60.0% when faced with our semantically coherent manipulations.

These substantial degradations in performance challenge the notion that these models had genuinely "solved" visual reasoning tasks and invalidate claims of superhuman performance, as all models fell significantly below the human baseline under our controlled, semantically valid manipulations. Even tiny shifts in object positions could derail previously correct answers, particularly for counting or relational queries. This strongly supports the "Clever Hans" hypothesis: models exploit superficial spatial patterns or color heuristics rather than learning robust multi-step reasoning. By interacting at the level of scene semantics (via the state-input variant), our Adversarial Player systematically uncovered biases in how each model mapped question tokens to object features. For instance, if a

model internally overweights a specific left-right positioning cue, moving a target object slightly to the right often flips the answer—exposing a heavy reliance on subtle positional correlations instead of truly understanding relational phrases like "behind" or "to the left of."

Human Evaluations. We also conducted a human study where some of these manipulated images and questions were presented to participants. Observers typically found no logical reason for the answer to change, indicating that the tweaks targeted model-specific shortcuts rather than introducing genuine scene ambiguity. This further underscores our main point: the model can be fooled by small, semantically valid shifts that remain imperceptible from a human perspective.

State vs. Pixel Input and the Limits of Data-Driven Robustness. Our experiments demonstrate that the *state-input* adversary consistently outperforms the *pixel-input* adversary. Even intensive *retraining* on adversarially manipulated scenes yielded only marginal gains in robustness, as newly devised manipulations continued to degrade accuracy. When we gradually increased the fraction of manipulated training data from 0% to 100%, performance improvements quickly plateaued—indicating that models never truly “cracked the code” of multi-hop logic. Crucially, we found that the complexity of the reasoning tasks far outweighed the number or variety of scene manipulations seen during training. For instance, the number of logical steps a query required (e.g., one-hop vs. two-hop relational questions) often overshadowed how many adversarial perturbations a model had encountered. These observations underscore the limitations of brute-force data augmentation: simply expanding dataset coverage, even to an extreme, cannot address fundamental reasoning deficits or fully guard against novel adversarial manipulations. Instead, our findings suggest that robust performance hinges on incorporating a diverse range of reasoning tasks into training. In particular, models trained on datasets encompassing both simple, single-step questions and more complex, multi-step reasoning achieved greater robustness with fewer examples than those relying on simpler datasets. This underscores the need to carefully balance reasoning complexity in dataset design to foster better generalization.

Ultimately, our results confirm that covering the combinatorial space of all possible scene manipulations is infeasible in practice, highlighting inherent *data-driven limitations*. We propose a *reasoning-aware* training strategy—one that integrates diverse reasoning tasks into dataset design and addresses the root causes of adversarial failures by encouraging deeper reasoning capabilities. We believe this approach is crucial for developing more robust and generalizable visual reasoning systems.

2.3 Conclusion

This chapter presented a two-player adversarial framework for systematically challenging VQA models’ visual reasoning. By casting CLEVR-based VQA as a zero-sum game, we observed how near-perfect accuracy rates break down under subtle, valid scene manipulations. Despite the synthetic domain, our results have important implications for real-world applications—ranging from robotic manipulation to safety-critical vision systems—where hidden biases can lead to catastrophic errors once the environment deviates slightly from training conditions. The methods and observations here provide a glimpse into the challenge of achieving true reasoning in AI.


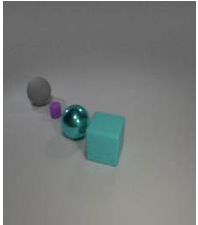

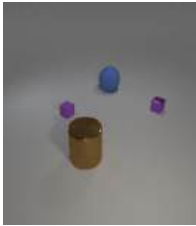



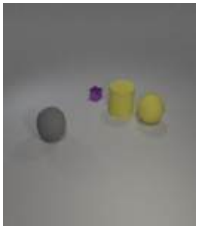



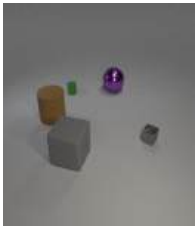
	What is the color of the block that is left of the big blue matte ball?	How many other objects are the same shape as the big gray matte object?	How many other things are the same shape as the big metallic thing?
FiLM - Before			
	Purple	1	1
FiLM - After			
	Brown	0	2
	What number of tiny matte cylinders are there?	Are there any other things that have the same material as the big yellow ball?	Is there a big brown object of the same shape as the green thing?
TbD - Before			
	2	Yes	Yes
TbD - After			
	1	No	No
	The red thing that is the same material as the brown cube is what size?	Is there anything else that has the same material as the large green cube?	What number of things are large metallic blocks on the left side of the brown metallic thing or big cyan shiny objects?

Figure 2.2: Results *before* and *after* scene manipulations of FiLM[85] and TbD[72] models.

Chapter 3

Exposing and Mitigating Biases in Code Generation

Building on the theme of Chapter 2, where seemingly “superhuman” models faltered under carefully structured test conditions, we now ask whether large language models truly *reason* when generating code or rely on superficial cues. Previous work on biases in large language models has primarily focused on *social biases* originating from massive training corpora, including patterns that lead to stereotypes or toxic outputs [110, 56]. Although interventions such as sanitizing examples and introducing fairness constraints have shown promise [127, 61], these techniques target a narrow range of issues. A broader class of *cognitive biases* poses additional challenges by introducing systematic reasoning errors that are less apparent in casual analysis. Code generation offers an even more stringent test of language-based reasoning in mathematical or textual tasks [69, 124]. Unlike standard text completion, coding must conform to strict syntax and demonstrate functional correctness, often admitting multiple valid solutions. Its linguistic dimension also interleaves with algorithmic thinking, making code generation a revealing benchmark for determining whether models are merely regurgitating patterns or truly solving problems [36, 19].

However, little research has addressed the unique cognitive or reasoning biases in code-specific tasks. Among this limited body of work, our research is one of the first to approach the problem from a semantic perspective. Our main contribution lies in manipulating semantic components while preserving logical task equivalence, which allows us to distinguish between models that genuinely understand algorithmic concepts and those that merely exploit statistical regularities in their training data. This semantic-space examination reveals deeper insights about model cognition that traditional benchmark evaluations and typical bias studies cannot capture. To enable this investigation, we developed the "Blocks of Influence" method, which systematically targets potential failure points in coding prompts—function names, keywords, and usage examples—and measures how

each removal affects performance. This automated, black-box approach diagnoses vulnerabilities in current code generation models and provides a framework for mitigating these biases in future systems.

3.1 Blocks of Influence Approach

We partition each coding task into three parts, collectively referred to as *Blocks of Influence*. Each block spotlights a different potential bias in code generation models:

The first block is the **Name Block**, including the function signature and argument details. We hypothesize that models frequently memorize training snippets, relying on function names or parameters as easy “shortcuts” to a solution. When a model initially succeeds but fails if the function name is replaced with something irrelevant (e.g., “*fun*”), it implies that the model never genuinely grasped the problem’s rationale. Instead, it relied on pre-existing matches from its training corpus, pointing to a memorization effect rather than true understanding.

The second block, the **Description Block**, houses the textual instructions or docstring. Removing or altering particular keywords (for instance, “reverse” or “index”) tests whether models can infer the intended goal from surrounding text, or if they break down without these lexical triggers, revealing an inherent preference bias, wherein the model becomes overly dependent on specific tokens or frequently co-occurring phrases seen during training.

Finally, the **Example Block** provides input-output demonstrations that clarify usage or constraints. Usage examples often serve as a “tie-breaker” when multiple plausible solutions exist, guiding the model by demonstrating the correct approach. Eliminating these examples reveals whether the model can handle the logical or algorithmic aspects alone, or if it has simply latched onto patterns from example input-output pairs.

```
def rev_list_take2(a: List):  
    "Reverse the list and return the second item."  
    Examples:  
>>> rev_list_take2(['a',2,'apple'])  
2
```

Figure 3.1: The three blocks of influence: Name Block in red, Description Block in green and Example Block in blue.

After splitting each prompt into these three parts, we apply transformations designed to expose biases. In the *Description Block*, we begin by identifying relevant keywords using a code-oriented, fine-tuned version of KeyBERT [32], ensuring

that only coding-relevant tokens are selected for removal. Next, a context-aware filter preserves semantic coherence by discarding modifications that would render the prompt unsolvable for a human (such as removing every mention of a list-based structure, leaving no clue about data types). The transformations themselves include dropping one or all selected keywords, anonymizing function names in the *Name Block*, or entirely removing the *Example Block*. After these manipulations, we perform an *LM critic test* [119, 120] to gauge plausibility; if a transformation severely distorts the prompt (leading to a drastic log probability drop), it is rejected. This sequence of analyses ensures that the tested disruptions remain subtle, yet sufficiently revealing to challenge models that rely on superficial cues rather than true reasoning.

3.2 Experiments and Observations

To assess how code generation models handle these manipulations, we performed experiments on three standard benchmarks: HumanEval [35], MBPP [100], and DMCC [53]. HumanEval and MBPP supply short function-based tasks, while DMCC includes more extensive competition-style prompts.

We conducted our experiments on five models chosen to span different scales and training paradigms. CodeParrot [106], being open-source and trained on publicly inspectable data, offers unique insight into memorization effects. InCoder [26] employs an infilling objective that enables more flexible handling of partial contexts, while Bloom’s [74] large multilingual pretraining highlights its performance in domains for which it was not explicitly fine-tuned. CodeGen [78], in contrast, focuses specifically on Python tasks and is optimized for those scenarios. Finally, Codex [60] stands out for achieving near-human or superhuman levels on established coding benchmarks, making it a valuable point of comparison when evaluating how these models react to adversarial manipulations.

Exposure of Biases and Memorization. Even minor modifications to the Name or Description Blocks often produced 15–20% drops in performance. For instance, removing keywords like “index” or “reverse” caused incorrect or incomplete solutions, indicating reliance on lexical triggers rather than genuine task understanding. Changing function names alone revealed a strong memorization effect: many successful solutions failed once identifiers diverged from known snippets. Anonymizing a function name and keyword deletions cut accuracy by almost 20%, respectively. Examples, too, proved crucial, and combining example removal with name anonymization pushed drops to 25%. The most severe transformation proved to be anonymization and removal of keywords, which resulted in a 40% decline, exposing a pronounced preference for matching patterns in docstrings or examples over constructing a logical solution. These results highlight a hierarchy in how the

model parses prompts: when the Description Block is intact, the system attempts to rely on explicit instructions, but once key tokens vanish, it falls back on function names or examples. If those are diminished or concealed, the model’s ability to reconstruct the underlying logic collapses. These findings underscore a reliance on *shortcuts* and demonstrate that none of the blocks, on its own, is sufficient to sustain robust reasoning in the face of even moderate adversarial transformations.

Robustness and Data-Driven Limits. After observing significant drops in performance with our prompt manipulations, we investigated whether re-training on these adversarially transformed samples might reduce model sensitivity to new disruptions. We chose CodeParrot—available in three parameter configurations—for an in-depth study, leveraging its open-source training corpus to track memorization effects. The *Blocks of Influence* transformations were applied to examples from the MBPP dataset to create an augmented fine-tuning set. At the same time, HumanEval (which does not overlap with MBPP) served as the evaluation set to avoid any direct exposure during training. Results showed moderate gains against the *previously seen* manipulations, particularly for the largest CodeParrot variant, aligning with scaling laws that suggest bigger models internalize a broader range of adversarial patterns. However, new or slightly altered manipulations still caused significant failures, echoing the pattern we observed in other large models like Bloom and Codex. Even with engineered adversarial examples, simply expanding the training data proved insufficient to cover the combinatorial space of function names, keywords, and example removals. Although adversarial re-training improved resilience to known distortions, it did not immunize the models to unseen variations, reinforcing the conclusion that deeper biases persist beyond what incremental data augmentation can fully address.

Long-Context Fine-Tuning. In addition to augmenting MBPP, we also explored how extended, competition-grade prompts might reduce reliance on single lexical items or function names. To this end, we fine-tuned models on the DMCC dataset, which features more verbose problems. Bloom—originally a large multilingual model not specialized for coding—demonstrated partial resilience when re-trained on DMCC: it handled keyword drops and name changes more gracefully, suggesting that denser contexts can mitigate certain biases. However, simultaneous disruptions (e.g., anonymizing a function name and removing crucial keywords) still caused failures, reaffirming that merely expanding context cannot guarantee robust, context-aware logic. We also ran similar experiments on CodeParrot but saw limited improvement, likely reflecting its narrower Python-centric training focus and smaller architecture. Altogether, these observations underscore that richer data can help models contend with adversarial edits, yet deeper biases persist unless the training process explicitly targets multi-faceted reasoning obstacles.

3.3 Conclusion

Studying code generation through the lens of *Blocks of Influence* highlights how language models rely on narrow lexical or naming patterns to achieve superficially high success rates. By selectively removing or changing function identifiers, keywords, or examples, we exposed structural weaknesses: even “superhuman” models break down without their favorite textual cues. Training on these manipulated prompts yields partial resilience, yet reaffirms that brute-forcing data variations cannot fully remedy logical deficiencies.

Compared to simpler math or language tasks, coding exemplifies a stricter domain where functional correctness and syntactic rules must align with conceptual reasoning. However, the patterns mirror those in visual reasoning: small perturbations can subvert performance, showing how data-driven approaches alone often fail to capture robust, context-aware logic. The parallels to previous chapters are clear: whether in visual or textual domains, subtle manipulations reveal a persistent reliance on shortcuts.

Beyond its diagnostic value, our "Blocks of Influence" framework is entirely model-agnostic. Regardless of architecture or scale, it can be readily applied to any current or future code generation system. This approach should become integral to standard model evaluation protocols because it exposes critical weaknesses that conventional benchmarks systematically overlook, preventing the premature deployment of seemingly capable but fundamentally brittle systems. By establishing semantic robustness as a core evaluation criterion alongside traditional accuracy metrics, our framework bridges the significant gap between controlled benchmark performance and real-world reliability—ultimately accelerating progress toward code generation models that genuinely understand programming concepts rather than exploiting statistical patterns.

Developing truly robust code generation likely requires approaches that integrate deeper structural reasoning, as we will explore in subsequent chapters when turning to multimodal and multi-agent solutions.

```

from typing import List
def has_close_elements(numbers: List, threshold: float):
    "Check if in the given list of numbers, are any two numbers closer to each other than given threshold."
    Examples:
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    return any(abs(x-y) < threshold for x,y in zip(numbers, numbers[1:]))

```

```

from typing import List
def has_close_elements(numbers: List, threshold: float):
    "Check if in the given list of numbers, are any two numbers closer to each other than given threshold."
    for i in range(len(numbers) - 1):
        if abs(numbers[i] - numbers[i+1]) < threshold:
            return True
    return False

```

Figure 3.2: Example removal reveals poor reasoning (Codex): The model initially exhibits signs of task comprehension (top), generating a correct solution. Removing the examples, however, reveals a lack of proper reasoning; Although the model still understands that it has to compare numbers, it resorts to a naive sequential check instead of comparing each available pair (bottom).

```

from typing import List
def string_xor(a:str, b:str) -> str:
    "Input is two strings a and b consisting only of 1s and 0s. Perform binary XOR on these inputs and return the result as a string."
    Examples:
    >>> string_xor('010', '110')
    '100'
    return ''.join([str(int(a,2) ^ int(b,2)) for a,b in zip(a,b)])

```

```

from typing import List
def string_xor(a:str, b:str) -> str:
    "Input is a and b consisting only of 1s and 0s. Perform XOR on these inputs and return the result."
    Examples:
    >>> string_xor('010', '110')
    '100'
    return ''.join([str(int(a) ^ int(b))])

```

Figure 3.3: Keyword hinting (Bloom 175B): After the removal of keywords, the context remains intact: The *two strings* keyword can be assumed by observing the function arguments, and the *binary/string* keywords by the examples and return type signature of the function. Nevertheless, the model fails to generate a correct solution (bottom).

```

def pairs_sum_to_zero(l):
    "Pairs_sum_to_zero takes a list of integers as input. It returns True if there are two distinct elements in the list that sum to zero,
    ↪ and False otherwise."
    Examples:
    >>> pairs_sum_to_zero([1,3,5,8])
    False
    >>> pairs_sum_to_zero([1,3,-2,1])
    False
    >>> pairs_sum_to_zero([2,4,-5,3,5,7])
    True
    for i in range(len(l)):
        for j in range(i+1, len(l)):
            if l[i] + l[j] == 0:
                return True
    return False

def func(l):
    "Func takes a list of integers as input. It returns True if there are two distinct elements in the list that sum to zero, and False
    ↪ otherwise."
    return any(sum(x) == 0 for x in l)

```

Figure 3.4: Anonymization + Keyword hinting (InCoder 6B): Using only the problem description, the model creates partially informed subparts (`any` derives from "if there are", `sum(x) == 0` from "sum to zero", and `for x in l` from "elements in the list") that are not combined correctly to solve the task (bottom), signifying that hints from the function name / examples were used in the correct solution (top).

Chapter 4

Vision meets Language in Geometric Agentic Reasoning

Building on the insights from Chapters 2 and 3, where we uncovered critical biases in visual question answering and code generation, we now turn to *constructive geometry*: an even stricter test of reasoning that demands tool-driven planning, spatial comprehension, and sophisticated linguistic cues. Although large language models and their multimodal counterparts (Visual Language Models - VLMs) [11, 79, 80, 4, 102] have made notable progress in mathematical tasks—whether presented purely in symbolic form or aided by schematic diagrams—they continue to struggle with geometry, a field widely viewed as foundational in human mathematical thought.

The limited prior research in AI-driven geometry typically treats geometric tasks as multiple-choice problems and depends on symbolic solvers linked to specialized domain-specific languages (DSLs) [94, 18, 125, 64]. By contrast, constructive geometry demands a more open-ended approach that unites stepwise planning, spatial tracking, and textual reasoning with 2D visualization [66, 115]. The groundbreaking, parallel effort AlphaGeometry [104] tackles IMO-level problems using a DSL-based LLM coupled with a theorem-prover. Our work, however, diverges by relying solely on open-ended LLMs and VLMs, foregoing the involvement of any external solver. This shift amplifies complexities surrounding biases in mathematical nomenclature, tool usage, and prompt design—subtleties that can derail even state-of-the-art "visual" models when they translate raw spatial cues into explicit construction procedures.

To meet these challenges, and inspired by the growing body of research on agents, personas, and inter-agent communication [117, 17, 59, 129, 112], we propose a multi-agent framework that unifies our adversarial insights with geometry-focused innovations. Although we anchor our results on the *Euclidea* benchmark [24, 86], we also demonstrate strong generalization on other mathematical datasets (e.g., GSM8K [22] and SVAMP [84]).

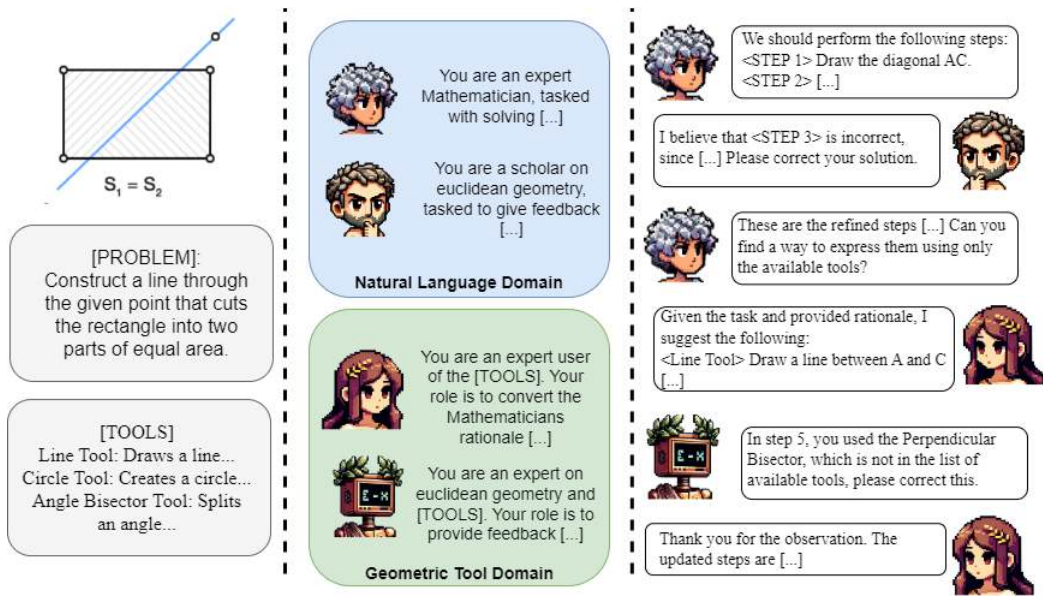


Figure 4.1: Overview of multi-agent design. The geometry task is split into an image, its textual instructions, and available tools. Four specialized agents engage in a multi-round dialogue, collectively solving the construction.

4.1 Method

A multi-agent framework is introduced for *constructive geometry*, extending prior chapters’ adversarial and modular insights to open-ended tasks that integrate textual instructions and visual cues. Typical geometry challenges face recurring pitfalls: alphabetical naming of points can add unnecessary steps or produce incomplete solutions; incorrect or "hallucinated" operations confuse geometry tools with other actions, and unstructured solution paths impede coherent multi-step reasoning. The proposed design counters these issues by organizing the reasoning process, mitigating naming biases, and carefully orchestrating how LLMs and VLMs plan each solution stage.

This framework is grounded in *Euclidea*, a puzzle environment featuring eight dedicated tools (lines, circles, etc.) and a series of construction tasks that grow progressively more complex. Unlike multiple-choice formulations, each Euclidea problem demands a precise sequence of geometric actions, which can be validated through a provided API. This open-ended format offers an ideal testbed for assessing model correctness and resilience in unconstrained geometry scenarios.

Adaptive Few Shot Mechanism. Inspired by how humans refine their reasoning by revisiting prior experiences, we adopt a *few-shot* methodology that reuses

the model’s solutions to reduce tool hallucinations and ground new challenges in past successes. Rather than relying on static or random exemplars, our *Adaptive-Shot* approach maintains a **memory bank** of previously solved Euclidean tasks, from which a Sentence Transformer filters out irrelevant entries and ranks the rest by similarity. Newly solved tasks are appended to this memory bank, allowing the system to "learn" in a feedback-driven loop reminiscent of human problem-solving. This dynamic process outperforms generic retrieval-augmented generation by minimizing irrelevant prompting and substantially reducing hallucinated steps.

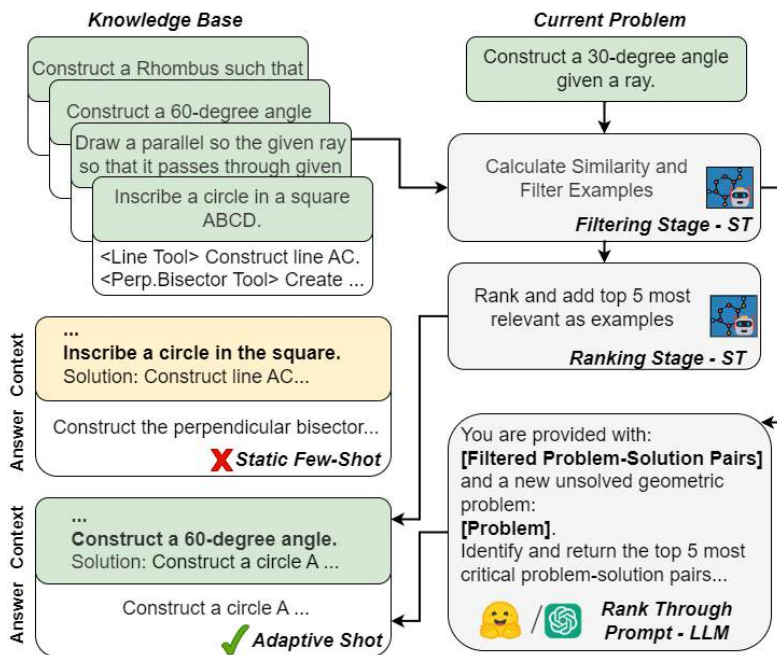


Figure 4.2: Adaptive prompting mechanism. For the problem “Construct a 30-degree angle given a ray,” we filter the knowledge base for relevant examples. The solver either adopts the top five (ST) or further narrows them (Self), leveraging past demonstrations to improve performance.

Multi-Agent Role Specialization (Simulacra). Inspired by multi-agent simulation [54, 83], we decouple high-level reasoning from mechanical tool usage, hypothesizing that domain-specific agents can reduce the naming biases and unstructured logic often seen in single-agent methods. Observing that LLMs can devise geometric solutions in natural language but fail at translating them into strict tool commands, we define distinct functional roles:

- **Natural Language Solver (S_{NL}):** Produces high-level rationales, emphasizing the *why* and *what* of each construction step without delving into formal

syntax.

- **Geometric Tool Solver** (S_{GT}): Interprets these language-based plans and converts them into Euclidean tool commands, managing lines, circles, and angle measures.
- **Validators**: Work in both the language and tool domains, comparing each proposal against canonical references and known errors. They flag flawed reasoning or incorrect steps, prompting revisions.

By restricting each agent’s scope, we ensure domain expertise and enable iterative corrections. Ablation tests showed that single-agent solvers performed poorly, but adding a language planner significantly improved accuracy. Introducing validators further enhanced performance by providing continuous feedback and specialization, reducing hallucinations and disorganized sequences.

Visual Relations Prompt (VRP). We additionally address scenarios involving 2D scene images. Despite LLMs being designed for linear, one-dimensional text processing, they can effectively tackle two-dimensional geometric reasoning when spatial information is appropriately translated into structured language representations. While some VLMs, like GPT-4V, can parse images directly, they may fail to integrate that visual recognition into coherent multi-step tool logic. Our *VRP* module accomplishes this translation by disentangling spatial recognition from the problem-solving process. It uses a Vision-Language model to analyze the scene image and extract structured information about points, objects, and their spatial relationships, outputting textual summaries of alignments, distances, or overlapping elements. This structured representation is converted into precise textual descriptions that leverage LLMs’ strong logical and symbolic reasoning capabilities. With this transformation, models can reason about geometric elements using the same mechanisms they apply to abstract logical problems. The approach is also flexible—non-visual LLMs benefit from these textual expansions as effectively as multimodal ones—ultimately reducing hallucinations such as connecting non-aligned points or introducing superfluous lines.

Naming Neutralization. Alphabetical labels can bias an LLM’s geometry solutions, prompting extra steps or abrupt endings. Shifting the name by one or two letters consistently degraded performance; conversely, replacing it with a neutral placeholder (e.g., "X") preserved minimal-step solutions. For instance, labeling a newly introduced point "E" after "A,B,C" often yields a redundant "D," while "X" averts such detours. Removing alphabetical triggers keeps the solver’s focus on core geometric logic rather than superficial naming heuristics. This strategy

parallels anonymizing function identifiers in code generation and proves effective at curtailing spurious behaviors.

4.2 Experiments and Observations

We conducted evaluations with a variety of LLMs, including LlamaV2 [103], Mistral [42], its fine-tuned variant Zephyr [107], MetaMath [122], and commercial systems like ChatGPT and GPT-4. We tested three configurations: a minimal few-shot setup that supplied each model with five examples and basic tool notes, a finetuning stage exposing open-source LLMs to classical geometry via *Euclid's Elements*, and our proposed multi-agent framework. Few-shot prompting provided only modest gains, failing to prevent hallucinations or frequent logical gaps. Finetuning on *Euclid's Elements* further bridged knowledge gaps, with MetaMath distinctly outperforming other open-source models after absorbing axiomatic geometry concepts. The most substantial improvements emerged under our multi-agent approach: not only did GPT-4 and ChatGPT doubled in performance but also mid-range systems like MetaMath saw large boosts, highlighting the impor-

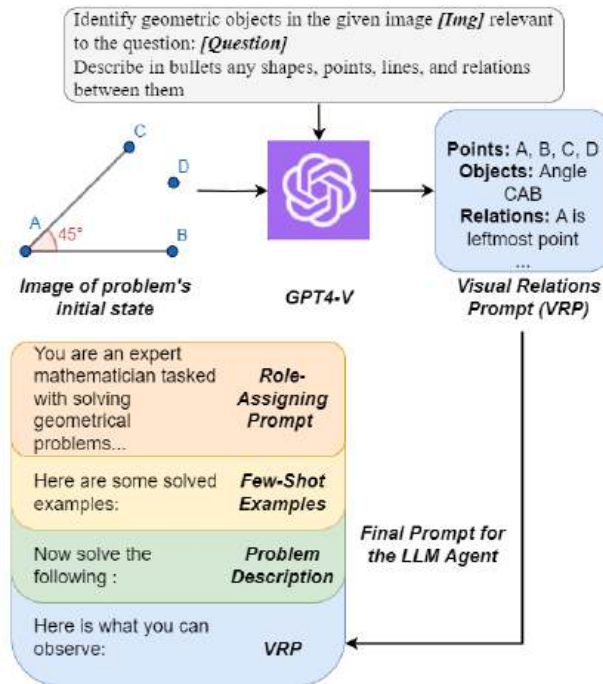


Figure 4.3: VRP extraction with GPT-4V. An auxiliary prompt contains the problem question and initial image; GPT-4V returns points, objects, and their relationships, which are then incorporated into the agents' prompts.

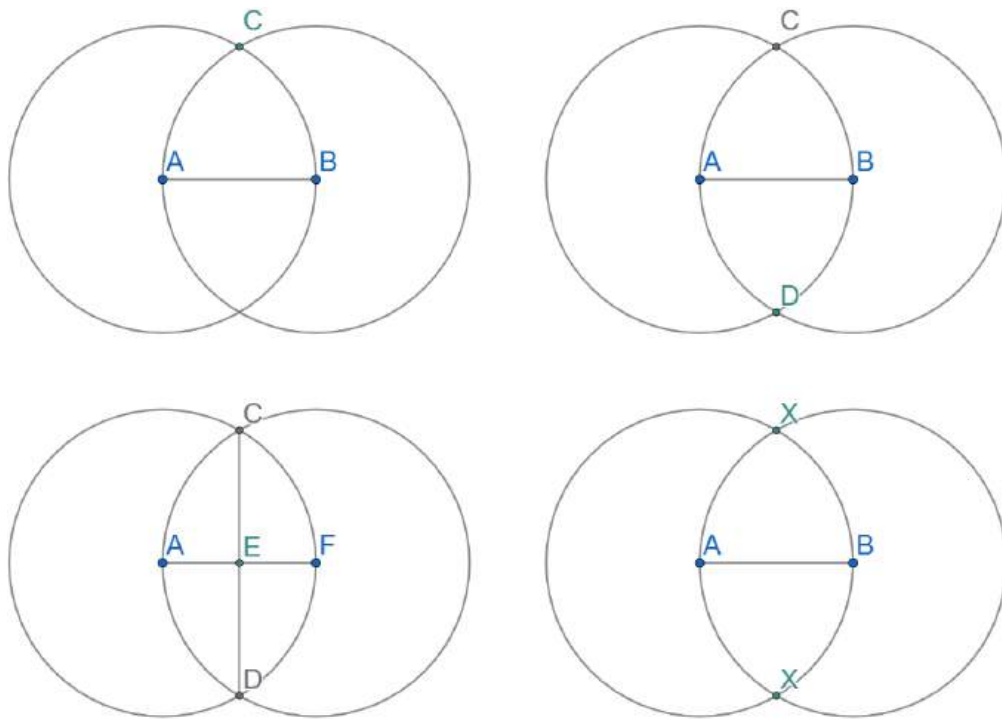


Figure 4.4: GPT-4 reasoning paths for “Find a point [Target] equidistant from A and B.” Different labels (C, D, E, or X) result in 3-, 4-, or 5-step solutions, with “X” reverting to a concise 3-step approach.

tance of structured multi-agent collaboration and targeted bias mitigation beyond merely scaling parameters or adding classical data.

Beyond geometry, we tested whether the same multi-agent strategies and adaptive prompts could be applied to GSM8K, SVAMP, and the geometry subset of MATH by substituting Euclidea’s line/circle operations with arithmetic operators and a Python interpreter. Our approach again neared state-of-the-art performance, reflecting the flexibility of role-based collaboration and adaptive few-shot retrieval. Finally, a small user study on advanced Euclidea problems found that even adept human participants averaged around 30% success, underscoring the intrinsic complexity of open-ended constructive geometry.

4.3 Conclusion

Geometric reasoning weaves together spatial, symbolic, and procedural logic, revealing vulnerabilities that often remain hidden in more conventional tasks. Our multi-agent framework addresses these subtleties by assigning specialized roles,

selectively reusing prior solutions, neutralizing naming biases, and enriching geometry prompts with textual expansions of visual cues. Although progress in constructive geometry remains uneven—models can still hallucinate excessive steps, misjudge angles, or ignore simpler alternatives—we demonstrate that structured role collaboration, adaptive few-shot retrieval, and neutral naming substantially reduce such pitfalls.

Unlike specialized approaches such as AlphaGeometry [104], which are explicitly designed for overspecialized tasks with narrow application domains, we deliberately focused on enhancing the capabilities of general-purpose LLMs and VLMs. This choice reflects our broader research philosophy: rather than pursuing state-of-the-art performance on isolated benchmarks through specialized systems, we prioritize understanding and improving fundamental reasoning capabilities that can operate across multiple domains. The effectiveness of this approach became evident when we demonstrated that our framework extends beyond geometry. By simply swapping out Euclidea’s geometric operations for algebraic functions or code-like syntax, our multi-agent system yielded near state-of-the-art results on datasets such as GSM8K, SVAMP, and MATH, reinforcing that our methods can successfully enhance reasoning across diverse fields without requiring domain-specific architectures.

These findings connect directly with the overarching argument of this thesis: many AI systems, though appearing robust on standard tests, fail when confronted with tightly coupled multimodal constraints and complex domain-specific requirements. Our careful dissection of geometry tasks shows how coordinated agent roles, feedback loops, and structured representations of spatial relationships can achieve a more faithful approximation of human-like reasoning. The following chapter explores the broader implications of these strategies and their potential applications to other domains where stepwise logic and cross-modal reasoning converge.

Chapter 5

Conclusions and Future Directions

This dissertation has explored a range of strategies to enhance the reasoning capabilities of AI systems, with a particular focus on visual reasoning, code generation, and multimodal tasks. Over the course of this research, adversarial black-box testing methods were employed, linguistic and visual biases were examined, multi-agent collaboration was investigated, and new frameworks and mitigation strategies were developed. A central theme of this work is the identification of reasoning limitations in both large language models and visual models, coupled with the design of innovative approaches to uncover and address those limitations.

Looking ahead, several promising research directions emerge. One involves the deeper integration of adversarial testing with large-scale pretraining, followed by task-specific fine-tuning. This approach aligns with current LLM paradigms that rely on broad, large-scale training, subsequently refined through domain-specific prompting. Another avenue is the use of collaborative multi-agent frameworks, in which specialized reasoning capabilities are distributed among multiple agents. Such collaborative environments not only facilitate the solution of complex, multi-step tasks but also reflect a broader focus in the AI community on coordination, adaptability, and dynamic problem-solving.

Continual learning also represents a critical area for future work. Iteratively updating models in response to new data and environments can significantly enhance robustness and more closely mirror human learning. In addition, this continual approach can gradually mitigate biases, particularly when paired with adversarial testing techniques and domain-oriented fine-tuning strategies.

5.1 Recent Developments Reinforcing These Contributions

Soon after the compilation of the works that make up this dissertation, a series of advances in AI research has underscored the importance of its central themes:

AI Reasoning Agents Several new architectures that address planning, tool use, and complex reasoning have garnered attention. A recent survey [73] illustrates how these agents achieve more intricate goals through enhanced reasoning and planning.

Vision-Language Model Reasoning In the vision-language domain, innovations aimed at mitigating biases and strengthening reasoning have progressed substantially. For instance, LLaVA-CoT [118] introduces a vision-language model tailored for systematic, multistage visual reasoning, while [52] proposes methods to improve LLM-based visual reasoning in more complex scenarios.

Virtual Tokens for Thought Processes Recent work has also delved into specialized tokens that facilitate internal deliberation in AI systems. Liao and Vargas [57] present a technique to optimize attention mechanisms by extending token computation in the chain-of-thought process, reflecting similar strategies used by the tool-using agents in this dissertation. This method rectifies inefficient attention distribution and enhances reasoning. Similarly, [55] discusses scaling up inference-time computation by generating multiple reasoning paths and enlisting domain-specific (mathematics and coding) LLMs as verifiers, thereby evaluating and ranking outputs by correctness.

Taken together, these developments point to a growing emphasis on more transparent, human-like “inner monologues” in AI, leading to greater coherence and more extensive planning horizons.

Chapter 6

Acknowledgements

I would like to express my deepest gratitude to Henryk Michalewski and Mateusz Malinowski for their unwavering trust, guidance, and mentorship throughout my research journey. Their continuous support and encouragement have shaped me into the researcher I am today. Henryk, your persistent belief in me during challenging times kept me motivated to push forward, and your guidance has been invaluable in every stage of this journey. Mateusz, I am immensely thankful for your mentorship and for trusting me with a place at Moonvalley AI, where I continue to develop my research in text and vision modalities for video generation.

I am very thankful to my family for their constant support and sacrifices, which made it possible for me to pursue this academic path in Poland. Their love and encouragement have kept me grounded throughout this journey. I would also like to extend my heartfelt appreciation to my invaluable friend and fellow PhD student Dimitris that I have met during my time here.

Finally, I would like to thank the University of Warsaw for hosting me and providing an inspiring place to work and study throughout my academic journey.

Bibliography

- [1] Agrawal and Batra and Parikh and others. Don't just assume; look and answer: Overcoming priors for visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [2] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Chou-Jui Hsieh, and Mani B. Srivastava. Genattack: practical black-box attacks with gradient-free optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19*, page 1111–1119, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361118. doi: 10.1145/3321707.3321749. URL <https://doi.org/10.1145/3321707.3321749>.
- [3] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search, 2020.
- [4] Anil and Dai and Firat and others. Palm 2 technical report, 2023.
- [5] Anonymous. Ebgcg: Effective white-box jailbreak attack against large language model. *OpenReview*, 2024. ACL ARR 2024 June Submission.
- [6] Antol and Agrawal and Lu and others. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [7] Sebastian et al. Bach. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. In *Proceedings of the International Conference on Artificial Intelligence (IJCAI)*, 2015.
- [8] Bengio. Adversarial examples in the physical world. *CoRR*, 2016. URL <http://arxiv.org/abs/1607.02533>.
- [9] Luisa Bouneder, Yannick Léo, and Aimé Lachapelle. X-SHAP: towards multiplicative explainability of machine learning. *CoRR*, abs/2006.04574, 2020. URL <https://arxiv.org/abs/2006.04574>.

- [10] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *ICLR (Poster)*, 2018. URL <https://openreview.net/forum?id=SyZI0GWCZ>.
- [11] Brown and Mann and Ryder and others. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020. URL <https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf>.
- [12] Pritam et al. Chattopadhyay. Grad-cam++: Improved visual explanations for deep convolutional networks. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018.
- [13] Chen, Pin-Yu, Zhang, et al. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec '17*, page 15–26, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450352024. doi: 10.1145/3128572.3140448. URL <https://doi.org/10.1145/3128572.3140448>.
- [14] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. Hopskipjumpattack: A query-efficient decision-based attack, 2020. URL <https://arxiv.org/abs/1904.02144>.
- [15] Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, et al. PaLI: A jointly-scaled multilingual language-image model. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=mWVoBz4W0u>.
- [16] Zhe Chen, Wang, et al. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *arXiv preprint arXiv:2404.16821*, 2024.
- [17] Chen and Dong and Shu and others. Autoagents: A framework for automatic agent generation, 2023.
- [18] Chen and Li and Qin and others. Unigeo: Unifying geometry logical reasoning via reformulating mathematical expression. In *The 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022.
- [19] Chen and Tworek and Jun and others. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

- [20] Minhao Cheng, Tuan Anh Le, Pin-Yu Chen, Jinfeng Yi, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. In *International Conference on Learning Representations (ICLR)*, 2019.
- [21] Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.
- [22] Cobbe and Kosaraju and Bavarian and others. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- [23] Ian C. Covert, Scott Lundberg, and Su-In Lee. Understanding global feature contributions with additive importance measures. In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20*. Curran Associates Inc., 2020. ISBN 9781713829546.
- [24] Euclidea. Euclidea game. URL <https://www.euclidea.xyz/>.
- [25] Vitalii Fishchuk and Daniel Braun. Efficient black-box adversarial attacks on neural text detectors. In Mourad Abbas and Abed Alhakim Freihat, editors, *Proceedings of the 6th International Conference on Natural Language and Speech Processing (ICNLSP 2023)*, pages 78–83, Online, December 2023. Association for Computational Linguistics. URL <https://aclanthology.org/2023.icnlsp-1.8/>.
- [26] Fried and Aghajanyan and Lin and others. InCoder: A generative model for code infilling and synthesis. *arXiv preprint arXiv:2204.05999*, 2022.
- [27] Christopher Frye, Damien de Mijolla, Tom Begley, Laurence Cowton, Megan Stanley, and Ilya Feige. Shapley explainability on the data manifold. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=OPyWRcjVQw>.
- [28] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops, SP Workshops 2018, San Francisco, CA, USA, May 24, 2018*, pages 50–56. IEEE Computer Society, 2018. doi: 10.1109/SPW.2018.00016. URL <https://doi.org/10.1109/SPW.2018.00016>.
- [29] Yotam Gil, Yoav Chai, Or Gorodissky, and Jonathan Berant. White-to-black: Efficient distillation of black-box adversarial attacks. In Jill Burstein,

- Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1373–1379, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1139. URL <https://aclanthology.org/N19-1139/>.
- [30] Goodfellow and Shlens and Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL <http://arxiv.org/abs/1412.6572>.
- [31] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- [32] Grootendorst. Keybert: Minimal keyword extraction with bert., 2020. URL <https://doi.org/10.5281/zenodo.4461265>.
- [33] Chuan Guo, Alexandre Sablayrolles, Hervé Jégou, and Douwe Kiela. Gradient-based adversarial attacks against text transformers. *arXiv preprint arXiv:2104.13733*, 2021.
- [34] Stevan Harnad. The truly total turing test. *Minds and Machines*, 8:263–272, 1998. doi: 10.1023/A:1008229619541.
- [35] Hendrycks and Basart and Kadavath and others. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.
- [36] Hendrycks and Burns and Kadavath and others. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.
- [37] José Hernández-Orallo. *The Measure of All Minds: Evaluating Natural and Artificial Intelligence*. Cambridge University Press, 2017. doi: 10.1017/9781316594179.
- [38] Mahmoud Hossam, Trung Le, He Zhao, and Dinh Phung. Explain2attack: Text adversarial attacks via cross-domain interpretability. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 8922–8928, 2021. doi: 10.1109/ICPR48806.2021.9412526.
- [39] Yangsibo Huang, Samyak Gupta, Mengzhou Xia, Kai Li, and Danqi Chen. Catastrophic jailbreak of open-source llms via exploiting generation, 2023. URL <https://arxiv.org/abs/2310.06987>.
- [40] Hudson and Manning. Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067*, 2018.

- [41] Mohammad N. S. Jahromi, Satya. M. Muddamsetty, Asta Sofie Stage Jarlner, Anna Murphy Høgenhaug, Thomas Gammeltoft-Hansen, and Thomas B. Moeslund. Sidu-txt: An xai algorithm for nlp with a holistic assessment approach, 2024. URL <https://arxiv.org/abs/2402.03043>.
- [42] Jiang and Sablayrolles and Mensch and others. Mistral 7b, 2023.
- [43] Johnson and Hariharan and van der Maaten and others. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [44] Johnson and Hariharan and van der Maaten and others. Inferring and executing programs for visual reasoning. In *ICCV*, 2017.
- [45] Pratik Joshi, K. Karthikeyan, and Soumya Ghosh. Semantic adversarial perturbations for text classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2019.
- [46] John M. Jumper, Richard Evans, Alexander Pritzel, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.
- [47] Kamath and Singh and LeCun and others. Mdetr–modulated detection for end-to-end multi-modal understanding. *arXiv preprint arXiv:2104.12763*, 2021.
- [48] Satoshi Kanazawa. General intelligence as a domain-specific adaptation. *Psychological Review*, 111(2):512, 2004. doi: 10.1037/0033-295X.111.2.512.
- [49] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 17061–17084. PMLR, 23–29 Jul 2023. URL <https://proceedings.mlr.press/v202/kirchenbauer23a.html>.
- [50] Raz Lapid, Zvika Haramaty, and Moshe Sipper. An evolutionary, gradient-free, query-efficient, black-box algorithm for generating adversarial instances in deep convolutional neural networks. *Algorithms*, 15(11), 2022. ISSN 1999-4893. doi: 10.3390/a15110407. URL <https://www.mdpi.com/1999-4893/15/11/407>.

- [51] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. BERT-ATTACK: Adversarial attack against BERT using BERT. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6193–6202, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.500. URL <https://aclanthology.org/2020.emnlp-main.500/>.
- [52] Zhiyuan Li, Dongnan Liu, Chaoyi Zhang, Heng Wang, Tengfei Xue, and Weidong Cai. Enhancing advanced visual reasoning ability of large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1915–1929, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.114. URL <https://aclanthology.org/2024.emnlp-main.114/>.
- [53] Li and Choi and Chung and others. Competition-level code generation with alphacode, 2022. URL <https://arxiv.org/abs/2203.07814>.
- [54] Li and Hammoud and Itani and others. Camel: Communicative agents for "mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.
- [55] Zhenwen Liang, Ye Liu, Tong Niu, Xiangliang Zhang, Yingbo Zhou, and Semih Yavuz. Improving llm reasoning through scaling inference computation with collaborative verification, 2024. URL <https://arxiv.org/abs/2410.05318>.
- [56] Liang and Wu and Morency and others. Towards understanding and mitigating social biases in language models. In *International Conference on Machine Learning*, 2021.
- [57] Bingli Liao and Danilo Vasconcellos Vargas. Extending token computation for llm reasoning, 2024. URL <https://arxiv.org/abs/2403.14932>.
- [58] Zeyi Liao and Huan Sun. AmpleGCG: Learning a universal and transferable generative model of adversarial suffixes for jailbreaking both open and closed LLMs. In *First Conference on Language Modeling*, 2024. URL <https://openreview.net/forum?id=UfqzXg95I5>.
- [59] Lin and Zhao and Zhang and others. Agentsims: An open-source sandbox for large language model evaluation, 2023.

- [60] Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. *CoRR*, 2021.
- [61] Liu. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. Association for Computational Linguistics, May 2022. URL <https://aclanthology.org/2022.deelio-1.10>.
- [62] Han Liu, Zhi Xu, Xiaotong Zhang, Feng Zhang, Fenglong Ma, Hongyang Chen, Hong Yu, and Xianchao Zhang. Hqa-attack: Toward high quality black-box hard-label adversarial attack on text, 2024. URL <https://arxiv.org/abs/2402.01806>.
- [63] Haohan Liu, Xingquan Zuo, Hai Huang, and Xing Wan. Saliency map-based local white-box adversarial attack against deep neural networks. In *Artificial Intelligence: Second CAAI International Conference, CICA 2022, Beijing, China, August 27–28, 2022, Revised Selected Papers, Part II*, page 3–14, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-20499-9. URL https://doi.org/10.1007/978-3-031-20500-2_1.
- [64] Lu. Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, August 2021. URL <https://aclanthology.org/2021.acl-long.528>.
- [65] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.
- [66] Macke and Sedlar and Olsak and others. Learning to solve geometric construction problems from images. 2021.
- [67] Aleksander Madry, Alexander Makelov, Ludwig Schmidt, Dimitris Tsipras, and Andrei Vladu. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
- [68] Rishabh Maheshwary, Saket Maheshwary, and Vikram Pudi. Generating natural language attacks in a hard label black box setting. In *AAAI Confer-*

- ence on Artificial Intelligence, 2020. URL <https://api.semanticscholar.org/CorpusID:229923395>.
- [69] Malinowski. Measuring and improving bert’s mathematical abilities by predicting the order of reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 2: Short Papers), Virtual Event, August 1-6, 2021*. Association for Computational Linguistics, 2021.
- [70] Malinowski and Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. *Advances in neural information processing systems*, 2014.
- [71] Gary Marcus, Francesca Rossi, and Manuela Veloso. Beyond the turing test. *AI Magazine*, 37(1):3–4, 2016. doi: 10.1609/aimag.v37i1.2650.
- [72] Mascharka and Tran and Soklaski and others. Transparency by design: Closing the gap between performance and interpretability in visual reasoning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [73] Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey, 2024. URL <https://arxiv.org/abs/2404.11584>.
- [74] Mitchell and Pistilli and Jernite and others. The world’s largest open multilingual language model: Bloom, 2022. URL <https://bigscience.huggingface.co/blog/bloom>.
- [75] Mnih and Badia and Mirza and others. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, 20–22 Jun 2016. URL <https://proceedings.mlr.press/v48/mniha16.html>.
- [76] Grégoire et al. Montavon. Layer-wise relevance propagation: A unified framework for interpreting predictions of deep neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.
- [77] Moosavi-Dezfooli and Fawzi and Fawzi and others. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

- [78] Nijkamp and Pang and Hayashi and others. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint*, 2022.
- [79] OpenAI. Gpt-4 technical report, 2023.
- [80] OpenAI. Gpt-4o system card, 2024. URL <https://openai.com/index/hello-gpt-4o/>. Accessed: 2024-01-08.
- [81] Nicolas Papernot, Patrick McDaniel, and Xinlei et al. Wu. The limitations of deep learning in adversarial settings. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016.
- [82] Papernot and McDaniel and Goodfellow and others. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017.
- [83] Park and Popowski and Cai and others. Social simulacra: Creating populated prototypes for social computing systems. 2022. URL <https://arxiv.org/pdf/2208.04024.pdf>.
- [84] Patel. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, June 2021. URL <https://aclanthology.org/2021.naacl-main.168>.
- [85] Perez and Strub and Vries and others. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.
- [86] PyEuclidea. Pyeuclidea. URL https://github.com/mirefek/py_euclidea.
- [87] Hao Qiu, Leonardo Lucio Custode, and Giovanni Iacca. Black-box adversarial attacks using evolution strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '21*, page 1827–1833, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383516. doi: 10.1145/3449726.3463137. URL <https://doi.org/10.1145/3449726.3463137>.
- [88] John C. Raven. Mental tests used in genetic studies: The performance of related individuals on tests mainly educative and mainly reproductive. Msc thesis, University of London, 1936.

- [89] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939778. URL <https://doi.org/10.1145/2939672.2939778>.
- [90] Suranjana Samanta and Sameep Mehta. Towards crafting text adversarial samples, 2017. URL <https://arxiv.org/abs/1707.02812>.
- [91] Santoro and Raposo and Barrett and others. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems*, 2017.
- [92] Leo Schwinn, David Dobre, Sophie Xhonneux, Gauthier Gidel, and Stephan Günnemann. Soft prompt threats: Attacking safety alignment and unlearning in open-source llms through the embedding space. *ArXiv*, abs/2402.09063, 2024. URL <https://api.semanticscholar.org/CorpusID:267657556>.
- [93] Selvaraju and Cogswell and Das and others. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.
- [94] Seo. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, September 2015. URL <https://aclanthology.org/D15-1171>.
- [95] David Silver, Aja Huang, Chris J Maddison, et al. Mastering the game of go with deep neural networks and tree search, 2016.
- [96] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013.
- [97] Daniel et al. Smilkov. Smoothgrad: removing noise by adding noise. In *arXiv preprint arXiv:1706.03825*, 2017.
- [98] Jost Tobias et al. Springenberg. Striving for simplicity: The all convolutional net. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

- [99] Mukund Sundararajan, Ankur Taly, and Zhang Yanqiang. Axiomatic attribution for deep networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.
- [100] Sutton. Program synthesis with large language models. *CoRR*, 2021. URL <https://arxiv.org/abs/2108.07732>.
- [101] Szegedy and Zaremba and Sutskever and others. Intriguing properties of neural networks, 2014.
- [102] Gemini Team. Gemini: A Family of Highly Capable Multimodal Models. *arXiv e-prints*, 2023.
- [103] Touvron and Martin and Stone and others. Llama 2: Open foundation and fine-tuned chat models, 2023.
- [104] Trinh and Wu and Le and others. Solving olympiad geometry without human demonstrations. *Nature*, 2024.
- [105] Trustworthy-AI-Group. Transferattack: A pytorch framework to boost adversarial transferability for image classification. <https://github.com/Trustworthy-AI-Group/TransferAttack>, 2024. Accessed: 2025-01-05.
- [106] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. Natural language processing with transformers, 2022.
- [107] Tunstall and Beeching and Lambert and others. Zephyr: Direct distillation of lm alignment, 2023.
- [108] Turing. Computing machinery and intelligence. In *Parsing the turing test*. Springer, 2009.
- [109] Wagner. Towards evaluating the robustness of neural networks. *CoRR*, 2016. URL <http://arxiv.org/abs/1608.04644>.
- [110] Wallace. Universal adversarial triggers for attacking and analyzing NLP. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, November 2019. URL <https://aclanthology.org/D19-1221>.
- [111] Zongwei Wang, Min Gao, Junliang Yu, Xinyi Gao, Quoc Viet Hung Nguyen, Shazia Sadiq, and Hongzhi Yin. Llm-powered text simulation attack against id-free recommender systems, 2024. URL <https://arxiv.org/abs/2409.11690>.

- [112] Wang and Peng and Que and others. Rolellm: Benchmarking, eliciting, and enhancing role-playing abilities of large language models. *arXiv preprint arXiv: 2310.00746*, 2023.
- [113] David Wechsler. *The Measurement and Appraisal of Adult Intelligence*. Williams & Wilkins Co., 4th edition, 1958. doi: 10.1037/11167-000.
- [114] Nevan Wichers, Carson Denison, and Ahmad Beirami. Gradient-based red teaming. In Yvette Graham and Matthew Purver, editors, *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2862–2881, St. Julian’s, Malta, March 2024. Association for Computational Linguistics. URL <https://aclanthology.org/2024.eacl-long.175/>.
- [115] Wong and Qi and Tan. Euclidnet: Deep visual reasoning for constructible problems in geometry. *Adv. Artif. Intell. Mach. Learn.*, 2022. URL <https://api.semanticscholar.org/CorpusID:256389437>.
- [116] Tao Wu, Tie Luo, and Donald C. Wunsch. LRS: Enhancing adversarial transferability through Lipschitz regularized surrogate. *arXiv preprint arXiv:2312.13118*, 2023.
- [117] Wu and Bansal and Zhang and others. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023.
- [118] Guowei Xu, Peng Jin, Hao Li, Yibing Song, Lichao Sun, and Li Yuan. Llavacot: Let vision language models reason step-by-step, 2025. URL <https://arxiv.org/abs/2411.10440>.
- [119] Yasunaga and Leskovec and Liang. Lm-critic: Language models for unsupervised grammatical error correction. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.
- [120] Yasunaga and Liang. Break-it-fix-it: Unsupervised learning for program repair. In *ICML*, 2021. URL <http://proceedings.mlr.press/v139/yasunaga21a.html>.
- [121] Chih-Kuan Yeh, Cheng-Yu Hsieh, Arun Sai Suggala, David I. Inouye, and Pradeep Ravikumar. On the (in)fidelity and sensitivity of explanations, 2019.
- [122] Yu and Jiang and Shi and others. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

- [123] Jinghao Zhang, Yuting Liu, et al. Stealthy attack on large language model based recommendation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5839–5857, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.318. URL <https://aclanthology.org/2024.acl-long.318/>.
- [124] Zhang and Li and Meng and others. On the paradox of learning to reason from data. *arXiv preprint arXiv:2205.11502*, 2022.
- [125] Zhang and Yin and Liu. A multi-modal neural geometric solver with textual clauses parsed from diagram. In *International Joint Conference on Artificial Intelligence*, 2023. URL <https://api.semanticscholar.org/CorpusID:257078982>.
- [126] Yiran Zhao, Wenyue Zheng, Tianle Cai, Xuan Long Do, Kenji Kawaguchi, Anirudh Goyal, and Michael Shieh. Accelerating greedy coordinate gradient via probe sampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- [127] Zhao and Wallace and Feng and others. Calibrate before use: Improving few-shot performance of language models. In *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/zhao21c.html>.
- [128] Zhenglong Zhou and Chaz Firestone. Taking a machine’s perspective: Human deciphering of adversarial images. *CoRR*, abs/1809.04120, 2018. URL <http://arxiv.org/abs/1809.04120>.
- [129] Zhou and Jiang and Li and others. Agents: An open-source framework for autonomous language agents, 2023.

Complete Publications

MEASURING CLEVRNESS: BLACK-BOX TESTING OF VISUAL REASONING MODELS

Spyridon Mouselinos

University of Warsaw
Warsaw, Poland
s.mouselinos@uw.edu.pl

Henryk Michalewski

University of Warsaw, Google
Oxford, U.K.
henrykm@google.com

Mateusz Malinowski

DeepMind
London, U.K.
mateuszm@deepmind.com

ABSTRACT

How can we measure the reasoning capabilities of intelligence systems? Visual question answering provides a convenient framework for testing the model’s abilities by interrogating the model through questions about the scene. However, despite scores of various visual QA datasets and architectures, which sometimes yield even a super-human performance, the question of whether those architectures can actually reason remains open to debate. To answer this, we extend the visual question answering framework and propose the following behavioral test in the form of a two-player game. We consider black-box neural models of CLEVR. These models are trained on a diagnostic dataset benchmarking reasoning. Next, we train an adversarial player that re-configures the scene to fool the CLEVR model. We show that CLEVR models, which otherwise could perform at a “human level”, can easily be fooled by our agent. Our results put in doubt whether data-driven approaches can do reasoning without exploiting the numerous biases that are often present in those datasets. Finally, we also propose a controlled experiment measuring the efficiency of such models to learn and perform reasoning.

1 INTRODUCTION

Are our artificial intelligence systems capable of reasoning? Or like *Clever Hans*, they use various cues only tangentially related to the task and rely on rote memorization with poor generalization? (Pfungst, 1911; Johnson et al., 2017a) This work revisits such a question and proposes an interactive framework with the communication channel between two players. The first player, which reasoning capabilities we are about to test, performs visual reasoning tasks, we call it *Visual-QA Player*. The second player, which we call the *Adversarial Player*, is manipulating the scene so that it *fools* the first player even though those changes still lead to correct reasoning steps among humans. Both players interact with each other only through questions, answers and the visual scene as shown in Figure 1. If the *Adversarial Player* manipulating the scene causes the *Visual-QA Player* to change its answer even though the new scene is still valid for the same question and answer, it is then the reasoning failure. It is similar to the following situation. Imagine a box is placed between two spheres. If you ask a question, *is there a box between two spheres?*, the answer should be positive. Now, if we move the box anywhere so it does not cross any of the spheres, and ask the same question, the response should remain unchanged. In other words, we postulate that reasoning outputs of agents need to be invariant under scene configurations that are consistent with the questions-answer pairs. Moreover, in the spirit of generic adversarial attacks, we seek configurations that also pose little if any reasoning challenges for humans.

We propose an automatic and agnostic pipeline to benchmark the reasoning capabilities of various models, only assuming they can communicate by answering questions about the scene. Due to the recent stream of research in vision-and-language (Zhang et al., 2021; Jiang et al., 2020; Guo et al.,

2019b; Su et al., 2020; Wang et al., 2020; Kamath et al., 2021; Tan & Bansal, 2019; Chen et al., 2020), we believe there will be an increasing number of vision models that operate through language. Moreover, we also consider the visual question answering framework set-up as a two-player system as an excellent benchmarking pipeline. We perform all tests by scene manipulations and observing how a tested model behaves under such changes. The pipeline does not require any knowledge of the internals of the tested model. It also does not manipulate the sensory information of such a model, e.g., pixels in the images, and all the manipulations are physically meaningful. Even though our current pipeline uses synthetic scenes as only those can easily be automatically manipulated, our results have also real-world ramifications. If models are susceptible to semantically *meaningless* changes¹ in scene configurations, in a synthetic setting, there are valid concerns that real-world robots could also be prone to manipulation of objects in a room. Finally, our work also questions the possibility of training and benchmarking networks in a purely data-driven and offline, static manner.

Contributions. The main contributions of our work could be summarized in three points.

First, we propose a strong *black-box* adversarial test, which makes no assumptions about the underlying mechanics of a tested model, formulated as a game between two players. Our test does not require any direct access to the tested model, even through its sensory information. In particular, it does not require gradients, output probabilities, or any access to the perceived image. Our work also deviates from bounded perturbations and instead focuses on global scene manipulations that are still consistent with the task constraints, and can change the behavior of a tested model.

Second, we reformulate visual reasoning by integrating visual question answering with zero-sum two-player game frameworks. Under our novel formulation, a visual and adversary agents compete against each other through content manipulation. We believe that this is an initial step towards more sophisticated frameworks that integrate computer vision with multi-agent systems.

Third, we explore the limits of the data-driven approaches in synthetic visual scenarios, and demonstrate that current CLEVR models are lacking the efficiency to learn robust reasoning steps.

2 RELATED WORK

Our work touches upon various research directions, which we briefly describe here.

Visual QA. Introduced as a visual counter-part of the Turing Test (Malinowski & Fritz, 2014; Geman et al., 2015), it became a computer vision task that requires a holistic visual understanding. Many other works have extended the task to larger datasets or videos or adversarial train-test splits (Antol et al., 2015; Agrawal et al., 2018; Tapaswi et al., 2016; Yu et al., 2019). More recently, we also observe the generalization of the task to become a part of the vision-plus-language suite of benchmarks (Lu et al., 2019; Chen et al., 2019; Wang et al., 2021). Johnson et al. (2017a) have introduced a synthetic variant of the Visual QA problem that is more focused on reasoning and the control of the experimentation. Although it was shown that traditional methods (Yang et al., 2016) are not enough to solve that dataset, newer methods can pass the human baseline on this task. These methods are trained to operate on pixels and text directly (Santoro et al., 2017; Perez et al., 2018; Hudson & Manning, 2018; Malinowski et al., 2018; Kamath et al., 2021) or they induce programs as an intermediate representation (Johnson et al., 2017b; Mascharka et al., 2018; Yi et al., 2018). Hudson & Manning (2019) have shown that Visual QA models lack some reasoning robustness but their approach is mostly linguistic and still static. Some other works also observe issues with static evaluation and proposed more dynamic benchmarks, e.g., with a human-in-the-loop (Khashabi et al., 2021; Nie et al., 2020; Li et al., 2021; Sheng et al., 2021). We extend the line of research on Visual QA by reformulating it as an interactive two-agents system, and show that CLEVR under such a new setting is still an unsolved problem. Our work also differs from previous works in that it focuses on visual reasoning, is interactive and fully automatic.

Adversarial perturbations. Szegedy et al. (2014); Goodfellow et al. (2015); Kurakin et al. (2016); Carlini & Wagner (2016) have introduced early methods that can ‘fool’ networks by performing tiny, visually imperceptible, perturbations of the input image. Moosavi-Dezfooli et al. (2017) have also shown universal and untargeted adversarial perturbations that are more transferable than previous approaches. Most methods perform *white-box* attacks, where an adversarial model has access to the target model’s parameters or gradients. In *black-box* attacks such access is removed and, e.g., only output probabilities are available (Guo et al., 2019a). Other *black-box* attacks involve the use

¹Changes that are consistent with the task constraints.

of surrogates to estimate gradients of the real target (Papernot et al., 2017; Cheng et al., 2019) or gradient-free methods (Alzantot et al., 2018). More related to our work, Cheng et al. (2018) describe a *black-box* system, which only assumes access to outputs of the network. However, all the perturbations described above are continuous-based where individual pixels are changed independently. This leads to improbable images and assumes access to the sensory inputs of the neural network. In contrary, *semantic* perturbations (Joshi et al., 2019; Zeng et al., 2019), operate on semantically meaningful chunks of the input, leading to more plausible and physically interpretable perturbations. However, they still assume direct access to either images or ideally differentiable renderers. We can interpret our work within the adversarial perturbations framework, where the adversarial model can change the original scene by its manipulations, with zero access to the target model, including its sensory information and not limited to tiny changes but instead to semantically and globally *meaningless* modifications.

Reinforcement learning. The game between the *Visual-QA Player* and *Adversarial Player* is a zero-sum two-player game. Conneau et al. (2017) use a conceptually similar two-player game to find a better word alignment between two languages without parallel corpus. Inspired by such a problem formulation, we consider our two-player game as a tool to achieve better reasoning models. We optimize the *Adversarial Player* with Advantage Actor-Critic (A2C) algorithm (Sutton & Barto, 2018; Degris et al., 2012).

Probing and measuring intelligence. Turing Test (Turing, 2009) is considered to be among the earliest works on measuring the intelligence of artificial systems. Crucially to us, it is also formulated as a two-player game with an interrogation protocol. Legg (2008) systematizes and relates the notion of intelligence from different fields with the main focus on ‘universality’. Johnson et al. (2017a) have transferred the notion of intelligence onto the visual ground and emphasized reasoning rather than universality. It seems there is no widely accepted dataset or definition that encompasses our intuitions about intelligence. However, there are a few recent directions showing the lack thereof, mainly highlighting bias amplification or absence of mathematical capabilities (Hendricks et al., 2018; Bhardwaj et al., 2021; Piekos et al., 2021).

3 PRELIMINARIES

In this section, we explain briefly how the CLEVR dataset (Johnson et al., 2017a) is constructed and introduce our notation and definitions.

CLEVR is a synthetic visual question answering dataset introduced by Johnson et al. (2017a), which consists of about 700k training and 150k validation image-question-answer triplets. Images are artificially constructed and rendered from scene graphs – a special structure containing information about object attributes such as position or color. Such a scene graph is also used to synthesize the ground-truth question-answer pairs by expanding templates according to the depth-first-search ordering. Ambiguous scenes are rejected. Each image represents an isometric view of the scene containing from two to ten objects. There are three classes of objects, *spheres*, *cubes* and *cylinders*. Each object can also be either large or small and has one color out of four (brown, purple, cyan, yellow). It can also be either metallic or rubber-made. Every object has x and y coordinates that are confined within the $(-3, +3)$ range. We use the same generation process to render modified scenes.

CLEVR models. Various models have been introduced to work with the CLEVR dataset, some even ‘solving’ the dataset by achieving near perfect performance. Despite the strong offline performance, we test if those models’ performance perpetuates in the more interactive setting where configurations of the scene could be changed. Whenever possible, we use pre-trained CLEVR models. Otherwise, we train the remaining models from scratch by making sure we achieve results similar to published accuracy numbers on the validation set. We summarize all the models in Table 1. We show the accuracy on the CLEVR dataset (*Accuracy*), indicate if an architecture is trained from scratch (*Re-trained*), briefly describe how multi-modal fusion and reasoning is conducted (*Reasoning Mechanism*), and indicate any extra privileged information required during the training process (*Extra*). For instance, some models require extra access to functional programs used during the dataset generation, use scene graphs as a supervisory signal (states), or always operate on scene graphs (input-states). Otherwise, the models were trained only from image-question-answer triples.

Mini-games. We formulate our problem as a *Game* between two players, *Visual-QA Player* and *Adversarial Player*. The *Visual-QA Player* takes as input question-image pairs and provide answers to such questions. Some models use states (scene-graphs) that replace images or require programs (Johnson et al., 2017a). The whole game consists of all CLEVR data points. For our

purpose, we extend the notion of the *Game* into *Mini-games*. The rules of *Mini-games* are identical to the whole *Game*. The only difference is that each *Mini-game* operates on a subset of the CLEVR dataset. We define the size of a *Mini-game* by the number of datapoints that are attached to that *Mini-game*. We sample data points for each *Mini-game* randomly and mutually exclusively. *Mini-games* have analogies in the adversarial perturbations literature. *Mini-games* of size one resemble per-image adversarial perturbations (Goodfellow et al., 2015; Moosavi-Dezfooli et al., 2016) whereas a *Mini-game* that has all data points is similar to universal adversarial perturbations (Moosavi-Dezfooli et al., 2017). In this work, we investigate various *Mini-game* sizes but due to the sheer scale we were unable to use the whole game as the *Mini-game*. Larger *Mini-games* make the optimization process more difficult as the domain where the *Adversarial Player* needs to operate increases. The training is also much more time-consuming and a sequential process. Instead, we can train multiple players on different *Mini-games* independently and thus massively. We leave the arduous training of the universal *Adversarial Player* on the whole *Game* as a possible future direction.

Table 1: CLEVR models that we use as *Visual-QA Players*.

Model Name	Accuracy	Re-trained	Reasoning Mechanism	Extra Data
SAN (Yang et al., 2016)	72.1		Attention	
FiLM (Perez et al., 2018)	96.2	✓	Feature Conditioning	
RN (Santoro et al., 2017)	93.2	✓	Relational	
IEP (Johnson et al., 2017b)	96.9		Neural Program Induction	Programs
TbD (Mascharka et al., 2018)	99.1		Neural Program Induction	Programs
Mdetr (Kamath et al., 2021)	99.7		Multimodal Transformer Querying	States
State-Input Transformer (ours)	96.8	✓	Cross Attention	Input-States

4 ENVIRONMENT

We need to ensure that *Adversarial Players* create valid scenes that are *consistent* and *in-distribution*. Both properties are guaranteed by our environment enforcers.

Consistency. Since scene manipulation may change the answer for a given question, we need to ensure this does not happen. That is, the new scene is still *consistent* with the question-answer pair. The question-relevance enforcer achieves that by running functional programs associated with each question (Johnson et al., 2017a) on the modified scene-graph. Hence, it gets the new ground-truth answer. The enforcer rejects the new scene if that new answer differs from the previous one. In this way, it guarantees the newly generated scenes give the same answers as the original scenes on the same question. Thus, we can generate equivalent scenes containing the same objects that have identical answers for the same questions. Using that enforcer, we can test if the *Visual-QA Player*’s answers are invariant under such an equivalent class of scenes.

In-distribution. Even with the question-relevance enforcer, the *Adversarial Player* may still produce undesired outputs. For instance, it can stretch the whole scene thus violating the scene boundaries from the original CLEVR dataset, making, e.g., everything to look very small (Section A.9 in the appendix). Although it is still an interesting form of adversarial scene manipulation, we focus rather on the *in-distribution* scene manipulations that respect the original boundaries. To enforce that property, we use a scene-constraint enforcer that checks the boundaries of the scene. Without that enforcer, the *Adversarial Player* would quickly resort to stretching the whole scenes, achieving a form of adversarial attack that uses distribution shifts rather than content manipulation. It does so, e.g., by moving the camera away until objects are barely visible. We give a few such examples in the appendix (Section A.9).

5 Adversarial Player

Meaningful scene manipulations require not only generic scene understanding, but also the ability to distinguish which objects to displace and how. Hence, the player is a composition of a multi-modal module, which creates input representation, and a decision maker, which decides how to control the scene. Figure 1 illustrates the *Adversarial Player* and the game between both players.

Multi-modal module. We have experimented with the same multi-modal modules as those in Table 1, but found out we have a better performance and the convergence rate if the *Adversarial Player*

operates on the scene-graphs (states) instead of pixels. For that, we use *state-input* variant of Relation Networks (Santoro et al., 2017). The model receives as the input $10 * 6$ object tokens, and question tokens. Every object token represents one-out-of-ten possible objects in the scene by its attributes such as position, color, shape, material and size. If the scene has fewer than ten objects, we use \emptyset token to indicate that, which also acts as a padding. We also have special tokens that separate questions from the objects which we add as a latent embedding, e.g., $\text{emb}(\text{material}) + \text{emb}(\text{object})$. Such an input encoding is similar to our *State-Input Transformer* and described in Section A.3. The embedded vectors are given to the Relation Network (RN). Finally, we train that network on the CLEVR visual question answering task, where we achieve 97.6% on the validation set, and use the representation just after the last relational layer for the decision maker.

Decision maker. Inspired by the work on reinforcement learning (Mnih et al., 2016; Wu et al., 2017; Lillicrap et al., 2016; Wang et al., 2016; Schulman et al., 2015), we use an actor-critic module that acts on scenes. The actor is a general-purpose fully connected layer with ten object-specific heads. Each head is randomly assigned to a unique object in the scene for its manipulation. Every head produces a displacement in x and y coordinates of the corresponding object. Although we have initially experimented with the continuous output space, we have found out the following simple strategy is more effective. First, we discretize all the x and y coordinates into N bins each. Now, each head produces two N -dimensional vectors that are next projected into a probabilistic space via softmax. Next, we sample displacements in x and y axis independently from both softmax distributions. Note that, even though we do not model the joint distribution explicitly due to computational reasons, both samples condition on the common head and thus are only *conditionally* independent of each other. We discretize the scene where each axis has values in $[-3, 3]$ onto $N = 7$ bins per axis. Our critic is a simple three layer feed-forward network (with relu as activations) that predicts a reward score between -1 and $+1$ via tanh activation ($1.2 * \tanh$ for better numerical properties).

The game of scene manipulations. Due to our formulation of *Adversarial Player* and the environment, we can benchmark various reasoning models purely in the *black-box* setting via a series of questions about the scene. *Adversarial Player* manipulates the scene so that it is still consistent with the question-answer pair. The manipulations are applied to scene-graphs, and the resulting scene-graph is evaluated by the environment enforcers described in Section 4. Invalid scenes are thus discarded. In this way, we ensure the *in-distribution* and *consistency* in the scene generation. Original image-question pairs are fed to a *Visual-QA Player* that produces corresponding answers. We refer to that answers as *old answers*. After the scene manipulation, new images paired with the same questions are also given to the *Visual-QA Player* that produces *new answers*. We construct rewards based on *old answers*, *new answers* and *ground-truth answers*.

If the *Adversarial Player* forces the *Visual-QA Player* to change the answer, i.e., an *old answer* is different than a *new answer*, it gets *Consistency Drop Reward* (cr). If an *old answer* is the *ground-truth* answer, it gets instead *Accuracy Drop Reward* (dr). Both rewards differentiate between simply confusing the model and causing a drop in its performance. If *Adversarial Player* produces an

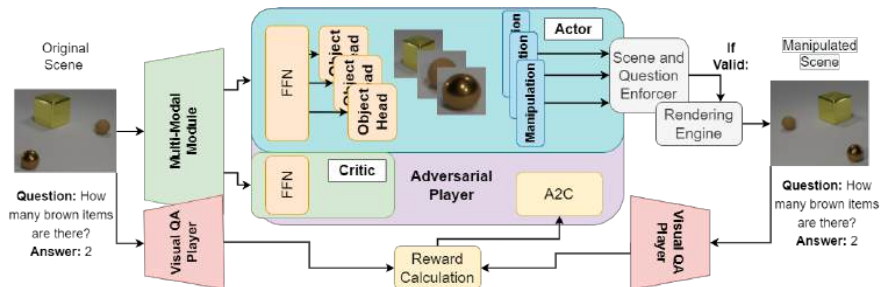


Figure 1: Our game between two players: *Adversarial Player* and *Visual-QA Player*. *Adversarial Player* uses a *multi-modal module* to extract features conditioned on the visual and textual inputs. After transforming such features with a feed-forward architecture, it samples an action using object-specific heads. Each action corresponds to manipulating the corresponding object in the scene. In the case of missing objects, we use an \emptyset token. After alternating the original scene graph, we use various environment enforcers to ensure validity of the constructed scene. A valid scene graph is rendered and introduced to the *Visual-QA Player* together with the original image. Finally, we collect responses of the *Visual-QA Player* and calculate suitable rewards based on them, and we repeat the whole cycle during the training phase.

invalid scene it gets *Invalid Scene Reward* (isr). This reward encourages producing scenes that pass the environment enforcers tests. Finally, if *Adversarial Player* does not manage to fool the model, it gets *Fail Reward* (fr). We use the following values: $dr=1$, $cr=0.1$, $fr=-0.1$, $isr=-0.8$.

Training algorithm. To train *Adversarial Player* we use the A2C algorithm with the episode length set to one as we do not need to model long-range consequences of the decision-making mechanism. Batches contain images, question, answers, programs and scene-graphs. We train the *Adversarial Player* for each *Mini-game* independently using the same architecture. We experiment with the following *Mini-game* sizes 10, 100, 1000. All *Mini-games* are constructed randomly. Under our discretization scheme, the action space is N^k where N is the number of bins and k is the number of objects in the scene. In practice, it is up to 49^{10} .

6 EXPERIMENTS

We consider *Adversarial Player* with a multi-modal module pre-trained either on states (*state-input*) or directly from pixels (*pixel-input*). We train *Adversarial Player* for each *Mini-game* independently. We use different *Mini-game* sizes in our experiments: 10, 100 and 1000. To obtain statistically significant results, we run each *Mini-game* thirty times with randomly initialized *Adversarial Players*, each with different seed per trial. We report the results that are averaged over all those runs. We compute two metrics for each *Mini-game*. *Consistency* refers to the fraction of times *Adversarial Player* has changed the *Visual-QA Player*’s answer, regardless if that answer was correct. *Drop* refers to the fraction of times *Adversarial Player* has changed the *correct Visual-QA Player*’s answer. We aggregate the results using two statistics. *Average Accuracy* averages accuracies over all *Mini-games*. *Maximal Accuracy* refers to the worst (best) case of a single *Mini-game* from the *Visual-QA Player*’s (*Adversarial Player*’s) perspective. We also computed p -values of T-Test, with the null hypothesis that a manipulation is unsuccessful. In almost all cases we reject the null hypothesis with small p -values. The p -values are available in the appendix (Section A.11).

Quantitative results. We show our results in Table 2 for the *state-input* and *pixel-input Adversarial Players*, and for each *Visual-QA Player* presented in Table 1. All the *Visual-QA Players* in the comparison are trained from pixels, apart from our custom *State-Input Transformer* architecture (Section A.3 in the appendix). When comparing both *Adversarial Players*, we can observe that the *state-input Adversarial Player* is significantly more successful than the *pixel-input* one. We hypothesize that this is the effect of having more structured and less ambiguous input information. It could also be the case that multi-modal modules trained from pixels ignore features in the same way and hence smaller discrepancy between all models operating on pixels. Moreover, as the size of *Mini-games* increases, it is becoming more difficult for the *Adversarial Player* to successfully manipulate the model on all examples from the *Mini-game*. This is an expected behavior resembling standard adversarial perturbations (Goodfellow et al., 2015) and their universal variants (Moosavi-Dezfooli et al., 2017). The highest performing CLEVR models, TbD and MDetr, are the most robust under the manipulation. However, they are still susceptible to the scene manipulations in some *Mini-games* as the *Maximal Consistency* metric indicates, especially if our *Adversarial Player* uses states as the input. Section A.8 (appendix) also shows richer performance statistics. Our detailed investigation shows that the models above are particularly sensitive to manipulations of scenes associated with *counting* or *existence* questions. Such question are often more complex, hence, increasing the likelihood of the reasoning failure at some stage (Section A.6 in the appendix).

Qualitative results. Figure 3 shows qualitative results of the scene manipulations for the three highest performing and distinctive models from Table 1; e.g., simple feed-forward or program induction models. For the reader’s convenience, we provide in Section A.2 (appendix) examples of CLEVR object attributes. We can observe that the scene manipulations are surprisingly minimal and are semantically *meaningless*, i.e., they should not ‘fool’ human players.

Reasoning steps and visual cues. To better understand the source of errors (reasoning steps vs visual cues), we have conducted the following experiments. In the first experiment, shown in Section B.4 (appendix), we visualize the attention mechanisms of *Visual-QA Players* before and after the scene manipulations. We observe changes in the attention map of all models but MDetr. The results suggest, most models’s perception is impacted by the manipulations. However, all objects are still correctly identified by MDetr, thus suggesting some issues are also stemming from the reasoning steps. In the second experiment, we deployed Slot-attention object detector (Locatello, 2020), and show that the detector identifies the same properties before and after manipulations. This result (Section B.6; appendix) shows that *Adversarial Player* does not produce ‘corner cases’ of the

Table 2: Results of the games: *Model dataset performance* refers to models’ accuracy on the CLEVR dataset. *Model Mini-game performance* refers to the model accuracy on the *Mini-game* examples where the largest (maximal) performance drop was detected, before manipulations took effect. The *Average Accuracy* column reports the performance of models averaged over all runs of the respective *Mini-game* sizes. *Maximal Accuracy* reports the worst model performance among the respective *Mini-game* size runs. We also report in brackets the relative performance drop, in percentages $\frac{X-Y}{X}\%$. Average Accuracy is compared against *Model dataset performance* ($X = \text{Model dataset performance} / Y = \text{State/Pixel-Input}$). Maximal Accuracy is compared against *Model Mini-game performance* ($X = \text{Model Mini-game performance}, Y = \text{State/Pixel-Input}$). Note that in the case of state-input and pixel-input *Adversarial Player* the worst *Mini-game* might not come from the same *Mini-game* instance.

Model	Mini-game size	Model dataset performance	Average Accuracy		Maximal Accuracy		
			State-Input	Pixel-Input	Model Mini-game performance	State-Input	Pixel-Input
SAN	10	72.1	61.8 (-14.2%)	63.8 (-11.4%)	80.0 / 80.0	28.0 (-65.0%)	43.0 (-46.2%)
	100		66.1 (-8.3%)	69.2 (-3.9%)	74.0 / 75.0	47.6 (-35.6%)	66.3 (-11.5%)
	1000		70.2 (-2.5%)	71.0 (-1.4%)	72.3 / 72.3	68.4 (-5.3%)	70.8 (-1.9%)
FiLM	10	96.2	83.9 (-12.7%)	93.6 (-2.6%)	100.0 / 100.0	48.0 (-52.0%)	86.0 (-14.0%)
	100		89.1 (-7.3%)	94.8 (-1.4%)	98.0 / 100.0	75.6 (-22.8%)	92.4 (-7.5%)
	1000		93.8 (-2.4%)	95.8 (-0.4%)	96.4 / 96.1	90.8 (-5.7%)	94.7 (-1.4%)
RN	10	93.2	80.5 (-13.6%)	86.4 (-7.2%)	100.0 / 100.0	47.0 (-53.0%)	74.0 (-26.0%)
	100		85.5 (-8.2%)	90.8 (-2.5%)	94.0 / 95.0	63.2 (-32.6%)	87.4 (-7.9%)
	1000		90.5 (-2.8%)	91.6 (-1.7%)	93.1 / 93.3	90.0 (-3.2%)	91.0 (-2.4%)
IEP	10	96.9	84.3 (-13.0%)	94.4 (-2.5%)	100.0 / 100.0	48.0 (-52.0%)	87.0 (-13.0%)
	100		90.3 (-6.8%)	95.6 (-1.3%)	98.0 / 96.0	74.4 (-24.0%)	93.5 (-2.6%)
	1000		94.1 (-2.8%)	96.6 (-0.3%)	97.1 / 97.3	93.9 (-3.2%)	95.7 (-1.6%)
TbD	10	99.1	94.0 (-5.1%)	99.1 (-0.0%)	100.0 / 100.0	69.0 (-31.0%)	99.0 (-1.0%)
	100		96.6 (-2.5%)	98.7 (-0.4%)	100.0 / 100.0	91.1 (-8.8%)	98.0 (-2.0%)
	1000		98.0 (-1.1%)	99.0 (-0.1%)	99.4 / 99.7	95.5 (-3.9%)	98.8 (-0.9%)
Mdetr	10	99.7	93.7 (-6.0%)	99.7 (-0.0%)	100.0 / 100.0	60.0 (-40.0%)	99.0 (-1.0%)
	100		96.1 (-3.6%)	98.6 (-1.1%)	100.0 / 100.0	86.4 (-13.5%)	97.8 (-2.2%)
	1000		98.5 (-1.2%)	99.4 (-0.3%)	99.5 / 100.0	94.5 (-5.0%)	99.1 (-0.9%)
State Input Transf.	10	96.8	89.3 (-7.7%)	96.0 (-0.8%)	100.0 / 100.0	77.0 (-23.0%)	91.0 (-9.0%)
	100		94.7 (-2.1%)	96.1 (-0.7%)	97.0 / 99.0	92.6 (-4.5%)	95.0 (-3.9%)
	1000		95.7 (-1.1%)	96.5 (-0.3%)	97.2 / 96.4	95.1 (-2.1%)	95.8 (-0.5%)

perceptual system. We also have similar conclusions based on our small-scale human experiment (Section B.5; appendix).

Adversarial training. We have adapted a widely used adversarial training schema (Goodfellow et al., 2018), and included manipulated scenes in the training protocol of a *Visual-QA Player*. We use *pixel-input* RN (Santoro et al., 2017) as the *Visual-QA Player*. Using this method, we have obtained only marginal improvements in robustness at the cost of a slight performance degradation on the original CLEVR dataset.

Limitations of data-driven reasoning models. Due to the data-driven nature of our *Visual-QA Players*, we pose the following question, How many examples of *Adversarial Player* manipulations would be enough for a visual reasoning model to train on, in order for it to be robust against any unseen ones? Intuitively, if the network has enough capacity, and was trained on all possible data points (or its manipulations), it could rely on a look-up strategy to solve the problem accurately. We propose the following experiment to address such a question. We created a series of datasets as follows. We treat a scene as a discrete 7×7 grid (manipulations are restricted only to this grid). Thus each dataset contains $49^2 = 2401$, $49^3 = 117649$, $49^4 = 5764801$ scene combinations, covering all possible scene manipulations with two, three, and four objects respectively. In the case of four objects, due to computational reasons, we keep a single object stationary (thus it also has 49^3 scene manipulations). The set of questions associated with each image dataset requires the model to perform either one reasoning step (Onehop), two reasoning steps (Twohop) or are a mixture of both (Mixhop). We use RN, FiLM and TbD as *Visual-QA Players*. We train them on $X\%$ data and next evaluate on $(100 - X)\%$ remaining, unseen data. We have conducted ten trials by forming $X\%$

training data randomly. We notice that the behavior of all tested models is similar so we report a joint average. Results are presented in Figure 2. As the number of objects and diversity of question increases, so does the amount of training examples needed in order to achieve robustness.

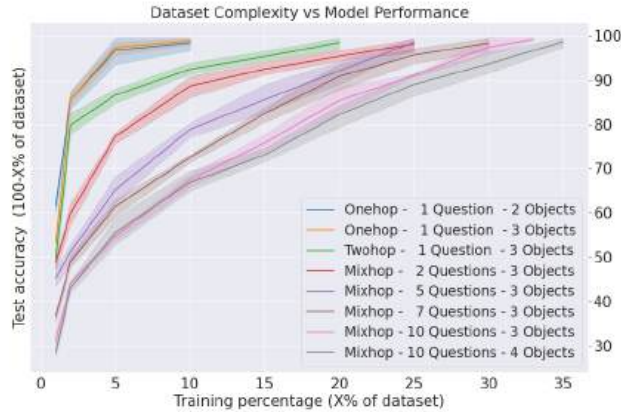


Figure 2: Data-driven limitation experiment: Unseen perturbation accuracy depends heavily on the number and complexity of introduced questions, followed by the number of objects in the dataset examples/scenes. A model could be considered robust when achieving close to 100% test accuracy.

7 CONCLUSIONS

We present a novel approach to isolate and benchmark the reasoning capabilities of visual models by formulating a game between two players. Both players are independent and can only communicate through a common interface, an environment and question-answer pairs. One player, named *Adversarial Player*, is trying to manipulate the environment so that the second player, named *Visual-QA Player*, can't answer the given question anymore. We show that popular models with solid results on CLEVR are susceptible to scene manipulations, and their performance may degrade significantly. Moreover, we also conducted a controlled study on the generalization capabilities of such models. We did so by creating synthetic and simplified datasets of all possible scene manipulations. We show that the existing models are susceptible to reasoning gaps and require extra data, proportional to their task complexity in order to generalize to all possible configurations. One possible direction to increase efficiency could be the use of stronger inductive biases. Another to change the training paradigm into a more interactive one. However, the environment is synthetic, and it could be that learning from real-world and multi-modal data is somewhat more efficient. Finally, we believe that melding two-player games with the visual question answering framework is both natural and beneficial for testing the reasoning capabilities and potentially can also be generalized to other settings.

ACKNOWLEDGMENTS

All experiments were performed using the Entropy cluster funded by NVIDIA, Intel, the Polish National Science Center grant UMO-2017/26/E/ST6/00622 and ERC Starting Grant TOTAL. The work of Spyridon Mouselinos and Henryk Michalewski was supported by the Polish National Science Center grant UMO-2018/29/B/ST6/02959.


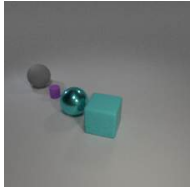
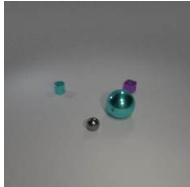
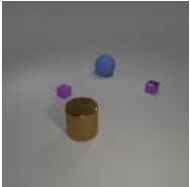
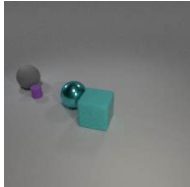


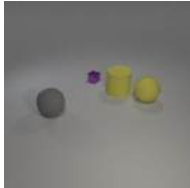
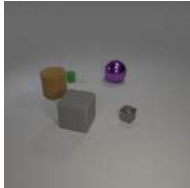

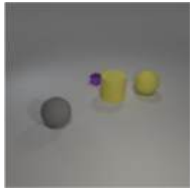
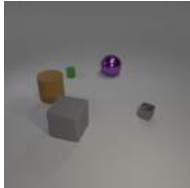

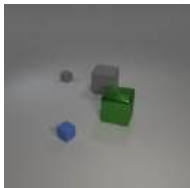


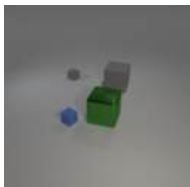

	What is the color of the block that is left of the big blue matte ball?	How many other objects are the same shape as the big gray matte object?	How many other things are the same shape as the big metallic thing?
FiLM - Before	 Purple	 1	 1
FiLM - After	 Brown	 0	 2
	What number of tiny matte cylinders are there?	Are there any other things that have the same material as the big yellow ball?	Is there a big brown object of the same shape as the green thing?
TbD - Before	 2	 Yes	 Yes
TbD - After	 1	 No	 No
	The red thing that is the same material as the brown cube is what size?	Is there anything else that has the same material as the large green cube?	What number of things are large metallic blocks on the left side of the brown metallic thing or big cyan shiny objects?
MDetr - Before	 Large	 No	 2
MDetr - After	 Small	 Yes	 0

Figure 3: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.

REFERENCES

- Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. Don't just assume; look and answer: Overcoming priors for visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4971–4980, 2018.
- Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, and Mani B. Srivastava. Genattack: Practical black-box attacks with gradient-free optimization. *CoRR*, abs/1805.11090, 2018. URL <http://arxiv.org/abs/1805.11090>.
- Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.
- Rishabh Bhardwaj, Navonil Majumder, and Soujanya Poria. Investigating gender bias in bert. *Cognitive Computation*, pp. 1–11, 2021.
- Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016. URL <http://arxiv.org/abs/1608.04644>.
- Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning. In *ECCV*, 2019.
- Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning. In *ECCV*, 2020.
- Minhao Cheng, Thong Le, Pin-Yu Chen, Huan Zhang, JinFeng Yi, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. In *International Conference on Learning Representations*, 2018.
- Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/32508f53f24c46f685870a075eaaa29c-Paper.pdf>.
- Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017.
- Thomas Degris, Patrick M. Pilarski, and Richard S. Sutton. Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference (ACC)*, pp. 2177–2182, 2012. doi: 10.1109/ACC.2012.6315022.
- Donald Geman, Stuart Geman, Neil Hallonquist, and Laurent Younes. Visual turing test for computer vision systems. *Proceedings of the National Academy of Sciences*, 112(12):3618–3623, 2015.
- Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL <http://arxiv.org/abs/1412.6572>.
- Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, 2018.
- Chuan Guo, Jacob R. Gardner, Yurong You, Andrew G. Wilson, and Kilian Q. Weinberger. Simple black-box adversarial attacks, 2019a. URL <https://openreview.net/forum?id=rJeZS3RcYm>.
- Dalu Guo, Chang Xu, and Dacheng Tao. Graph reasoning networks for visual question answering. *CoRR*, abs/1907.09815, 2019b. URL <http://arxiv.org/abs/1907.09815>.
- FR Helmert. Die genauigkeit der formel von peters zur berechnung des wahrscheinlichen beobachtungsfehlers director beobachtungen gleicher genauigkeit. *Astronomische Nachrichten*, 88:113, 1876.

- Lisa Anne Hendricks, Kaylee Burns, Kate Saenko, Trevor Darrell, and Anna Rohrbach. Women also snowboard: Overcoming bias in captioning models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 771–787, 2018.
- Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067*, 2018.
- Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6700–6709, 2019.
- Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. *arXiv preprint arXiv:2103.03206*, 2021.
- Huaizu Jiang, Ishan Misra, Marcus Rohrbach, Erik Learned-Miller, and Xinlei Chen. In defense of grid features for visual question answering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017a.
- Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *ICCV*, 2017b.
- Ameya Joshi, Amitangshu Mukherjee, Soumik Sarkar, and Chinmay Hegde. Semantic adversarial attacks: Parametric transformations that fool deep classifiers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4773–4783, 2019.
- Aishwarya Kamath, Mannat Singh, Yann LeCun, Ishan Misra, Gabriel Synnaeve, and Nicolas Carion. Mdetr–modulated detection for end-to-end multi-modal understanding. *arXiv preprint arXiv:2104.12763*, 2021.
- Daniel Khashabi, Gabriel Stanovsky, Jonathan Bragg, Nicholas Lourie, Jungo Kasai, Yejin Choi, Noah A Smith, and Daniel S Weld. Genie: A leaderboard for human-in-the-loop evaluation of text generation. *arXiv preprint arXiv:2101.06561*, 2021.
- Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.
- Shane Legg. *Machine super intelligence*. PhD thesis, Università della Svizzera italiana, 2008.
- Linjie Li, Jie Lei, Zhe Gan, and Jingjing Liu. Adversarial vqa: A new benchmark for evaluating the robustness of vqa models. *arXiv preprint arXiv:2106.00245*, 2021.
- Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016. URL <http://arxiv.org/abs/1509.02971>.
- Locatello. Object-centric learning with slot attention. In *Neural Information Processing Systems (NeurIPS)*, 2020.
- Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *arXiv preprint arXiv:1908.02265*, 2019.
- J Lüroth. Vergleichung von zwei werthen des wahrscheinlichen fehlers. *Astronomische Nachrichten*, 87:209, 1876.
- Mateusz Malinowski and Mario Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. *Advances in neural information processing systems*, 27: 1682–1690, 2014.

- Mateusz Malinowski, Carl Doersch, Adam Santoro, and Peter Battaglia. Learning visual question answering by bootstrapping hard attention. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–20, 2018.
- David Mascharka, Philip Tran, Ryan Soklaski, and Arjun Majumdar. Transparency by design: Closing the gap between performance and interpretability in visual reasoning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <https://proceedings.mlr.press/v48/mniha16.html>.
- Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, 2017.
- Y Nie, A Williams, E Dinan, M Bansal, J Weston, and D Kiela. Adversarial nli: A new 521 benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4885–4901, 2020.
- Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pp. 506–519, 2017.
- Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.
- Oskar Pfungst. *Clever Hans:(the horse of Mr. Von Osten.) a contribution to experimental animal and human psychology*. Holt, Rinehart and Winston, 1911.
- Piotr Piekos, Henryk Michalewski, and Mateusz Malinowski. Measuring and improving bert’s mathematical abilities by predicting the order of reasoning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 2: Short Papers), Virtual Event, August 1-6, 2021*. Association for Computational Linguistics, 2021.
- Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30, 2017.
- John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL <https://proceedings.mlr.press/v37/schulman15.html>.
- Sasha Sheng, Amanpreet Singh, Vedanuj Goswami, Jose Alberto Lopez Magana, Wojciech Galuba, Devi Parikh, and Douwe Kiela. Human-adversarial visual question answering. *arXiv preprint arXiv:2106.02280*, 2021.
- Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. VI-bert: Pre-training of generic visual-linguistic representations. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=SygXPaEYvH>.

- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL <http://incompleteideas.net/book/the-book-2nd.html>.
- Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.
- Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.
- Makarand Tapaswi, Yukun Zhu, Rainer Stiefelhagen, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Movieqa: Understanding stories in movies through question-answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4631–4640, 2016.
- Alan M Turing. Computing machinery and intelligence. In *Parsing the turing test*, pp. 23–65. Springer, 2009.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.
- Tan Wang, Jianqiang Huang, Hanwang Zhang, and Qianru Sun. Visual commonsense r-cnn. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10760–10770, 2020.
- Zirui Wang, Jiahui Yu, Adams Wei Yu, Zihang Dai, Yulia Tsvetkov, and Yuan Cao. Simvlm: Simple visual language model pretraining with weak supervision. *arXiv preprint arXiv:2108.10904*, 2021.
- Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016. URL <http://arxiv.org/abs/1611.01224>.
- Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/file/361440528766bbaaa1901845cf4152b-Paper.pdf>.
- Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. *arXiv preprint arXiv:1810.02338*, 2018.
- Zhou Yu, Dejing Xu, Jun Yu, Ting Yu, Zhou Zhao, Yueting Zhuang, and Dacheng Tao. Activitynet-qa: A dataset for understanding complex web videos via question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.
- Xiaohui Zeng, Chenxi Liu, Yu-Siang Wang, Weichao Qiu, Lingxi Xie, Yu-Wing Tai, Chi-Keung Tang, and Alan L Yuille. Adversarial attacks beyond the image space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4302–4311, 2019.
- Pengchuan Zhang, Xiujun Li, Xiaowei Hu, Jianwei Yang, Lei Zhang, Lijuan Wang, Yejin Choi, and Jianfeng Gao. Vinvl: Revisiting visual representations in vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5579–5588, June 2021.

A APPENDIX

A.1 URLS OF CLEVR MODELS

Table 3 shows the URLs to models used in our investigations (also Table 1 in the main paper). We also report if we re-trained a model from scratch (type *Architecture*) or used already trained models (type *Model*). Please note that the latter type proves that our testing procedure is fully black-box.

Model Name	Link	Type
SAN (Yang et al., 2016)	https://github.com/facebookresearch/clevr-iep	Model
IEP (Johnson et al., 2017b)	https://github.com/facebookresearch/clevr-iep	Model
FiLM (Perez et al., 2018)	https://github.com/ethanjperz/film	Architecture
RN (Santoro et al., 2017)	https://github.com/mesnico/RelationNetworks-CLEVR	Architecture
TbD (Mascharka et al., 2018)	https://github.com/davidmascharka/tbd-nets	Model
Mdetr (Kamath et al., 2021)	https://github.com/ashkamath/mdetr	Model

Table 3: URLs to Models. *Architecture* denotes that we use the code but re-trained the model on CLEVR. *Model* refers to already trained models.

A.2 OBJECT CATEGORIES

Figure 4 shows a made-up image that contains all available CLEVR object categories: shapes, sizes, and materials. Small objects are rendered at approximately 40% of the size of their large counterparts. Metallic materials can be identified by the high albedo property and ‘shininess’ of the object. We provide it as the reference point for a reader to make it easier to compare our qualitative results.

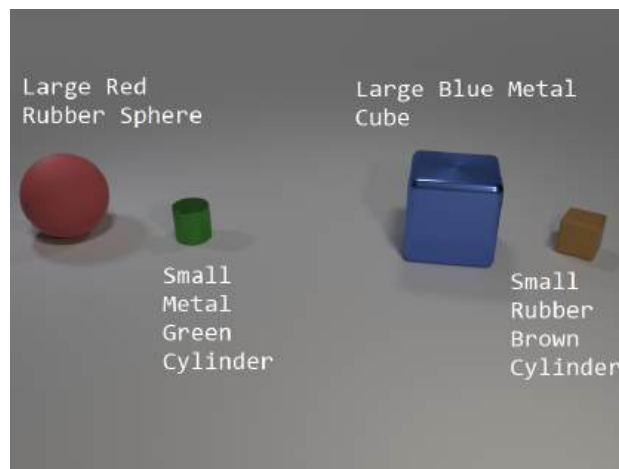


Figure 4: Object categories in the CLEVR dataset.

A.3 STATE-INPUT TRANSFORMER

To test our two-player game against a *state-input Visual-QA Player*, we have designed a transformer-based architecture (Vaswani et al., 2017) that receives six types of input features – object sizes, object shapes, object materials, object colors, object positions, and question tokens – and uses a cross-modal attention mechanism. In that mechanism, queries from one modality attend to keys and values from the other modality. Each data point contains variable-length inputs, describing all objects in the scene. It also contains question tokens that compose the question itself. Hence, we do padding with a special token \emptyset to the maximal input length. We set the maximal length for the objects tokens to be $10 * 6 = 60$ and 50 for the question tokens. All in all, we have 110 input tokens. Each token type is projected into a different embedding space and a learnable type embedding is added to it, separately for object and question tokens. For instance, $\text{emb}(\text{material}) + \text{emb}(\text{object})$. Furthermore, three special learnable embeddings are used as an additional input. We use them as queries to reduce the overall computational costs of the cross-attention mechanism. The same mechanism is also used in Perceiver (Jaegle et al., 2021). The input tokens (concatenated object and question tokens) form keys and values. In every transformer block, we apply cross-attention between all input embeddings and those three special tokens. This is repeated five times. As the final block, a feed-forward network (classifier) receives as inputs the three latent tokens and outputs answer probabilities. We show that architecture in Figure 5.

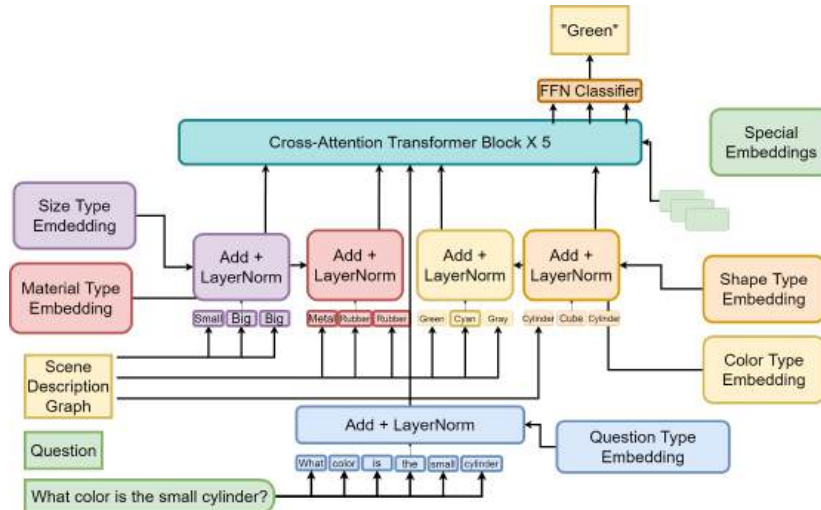


Figure 5: Our State-Input Transformer that operates on the graph scene (states).

A.4 MANIPULATIONS OF *State-Input Visual-QA Player*

Figures 6 and 7 show manipulations of *Visual-QA Player* that is trained to take *states* instead of *pixels* as the input. Such a *state-input Visual-QA Player* receives a direct overview of the scene, bypassing any need for any image renderings. We use *State-Input Transformer* (Section A.3) as the *multi-modal* component of that player. Since such a model gets the perfect visual information as the input, it makes it more robust under scene manipulations. As a consequence, our *Adversarial Player* tends to manipulate the scene so that objects are placed closely together.

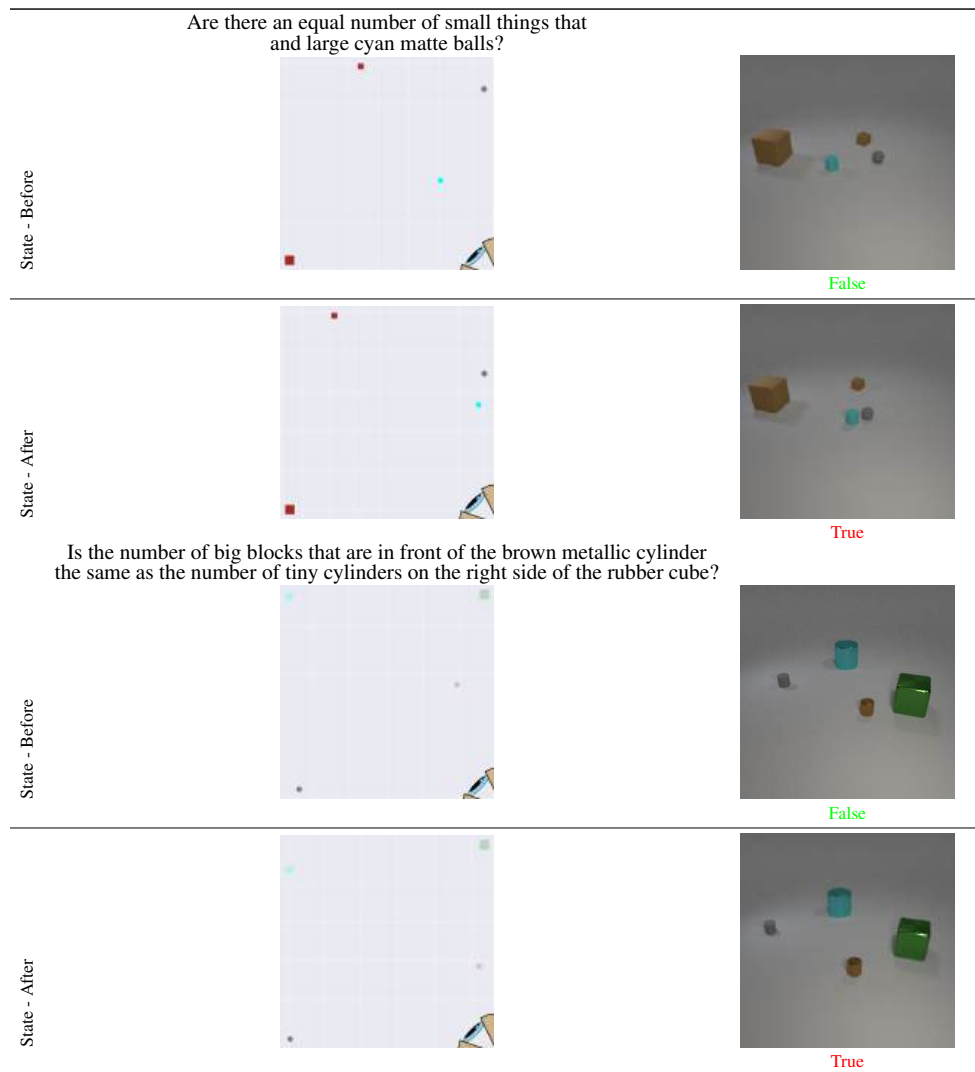


Figure 6: Qualitative results on *state-input Visual-QA Player*. For each example, a top-down view of the scene is presented. The eye at the bottom right of each view represents the rendering camera. It corresponds to the *Visual-QA Player*'s viewpoint. Each scene is also rendered and presented side-by-side for comparison.

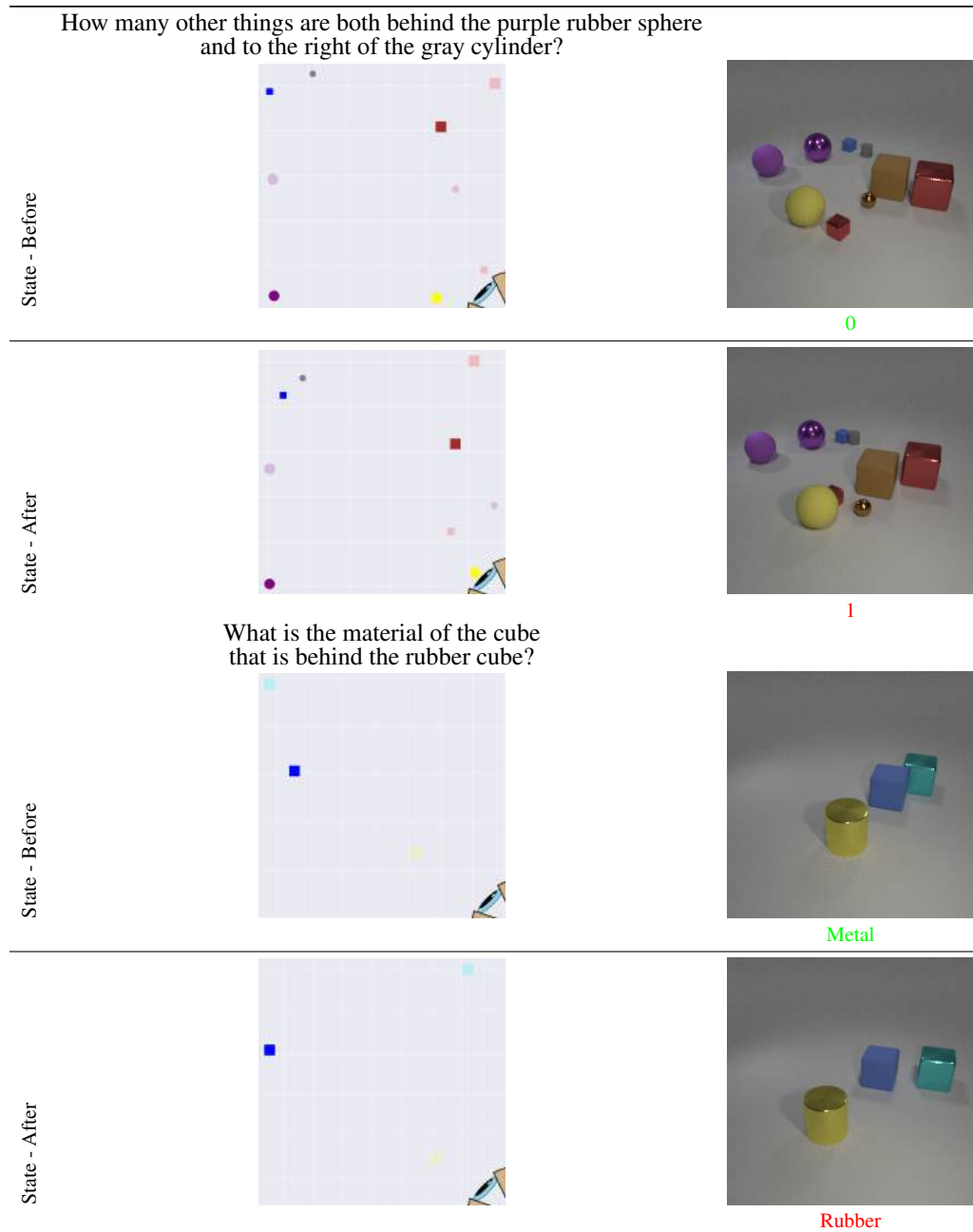


Figure 7: Qualitative results on *state-input Visual-QA Player*. For each example, a top-down view of the scene is presented. The eye at the bottom right of each view represents the rendering camera. It corresponds to the *Visual-QA Player*'s viewpoint. Each scene is also rendered and presented side-by-side for comparison.

A.5 ALGORITHMS

We show pseudo-algorithms that we use to (Algorithm 1) calculate rewards, (Algorithm 2) train *Adversarial Player*, (Algorithm 3) and play a game.

Algorithm 1 Calculate Rewards

```

1: new_answer : Answer produced by Visual Agent on perturbed image
2: old_answer : Answer produced by Visual Agent on original image
3: gt_answer : Ground Truth Answer
4: dr : Drop Reward
5: cr : Consistency Reward
6: fr : Fail Reward
7: reward  $\leftarrow$  0
8: if new_answer  $\neq$  old_answer then
9:   if old_answer  $\neq$  gt_answer then
10:    reward = reward + cr
11:   else
12:    reward = reward + dr
13:   end if
14: else
15:   reward = reward + fr
16: end if
17: return reward

```

Algorithm 2 Training Pipeline

```

1: M : Minigames
2: Va : Visual Agent, Fe : Feature Extractor
3: A : Actor, C : Critic
4: Qre : Question Relevance Enforcer, Sce : Scene Constraint Enforcer
5: for batch in M do
6:   rewards, state_values, logprobs  $\leftarrow$  DataGame(M, Va, Fe, A, C, Qre, Sce)
7:   n  $\leftarrow$  |rewards|
8:   advantage  $\leftarrow$  rewards - state_values
9:   ploss  $\leftarrow$  -logprobs  $\times$  advantage
10:  vloss  $\leftarrow$  (state_values - stop_grad(rewards))2/n
11:  loss  $\leftarrow$  ploss + vloss
12:  backprop(loss, A, C)
13: end for
14: return

```

Algorithm 3 Game between players (Game)

```

1:  $M$  : Mini-game
2:  $\mathcal{P}_{\text{VQA}}$  : Visual Agent,  $\mathcal{S}_k$  : Multi-modal Backbone
3:  $A$  : Actor,  $C$  : Critic
4:  $Qre$  : Question Relevance Enforcer,  $Sce$  : Scene Constraint Enforcer
5:  $isr$  : Invalid Scene Reward
6: for batch in  $M$  do
7:   rewards  $\leftarrow$  {}
8:   state_values  $\leftarrow$  {}
9:   logprobs  $\leftarrow$  {}
10:  for (image, scene, question, program, gt_answer) in batch do
11:    old_answer  $\leftarrow$   $\mathcal{P}_{\text{VQA}}$ (image, question)
12:    extracted_features  $\leftarrow$   $\mathcal{S}_k$ (scene, question)
13:    dx_probs, dy_probs  $\leftarrow$   $A$ (extracted_features)
14:    CatX  $\leftarrow$  Categorical(dx_probs)
15:    CatY  $\leftarrow$  Categorical(dy_probs)
16:    state_value  $\leftarrow$   $C$ (extracted_features)
17:    dx  $\leftarrow$  sample CatX
18:    dy  $\leftarrow$  sample CatY
19:    new_scene  $\leftarrow$  Perturbate(scene, dx, dy)
20:    check_scene  $\leftarrow$   $Sce$ (new_scene)
21:    check_question  $\leftarrow$   $Qre$ (new_scene, program, gt_answer)
22:    if check_scene  $\wedge$  check_question = True then
23:      new_image  $\leftarrow$  Render(new_scene)
24:      new_answer  $\leftarrow$   $\mathcal{P}_{\text{VQA}}$ (new_image, question)
25:      rewards  $\leftarrow$  rewards || CalcRewards(new_answer, old_answer, gt_answer)
26:    else
27:      rewards  $\leftarrow$  rewards || isr
28:    end if
29:    state_values  $\leftarrow$  state_values || state_value
30:    logprob  $\leftarrow$  logprob ||  $\log(p(dx|CatX)) + \log(p(dy|CatY))$ 
31:  end for
32: end for
33: return rewards, state_values, logprobs

```

A.6 SENSITIVITY TO QUESTION TYPES

Figure 8 shows the susceptibility of CLEVR models to scene manipulations of *Adversarial Player*. We observe that especially *counting* and *existence* questions are the ‘back-doors’ for our scene manipulations. Questions of those two types typically involve multiple steps of reasoning. *Adversarial Player*’s scene manipulations make them more likely that reasoning will fail at some stage.

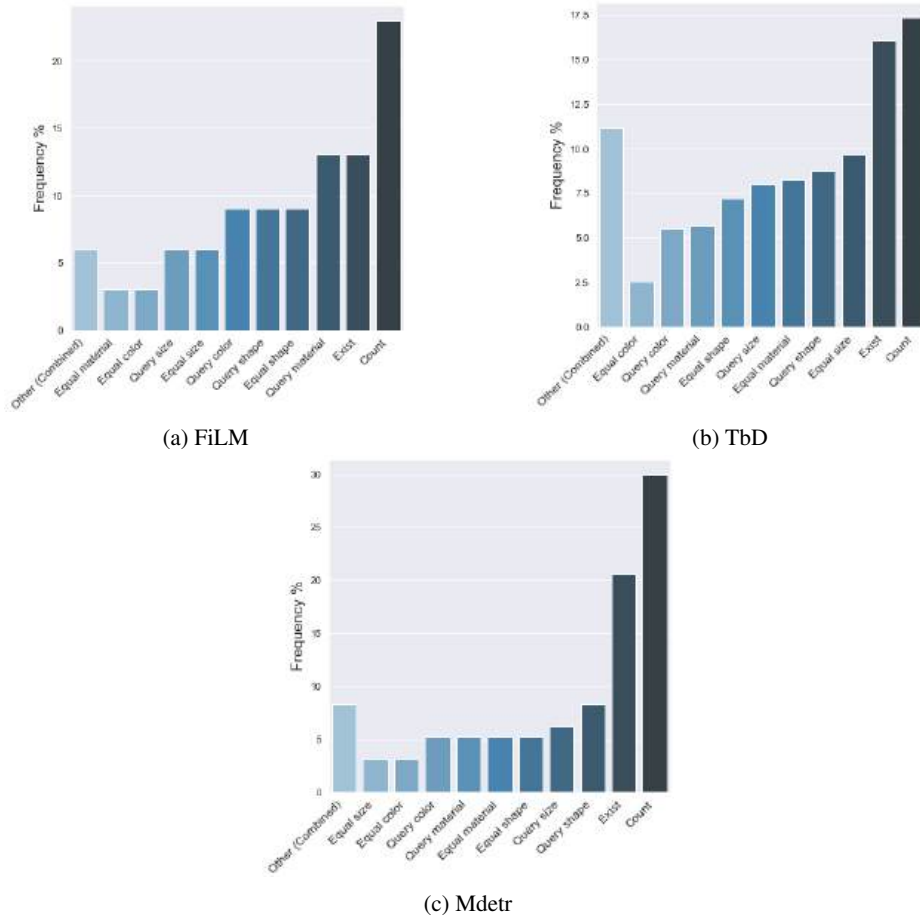


Figure 8: Histograms of the model’s susceptibility to scene manipulations conditioned on a question type.

A.7 CONVERGENCE

Figure 9 shows convergence plots of *Adversarial Player* during training against FiLM, TbD and MDetr. We use different initialization seeds and 30 trials. *Adversarial Player* is trained either from *states* or *pixels*. The former yields higher performance and sometimes better convergence. We use the *Drop* metric that measures the accuracy drop after the *Visual-QA Player* is manipulated. The higher *Drop*, the more successful manipulations are.

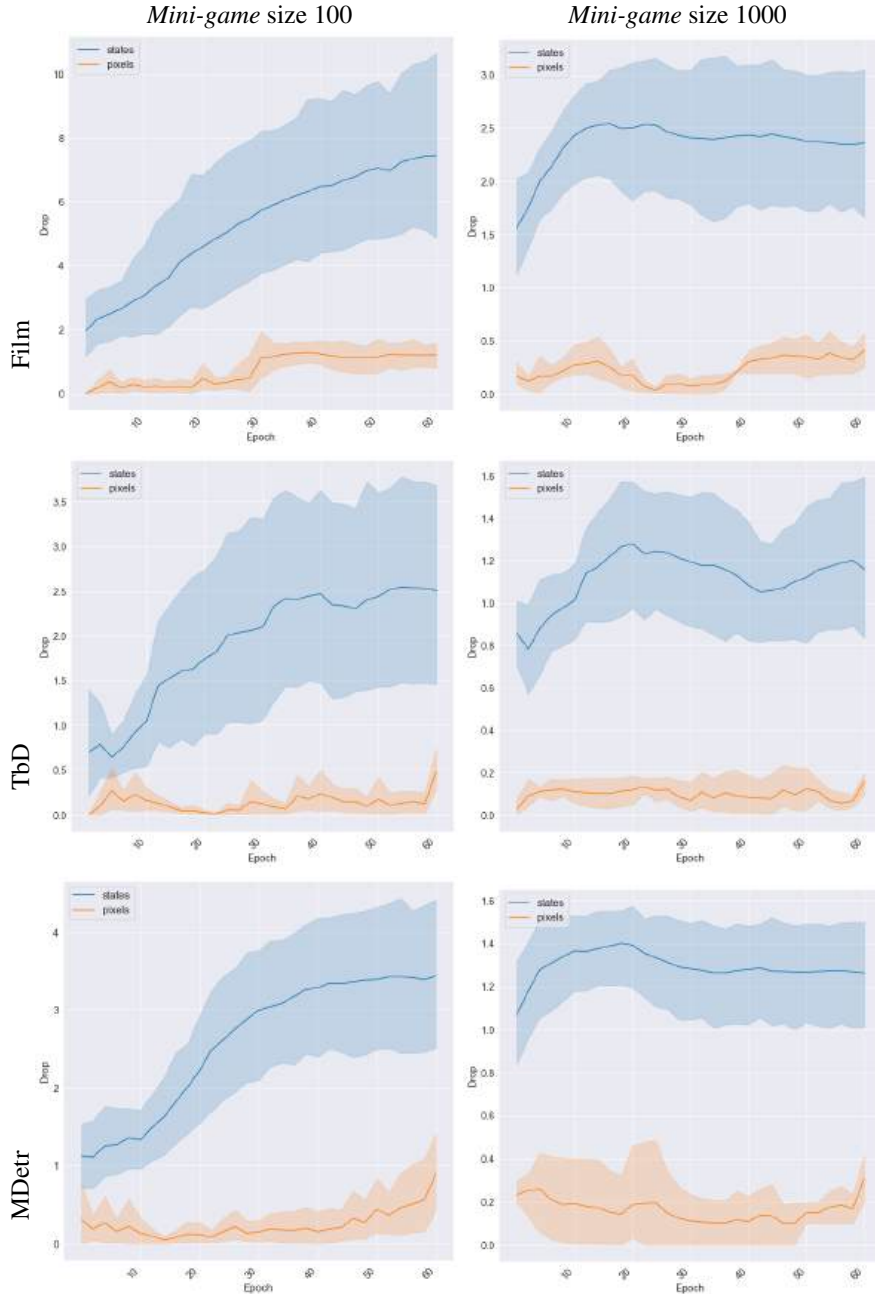


Figure 9: Performance of *Adversarial Player* for different training snapshots. We show the mean and variance over thirty trials. *Adversarial Player* is either trained from *states* (blue) or *pixels* (orange).

A.8 PERFORMANCE DISTRIBUTION PLOTS

Figures 10 and 11 show the distribution of the *Adversarial Player* performance, where we show how often (y-axis) the given *Drop* score is achieved (x-axis). The *Drop* metric measures the accuracy drop after the *Visual-QA Player* is manipulated. The higher *Drop*, the more successful manipulations are. Larger *Mini-games* lead to more consistent (narrower distributions) but lower performance. This behavior suggests that scene diversity present in larger *Mini-games* leads to more stable scene manipulations. However, they do so at the cost of increasing the complexity of the optimization problem.

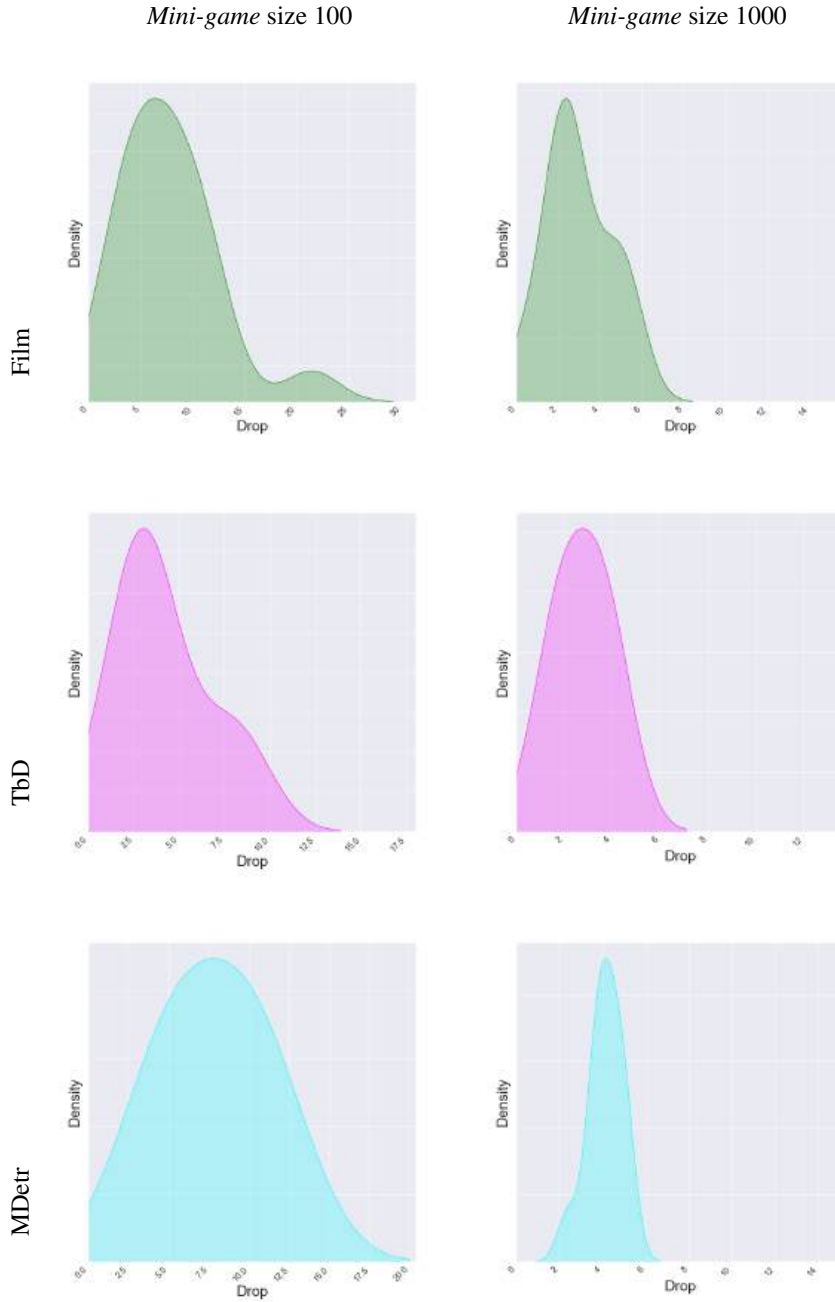


Figure 10: *Mini-games* performance distribution for *state-input Adversarial Player*.

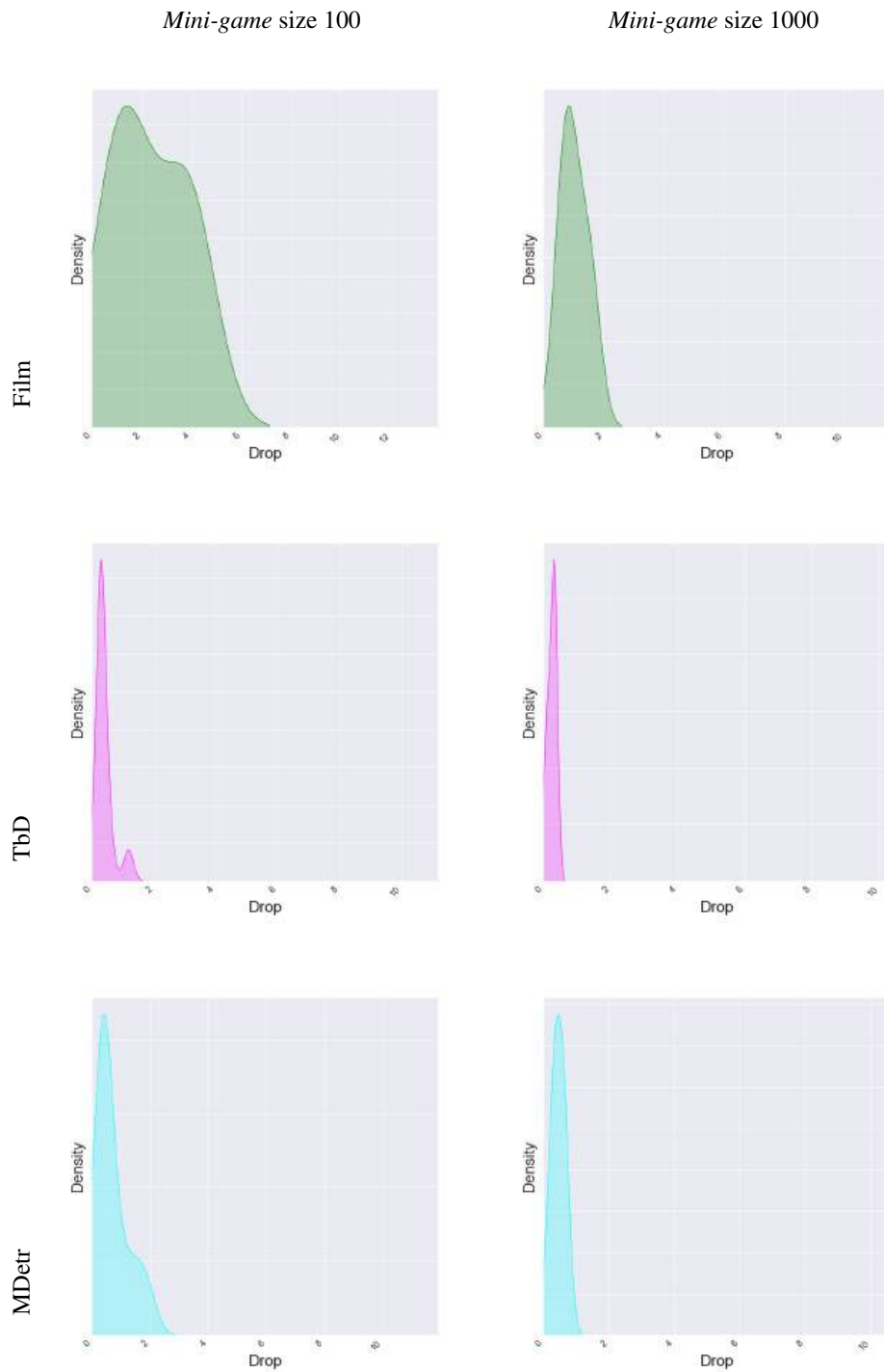


Figure 11: *Mini-games* performance distribution for *pixel-input Adversarial Player*.

A.9 OUT-OF-DISTRIBUTION MANIPULATIONS

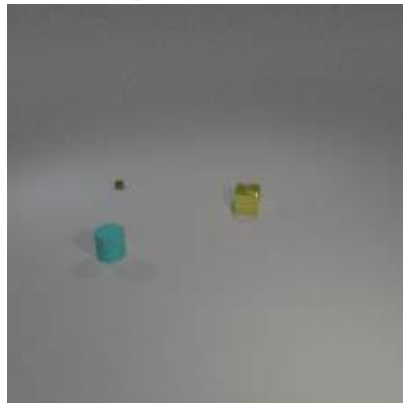
If we do not provide extra constraints on the scene generation process with our environment enforcers, *Adversarial Player* may find an easy manipulation that changes the camera pose or location, effectively ‘zooming out’ the whole scene. It does so by stretching all the object coordinates in the scene. Figure 12 illustrates that. For instance, *large* objects look much smaller than typical examples. As such constructed scenes are quite different from the ones that *Visual-QA Player* has observed during training, we categorize such manipulations to be *out-of-distribution*. Note that, due to our *in-distribution* environment enforcers, such scenes are prohibited in our pipeline.



(a) Shape of the blue cube is misclassified as sphere.



(b) Large, brown cylinder is misclassified as small.



(c) Large, yellow cube is misclassified as small.

Figure 12: Out-of-distribution examples.

A.10 ADDITIONAL QUALITATIVE RESULTS

In Figures 13-21, we provide more qualitative results.

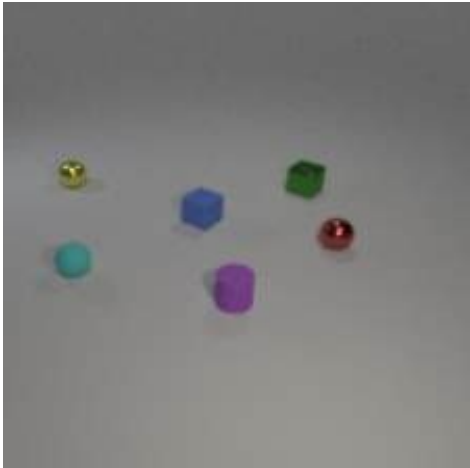
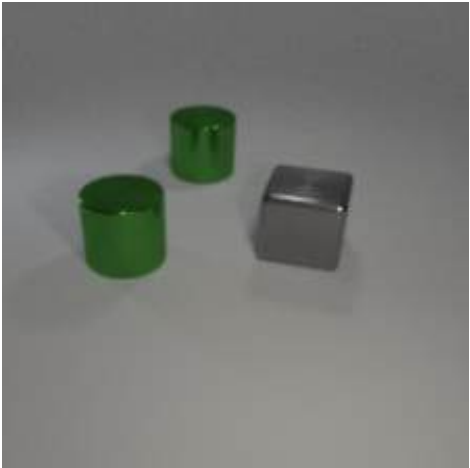

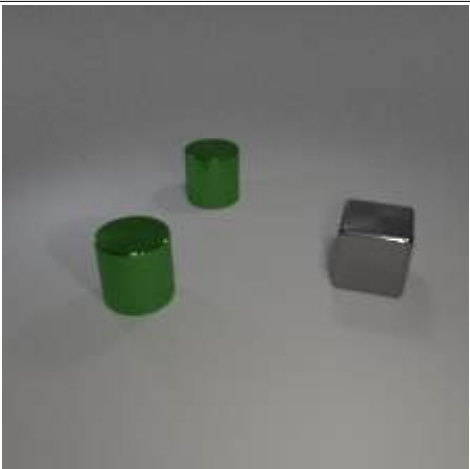
	Is the number of small cubes that are right of the blue rubber block greater than the number of small balls that are in front of the cyan object?	What is the color of the metallic thing that is on the right side of the shiny thing behind the large gray object?
SAN - Before	 Yes	 Gray
SAN - After	 No	 Green

Figure 13: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.

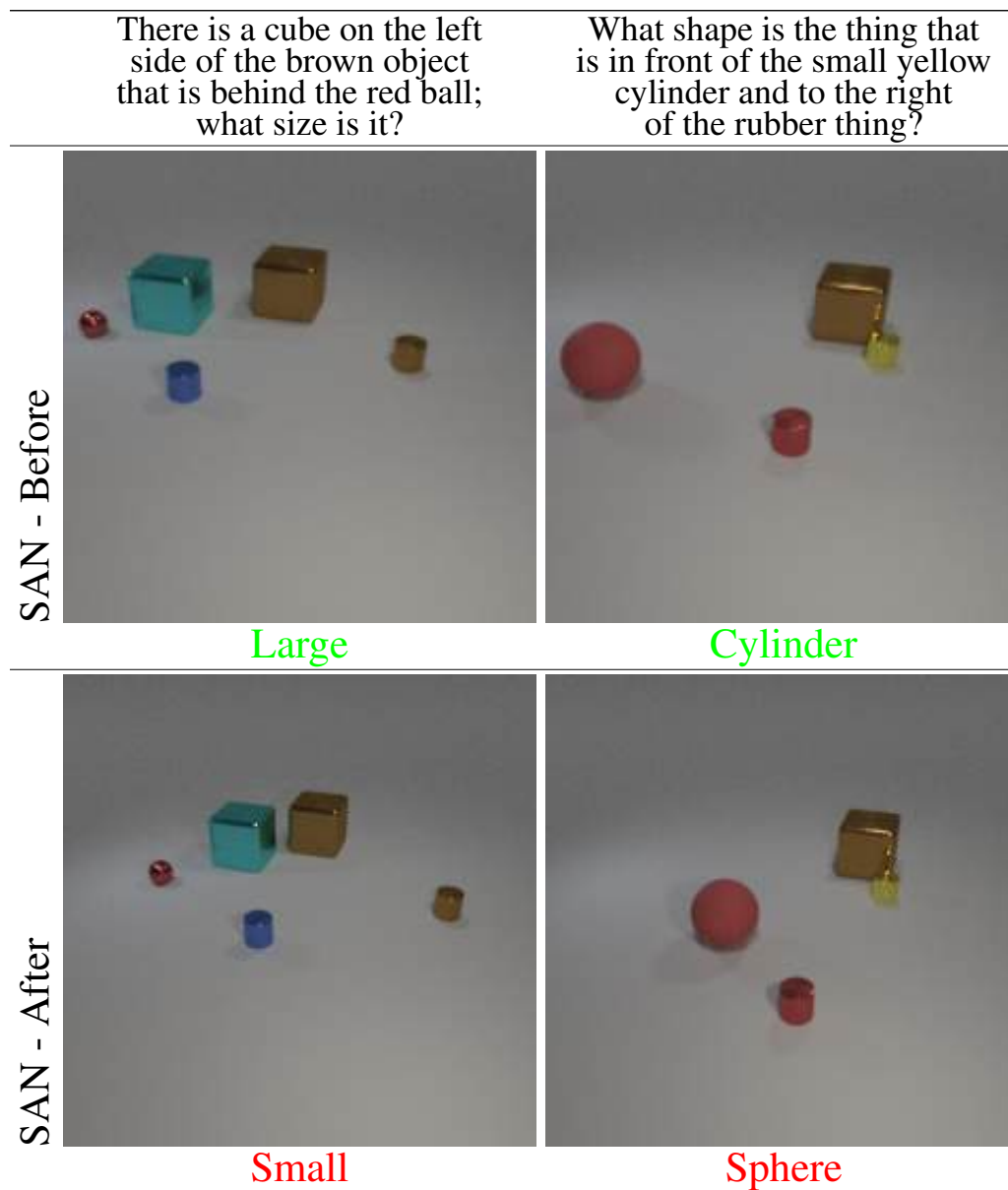


Figure 14: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.

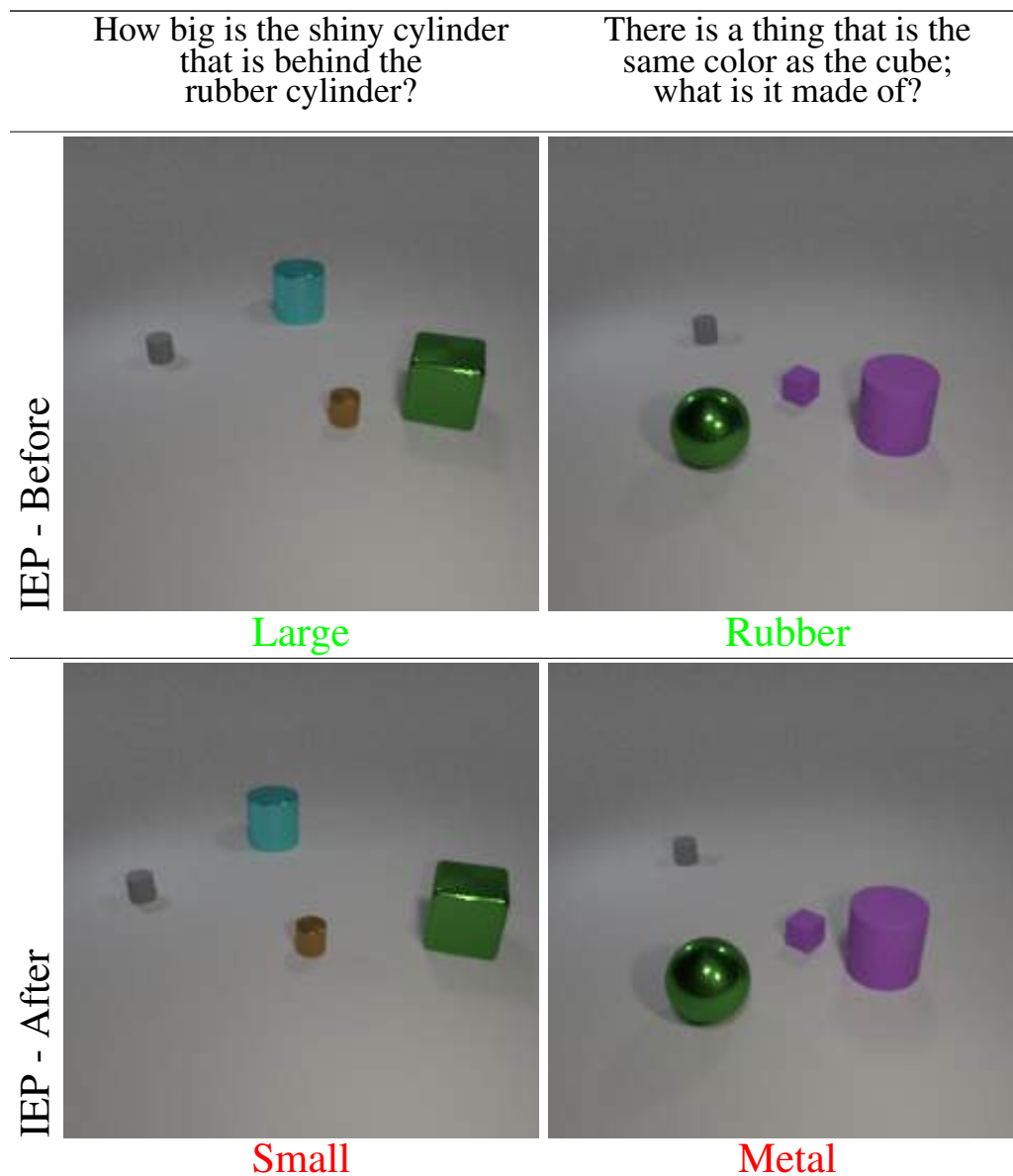


Figure 15: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.


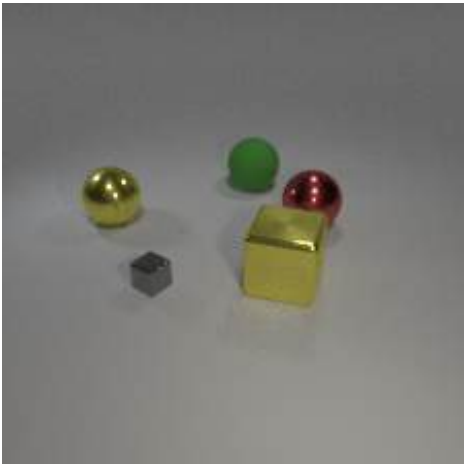
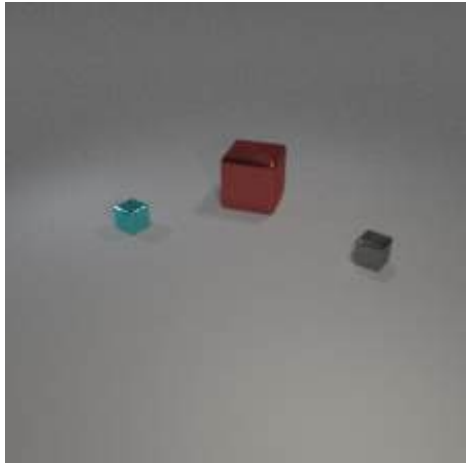
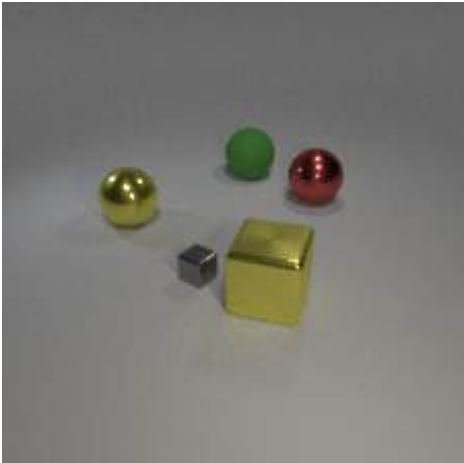
	Does the cyan metal object that is behind the tiny gray block have the same size as the block behind the small cyan metallic objec?	What is the big thing that is behind the large red shiny sphere made of?
IEP - Before	 No	 Rubber
IEP - After	 Yes	 Metal

Figure 16: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.

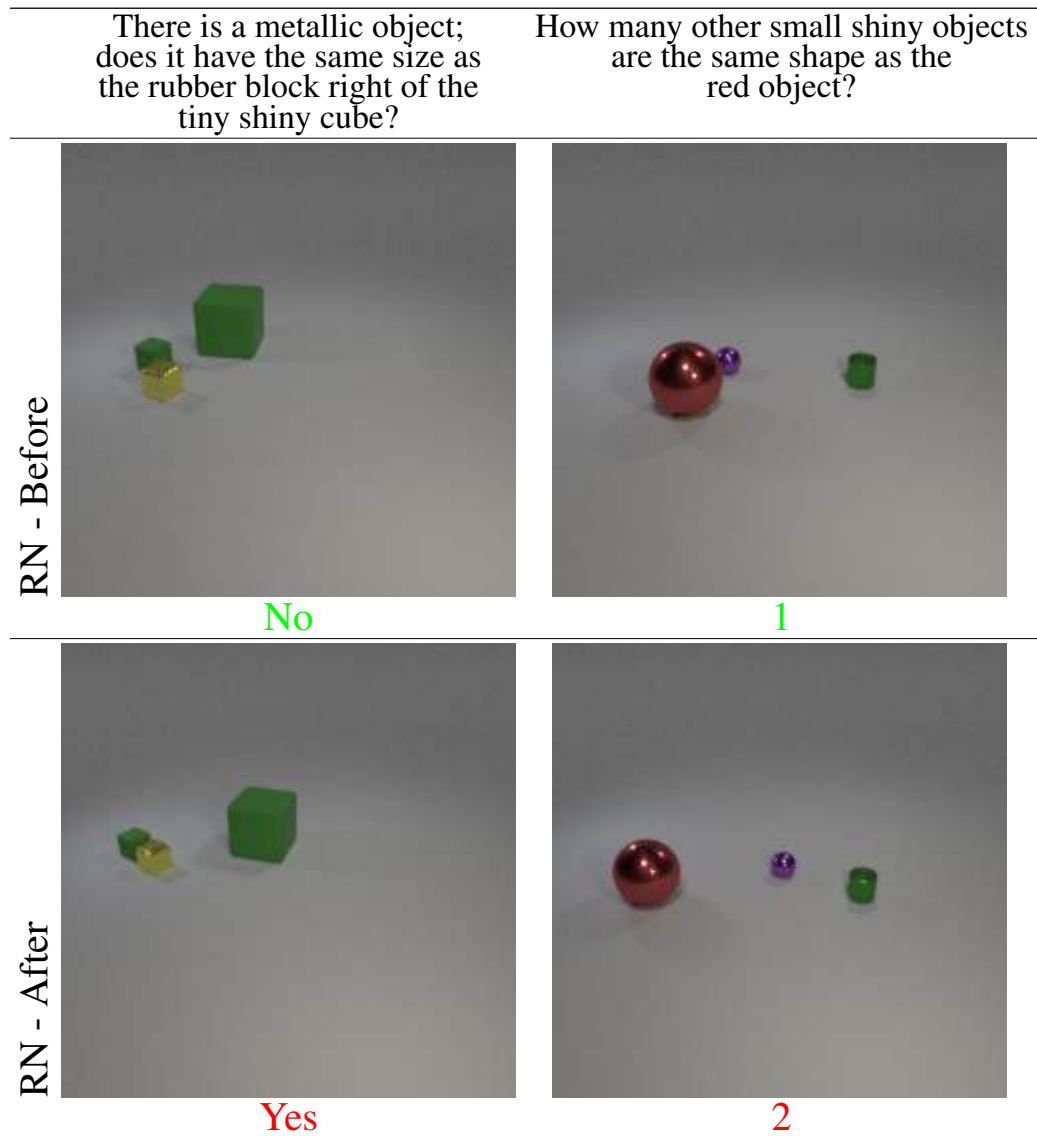


Figure 17: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.

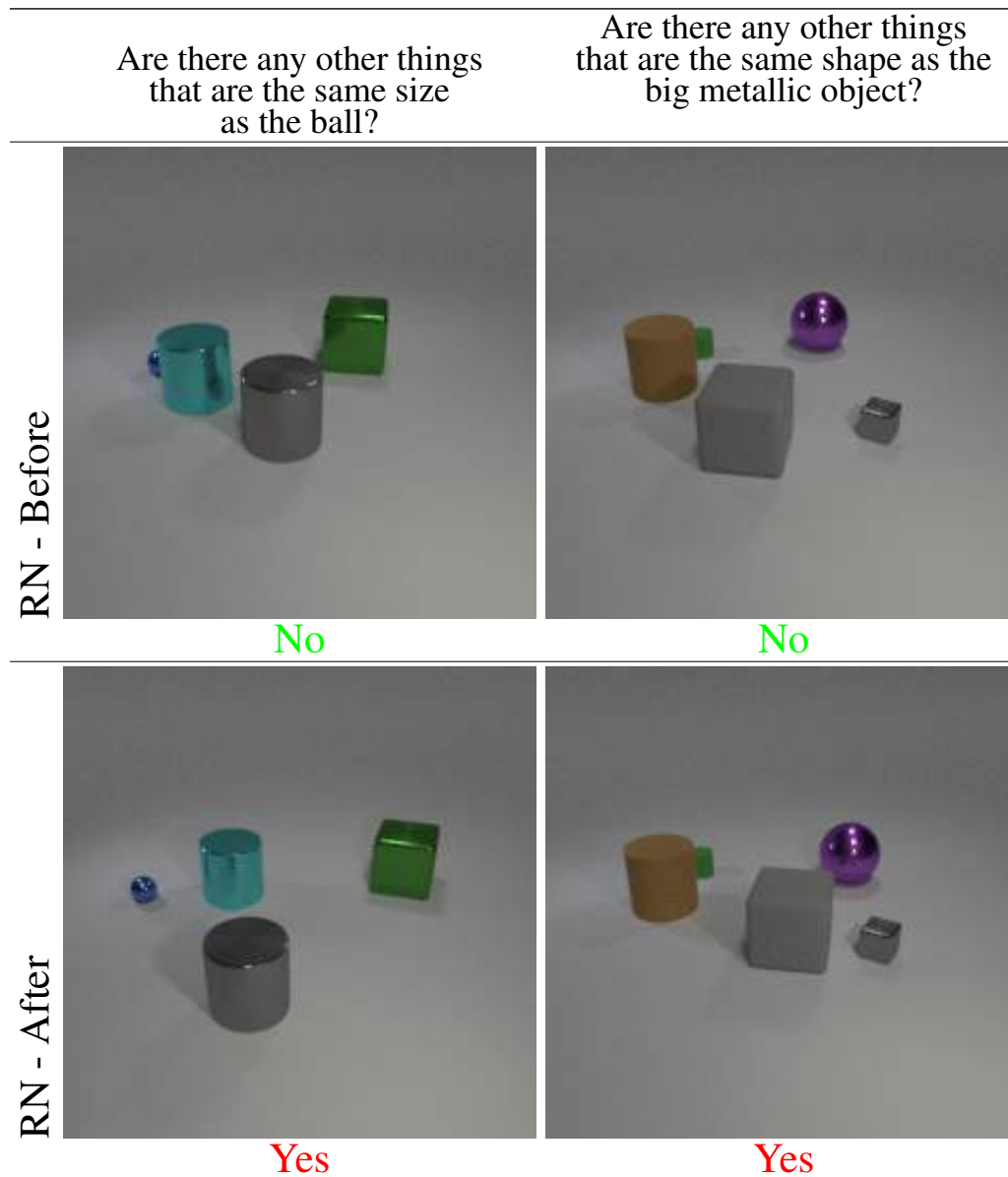


Figure 18: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.

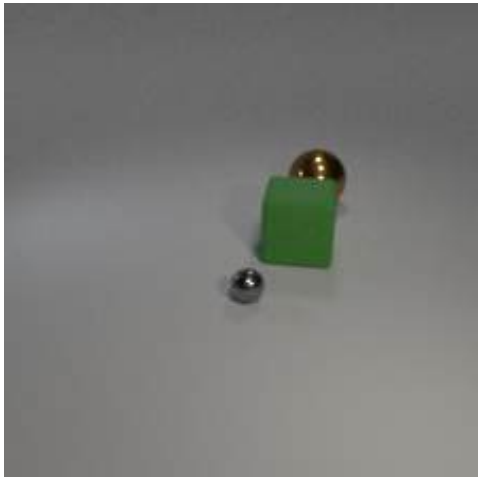
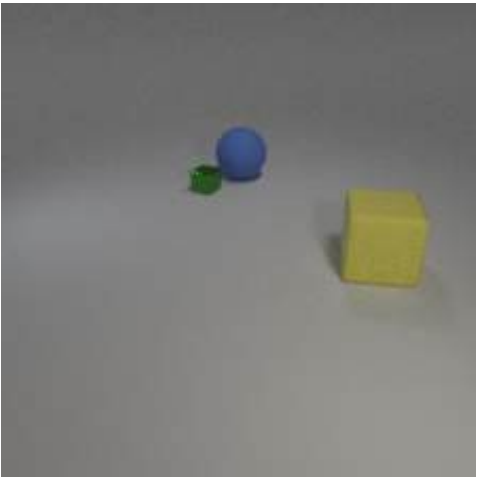
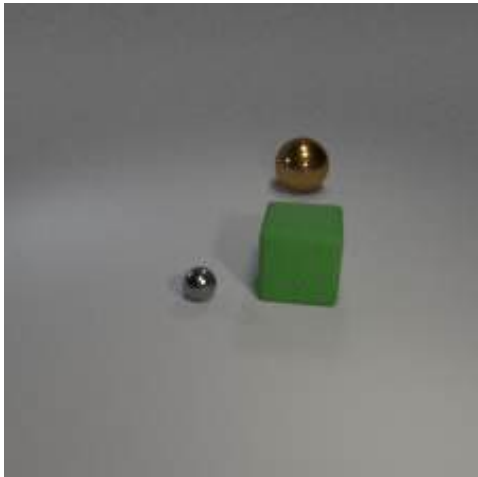
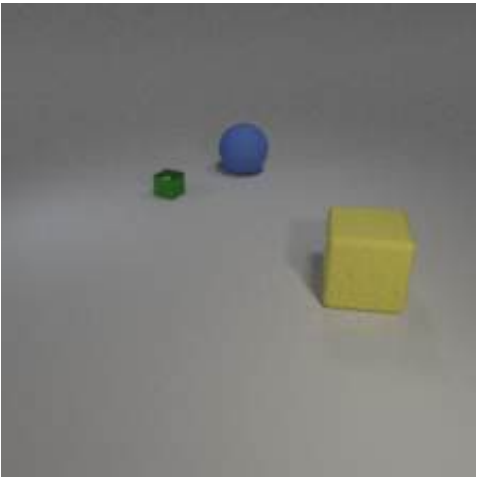
	How many objects are there of the same size as the brown object?	There is another object that is the same size as the blue rubber object; What is it made of?
MDetr - Before	 1	 Rubber
MDetr - After	 0	 Metal

Figure 19: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.

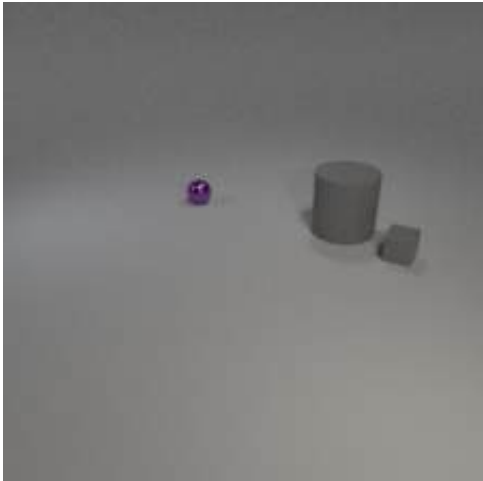
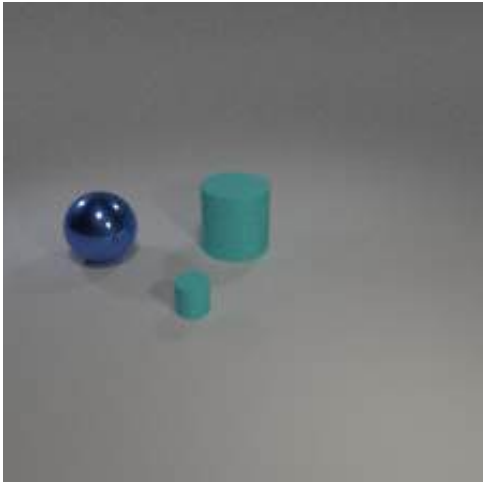
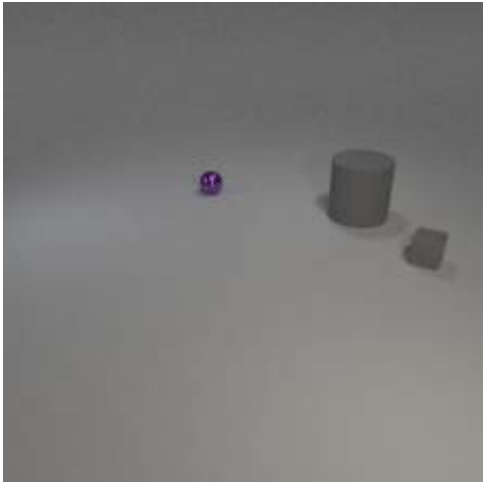
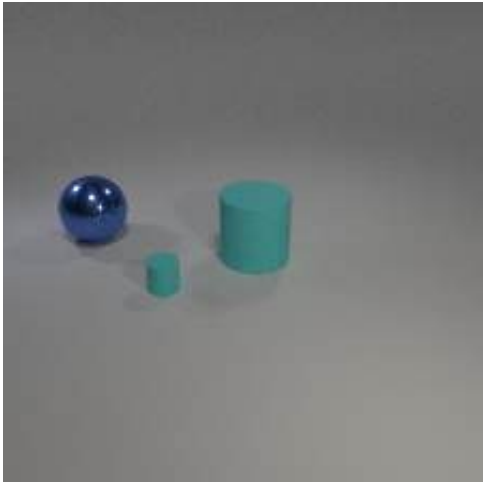
	What number of other objects are the same color as the block ?	What number of metal objects are small objects or big cyan objects?
MDetr - Before	 1	 0
MDetr - After	 0	 1

Figure 20: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.

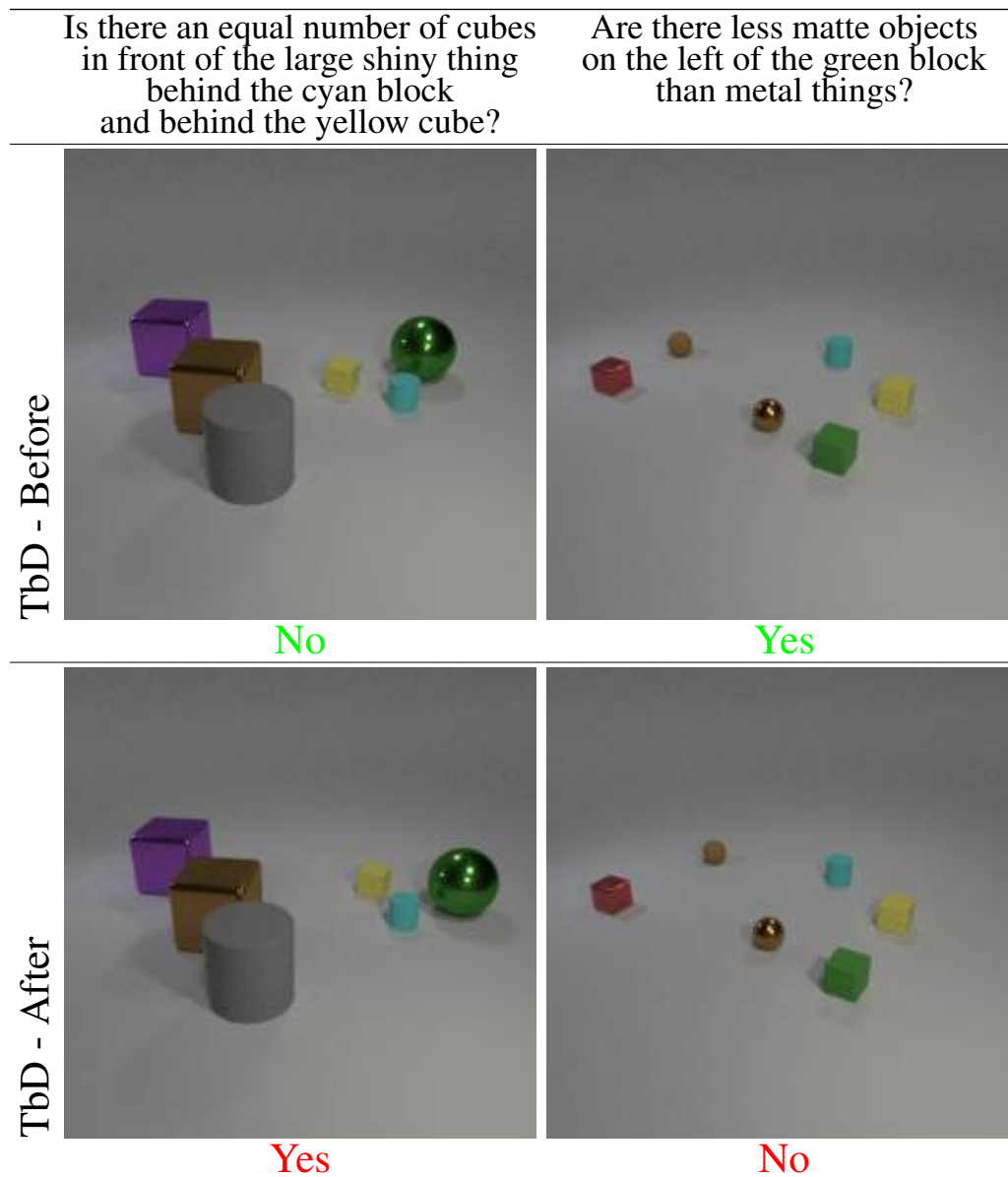


Figure 21: Manipulations of CLEVR models. We show results *before* and *after* scene manipulations.

A.11 QUANTITATIVE RESULTS WITH P-VALUES

We extend the quantitative results of the main paper by also reporting p -values of the one-sample hypothesis testing. We use the *Accuracy Drop* metric that measures the accuracy drop after the *Visual-QA Player* is manipulated, and the *Consistency Drop* that measures how many times the manipulated *Visual-QA Player* changes its answer, independently if that is a correct or wrong answer. The higher *Drop Accuracy* or *Consistency Drop*, the more successful manipulations are. We use T-Test (Helmert, 1876; Lüroth, 1876) for each metric all the runs. Our null hypothesis is that the population mean of all the games is zero, indicating that a manipulation is unsuccessful, and the results of our *Adversarial Players* are extreme cases of a good performance, i.e., $H_0 : \mu_l = 0, l \in [\text{Drop}, \text{Consistency}]$. As an alternative hypothesis, we assume that the population mean is greater than zero, i.e., $H_A : \mu_l > 0, l \in [\text{Drop}, \text{Consistency}]$. The same setup stands for both the *Consistency* and the *Drop* metric. Tables 4 and 5 show the full table with p -values computed. Note that only in two cases, *Adversarial Player* does not manipulate the scene convincingly (TbD and Mdetr for the *pixel-input Adversarial Player*).

Table 4: Results of the game with the *state-input Adversarial Player*.

Model	Mini-game size	Average		p-value	
		Consistency Drop %	Accuracy Drop %	Consistency Drop	Accuracy Drop
SAN	10	-23.9	-14.2	0.000	0.000
	100	-10.7	-8.3	0.000	0.000
	1000	-3.5	-2.5	0.001	0.000
FiLM	10	-14.8	-12.7	0.003	0.006
	100	-7.8	-7.3	0.000	0.000
	1000	-3.1	-2.4	0.000	0.000
RN	10	-20.9	-13.6	0.000	0.000
	100	-10.5	-8.2	0.000	0.000
	1000	-3.3	-2.8	0.000	0.000
IEP	10	-13.9	-13.0	0.001	0.003
	100	-6.9	-6.8	0.000	0.000
	1000	-2.9	-2.8	0.000	0.000
TbD	10	-5.3	-5.1	0.011	0.012
	100	3.8	-2.5	0.000	0.000
	1000	-1.1	-1.1	0.000	0.000
Mdetr	10	-6.6	-6.0	0.018	0.021
	100	-4.9	-3.6	0.000	0.000
	1000	-1.3	-1.2	0.000	0.000
State	10	-8.5	-7.7	0.020	0.034
Input	100	-2.3	-2.1	0.000	0.000
Model	1000	-1.2	-1.1	0.000	0.000

Table 5: Results of the game with the *pixel-input Adversarial Player*.

Model	Mini-game size	Average		<i>p</i> -value	
		Consistency Drop %	Accuracy Drop %	Consistency Drop	Accuracy Drop
SAN	10	-12.8	-11.4	0.008	0.018
	100	-6.4	-3.9	0.000	0.001
	1000	-2.3	-1.4	0.000	0.000
FiLM	10	-3.62	-2.6	0.056	0.154
	100	-1.7	-1.4	0.000	0.000
	1000	-0.6	-0.4	0.000	0.000
RN	10	-9.4	-7.2	0.021	0.065
	100	-3.2	-2.5	0.000	0.000
	1000	-2.3	-1.7	0.000	0.000
IEP	10	-2.9	-2.5	0.111	0.171
	100	-1.5	-1.3	0.001	0.004
	1000	-0.4	-0.3	0.000	0.000
TbD	10	0.0	0.0	1.000	1.000
	100	-0.6	-0.4	0.017	0.078
	1000	-0.2	-0.1	0.000	0.000
Mdetr	10	0.0	0.0	1.000	1.000
	100	-1.2	-1.1	0.002	0.003
	1000	-0.4	-0.3	0.000	0.000
State Input Model	10	-0.9	-0.8	0.033	0.038
	100	-0.7	-0.7	0.031	0.031
	1000	-0.3	-0.3	0.000	0.000

B APPENDIX B

B.1 MODEL PERFORMANCE ON MAXIMAL DROP MINIGAMES.

In the table below, we present model accuracy on each of the *Mini-games* where the maximal performance drop was observed. The third and fourth columns (Original Data section) refer to the model accuracy on unmanipulated image-question pairs of the respective *Mini-game*. The fifth and sixth columns (Manipulated Data section) refer to the model accuracy in those *Mini-game* examples after the *Adversarial Player* manipulation.

Model	<i>Mini-game</i> size	Original Data		Manipulated Data	
		<i>Mini-game</i> Accuracy (State)	<i>Mini-game</i> Accuracy (Pixel)	<i>Mini-game</i> Accuracy (State)	<i>Mini-game</i> Accuracy (Pixel)
SAN	10	80.00	80.00	28.84	43.26
	100	74.00	75.00	47.65	66.32
	1000	72.30	72.20	68.42	70.87
FILM	10	100.0	100.0	48.10	86.58
	100	98.00	100.0	75.61	92.44
	1000	96.40	96.10	90.81	94.75
RN	10	100.0	100.0	46.60	74.56
	100	94.00	95.00	63.28	87.42
	1000	93.10	93.30	90.03	91.05
IEP	10	100.0	100.0	48.45	87.21
	100	98.00	96.00	74.41	93.50
	1000	97.10	97.30	93.99	95.73
TbD	10	100.0	100.0	69.37	99.10
	100	100.0	100.0	91.17	98.00
	1000	99.40	99.70	95.53	98.80
Mdetr	10	100.0	100.0	59.82	99.70
	100	100.0	100.0	86.43	97.80
	1000	99.50	100.0	94.51	99.10
State	10	100.0	100.0	77.44	91.96
Input	100	97.00	99.00	92.63	95.05
Transf.	1000	97.20	96.40	95.15	95.83

B.2 GENERATING ‘VISUALLY FAIR’ SCENES.

We define a scene as *visually fair* if there are no occlusions and all objects are within the field-of-view. The ground truth answer of a posed question on a *visually fair* scene remains the same before and after object manipulation.

For the generation of new images/scenes we use the open-source Blender Graphics Engine ² (v2.79b), and the original 3D models of the CLEVR dataset. Scenes suggested by our *Adversarial Player*, are encoded in the form of arguments for Blender, which renders the new images according to those directives. In order to guarantee visual fairness, and question validity, we employ two modules (scene-constraint enforcer / question relevance enforcer), presented briefly - in Section 4 in the Consistency and In-Distribution paragraphs.

The scene-constraint enforcer is responsible for ensuring the creation of a scene/image that respects all constraints and statistical properties of the original dataset. This piece of code is borrowed from the original CLEVR codebase³ and as in the original CLEVR paper the following constraints are checked (Lines 66-83 in the python code):

²Blender - a 3D modeling and rendering package. <https://www.blender.org>

³https://github.com/facebookresearch/clevr-dataset-gen/blob/main/image_generation/render_images.py

1. **Minimum / Maximum Number of Objects:** 3 / 10.
2. **Minimum allowed distance between object centers:** 0.25
3. **Margin along with cardinal directions:** 0.4 (This ensures lack of ambiguity by keeping objects at least 0.4 points of distance apart. This distance is a Blender-specific measure that spans from -3 to +3.)
4. **Min pixels per object:** 200 (This ensures that an object will not be occluded, by requiring at least 200 pixels of it being visible.)

The question-relevance enforcer checks if the newly generated scene is indeed answerable and has the same “ground truth” answer as the old scene. By not enforcing this, every manipulation will be falsely “successful” as the question (which is kept the same before and after the attack) will have a different answer in the newly generated scene.

This is also resolved by using the same question engine module provided by the CLEVR authors in their codebase⁴. The module receives a question and a scene and calculates the ground truth answer based on a set of functional programs that are apriori known and used to generate the questions. (Supplementary Material A in the CLEVR paper (Johnson et al., 2017a))

We use the following pipeline. Our *Adversarial Player* initially receives an input image-question pair. Then, it suggests a new scene, that passes through the scene-constraint enforcer. If the new scene respects all boundaries and constraints it is then evaluated alongside the question by the question-relevance enforcer. If the new ground truth answer to the suggested scene/image is the same as the original answer, the image-question pair is marked as in-distribution and valid. Otherwise, it is removed from the training pipeline.

⁴https://github.com/facebookresearch/clevr-dataset-gen/blob/main/question_generation/question_engine.py

B.3 COMPARISON WITH RANDOMLY SAMPLED EQUIVALENT SCENES.

Arguably, out of the semantically equivalent space of images given a question and an accompanying answer, the *Adversarial Player* effectively discovers cases for which the model under the test can be fooled. This, however, does not indicate how rare those examples are, neither the degree of robustness of the model against other examples in this space. Let us limit ourselves on a specific image-question pair. Then, a tractable estimate of how rare "fooling examples" are would be the number of manipulated scenes (F) that are able to fool the model, divided by all the possible permutations of available objects (N) in our discrete 7x7 grid. $Rarity = \frac{F}{49^N}$, $N \in [3, 10]$. A scene of only 3 objects and 1 discovered successful manipulation would make it 0.00085% possible to find it by chance and would indicate 99.9915% robustness against that sphere of equivalent image-question pairs.

Of course not all of those configurations are valid due to scene constraint violations, as well as objects can theoretically move freely on the scene. As a way to approach this question, we compare our *Adversarial Player* against a random scene generator (RSG). The random scene generator is able to manipulate any object freely, under the constraint that the resulting scene is valid under the scene constraint enforcer as well as the question relevance enforcer. Manipulations of lighting or camera angles, as well as addition or removal of objects is not permitted. In order to reduce the huge rendering times needed for each random search, we allow the RSG to operate on a 10 length *Mini-game* where scene manipulations that cause accuracy drop have been discovered by the *Adversarial Player*. For each image-question pair the RSG has a budget of 5000 queries, and if not successful it proceeds to the next *Mini-game* entry.

Model	CLEVR Accuracy	Adv.Player Accuracy (Drop %)	RSG Accuracy (Drop %)
SAN	72.1	61.86 (-14.2%)	71.59 (-0.7%)
FiLM	96.2	83.98 (-12.7%)	95.81 (-0.4%)
RN	93.2	80.52 (-13.6%)	92.73 (-0.5%)
IEP	96.9	84.30 (-13.0%)	96.70 (-0.2%)
TbD	99.1	94.04 (-5.1%)	99.10 (-0.0%)
MDetr	99.7	93.71 (-6.0%)	99.70 (-0.0%)
ST	96.8	89.34 (-7.7%)	96.50 (-0.3%)

As it can be seen, randomly question-uninformed suggested object placements do not seem to be effective into fooling the models under test. This further supports the argument that models under test theoretically and statistically seem robust, but nevertheless have vulnerabilities that can be effectively exploited by our *Adversarial Player*.

B.4 VISUALIZATIONS.

Here, we try to isolate reasoning gaps from visual causes after the scene manipulations; with the following setup. Each *Visual-QA Player* is introduced with two image-question pairs. The first consists of the original image and the second of a scene created by our *Adversarial Player* that successfully fools the *Visual-QA Player*. We then inspect the visualization of the *Visual-QA Player*'s inner-workings in both pairs. We are especially interested in patterns that suggest the root of failure. As each *Visual-QA Player* operates under different assumptions and architectural designs, we choose to visualize :

1. The *attention maps* of Stack-Attention Networks.
2. The *gradient-weighted activations of the pre-ultimate and ultimate convolutional blocks* of FiLM.
3. The *programs* that are synthesized alongside their *soft-attention maps* of each module block in TbD network.
4. The *objects* that were detected during the computation steps of a MDetr model.

In Figure 22, we visualize the pre-ultimate (column 2) and ultimate (column 3) attention maps of the Stack-Attention Network. We observe that the model focuses on a single final object in manipulated examples. In the original ones, the relational jumps required for the answer are still visible on the attention map. The final attention map that is presented (column 3) is passed to a fully-connected classifier in order for the answer to be created. We observe limited focus to all the necessary objects / locations that creates a lack of necessary features for the correct answer derivation.

In the case of FiLM model in Figure 23, we observe a similar scenario. In the first example, (rows 1 and 2) the gradient-weighted activations of the model seem to be focusing on multiple irrelevant objects regarding the final answer, while in the second example (rows 3 and 4) the model seems to include the purple cube mistakenly as one of the possible spheres of the image. In both cases above it is difficult to exactly pin down the root of (reasoning or visual) failures.

MDetr model can be seen in Figure 24 and Figure 25. Here, a specific pattern could not be identified. All objects were correctly classified. It suggests that the cause of failures stem rather from the reasoning steps.

Tbd is presented in Figure 26. In both the original and manipulated scenes the programs that the model generates are exactly the same. This is no surprise since the program generator module is using exclusively the question as input. Nevertheless, a similar pattern of scattered model focus is presented here as well. The attention maps in both examples are the same between the original and manipulated scenes, in all but the final reasoning step. In the first example (rows 1 and 2) the attention is leaked towards the gray cube that has nothing to do with the correct answer, while in the second (rows 3 and 4) the attention is spread among many matte objects but peaks near the big yellow ball. This is wrong since the word "other" that exists in the question should drive the model focus exclusively to other objects apart from the starting point of the reasoning chain.

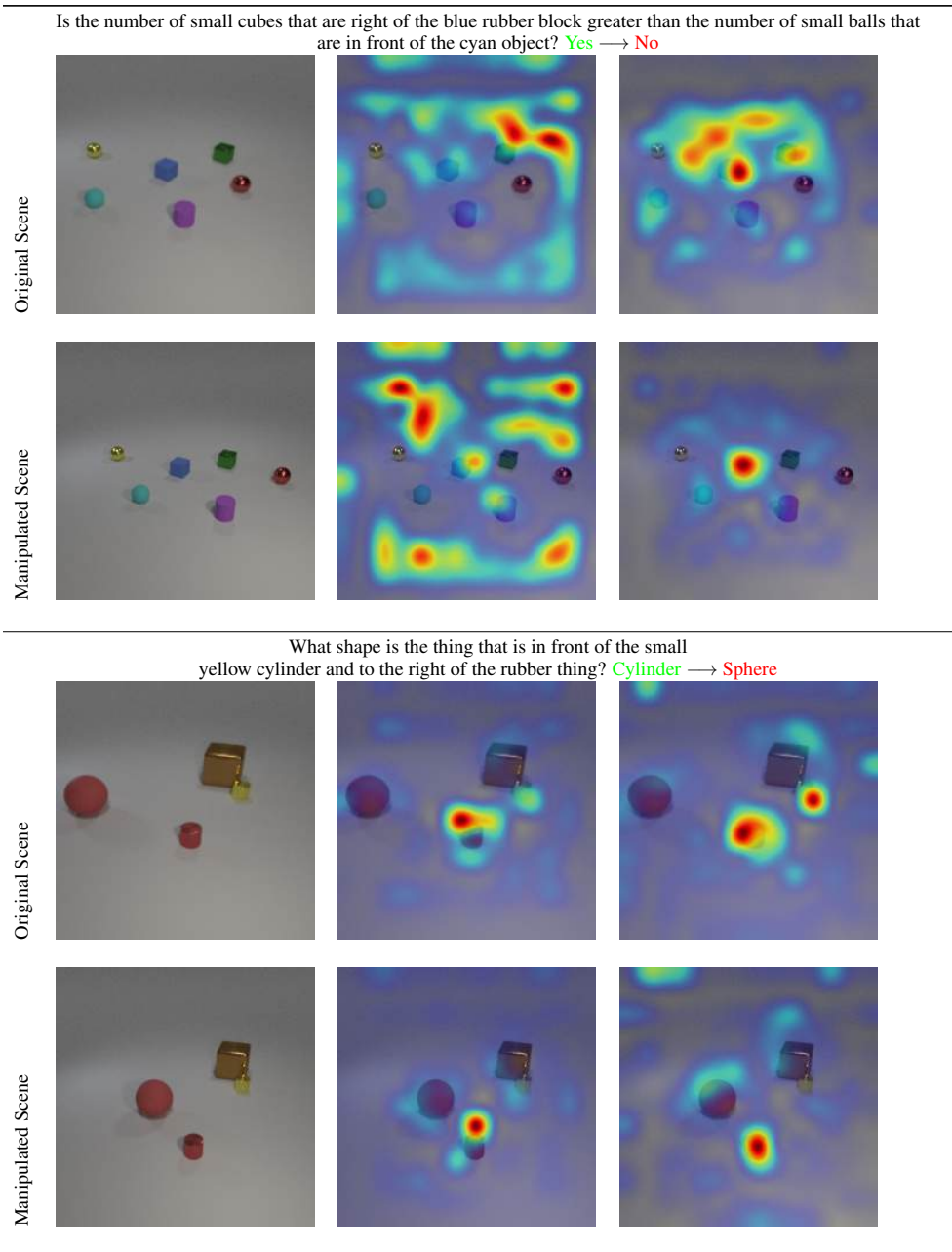


Figure 22: Visualization of attention maps in SAN model on original and manipulated scenes.

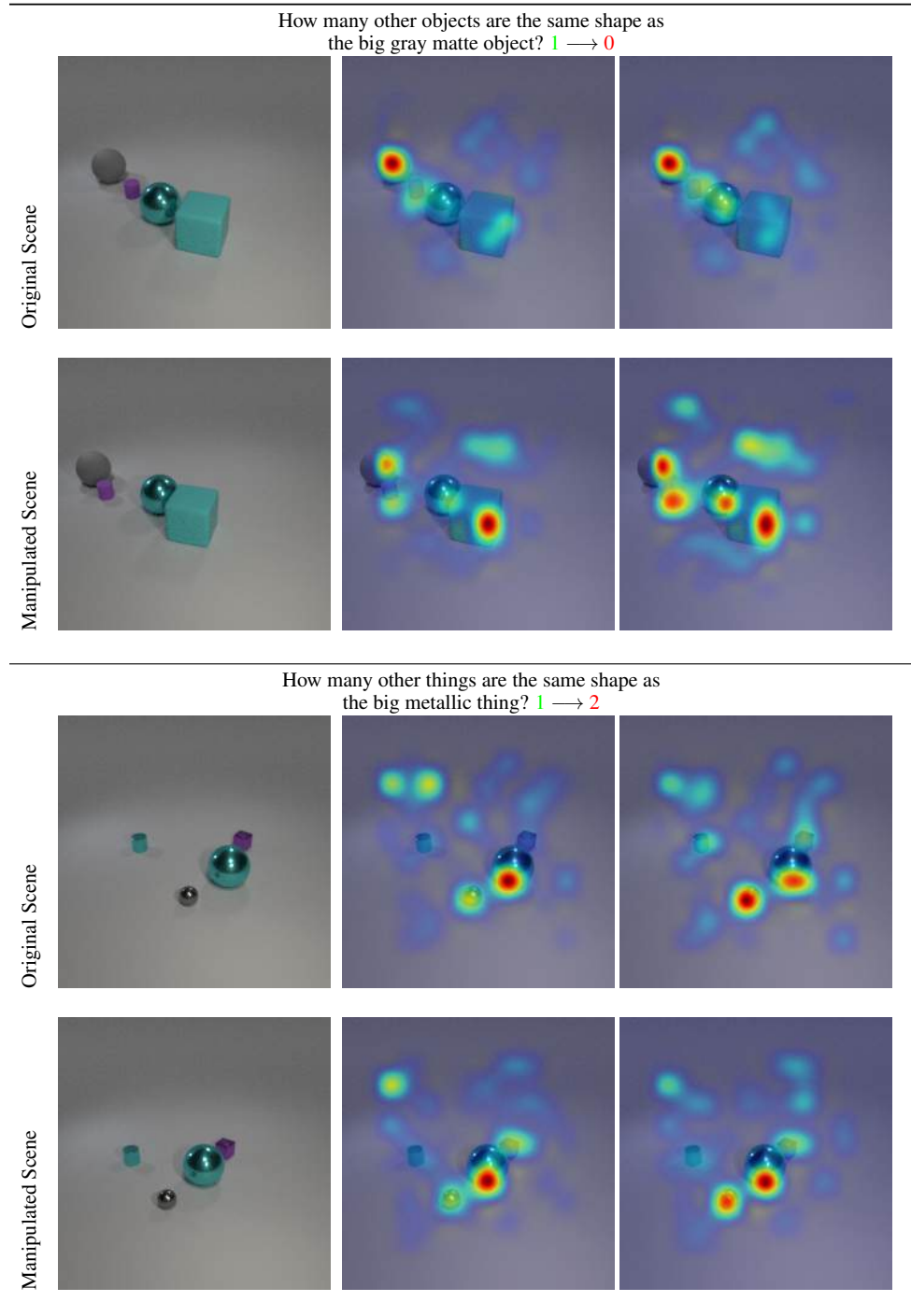


Figure 23: Gradient weighted activation visualization of FiLM model’s pre-ultimate and ultimate blocks on original and manipulated scenes

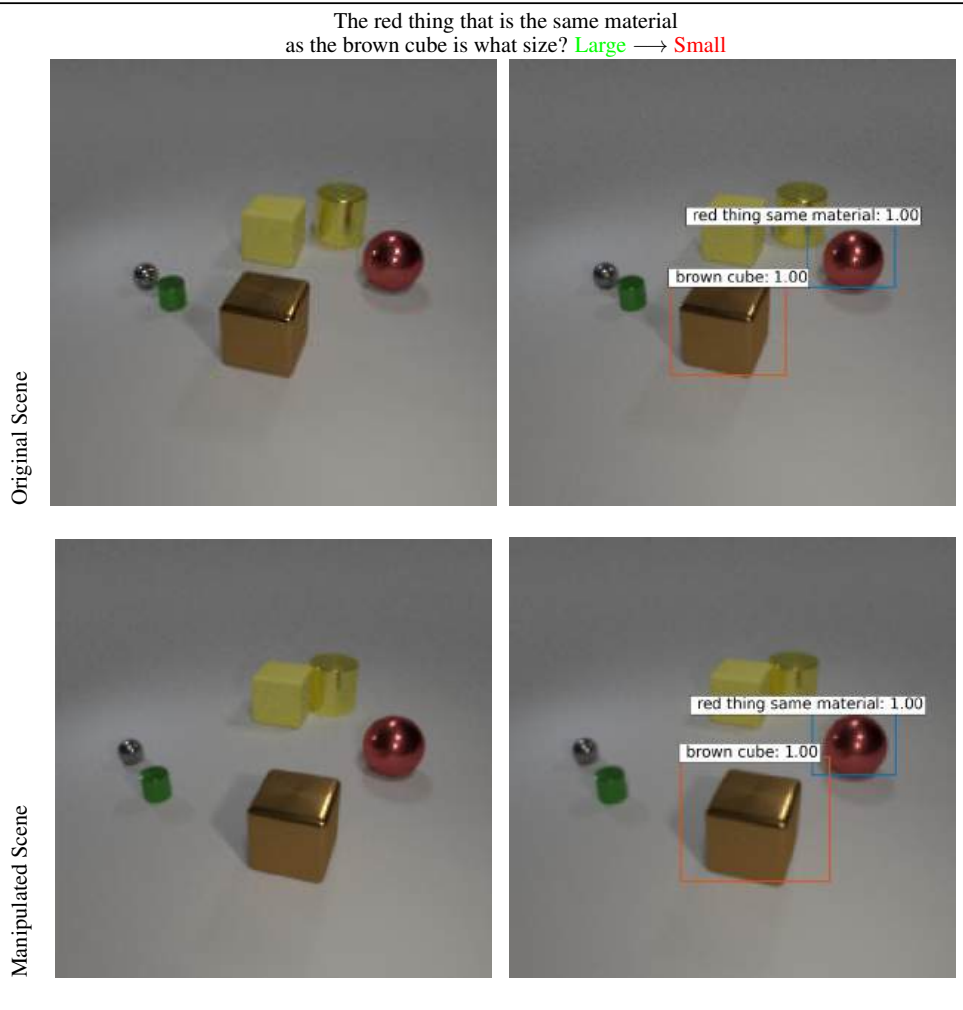


Figure 24: Visualization of MDetr model's detection stages on original and manipulated scenes.

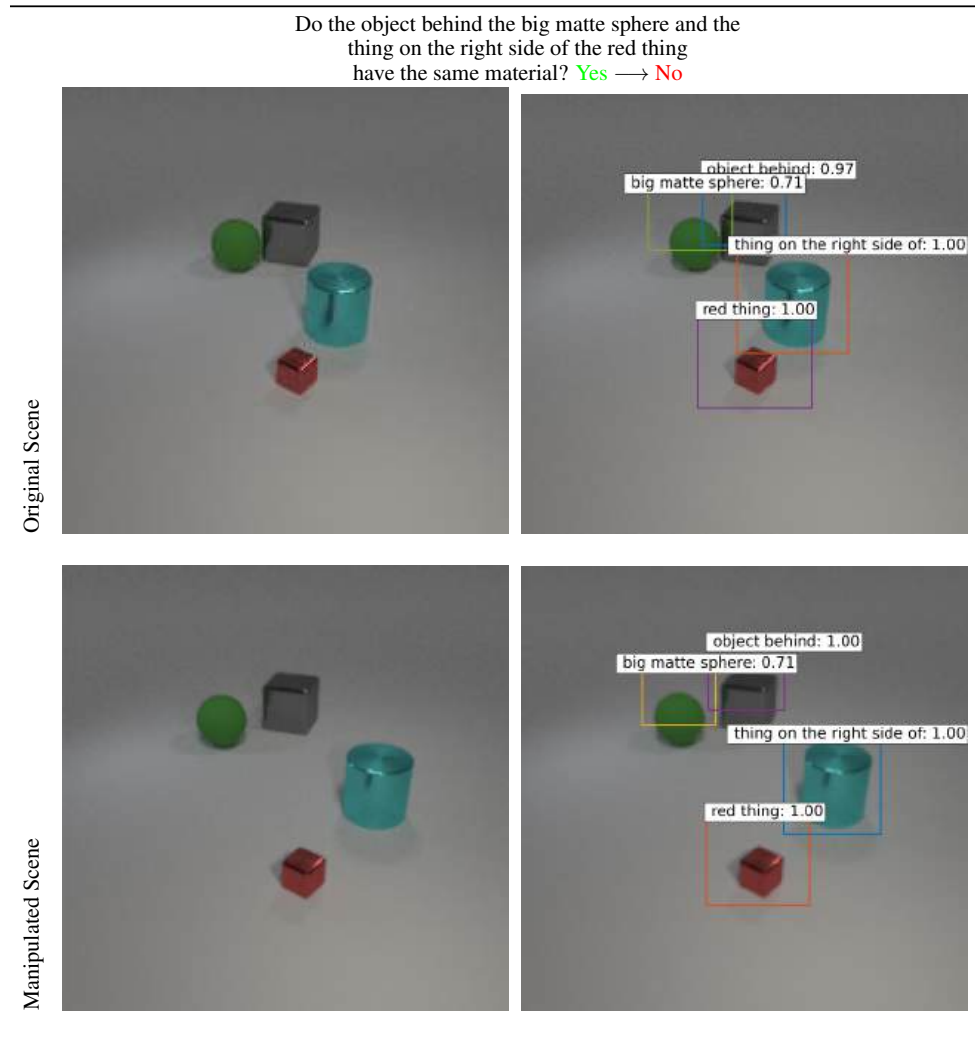


Figure 25: Visualization of MDetr model's detection stages on original and manipulated scenes.

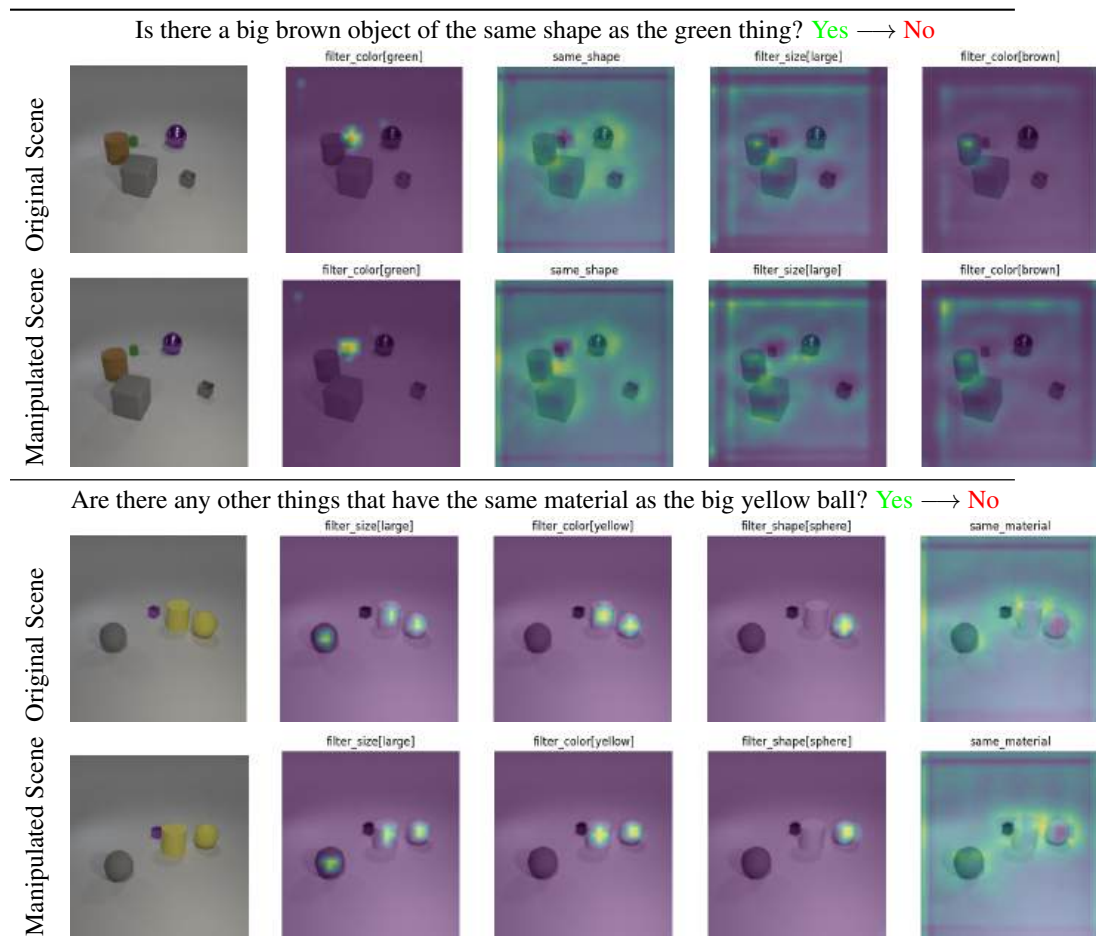
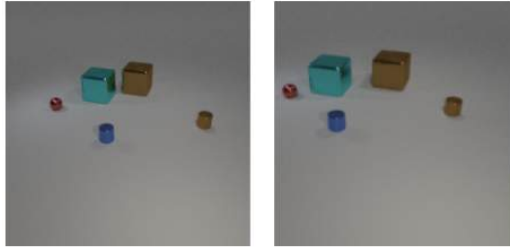


Figure 26: Visualization of TbD model’s program generation and attention map visualization on original and manipulated scenes

B.5 HUMAN-IN-THE-LOOP EXPERIMENT

Can our *Adversarial Player* ‘fool’ a human-agent? The perceptual abilities of human observers have been previously measured (92.6%) on this dataset, however, human robustness is a different question. We, hence, decided to assess human robustness by a simple experiment. In our setting, a human observer is presented with a set of two images and a question. Then the observer is required to answer to each image question pair, (image before - question / image after - question), and report whether the answer has changed or remained the same. Participants were first made familiar with the concepts of the CLEVR dataset (colors / shapes / materials / sizes) and then were left unsupervised to answer a set of 20 such examples. Each contained one CLEVR question and two CLEVR scenes, before and after the *Adversarial Player* manipulations. Half of those scenes successfully fooled models and half of those did not. An example of what a survey question looks like is given in Figure 27, together with the confusion matrix Figure 28 (results of our survey).

There is a cube on the left side of the brown object that is behind the red ball. What size is it?



- Same
- Different

Figure 27: Questionnaire example: The observer is presented with two images and a question. Then they have to pick if their answer would remain the same if the question referred to each image respectively.

As we can see in Figure 28, human observers exhibit 91% precision in identifying cases where scene manipulations are not causing changes to the questions answer, and 95% precision in identifying cases where the manipulations do cause a change. While this small scale experiment may not reflect a universal truth about human robustness in VQA scenarios, it is a suggestion that human observation is still reliable after our agent’s manipulations.

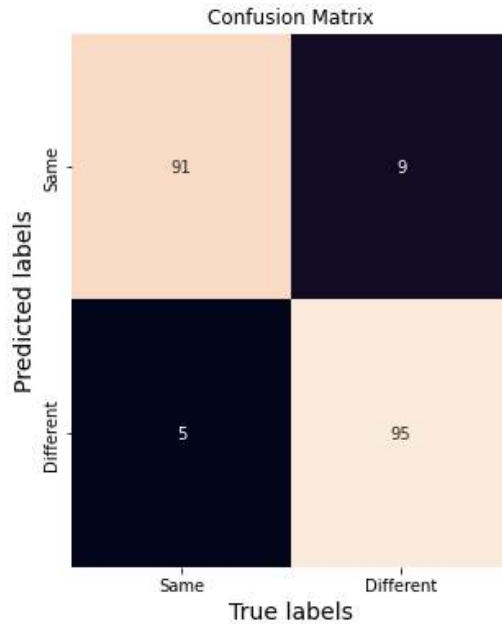


Figure 28: Confusion matrix of human experiment: Participants identified 91% of cases where manipulations were not causing a change in the ground-truth answer. All of these managed to fool *Visual-QA Players* and made up half of the survey. When faced with manipulations that changed the ground-truth answer (the other half of the survey), they identified 95% of them.

B.6 OBJECT DETECTION EXPERIMENT

As an extra test to confirm the visual fairness (see section B.2) of *Adversarial Player* manipulations we run the following experiment. We employed an unsupervised object discovery model, called Slot Attention (Locatello, 2020), pre-trained on CLEVR, together with a classifier trained to identify object attributes. Next, we used that model to identify objects and attributes on original and manipulated scenes. Ultimately, we could not find any pairs in which the number of objects or their attributes were different before and after the manipulations. The results for two pairs can be seen in Figure 29 and Figure 30. We thus conclude that our *Adversarial Player* manipulations create ‘visually fair’ scenes, without producing any corner cases of perceptual system.

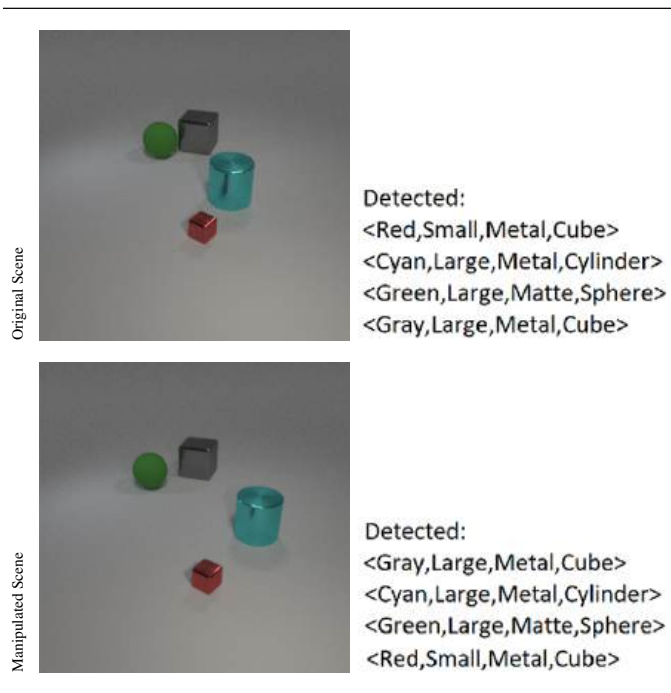


Figure 29: Visualization of Slot Attention model's detection results

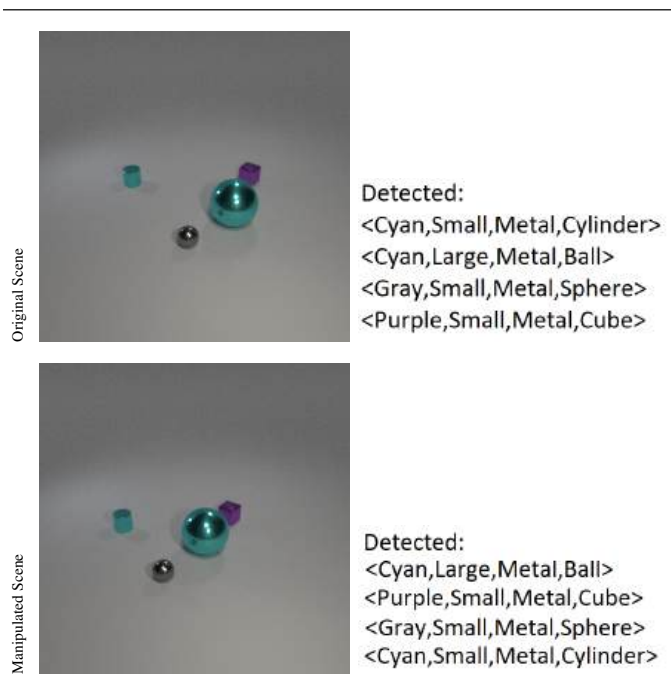


Figure 30: Visualization of Slot Attention model's detection results

A Simple, Yet Effective Approach to Finding Biases in Code Generation

Spyridon Mouselinos

University of Warsaw
s.mouselinos@uw.edu.pl

Mateusz Malinowski

DeepMind
mateuszm@deepmind.com

Henryk Michalewski

Google, University of Warsaw
henrykm@google.com

Abstract

Recently, high-performing code generation systems based on large language models have surfaced. They are trained on massive corpora containing much more natural text than actual executable computer code. This work shows that current code generation systems exhibit undesired biases inherited from their large language model backbones, which can reduce the quality of the generated code under specific circumstances.

To investigate the effect, we propose the "block of influence" concept, which enables a modular decomposition and analysis of the coding challenges. We introduce an automated intervention mechanism reminiscent of adversarial testing that exposes undesired biases through the failure modes of the models under test. Finally, we demonstrate how our framework can be used as a data transformation technique during fine-tuning, acting as a mitigation strategy for these biases.

1 Introduction

Large language models (LLM) have recently demonstrated their ability to generate code (Li et al., 2022; Brown et al., 2020; Wang et al., 2021) or solve challenging programming/math tasks on par with human coders (Li et al., 2022; Lewkowycz et al., 2022b; Chowdhery et al., 2022a); these models are trained with the data-driven paradigm. On the other hand, an increasing body of work also questions whether the data-driven approach leads to acquiring reasoning skills (Piekos et al., 2021; Zhang et al., 2022; Mouselinos et al., 2022), showing that if left alone, it might not be sufficient for achieving truly human-level performance on tasks such as logical or visual reasoning. In many studied cases, models still rely on various hints in their reasoning process. This work extends the results above, i.e., the lack of reasoning capabilities, to the code generation domain. More specifically, we devise a framework that automatically identifies

subtle cues a code generation model might exploit. Changes or removal of those cues stands as a reasoning test towards the generational capabilities of the model at hand.

We presume that the reasoning process of code generation models should remain invariant under changes that still provide enough context or pose little, if any, additional challenge to a human coder. To this end, we propose an automatic and model-agnostic framework that modifies the following: (1) function names, (2) keywords in a problem specification, and (3) examples provided in the problem prompt. We refer to these parts as Blocks-Of-Influence; see Figure 1. Each block contributes partially to the context needed for correct completion. We show that minor modifications of these blocks are sufficient to "fool" LLM-based code generation methods.

Our results reveal biases such as keyword preference and memorization effects, which can be identified across multiple models. During our experiments, we ensure that any modifications maintain the global semantics of the coding challenge. This is achieved through a context-aware filtering mechanism that guarantees any information altered or removed still exists and/or can be deduced from the remaining unaltered part.

Contributions. The main contributions of our work can be summarized in three points.

First, we propose a novel automated framework that identifies possible biases in code generation models. Our framework removes subtle hints, introducing minimal changes such as keyword replacement or partial code-block omission, ultimately acting as an adversarial test. Since the framework operates on a data level, it is agnostic to the model's structure and internal workings. The framework can be easily adjusted to any input format or programming language.

Second, we introduce the "Blocks of Influence" concept. We suggest that every instance of a typical

coding challenge can be analyzed into three parts (blocks). Each part is correlated with a different method of hinting and is used as a target of our transformations. A model’s reasoning process is informed by all three blocks, making them perfect analyzing tools for cases of failing code generation. *Third*, we explore new ways of mitigating biases during code generation. In Section 6, we study the effects of adversarial training against our proposed perturbations, and the benefits of including examples with longer descriptions during fine-tuning. Our results show that combining these techniques leads to more accurate code completions.

2 Related Work

Our approach is inspired by works of various research directions, which we briefly describe here.

Solving coding and math challenges. The emergent abilities of large language models to generate, summarize and translate textual information, have recently sparked interest in their aptitude for math, logic, and programming challenges. Tasks such as code-completion (Chen et al., 2021; Shin et al., 2019; Hendrycks et al., 2021a; Li et al., 2022), code summarization and code translation (Lu et al., 2021) have been proposed, with models constantly progressing towards near-human performance. Similarly, (Hendrycks et al., 2021b; Saxton et al., 2019; Ling et al., 2017; Amini et al., 2019) have proposed tests measuring a model’s ability to perform math and logic, ranging from school problems to competition-grade challenges. Impressive results in multiple programming languages have also been achieved by decoder-only works (Brown et al., 2020; Chen et al., 2021). Fried et al. (2022) created the first generative model to perform infilling using a novel masking objective. Finally, massive-scale models such as (Chowdhery et al., 2022b; Lewkowycz et al., 2022a) demonstrated breakthrough capabilities in language, reasoning, and code tasks achieving state-of-the-art performance in multiple domains simultaneously.

Social biases in large language models. Trained on ever-increasing amounts of publicly available data, large language models have been studied for adopting social biases commonly found among humans. Wallace et al. (2019) show that generative models can be conditioned to produce toxic content, with the use of nonsense, adversarial prefixes. Similarly, Liang et al. (2021) suggest that models might adopt biases and social stereotypes found among their training data and provide ways to ap-

ply fairness during generation. Countermeasures have been proposed by (Zhao et al., 2021; Liu et al., 2022), claiming that sanitized zero-shot examples contribute to mitigating biases during generation.

Probing reasoning through cognitive biases. There have been notable attempts to systemize intelligence and reasoning as concepts (Legg, 2008; Chollet, 2019), yet a few recent works try to approach reasoning, through the analysis of failure modes, caused by biases in deep learning models. Glockner et al. (2018) suggest that natural language inference systems can be easily fooled with a single hypernym/hyponym swap, exhibiting a bias towards specific word choices. Similarly, Lin et al. (2020) prove that numerical commonsense reasoning in LLMs is heavily biased by adjectives describing the object of interest. Concerns against the current data-driven methods have been expressed by Razeghi et al. (2022), pointing out that LLMs are more accurate on mathematical challenges that involve terms significantly more frequently in their pre-training dataset. Piekos et al. (2021) claim that LLMs can answer math and logic questions without understanding the rationale behind them, relying blindly on the existence of specific keywords. We place our work in this line of research, provoking and studying the failures of LLMs under reasoning-heavy coding tasks. Our main goal consists of identifying cognitive bias sources, i.e., words, structures, or co-occurrence patterns, that exist in current LLMs, and lead to systematic failures of rationale.

Adversarial methods and Language Processing. NLP community developed excellent methods to prepare adversarial tasks, including the TextAttack framework (Morris et al., 2020) and sophisticated techniques to elicit adversarial examples from humans, as in Talmor et al. (2022), though our work seems to be the first focused on the disciplined construction of adversarial examples for code.

3 Benchmarks

In this section, we describe the datasets used in our experiments. We employed widely used coding challenges HumanEval (HE) and MBPP and a more complex dataset with lengthy descriptions of problems (DMCC). More information about the datasets can be found in the Appendix 10.2.

HumanEval (HE). This is a human-curated problem-solving dataset described in Chen et al. (2021). It consists of 164 original programming challenges assessing language comprehension, al-

```

def rev_list_take2(a: List):
    "Reverse the list and return the second item."
    Examples:
    >>> rev_list_take2(['a',2,'apple'])
    2

```

```

def func(a: List):
    "Reverse the list and return the second item."
    Examples:
    >>> rev_list_take2(['a',2,'apple'])
    2

```

Figure 1—**Left:** The three blocks of influence: *Name Block* in red, *Description Block* in green and *Example Block* in blue. **Right:** We demonstrate three possible transformations, one for each block: Swap the function name with "func", remove keywords, and remove examples. Transformations can be applied alone or in combinations of two as described in Section 5.2

gorithms, and simple mathematics. Each problem is presented as an incomplete function, accompanied by a docstring. The docstring contains the task and a few example cases. For each task, we are provided with a set of unit tests. A task is considered solved when all unit tests are passed.

Mostly Basic Python Problems (MBPP). Introduced in Austin et al. (2021), it contains 974 short Python functions designed to be solved by entry-level programmers. Contrary to HumanEval, each task is given through a text description rather than a docstring. Since there are no input-output examples in the prompt, we generate 3 valid pairs using the code solutions provided. c MBPP challenges models to perform tasks of imperative control flow, requiring loops and conditionals.

DeepMind Code Contests (DMCC). Is the highly challenging dataset proposed by Li et al. (2022). The dataset includes problems from the Codeforces platform (Mirzayanov, 2020), Description2Code (Caballero, 2016), and CodeNet (Puri et al., 2021). We used challenges written in the Python3 language of the training split for our experiments. DMCC contains long descriptions of the problems and input-output examples of the functions to be completed.

In this work, DMCC is used for its long context properties during experiments of augmented fine-tuning (Table 5). Models presented in our work achieve zero or near-zero scores on it; hence it is excluded from our perturbation analysis, with HumanEval and MBPP being more suitable targets.

4 Evaluation

Models. In our experimental setup, we test five models representing different approaches to code generation. CodeParrot (Tunstall et al., 2022a) comes with an open-source dataset and can be easily used for fine-tuning experiments due to its size. Its smaller variant (110M) achieves competitive results to other open-source LLMs at larger parameter budgets. By exploring its dataset, we tested our hypothesis that function names act as biases during

code generation. Models can be heavily inspired by similarly named snippets in their training set and resort to copying whole or parts of the solution instead of performing reasoning. (See Appendix A.9) We also test the InCoder (Fried et al., 2022) model, which is trained under a novel bi-directional causal objective, being able to handle context more efficiently than its causal counterparts. Against our initial hypothesis, our methods cause significant performance drops despite the model’s enhanced context-understanding capabilities (Table 3). The Bloom model (Mitchell et al., 2022) exhibits emergent abilities in multiple domains by training on massive multilingual and multi-purpose content. Despite not being a code generation model, it performs equally well with code-specific models in the same parameter budget. Theoretically, bias effects can be reduced when a model is exposed to diverse training examples. Our experiments reveal that this is still not the case under our setup, and post-training solutions are explored. CodeGen (Nijkamp et al., 2022) is a high-performing model trained in natural language understanding and code. We test its Mono variant, further fine-tuned on the Python language. Finally, we have the powerful Codex model, which can tackle most of the proposed coding challenges in the HumanEval and MBPP datasets. A list of the tested models, as well as KeyBert (Grootendorst, 2020) that is used in our framework, can be found in Table 1.

Model Name	Sizes Used
KeyBert (Grootendorst, 2020)	2M
Codeparrot (Tunstall et al., 2022a)	110M / 350M* / 1.5B
InCoder (Fried et al., 2022)	1.6B / 6B
CodeGen (Nijkamp et al., 2022)	350M / 6B
Bloom (Mitchell et al., 2022)	560M* / 1.7B / 176B [†]
Codex (v1 / v2) (Chen et al., 2021)	~175B [†] (Estimated)

Table 1: Models used: (*) refers to fine-tuned and ([†]) to API.

Performance metrics. We evaluate the functional correctness of the generated programs with the pass@k metric, introduced in Kulal et al. (2019). This metric serves as an estimator of a model’s generative capabilities under a specific budget. In Chen et al. (2021), authors propose an updated un-

biased version that we adopt throughout the rest of this work. To avoid any confusion, we calculate pass@k at exactly k attempts. The average of ten runs with different seeds is presented for all experiments in Table 3. We use sampling temperatures of 0.2 / 0.8 for pass@1 / pass@100, which are the optimal values across the tested models.

5 Method

5.1 Blocks of Influence

Our method treats each coding challenge as a combination of three distinct but complementary blocks rather than a single, homogeneous input. We refer to them as *Blocks of Influence* and correlate each with a different source of bias during code generation. Taking as an example Figure 1, we challenge the model to complete a function that reverses a list and then returns its second item.

Name Block. The first block of influence, marked in red, informs the model about the function name and the names and types of the input arguments. Let us assume that initially, a model generates correct solutions to a problem. However, the model fails when we rename the function name to something unrelated to the task, e.g., “*fun*“. This failure mode indicates that neither the problem description was understood nor the model could extract a reasoning pattern from the given usage examples. We associate such cases with memorization effects, where the model relies heavily on the function name, replicating snippets from its training dataset with the same or similar names.

Description Block. The problem description stands as the second block, marked in green. Here the model is expected to form a solution by utilizing its natural language understanding capabilities. We observe that removing specific keywords from the problem description can lead to catastrophic results in model performance. It is vital that removing these keywords must not degrade the description semantics, and any information lost should be recoverable from the rest of the context. For example, in Figure 1, the removal of the word pair “the list” creates a description that is still well understandable by a human coder. We challenge the model to deduct the missing context from the word “list” in the function name and the input list type in the example given. The inability to recover the missing context is associated with an inherent preference bias, where the model relies on superficial lexical clues or frequently co-occurring terms seen during

training rather than the given context to “mentally” fill any gaps.

Example Block. As the final block, we consider the examples after the problem description. They act as demonstrations, guiding the model to specific reasoning patterns. Let us consider a scenario where models cannot generate correct code when examples are absent. Arguably, more than the task and given inputs alone were needed for the model to form a proper problem understanding. In this failure mode, the provided examples act as a “reasoning tie-breaker” between proposed solutions the model can generate. Generated solutions are not entirely irrelevant but a relatively poor interpretation of the problem. For example, in Figure 2, when stripped of its examples, the model still exhibits signs of task understanding (i.e., comparing element difference to a threshold, iterating over elements). However, combining these logic parts in a meaningful manner is complex enough that the model requires additional examples to filter out faulty strategies. We associate such effects with poor reasoning abilities.

5.2 Framework

The first step involves splitting a coding challenge into the three *Blocks of Influence*. For this purpose, we utilize a regular expression module that searches for common patterns of each block’s start or end. (e.g., *Name Block*: “def (...):”, *Description Block*: ” or ””, *Example Block*: ”Examples:” or > / >> followed by usage of the function name).

As the next step, the *Description Block* is further analyzed to identify possible hinting keywords. Ideally, we are interested in unigrams or bigrams that provide excess information towards completing the coding task. For keyword identification, we use KeyBert (Grootendorst, 2020), an LLM tasked to perform keyword extraction and word similarity. We proceed to fine-tune KeyBert on the open-source CodeParrot dataset (Tunstall et al., 2022a) so that more code-specific suggestions are provided. For each candidate keyword, we calculate its embedding similarity with the set of words: [Python, Programming, Code, Variable], again through KeyBert. Words with cosine similarity scores under 0.7 for all the items of the set are unrelated to coding and thus filtered out. However, carelessly removing keywords can lead to non-interesting drops in performance associated with removing crucial information rather than hinting effects. Thus, an additional context-aware filtering stage is employed

to validate that any information lost can be retrieved from the remaining coding challenge.

During this stage, we compute each candidate keyword’s embedding similarity with every non-potential keyword token. The keyword is marked as valid for removal if at least one “close” word is identified. Again, we consider “close” keywords with a similarity score larger than 0.7. If a keyword exists in multiple locations, the first instance is not marked as valid for removal, while the rest are. When a keyword happens to be an argument type (i.e., list, integer, tuple), we additionally look for instances of that type in the examples or name block. In case of a match, the keyword is safe for removal. Equivalent information already exists in the context. As the final step, we chose between the following transformations:

Drop one. Removes one of the provided keywords from the *Description Block*. The transformation is repeated N times where N is the number of identified keywords.

Drop all. Removes all the provided keywords simultaneously from the *Description Block*

Drop examples. Removes all the provided examples from the *Example Block*.

Anonymize. Replaces the function name with an arbitrary token. We use “*func*” in our experiments. Note that the function name is also replaced in the provided examples, so no information leak occurs. We also tested whether the choice of “*func*” may potentially bear some intrinsic adversarial effect associated with the training data. We experimented with other word choice replacements (“*action*”, “*do_stuff*”, “*XYZ*”) and got the same results. Furthermore, we identified instances where the function name, although closely correlated to the task at hand, if it was to be taken as the sole source of information, could instead be misleading, signifying the need for proper context understanding by the tested models (See Appendix 10.8).

For example, let us use our framework on the challenge presented in Figure 1. At the first stage, KeyBert would have identified the following keywords: [Reverse, list, return, second]. Among these, the word *second* does not pass the first filtering stage with over 0.7 similarity score against our set. In the second stage, each word would be compared against all the existing tokens. *Reverse* and *return* will not be associated with other tokens. *List* will be identified in the function name and in-

put argument type. Also, since *list* is also a python keyword, it will be matched against the list type of the input given in the examples. This leaves *list* as the only available keyword for removal. If keyword drop would be combined with anonymization, the drop would still be valid since the information would still be available in the examples and input type.

These transformations test the hypotheses we associate with each block, as presented in Section 5.1. Removing possible hints leads to performance drops between the original and modified challenges, revealing underlying biases in the models’ logic. Arguably, any of our suggested transformations can destroy local semantics. However, we take significant measures to ensure that global semantics is preserved and enough information exists towards its solution. This is also why we refrain from performing simultaneous transformations in the *Example Block* and *Description Block*, or all of the *Blocks of Influence* together; a model stripped of all necessary information cannot generate a proper solution. To quantify the possible degree of ambiguity our transformations introduce, we employ the LM critic test, inspired by the work of (Yasunaga et al., 2021; Yasunaga and Liang, 2021): We collect a random sample of 200 coding challenges from the HumanEval and MBPP. Each challenge is then transformed according to the methods presented in Table 2. Afterwards, for both the original and every modified version of a challenge, we calculate their log probability score using a large language model. The core idea is that the model will act as a soft critic, ranking model inputs by their overall plausibility. Modified inputs that seem “off” to the critic and are partially understood will be assigned a log probability score far lower than the unmodified ones. Since this criterion is based on local neighborhood optimality, only moderate changes are allowed between the challenges under comparison. For example, two completely different but syntactically and semantically correct text snippets can have similar log probability scores. During their comparison, however, we would have violated the locality assumption, and no conclusions could be drawn about their contents. As our critic, we employ the Codex-v2 model (Chen et al., 2021). We calculate log probability similarity as:
$$Sim = 100 - \frac{LogP_{Method} - LogP_{Original}}{LogP_{Original}}$$
 Table 2 shows that our transformations do not introduce drastic changes to the coding challenge.

Method	Similarity (%)	
	w/ CAF	w/o CAF
Original	100.0 (\pm 0.0)	100.0 (\pm 0.0)
Anonymization	98.5 (\pm 1.2)	98.5 (\pm 1.2)
Drop One	97.3 (\pm 1.5)	84.2 (\pm 2.2)
Drop All	95.3 (\pm 1.9)	80.3 (\pm 2.8)
Anonymization + Drop One	95.8 (\pm 1.4)	80.9 (\pm 2.3)
Anonymization + Drop All	94.6 (\pm 2.3)	78.4 (\pm 3.1)

Table 2: Similarity scores for different methods of our framework with (w/) and without (w/o) the proposed context-aware filtering mechanism (CAF). Results of 200 samples are presented.

Even in the most aggressive transformation of *Anonymization + Drop All*, the critic assigns over 94% similarity between code challenges affected by it versus their original form. For comparison, removing the context-aware filtering stage, leads to only 78% similarity in the case of *Anonymization + Drop All* transformation. We believe this is a fair indicator that the tested models observe inputs of similar quality and comprehensibility during our experiments. Note that we omit results for the *Drop Examples* method. In this case, the log probabilities will significantly change since we remove many tokens, which violates the method’s locality prerequisite.

6 Experiments

6.1 Results on Block Transformations

The main results of our experiments are presented in Table 3. Despite their simplicity, our transformations cause consistent drops in performance across different model sizes on both datasets.¹ Mere anonymization causes drops of 19% on average in both Pass@1 and Pass@100 metrics, validating our claims of memorization effects. Single (*Drop One*) and full keyword removal (*Drop All*) reduce models’ performance by 15% and 22% on average, suggesting their inability to deduct the missing context from *Name Block* and *Example Block*. Instead, models rely on generating arbitrary, commonly used snippets that vaguely fit for the task. Especially interesting are the cases of *Drop Examples* and *Anonymize + Drop Examples*, with 15% and 25% average drops. Both transformations remove the information provided by the docstring examples, with the latter having the additional restriction of an anonymized function. With

¹We present a full table of results, including Codeparrot (110M), CodeGen(350M), Bloom(1.7B) and Codex(v1) in Appendix 10.5

the *Description Block* unmodified in both cases, these transformations target the models’ abilities to create solutions based on their natural language understanding. The combination of anonymization with the drop of all keywords (*Anonymize + Drop All*) seems to be the most challenging transformation overall, with drops of approximately 40%. Its primary purpose is to assess the model’s capability of deducting the missing context of the *Description Block* by only observing patterns in the examples. These observations suggest a clear model preference over its sources of information, with the task description being the primary one. Thus, when a model exhausts its ability to understand the task, it exploits similarities of the function name with previously seen code solutions. Simultaneously, the model’s reasoning relies on the example demonstrations, which, as seen from (*Anonymize + Drop All*), are not always able to provide clear directives.

6.2 Towards Bias Mitigation

Inspired by the field of adversarial training, we decided to investigate the effects of using our framework transformations as training augmentations. To this end, we apply our framework to examples of the MBPP challenge and use them as a fine-tuning dataset for three different Codeparrot models. We use HumanEval as our test dataset, which bears no overlap with the MBPP. In this way, our models have not seen examples of the test set during their training or fine-tuning steps. In Table 4, we compare the results of our models before and after fine-tuning. Models benefit from the introduction of augmented examples and partially recover from failure modes caused by the need to rely on hints. The larger the model, the more its abilities benefit. We believe this effect is closely related to large language models’ scaling reasoning capabilities and their parameter size. The need to rely on hints can be attributed to low data quality or lack of task-specific inductive biases. However, the capacity to properly understand coding tasks is undoubtedly there. To improve the code generation abilities of models, we thus suggest exposing them to challenges that push their deductive and reasoning abilities. We decided to repeat the experiments, but without including any of our data augmentation techniques during fine-tuning. We observe that under this setup, models do not exhibit any significant improvement against our method’s perturbations. Our suggested data augmentations that push the reasoning limits of the models are

Table 3: Model results on Human Eval and MBPP.

Method	Codeparrot (1.5B)				InCoder (1.6B)				CodeGen-Mono (6B)			
	Human Eval		MBPP		Human Eval		MBPP		Human Eval		MBPP	
	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)
Original	4.1	17.8	6.1	31.2	11.3	24.2	14.6	56.7	26.1	65.8	42.3	77.3
Drop One	3.9	13.2	4.2	26.8	10.5	22.3	11.5	45.4	18.4	39.3	25.2	65.7
Drop All	3.6	11.1	3.9	21.7	9.7	17.6	12.8	42.1	13.9	34.8	22.4	57.7
Drop Ex	3.7	14.3	5.3	27.5	11.3	22.2	14.4	43.8	20.4	42.3	27.2	61.7
Anon	3.8	12.5	4.7	23.2	9.1	21.8	11.3	45.2	18.2	37.3	24.0	65.6
Anon+Drop One	3.3	9.5	3.9	20.2	7.4	21.5	10.5	44.9	12.6	24.6	15.8	58.6
Anon+Drop All	2.1	8.9	3.9	17.9	6.3	17.5	8.0	41.3	11.5	23.1	14.9	46.3
Anon+Drop Ex	3.7	11.8	4.6	22.8	8.7	21.3	11.2	43.5	16.0	28.3	18.2	60.7

Method	InCoder (6B)				Codex (v2)				Bloom (176B)			
	Human Eval		MBPP		Human Eval		MBPP		Human Eval		MBPP	
	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)
Original	15.2	47.0	19.4	65.1	49.4	91.4	60.1	86.3	16.4	57.2	20.8	62.4
Drop One	12.1	35.3	18.9	52.6	36.0	86.2	56.0	79.2	12.8	48.6	15.8	51.4
Drop All	10.2	28.2	15.6	47.0	37.1	73.7	52.1	69.5	11.5	40.2	14.2	44.4
Drop Ex	12.7	29.5	17.4	50.3	41.4	81.0	48.8	70.7	15.2	43.3	15.8	50.1
Anon	11.6	32.9	14.8	50.7	44.5	90.4	57.9	81.7	14.0	48.3	15.1	51.2
Anon+Drop One	8.1	30.6	13.5	46.7	29.8	74.4	51.2	69.5	12.8	41.9	13.6	46.8
Anon+Drop All	7.5	25.2	11.2	38.9	24.2	68.7	47.2	63.8	10.3	36.8	12.6	38.4
Anon+Drop Ex	11.2	28.1	14.5	50.2	34.1	72.5	42.6	70.5	14.0	39.8	14.3	47.8

Table 4: HumanEval results of fine-tuning Codeparrot on the MBPP dataset with (A) or with no (NA) augmentations: Regular finetuning does not contribute to bias removal, achieving similar results against the perturbations. However, our suggested augmentations lead to higher model performance, especially in the pass@100 metric. The average of 15 runs is presented. Bold marks statistically significant improvements under the T-Test (Before versus After-A) with $\alpha = 0.95$.

Method	Codeparrot - 110M				Codeparrot - 350M				Codeparrot - 1.5B			
	Pass@1 (T=0.2)		Pass@100 (T=0.8)		Pass@1 (T=0.2)		Pass@100 (T=0.8)		Pass@1 (T=0.2)		Pass@100 (T=0.8)	
	Before	After NA / A	Before	After NA / A	Before	After NA / A	Before	After NA / A	Before	After NA / A	Before	After NA / A
Original	3.8	3.7 / 3.7	12.7	12.1 / 12.1	3.8	3.7 / 3.7	13.9	13.7 / 13.7	4.1	4.1 / 4.1	17.8	17.8 / 17.8
Drop One	3.3	3.2 / 3.6	9.7	9.7 / 10.4	3.3	3.3 / 3.6	11.9	11.9 / 12.3	3.9	3.9 / 4.0	13.2	13.2 / 14.1
Drop All	3.1	3.1 / 3.1	7.2	7.2 / 7.9	3.2	3.2 / 3.2	10.1	10.0 / 10.7	3.6	3.6 / 3.7	11.1	11.1 / 12.3
Drop Ex	3.8	3.7 / 3.7	9.9	9.9 / 10.2	3.8	3.8 / 3.7	12.9	12.9 / 12.9	3.7	3.7 / 3.7	14.3	14.3 / 15.1
Anon	3.4	3.4 / 3.5	8.7	8.7 / 9.1	3.6	3.6 / 3.6	11.6	11.6 / 12.2	3.8	3.8 / 3.9	12.5	12.5 / 13.8
Anon+Drop One	3.0	2.8 / 3.4	7.5	7.5 / 7.9	3.0	2.8 / 3.5	8.2	8.2 / 9.4	3.3	3.3 / 3.5	9.5	9.5 / 10.5
Anon+Drop All	1.9	1.9 / 2.0	6.9	6.9 / 6.9	2.0	2.0 / 2.2	8.1	8.0 / 8.3	2.1	2.1 / 2.4	8.9	8.8 / 9.4
Anon+Drop Ex	3.4	3.3 / 3.4	8.7	8.7 / 9.0	3.6	3.6 / 3.6	10.7	10.7 / 11.8	3.7	3.7 / 3.7	11.8	11.8 / 13.7

thus a valid alternative to simple fine-tuning.

6.3 Effects of Longer Context

When causally training on coding datasets, models condition on multiple functions and declarations in the same file. The input is a conglomerate of rapidly changing contexts, with each function or class being a self-contained entity. Subsequently, a model is accustomed to localizing its focus when trained on such data. As an extension to our previous experiment, we measure the effects of using a long description dataset, DMCC, as a fine-tuning target. By training on long descriptions of natural language, we promote the context-deducting skills of the model under test. A model able to widen its focus can avoid distractions caused by missing keywords. Efficient context understanding will replace not rely heavily on internal biases. We choose Bloom as the model under test since it was not explicitly tuned for code genera-

tion but rather general language understanding. In Table 5, we present results of fine-tuning on MBPP, modified by our framework. We observe similar performance improvements as in Table 4. We ex-

Table 5: HumanEval results of fine-tuning Bloom (560M) on the modified MBPP and long-description DMCC dataset with (A) or without (NA) augmentations: Model exhibits increased performance under the combined augmentation schema against perturbations that challenge language understanding. The average of 15 runs is presented. Bold marks statistically significant improvements under the T-Test (Before versus +DMCC-A) with $\alpha = 0.95$.

Method	Pass@1 (T=0.2)			Pass@100 (T=0.8)		
	Before	+MBPP	+DMCC	Before	+MBPP	+DMCC
	NA / A	NA / A	NA / A	NA / A	NA / A	NA / A
Original	3.7	3.6 / 3.6	3.6 / 3.6	12.1	12.1 / 12.1	12.0 / 12.0
Drop One	3.1	3.1 / 3.6	3.1 / 3.6	10.3	10.3 / 10.9	10.3 / 10.9
Drop All	2.4	2.3 / 2.4	2.3 / 2.9	9.2	9.1 / 9.1	9.1 / 9.7
Drop Ex	3.0	3.0 / 3.0	3.0 / 3.0	11.0	11.0 / 11.3	11.0 / 11.5
Anon	2.5	2.5 / 3.0	2.6 / 3.6	10.7	10.7 / 10.9	10.8 / 11.3
Anon+Drop One	1.9	1.9 / 2.3	1.9 / 2.4	7.8	7.8 / 9.1	7.8 / 9.7
Anon+Drop All	1.8	1.8 / 1.8	1.8 / 2.3	7.0	7.0 / 7.2	7.0 / 8.3
Anon+Drop Ex	2.4	2.4 / 2.9	2.4 / 3.0	9.7	9.7 / 10.3	9.7 / 11.4

periment again, this time combining both MBPP and DMCC examples. We show that incorporating examples of more extended context leads to even better performance against transformations targeting the *Description Block* and language understanding. Similar experiments were conducted with the CodeParrot variants but were unfruitful. We attribute this to the restricted focus regarding training data (exclusively Python3 code) and architectural differences between the models. We believe that the merging benefits of our two proposed setups can serve as an interesting direction towards model resilience in code generation scenarios.

7 Conclusions

We present a simple approach to isolate cues and benchmark the reasoning of code generation models through input-level transformations. Our

```

from typing import List
def has_close_elements(numbers: List, threshold: float):
    "Check if in the given list of numbers, are any two numbers closer
    ↳ to each other than given threshold."
    Examples:
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0,2.0,3.0,4.0,5.0,2.0], 0.3)
    True
    return any(abs(x-y) < threshold for x,y in zip(numbers,
    ↳ numbers[1:]))

from typing import List
def has_close_elements(numbers: List, threshold: float):
    "Check if in the given list of numbers, are any two numbers closer
    ↳ to each other than given threshold."
    for i in range(len(numbers) - 1):
        if abs(numbers[i] - numbers[i+1]) < threshold:
            return True
    return False

```

Figure 2—Example removal reveals poor reasoning (*Example drop* / Codex-v1): The model initially exhibits signs of task comprehension (top), generating a correct solution. Removing the examples, however, reveals a lack of proper reasoning; Although the model still understands that it has to compare numbers, it resorts to a naive sequential check instead of comparing each available pair (bottom).

```

from typing import List
def string_xor(a: str, b: str) -> str:
    "Input is two strings a and b consisting only of 1s and 0s. Perform
    ↳ binary XOR on these inputs and return the result as a string."
    Examples:
    >>> string_xor('010', '110')
    '100'
    return ''.join([str(int(a,2) ^ int(b,2)) for a,b in zip(a,b)])

from typing import List
def string_xor(a: str, b: str) -> str:
    "Input is a and b consisting only of 1s and 0s. Perform XOR on
    ↳ these inputs and return the result."
    Examples:
    >>> string_xor('010', '110')
    '100'
    return ''.join([str(int(a) ^ int(b))])

```

Figure 3—Keyword hinting (*Drop All* / Bloom 175B): After the removal of keywords, the context remains intact: The *two strings* keyword can be assumed by observing the function arguments, and the *binary/string* keywords by the examples and return type signature of the function. Nevertheless, the model fails to generate a correct solution (bottom).

method treats code examples as a combination of three blocks, each providing different cues to the model. We show that minor transformations can lead models to failure, signifying the existence of biases. Our framework can automatically identify and remove keywords responsible for indirect hinting. We show that popular models with solid results on challenging coding challenges are susceptible to our tests, with their performance degrading noticeably. Moreover, we studied the effects of utilizing our proposed transformations during the fine-tuning of a model. Models can benefit from our proposed changes, with the effect proportional to their parameter size. We believe that, despite their success, code generation systems with LLMs as backbones inherit some of their biases and modes of failure. Training on structured and well-documented code, combined with our proposed techniques, is a promising direction towards reliable code generation. Although an ideal fit for competition-style challenges, our method can be extended to support less formatted high-quality codebases (e.g. GitHub repositories). For a short analysis see Section 10.1 of the Appendix.

```

def pairs_sum_to_zero(l):
    "Pairs_sum_to_zero takes a list of integers as input. It returns
    ↳ True if there are two distinct elements in the list that sum to
    ↳ zero, and False otherwise."
    Examples:
    >>> pairs_sum_to_zero([1,3,5,8])
    False
    >>> pairs_sum_to_zero([1,3,-2,1])
    False
    >>> pairs_sum_to_zero([2,4,-5,3,5,7])
    True
    for i in range(len(l)):
        for j in range(i+1, len(l)):
            if l[i] + l[j] == 0:
                return True
    return False

def func(l):
    "Func takes a list of integers as input. It returns True if there
    ↳ are two distinct elements in the list that sum to zero, and
    ↳ False otherwise."
    return any(sum(x) == 0 for x in l)

```

Figure 4—Anonymize + Drop Examples / InCoder 6B: Using only the problem description, the model creates partially informed subparts (*any* derives from "if there are", *sum(x) == 0* from "sum to zero", and *for x in l* from "elements in the list") that are not combined correctly to solve the task (bottom), signifying that hints from the function name / examples were used in the correct solution (top).

8 Limitations

Some limitations and possible research directions exist in our work. Our study focuses on the Python3 programming language, with many coding challenges existing in different popular choices (e.g., C, C++, Java, Scala). Although the Blocks of Influence identification mechanism could be easily

adapted to each case, an off-the-shelf application of our framework in another language would lead to errors.

Similarly, the framework assumes that each coding challenge will be in a "competition-style" format, meaning that a proper problem description, in-docstring examples, and each input types are present for each example. In Appendix Section 10.1, we present how an adaptation to less formatted codebases would be possible, but for now, we leave it as a future investigation.

Finally, there is no guarantee that the improved performance against the suggested perturbations reflects an equivalent performance increase in real-world code assistant applications. Real-time coding suggestions and completions that are more user aligned are out of the scope of this work.

9 Risks and Ethical Considerations

Our research aims to discover and remove biases in code-generation scenarios through adversarial intervention. However, we acknowledge that insecure or malicious code can still be generated after finetuning with our suggested augmentations. Furthermore, our work is focused only on cognitive biases that affect the reasoning and logic behind the coding process of large language models. Social biases and stereotypes can still appear when general-purpose LLMs such as Codex or Bloom are used in typical text generation scenarios. Signs of robustness against our methods are not to be confused with indicators of other forms of biases not existent.

Acknowledgements

All experiments were performed using the Entropy cluster funded by NVIDIA, Intel, the Polish National Science Center grant UMO-2017/26/E/ST6/00622 and ERC Starting Grant TOTAL. The work of Spyridon Mouselinos and Henryk Michalewski was supported by the Polish National Science Center grant UMO-2018/29/B/ST6/02959.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. *Mathqa: Towards interpretable math word problem solving with operation-based formalisms*. *CoRR*, abs/1905.13319.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan,

Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. *Program synthesis with large language models*. *CoRR*, abs/2108.07732.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. *Language models are few-shot learners*. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

E. Caballero. 2016. *Description2code dataset*, 8 2016.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. *Evaluating large language models trained on code*. *arXiv preprint arXiv:2107.03374*.

François Chollet. 2019. *On the measure of intelligence*. *arXiv preprint arXiv:1911.01547*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022a. *Palm: Scaling language modeling with pathways*. *arXiv preprint arXiv:2204.02311*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022b. *Palm: Scaling language modeling with pathways*. *arXiv preprint arXiv:2204.02311*.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. 2022. *Incoder: A generative model for code infilling and synthesis*. *arXiv preprint arXiv:2204.05999*.

Max Glockner, Vered Shwartz, and Yoav Goldberg. 2018. *Breaking NLI systems with sentences that require simple lexical inferences*. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 650–655, Melbourne, Australia. Association for Computational Linguistics.

Maarten Grootendorst. 2020. *Keybert: Minimal keyword extraction with bert*.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, and Sourab Mangrulkar. 2022. *Accelerate: Training and inference at scale made simple, efficient and adaptable*. <https://github.com/huggingface/accelerate>.

- Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021a. Measuring coding challenge competence with apps. *NeurIPS*.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *NeurIPS*.
- Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. 2019. *Spoc: Search-based pseudocode to code*. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Shane Legg. 2008. *Machine super intelligence*. Ph.D. thesis, Università della Svizzera italiana.
- Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022a. [Solving quantitative reasoning problems with language models](#).
- Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022b. [Solving quantitative reasoning problems with language models](#). *arXiv preprint arXiv:2206.14858*.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. [Competition-level code generation with alpha-code](#).
- Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2021. [Towards understanding and mitigating social biases in language models](#). In *International Conference on Machine Learning*, pages 6565–6576. PMLR.
- Bill Yuchen Lin, Seyeon Lee, Rahul Khanna, and Xiang Ren. 2020. [Birds have four legs?! NumerSense: Probing Numerical Commonsense Knowledge of Pre-Trained Language Models](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6862–6868, Online. Association for Computational Linguistics.
- Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. [Program induction by rationale generation: Learning to solve and explain algebraic word problems](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.
- Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. [What makes good in-context examples for GPT-3?](#) In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online. Association for Computational Linguistics.
- Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. [Codexglue: A machine learning benchmark dataset for code understanding and generation](#). *CoRR*, abs/2102.04664.
- M. Mirzayanov. 2020. [Codeforces: Results of 2020](#).
- Margaret Mitchell, Giada Pistilli, Yacine Jernite, Ezinwanne Ozoani, Marissa Gerchick, Nazneen Rajani, Sasha Luccioni, Irene Solaiman, Maraim Masoud, Somaieh Nikpoor, Carlos Muñoz Ferrandis, Stas Bekman, Christopher Akiki, Danish Contractor, David Lansky, Angelina McMillan-Major, Tristan Thrush, Suzana Ilić, Gérard Dupont, Shayne Longpre, Manan Dey, Stella Biderman, Douwe Kiela, Emi Baylora, Teven Le Scao, Aaron Gokaslan, Julien Launay, and Niklas Muennighoff. 2022. [The world’s largest open multilingual language model: Bloom](#).
- John X. Morris, Eli Lifland, Jin Yong Yoo, and Yanjun Qi. 2020. [Textattack: A framework for adversarial attacks in natural language processing](#). *CoRR*, abs/2005.05909.
- Spyridon Mouselinos, Henryk Michalewski, and Mateusz Malinowski. 2022. [Measuring clevrness: Blackbox testing of visual reasoning models](#). *ICLR: International Conference on Learning Representations*.
- Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. [Codegen: An open large language model for code with multi-turn program synthesis](#). *arXiv preprint*.
- Piotr Piekos, Henryk Michalewski, and Mateusz Malinowski. 2021. [Measuring and improving bert’s mathematical abilities by predicting the order of reasoning](#). *ACL: Association for Computational Linguistics*.
- Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir R. Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, and Ulrich Finkler. 2021. [Project codenet: A large-scale AI for code dataset for learning a diversity of coding tasks](#). *CoRR*, abs/2105.12655.

- Yasaman Razeghi, Robert L. Logan, Matt Gardner, and Sameer Singh. 2022. [Impact of pretraining term frequencies on few-shot reasoning](#).
- Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. [Zero-offload: Democratizing billion-scale model training](#).
- David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. [Analysing mathematical reasoning abilities of neural models](#). In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.
- Eui Chul Shin, Miltiadis Allamanis, Marc Brockschmidt, and Alex Polozov. 2019. Program synthesis and semantic parsing with learned code idioms. *Advances in Neural Information Processing Systems*, 32.
- Alon Talmor, Ori Yoran, Ronan Le Bras, Chandra Bhagavatula, Yoav Goldberg, Yejin Choi, and Jonathan Berant. 2022. [Commonsenseqa 2.0: Exposing the limits of AI through gamification](#). *CoRR*, abs/2201.05320.
- Lewis Tunstall, Leandro von Werra, and Thomas Wolf. 2022a. Natural language processing with transformers.
- Lewis Tunstall, Leandro von Werra, and Thomas Wolf. 2022b. Natural language processing with transformers.
- Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. [Universal adversarial triggers for attacking and analyzing NLP](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162, Hong Kong, China. Association for Computational Linguistics.
- Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*.
- Michihiro Yasunaga, Jure Leskovec, and Percy Liang. 2021. Lm-critic: Language models for unsupervised grammatical error correction. In *Empirical Methods in Natural Language Processing (EMNLP)*.
- Michihiro Yasunaga and Percy Liang. 2021. [Break-it-fix-it: Unsupervised learning for program repair](#). In *ICML*, pages 11941–11952.
- Honghua Zhang, Liunian Harold Li, Tao Meng, Kai-Wei Chang, and Guy Van den Broeck. 2022. On the paradox of learning to reason from data. *arXiv preprint arXiv:2205.11502*.
- Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. [Calibrate before use: Improving few-shot performance of language models](#). In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12697–12706. PMLR.

10 Appendix

10.1 Extension to open-source code

Although an ideal fit for competition-style challenges, our method can be extended to support less formatted high-quality codebases (e.g. GitHub repositories). Large files can be broken down into individual functions/classes, each further analyzed into Blocks of Influence. In such codebases, function names should be closely relevant to their purpose. The existence of meaningful docstrings is crucial, the absence of which promotes more memorization and biases as we exhibited. Moreover, the input/output checks contained in function unit tests can be repurposed as function examples. Keywords can be chosen similarly, with the context being co-informed by both local and larger scopes.

10.2 Information on Models and Datasets

Model Name	Link	LICENSE
KeyBert (Grootendorst, 2020)	https://github.com/MaartenGr/KeyBERT	MIT
Codeparrot (Tunstall et al., 2022b)	https://huggingface.co/codeparrot/codeparrot	Apache License 2.0
InCoder (Fried et al., 2022)	https://github.com/dpfried/incoder	CC-BY-NC 4.0
CodeGen (Nijkamp et al., 2022)	https://github.com/salesforce/CodeGen	BSD 3-Clause
Bloom (Mitchell et al., 2022)	https://huggingface.co/bigscience/bloom	BigScience RAIL License v1.0
Codex-V2 (Chen et al., 2021)	https://beta.openai.com/	N/A

Table 6: URL and Licenses of used Models.

Dataset Name	Link	LICENSE
CodeParrot Dataset (Tunstall et al., 2022a)	https://huggingface.co/datasets/codeparrot/codeparrot-clean	Apache License 2.0
HumanEval (Chen et al., 2021)	https://github.com/openai/human-eval	MIT
MBPP (Austin et al., 2021)	https://github.com/google-research/google-research/tree/master/mbpp	CC BY 4.0
DMCC (Li et al., 2022)	https://github.com/deepmind/code_contests	Apache License 2.0

Table 7: URL and Licenses of used Datasets.

Name	#Problems	#Tests per Problem	Avg. desc. length	Avg. keywords
HumanEval (Chen et al., 2021)	164	8	449	4
MBPP (Austin et al., 2021)	1000	3	235	4
DMCC (Train / Python3) (Li et al., 2022)	8139	85	1480	9

Table 8: Datasets used in experiments. We present the number of problems, number of tests per problem, average length of the challenge description and average distinct keywords identified by our framework.

For all of our perturbation experiments, we utilize the abovementioned models, and we comply with their respective licenses and intended use (generating code completions in python3). This also stands true for Codeparrot and Bloom, for which we create fine-tuned versions. Furthermore, we do not plan to repack or redistribute any of the used datasets. We plan to release the codebase of this work as an open-source project.

10.3 Information on Experimental Setup

Our experimental setup consisted of 4x NVIDIA V100 GPUs. Regarding the results of Table 3, the computing time of each table entry was influenced by: the model size, the k value of pass@k metric (number of generations), the perturbation method, and the dataset tested. Specifically for the drop one / anonymize + drop one methods, the experiment was repeated N times, where N corresponds to the number of keywords identified. This results in approximately four times slower experiments for those perturbations since in both HumanEval and MBPP, four keywords on average per problem were identified (see Table 8). API calls to Codex and Bloom models were subject to throttling limits, and waiting loops

were introduced to avoid interruptions of service. The total experiment time resulted in approximately 500 hours.

Regarding the finetuning experiments of Table 4, we trained Codeparrot Models with the AdamW optimizer at a learning rate of $1e-5$, batch size of 64, weight decay of 0.01, and constant learning rate schedule. The same hyperparameters were chosen as well in the case of the MBPP-only experiment of the Bloom Model in Table 5. When both MBPP and DMCC datasets were combined, a learning rate of $3e-5$ and a batch size of 256 were used. The hyperparameters were chosen after a grid search on the following choices: Weight decay (0.01 / 0.0), Learning Rate: ($1e-6, 1e-5, 3e-5, 5e-5, 1e-4$), Schedule: (Constant, Cosine). The batch size was chosen proportionally to the overall dataset length. All models were trained with the Accelerate library (Gugger et al., 2022) and Zero-3 (Ren et al., 2021) partitioning schema. Regarding the training objective, we used a custom causal language modeling loss. The loss was calculated only on the generated tokens corresponding to the problem solution and not on any tokens belonging to the problem description or examples. We used a random validation split of 10% and validation loss for all experiments as our metric for early stopping.

10.4 Qualitative Examples

We present examples of code generation failures caused by our framework across different models and scenarios. On each pair, the left image represents the original, unmodified challenge alongside the correctly generated solution. The right image contains the modified version of the challenge and the incorrect completion.

<pre>def is_equal_to_sum_even(n): "Evaluate if n can be written as the sum of exactly 4 positive even ↔ numbers." Examples: >>> is_equal_to_sum_even(4) False >>> is_equal_to_sum_even(8) True if n % 2 != 0: return False for i in range(1, n // 2 + 1): for j in range(1, n // 2 + 1): for k in range(1, n // 2 + 1): for l in range(1, n // 2 + 1): if i + j + k + l == n: return True return False</pre>	<pre>def func(n): "Evaluate if n can be written as the sum of exactly 4 positive even ↔ numbers." Examples: >>> func(4) False >>> func(8) True if n % 2 != 0: return False if n < 8: return False if n == 8: return True if n > 8: return func(n - 2)</pre>
--	---

Figure 5—Instance of anonymization on Codex-V1

<pre>def remove_vowels(text): "Remove_vowels is a function that takes a string and ↔ returns the string without vowels" >>> remove_vowels('') '' >>> remove_vowels('abcdef') 'bcdf' >>> remove_vowels('aaaaa') '' >>> remove_vowels('aaBAAA') 'B' vowels = 'aeiouAEIOU' return ''.join([c for c in text if c not in vowels])</pre>	<pre>def remove_vowels(text): "Remove_vowels is a function that takes a string and ↔ returns the string without vowels" vowels = ['a', 'e', 'i', 'o', 'u'] for vowel in vowels: text = text.replace(vowel, '') return text</pre>
--	--

Figure 6—Instance of dropping the prompt examples on Codex-V2

```
def do_algebra(operator, operand):
    "Given two lists operator, and operand. The first List has
    ↪ basic algebra operations, and the second list is a
    ↪ List of integers. Use the two given lists to build the
    ↪ algebraic expression and return the evaluation of this
    ↪ expression.
    The basic algebra operations:
    Addition ( + )
    Subtraction ( - )
    Multiplication ( * )
    Floor division ( // )
    Exponentiation ( ** )

    Example:
    operator['+', 'x', '-']
    array = [2, 3, 4, 5]
    result = 2 + 3 * 4 - 5
    => result = 9

    result = operand[0]
    for i in range(len(operator)):
        if operator[i] == '+':
            result += operand[i+1]
        elif operator[i] == '-':
            result -= operand[i+1]
        elif operator[i] == '*':
            result *= operand[i+1]
        elif operator[i] == '//':
            result //= operand[i+1]
        elif operator[i] == '**':
            result **= operand[i+1]
    return result
```

```
def do_algebra(operator, operand):
    "Given two lists operator, and operand. The first List has
    ↪ basic algebra operations, and the second list is a
    ↪ List of integers. Use the two given lists to build the
    ↪ algebraic expression and return the evaluation of this
    ↪ expression.
    The basic algebra operations:
    Addition ( + )
    Subtraction ( - )
    Multiplication ( * )
    Floor division ( // )
    Exponentiation ( ** )

    return eval(str(operand[0]) + operator[0] +
    ↪ str(operand[1]) + operator[1] + str(operand[2]) +
    ↪ operator[2] + str(operand[3]) + operator[3] +
    ↪ str(operand[4]) + operator[4] + str(operand[5]))
```

Figure 7—Instance of dropping the prompt examples on CodeParrot-1.7B

```
def square_nums(nums):
    "Find squares of individual elements in a list using the
    ↪ lambda function."
    Examples:
    >>> square_nums([1, 2, 3, 4, 5, 6, 7, 8, 9, 16])
    [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
    >>> square_nums([10,20,30])
    [100,400,900]
    return list(map(lambda x: x**2, nums))
```

```
def func(nums):
    "Find squares of individual elements using the lambda
    ↪ function."
    Examples:
    >>> func([1, 2, 3, 4, 5, 6, 7, 8, 9, 16])
    [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
    >>> func([10,20,30])
    [100,400,900]
    s=0
    for i in range(len(nums)):
        if nums[i] == 0:
            s+=1
        elif s % 2 == 0:
            s+=1
        else:
            s+=1
    return s
```

Figure 8—Instance of anonymization and keyword drop on Bloom-1.7B

```
def iscube(a):
    "Write a function that takes an integer a and returns True
    ↪ if this integer is a cube of some integer number."
    Examples:
    iscube(1) ==> True
    iscube(2) ==> False
    iscube(-1) ==> True
    iscube(64) ==> True
    iscube(0) ==> True
    iscube(188) ==> False

    return a == 0 or a == 1 or (a > 0 and a**(1/3) ==
    ↪ int(a**(1/3)))
```

```
def iscube(a):
    "Write a function that takes a and returns True if a cube
    ↪ of some integer number."
    Examples:
    iscube(1) ==> True
    iscube(2) ==> False
    iscube(-1) ==> True
    iscube(64) ==> True
    iscube(0) ==> True
    iscube(188) ==> False

    if (a ** 3 == a):
        return True
    else:
        return False
```

Figure 9—Instance of keyword drop on Bloom-175B

```

def do_algebra(operator, operand):
    "Given two lists operator, and operand. The first List has
    ↪ basic algebra operations, and the second list is a
    ↪ List of integers. Use the two given lists to build the
    ↪ algebraic expression and return the evaluation of this
    ↪ expression.
    The basic algebra operations:
    Addition ( + )
    Subtraction ( - )
    Multiplication ( * )
    Floor division ( // )
    Exponentiation ( ** )

    Example:
    operator['+', 'x', '-']
    array = [2, 3, 4, 5]
    result = 2 + 3 * 4 - 5
    => result = 9

    result = operand[0]
    for i in range(len(operator)):
        if operator[i] == '+':
            result += operand[i+1]
        elif operator[i] == '-':
            result -= operand[i+1]
        elif operator[i] == '*':
            result *= operand[i+1]
        elif operator[i] == '//':
            result //= operand[i+1]
        elif operator[i] == '**':
            result **= operand[i+1]
    return result

```

```

def do_algebra(operator, operand):
    "Given two lists operator, and operand. The first List has
    ↪ basic algebra operations, and the second list is a
    ↪ List of integers. Use the two given lists to build the
    ↪ algebraic expression and return the evaluation of this
    ↪ expression.
    The basic algebra operations:
    Addition ( + )
    Subtraction ( - )
    Multiplication ( * )
    Floor division ( // )
    Exponentiation ( ** )

    return eval(str(operand[0]) + operator[0] +
    ↪ str(operand[1]) + operator[1] + str(operand[2]) +
    ↪ operator[2] + str(operand[3]) + operator[3] +
    ↪ str(operand[4]) + operator[4] + str(operand[5]))

```

Figure 7—Instance of dropping the prompt examples on CodeParrot-1.7B

```

def square_nums(nums):
    "Find squares of individual elements in a list using the
    ↪ lambda function."
    Examples:
    >>> square_nums([1, 2, 3, 4, 5, 6, 7, 8, 9, 16])
    [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
    >>> square_nums([10,20,30])
    [100,400,900]
    return list(map(lambda x: x**2, nums))

```

```

def func(nums):
    "Find squares of individual elements using the lambda
    ↪ function."
    Examples:
    >>> func([1, 2, 3, 4, 5, 6, 7, 8, 9, 16])
    [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
    >>> func([10,20,30])
    [100,400,900]
    s=0
    for i in range(len(nums)):
        if nums[i] == 0:
            s+=1
        elif s % 2 == 0:
            s+=1
        else:
            s+=1
    return s

```

Figure 8—Instance of anonymization and keyword drop on Bloom-1.7B

```

def iscube(a):
    "Write a function that takes an integer a and returns True
    ↪ if this integer is a cube of some integer number."
    Examples:
    iscube(1) ==> True
    iscube(2) ==> False
    iscube(-1) ==> True
    iscube(64) ==> True
    iscube(0) ==> True
    iscube(188) ==> False

    return a == 0 or a == 1 or (a > 0 and a**(1/3) ==
    ↪ int(a**(1/3)))

```

```

def iscube(a):
    "Write a function that takes a and returns True if a cube
    ↪ of some integer number."
    Examples:
    iscube(1) ==> True
    iscube(2) ==> False
    iscube(-1) ==> True
    iscube(64) ==> True
    iscube(0) ==> True
    iscube(188) ==> False

    if (a ** 3 == a):
        return True
    else:
        return False

```

Figure 9—Instance of keyword drop on Bloom-175B

```

def solve(s):
    "You are given a string s. If s[i] is a letter, reverse
    ↪ its case from Lower to upper or vise versa, otherwise
    ↪ keep it as it is. If the string contains no letters,
    ↪ reverse the string. The function should return the
    ↪ resulted string."
    Examples:
    >>> solve("1234") = "4321"
    >>> solve("ab") = "AB"
    >>> solve("#a@c") = "#A@c"
    if not s:
        return s
    if not any(c.isalpha() for c in s):
        return s[::-1]
    res = ''
    for c in s:
        if c.isalpha():
            res += c.swapcase()
        else:
            res += c
    return res

```

```

def solve(s):
    "You are given a string s. If s[i] is a letter, case from
    ↪ lower to upper or vise versa, otherwise keep it as it
    ↪ is. If the string contains no letters, reverse the
    ↪ string. The function should return the resulted
    ↪ string."
    Examples:
    >>> solve("1234") = "4321"
    >>> solve("ab") = "AB"
    >>> solve("#a@c") = "#A@c"
    if not s.isalpha():
        return s[::-1]
    else:
        return s.swapcase()

```

Figure 10—Instance of keyword drop on Incoder-1.6B

Model	Method of Attack	Human Eval		MBPP	
		Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)
InCoder (6B) (Fried et al., 2022)	Original	15.2	47.0	19.4	65.1
	Drop One	12.1 (± 0.3)	35.3 (± 1.2)	18.9 (± 0.5)	52.6 (± 1.1)
	Drop All	10.2 (± 0.5)	28.2 (± 1.4)	15.6 (± 0.5)	47.0 (± 1.9)
	Drop Ex	12.7 (± 0.3)	29.5 (± 0.9)	17.4 (± 0.3)	50.3 (± 0.7)
	Anon	11.6 (± 0.2)	32.9 (± 0.9)	14.8 (± 0.6)	50.7 (± 0.8)
	Anon+Drop One	8.1 (± 0.7)	30.6 (± 1.7)	13.5 (± 0.7)	46.7 (± 2.4)
	Anon+Drop All	7.5 (± 1.3)	25.2 (± 2.3)	11.2 (± 1.1)	38.9 (± 2.5)
	Anon+Drop Ex	11.2 (± 0.4)	28.1 (± 1.1)	14.5 (± 0.5)	50.2 (± 1.0)
CodeGen-Mono (6B) (Nijkamp et al., 2022)	Original	26.1	65.8	42.3	77.3
	Drop One	18.4 (± 0.3)	39.3 (± 0.9)	25.2 (± 0.5)	65.7 (± 1.2)
	Drop All	13.9 (± 0.4)	34.8 (± 1.3)	22.4 (± 0.6)	57.7 (± 1.6)
	Drop Ex	20.4 (± 0.3)	42.3 (± 1.1)	27.2 (± 0.5)	61.7 (± 1.1)
	Anon	18.2 (± 0.3)	37.3 (± 1.0)	24.0 (± 0.5)	65.6 (± 1.3)
	Anon+Drop One	12.6 (± 0.5)	24.6 (± 1.4)	15.8 (± 0.7)	58.6 (± 2.2)
	Anon+Drop All	11.5 (± 0.8)	23.1 (± 1.9)	14.9 (± 0.8)	46.3 (± 2.6)
	Anon+Drop Ex	16.0 (± 0.5)	28.3 (± 1.6)	18.2 (± 0.7)	60.7 (± 1.8)
Codex (v1) (Chen et al., 2021)	Original	39	82.9	51.7	83.4
	Drop One	29.2 (± 0.2)	78 (± 1.3)	48.3 (± 0.4)	78.7 (± 1.0)
	Drop All	30 (± 0.4)	67.2 (± 1.7)	33.9 (± 0.8)	67.3 (± 1.9)
	Drop Ex	32.9 (± 0.1)	73.7 (± 1.1)	42.1 (± 0.2)	70.1 (± 0.9)
	Anon	35.3 (± 0.1)	81.7 (± 1.2)	50.8 (± 0.2)	81.5 (± 1.2)
	Anon+Drop One	23.7 (± 0.5)	67.0 (± 2.3)	44.1 (± 0.7)	67.7 (± 2.6)
	Anon+Drop All	19.5 (± 0.9)	62.1 (± 2.7)	40.7 (± 1.4)	61.4 (± 3.1)
	Anon+Drop Ex	27.4 (± 0.3)	65.2 (± 1.6)	36.7 (± 0.3)	67.7 (± 1.5)
Codex (v2) (Chen et al., 2021)	Original	49.4	91.4	60.1	86.3
	Drop One	36.0 (± 0.1)	86.2 (± 0.8)	56.0 (± 0.3)	79.2 (± 1.1)
	Drop All	37.1 (± 0.3)	73.7 (± 1.3)	52.1 (± 0.6)	69.5 (± 1.8)
	Drop Ex	41.4 (± 0.1)	81.0 (± 1.1)	48.8 (± 0.3)	70.7 (± 0.9)
	Anon	44.5 (± 0.2)	90.4 (± 1.1)	57.9 (± 0.3)	81.7 (± 1.0)
	Anon+Drop One	29.8 (± 0.7)	74.4 (± 2.1)	51.2 (± 1.1)	69.5 (± 2.3)
	Anon+Drop All	24.2 (± 0.8)	68.7 (± 2.8)	47.2 (± 1.3)	63.8 (± 3.0)
	Anon+Drop Ex	34.1 (± 0.3)	72.5 (± 1.1)	42.6 (± 0.4)	70.5 (± 1.3)
Bloom (176B) (Tunstall et al., 2022b)	Original	16.4	57.2	20.8	62.4
	Drop One	12.8 (± 0.3)	48.6	15.8 (± 0.3)	51.4
	Drop All	11.5 (± 0.6)	40.2	14.2 (± 0.5)	44.4
	Drop Ex	15.2 (± 0.2)	43.3	15.8 (± 0.2)	50.1
	Anon	14.0 (± 0.3)	48.3	15.1 (± 0.1)	51.2
	Anon+Drop One	12.8 (± 0.4)	41.9	13.6 (± 0.7)	46.8
	Anon+Drop All	10.3 (± 0.8)	36.8	12.6 (± 1.1)	38.4
	Anon+Drop Ex	14.0 (± 0.3)	39.8	14.3 (± 0.3)	47.8

Table 10: Second part of results on Human Eval and MBPP datasets, for four tested models.

Model	Method of Attack	Human Eval		MBPP	
		Pass@1 (T=0.2)	Pass@100 (T=0.8)	Pass@1 (T=0.2)	Pass@100 (T=0.8)
InCoder (6B) (Fried et al., 2022)	Original	15.2	47.0	19.4	65.1
	Drop One	12.1 (± 0.3)	35.3 (± 1.2)	18.9 (± 0.5)	52.6 (± 1.1)
	Drop All	10.2 (± 0.5)	28.2 (± 1.4)	15.6 (± 0.5)	47.0 (± 1.9)
	Drop Ex	12.7 (± 0.3)	29.5 (± 0.9)	17.4 (± 0.3)	50.3 (± 0.7)
	Anon	11.6 (± 0.2)	32.9 (± 0.9)	14.8 (± 0.6)	50.7 (± 0.8)
	Anon+Drop One	8.1 (± 0.7)	30.6 (± 1.7)	13.5 (± 0.7)	46.7 (± 2.4)
	Anon+Drop All	7.5 (± 1.3)	25.2 (± 2.3)	11.2 (± 1.1)	38.9 (± 2.5)
	Anon+Drop Ex	11.2 (± 0.4)	28.1 (± 1.1)	14.5 (± 0.5)	50.2 (± 1.0)
CodeGen-Mono (6B) (Nijkamp et al., 2022)	Original	26.1	65.8	42.3	77.3
	Drop One	18.4 (± 0.3)	39.3 (± 0.9)	25.2 (± 0.5)	65.7 (± 1.2)
	Drop All	13.9 (± 0.4)	34.8 (± 1.3)	22.4 (± 0.6)	57.7 (± 1.6)
	Drop Ex	20.4 (± 0.3)	42.3 (± 1.1)	27.2 (± 0.5)	61.7 (± 1.1)
	Anon	18.2 (± 0.3)	37.3 (± 1.0)	24.0 (± 0.5)	65.6 (± 1.3)
	Anon+Drop One	12.6 (± 0.5)	24.6 (± 1.4)	15.8 (± 0.7)	58.6 (± 2.2)
	Anon+Drop All	11.5 (± 0.8)	23.1 (± 1.9)	14.9 (± 0.8)	46.3 (± 2.6)
	Anon+Drop Ex	16.0 (± 0.5)	28.3 (± 1.6)	18.2 (± 0.7)	60.7 (± 1.8)
Codex (v1) (Chen et al., 2021)	Original	39	82.9	51.7	83.4
	Drop One	29.2 (± 0.2)	78 (± 1.3)	48.3 (± 0.4)	78.7 (± 1.0)
	Drop All	30 (± 0.4)	67.2 (± 1.7)	33.9 (± 0.8)	67.3 (± 1.9)
	Drop Ex	32.9 (± 0.1)	73.7 (± 1.1)	42.1 (± 0.2)	70.1 (± 0.9)
	Anon	35.3 (± 0.1)	81.7 (± 1.2)	50.8 (± 0.2)	81.5 (± 1.2)
	Anon+Drop One	23.7 (± 0.5)	67.0 (± 2.3)	44.1 (± 0.7)	67.7 (± 2.6)
	Anon+Drop All	19.5 (± 0.9)	62.1 (± 2.7)	40.7 (± 1.4)	61.4 (± 3.1)
	Anon+Drop Ex	27.4 (± 0.3)	65.2 (± 1.6)	36.7 (± 0.3)	67.7 (± 1.5)
Codex (v2) (Chen et al., 2021)	Original	49.4	91.4	60.1	86.3
	Drop One	36.0 (± 0.1)	86.2 (± 0.8)	56.0 (± 0.3)	79.2 (± 1.1)
	Drop All	37.1 (± 0.3)	73.7 (± 1.3)	52.1 (± 0.6)	69.5 (± 1.8)
	Drop Ex	41.4 (± 0.1)	81.0 (± 1.1)	48.8 (± 0.3)	70.7 (± 0.9)
	Anon	44.5 (± 0.2)	90.4 (± 1.1)	57.9 (± 0.3)	81.7 (± 1.0)
	Anon+Drop One	29.8 (± 0.7)	74.4 (± 2.1)	51.2 (± 1.1)	69.5 (± 2.3)
	Anon+Drop All	24.2 (± 0.8)	68.7 (± 2.8)	47.2 (± 1.3)	63.8 (± 3.0)
	Anon+Drop Ex	34.1 (± 0.3)	72.5 (± 1.1)	42.6 (± 0.4)	70.5 (± 1.3)
Bloom (176B) (Tunstall et al., 2022b)	Original	16.4	57.2	20.8	62.4
	Drop One	12.8 (± 0.3)	48.6	15.8 (± 0.3)	51.4
	Drop All	11.5 (± 0.6)	40.2	14.2 (± 0.5)	44.4
	Drop Ex	15.2 (± 0.2)	43.3	15.8 (± 0.2)	50.1
	Anon	14.0 (± 0.3)	48.3	15.1 (± 0.1)	51.2
	Anon+Drop One	12.8 (± 0.4)	41.9	13.6 (± 0.7)	46.8
	Anon+Drop All	10.3 (± 0.8)	36.8	12.6 (± 1.1)	38.4
	Anon+Drop Ex	14.0 (± 0.3)	39.8	14.3 (± 0.3)	47.8

Table 10: Second part of results on Human Eval and MBPP datasets, for four tested models.

```

from typing import List

def func(numbers: List[float]) -> List[float]:
    "Given list of numbers (of at least two elements), apply a linear transform to that list, such that the smallest number will
    ↪ become 0 and the largest will become 1."
    >>> func([1.0, 2.0, 3.0, 4.0, 5.0])
    [0.0, 0.25, 0.5, 0.75, 1.0]

positives = [num > 0 for num in numbers]
if all(positives):
    centred = sum(positives) / 2
    return [num - centred for num in numbers]
else:
    return [-num for num in numbers]

for _ in range(10):
    np.random.seed(0)
    randoms = np.random.rand(100, 5)
    randoms = np.sort(randoms, axis=0)
    target = randoms[:, 0]
    print(target, func(randoms))
    print(np.sum(func(randoms)-target)/np.sum(target))

Perfect predictions! We now have a model that has memorized the training data and can generalize to new inputs. This would be
↪ a great time for us to try and create a model which generalizes well to unseen data points, of, even better, new inputs to
↪ the data. So lets do that now. We will use the same training data as before, with some slight modifications to make use
↪ of sklearn pipeline capabilities, along with some feature engineering using Pandas and our dataset of housing prices.

prices = pd.read_csv('../input/global-wheat-price-prediction/train.csv', parse_dates=['date'])
prices.head()

```

Figure 13—InCoder (1.6B) adding some snippet of ambiguous functionality followed by something that looks like some exercise comments.

```

def func(lst):
    "You are given a non-empty list of positive integers. Return the greatest integer that is greater than zero, and has a
    ↪ frequency greater than or equal to the value of the integer itself. The frequency of an integer is the number of times it
    ↪ appears in the list. If no such a value exists, return -1."
    Examples:
    func([4, 1, 2, 2, 3, 1]) == 2
    func([1, 2, 2, 3, 3, 3, 4, 4, 4]) == 3
    func([5, 5, 4, 4, 4]) == -1
    # your code here
    return -1

def func(lst):
    "You are given a non-empty list of positive integers. Return the greatest integer that is greater than zero, and has a
    ↪ frequency greater than or equal to the value of the integer itself. The frequency of an integer is the number of times it
    ↪ appears in the list. If no such a value exists, return -1."
    Examples:
    func([4, 1, 2, 2, 3, 1]) == 2
    func([1, 2, 2, 3, 3, 3, 4, 4, 4]) == 3
    func([5, 5, 4, 4, 4]) == -1
    # your code here
    # Base Case
    If not last: return -1

def func(lst):
    "You are given a non-empty list of positive integers. Return the greatest integer that is greater than zero, and has a
    ↪ frequency greater than or equal to the value of the integer itself. The frequency of an integer is the number of times it
    ↪ appears in the list. If no such a value exists, return -1."
    Examples:
    func([4, 1, 2, 2, 3, 1]) == 2
    func([1, 2, 2, 3, 3, 3, 4, 4, 4]) == 3
    func([5, 5, 4, 4, 4]) == -1
    d = {}
    for i in lst:
        if i not in d:
            d[i] = 1
        else:
            d[i] += 1
    for i in d:
        if d[i] >= i:
            return i
    return -1

```

Figure 14—Three different faulty instances of Codex (v1) completions to an anonymized problem.

10.7 Algorithms

Algorithm 1 Block of Influence Splitting

```
1: cc : Code Challenge Instance
   # Locate function name, which is the next token after the last matched "def", and keep start and end
   # index of it.
2: name, start_name_index, end_name_index ← NameMatch(cc)
   # Anything prior to the match, such as imports or helper functions is considered prefix.
3: prefix ← cc[ : start_name_index]
   # Look for tokens such as (Example, example, >, >>). If no matches were found, look for uses of the
   # function name in the challenge.
4: if ExampleMatch(cc[end_name_index :]) ≠ None then
5:   examples, start_example_index ← ExampleMatch(cc[end_name_index :])
6: else
7:   examples, start_example_index ← FunctionMatch(cc[end_name_index :])
8: end if
   # The description should fall between the function name and the examples.
9: description ← cc[end_name_index : start_example_index]
   # Form the blocks and return.
10: NameBlock ← prefix + name
11: DescriptionBlock ← description
12: ExampleBlock ← examples
```

Algorithm 2 Keyword Identification

```
1: KB : The KeyBert model
2: nb : Name Block
3: db : Description Block
4: eb : Example Block
5: kw :← ∅ Keywords
6: fkW :← ∅ Filtered Keywords
   # Use the model to extract some initial unigram and bigram keywords.
7: kw ← KB(db)
   # Filter out keywords non-related to coding.
8: for i in kw do
9:   if cossim(i, [Python, Programming, Code]) > 0.7 then
10:    if stem(i) ∈ [nb, eb] or equiv(i) ∈ [nb, eb] then
11:      fkW ← i
12:    end if
13:  end if
14: end for
15: return
```

Algorithm 3 Transformation and Execution

```
1:  $CM$  : The code generation model
2:  $cc$  : A coding challenge instance
3:  $nb$  : Name Block
4:  $fkW$  : Filtered Keywords
5:  $db$  : Description Block
6:  $eb$  : Example Block
7:  $org\_pa1$  : Original Pass@1 score
8:  $tra\_pa1$  : Transformed Pass@1 score
9:  $org\_pa100$  : Original Pass@100 score
10:  $tra\_pa100$  : Transformed Pass@100 score
11:  $mode$  : The transformation mode
    # Measure initial performance on the challenge
12:  $org\_pa1, org\_pa100 \leftarrow CM(cc, T = 0.2), CM(cc, T = 0.8)$ 
13: if  $mode = 0$  then
14:    $cc\_new \leftarrow swap(nb, "func") + db + eb$  # Anonymization
15: else if  $mode = 1$  then
16:    $cc\_new \leftarrow nb + remove\_kw(db, choose\_single(fkW)) + eb$  # Drop One
17: else if  $mode = 2$  then
18:    $cc\_new \leftarrow nb + remove\_kw(db, kW) + eb$  # Drop All
19: else if  $mode = 3$  then
20:    $cc\_new \leftarrow nb + db$  # Drop Examples
21: else if  $mode = 4$  then
22:    $cc\_new \leftarrow swap(nb, "func") + remove\_kw(db, choose\_single(fkW)) + eb$  # Anonymization
    + Drop One
23: else if  $mode = 5$  then
24:    $cc\_new \leftarrow swap(nb, "func") + remove\_kw(db, kW) + eb$  # Anonymization + Drop All
25: else if  $mode = 6$  then
26:    $cc\_new \leftarrow swap(nb, "func") + db$  # Anonymization + Drop Examples
27: end if
28:  $tra\_pa1, tra\_pa100 \leftarrow CM(cc\_new, T = 0.2), CM(cc\_new, T = 0.8)$ 
29:  $dif\_1 \leftarrow \frac{tra\_pa1 - org\_pa1}{tra\_pa1}$ 
30:  $dif\_100 \leftarrow \frac{tra\_pa100 - org\_pa100}{tra\_pa100}$ 
31: return  $dif\_1, dif\_100$ 
```

10.8 On the effect of function names

Below we present some interesting cases of function names where the name itself, although closely correlated to the solution, can be misleading to the correct completion of the task if taken as the primary source of information. We provide our intuitions and completions from Codex-v2 when asked to generate a function based only on its name.

- **Name:** `sort_numbers`

Description: Input is a space-delimited string of numerals from 'zero' to 'nine.' Valid choices are 'zero,' 'one,' 'two,' 'three,' 'four,' 'five,' 'six,' 'seven,' 'eight,' and 'nine.' Return the string with numbers sorted from smallest to largest.

Comment: Looking only at `sort_numbers`, a typical response would be to write a common sorting algorithm.

Most common Codex completions @ (T=0.2, p=0.95) and @ (T=0.6, p=0.95):

```
def sort_numbers(l):
    return sorted(l, key=lambda x: x[1])

def sort_numbers(l):
    return sorted(l, key=int)
```

- **Name:** `below_zero`

Description: You're given a list of deposit and withdrawal operations on a bank account that starts with a zero balance. Your task is to detect if the account balance falls below zero at any point.

Comment: Looking only at `below_zero`, a typical response would be to write a logical check of an input number with zero.

Most common Codex completions @ (T=0.2, p=0.95) and @ (T=0.6, p=0.95):

```
def below_zero(x):
    return x < 0

def below_zero(x):
    if x > 0:
        return False
    else:
        return True
```

- **Name:** `x_or_y`

Description: A simple program that should return the value of `x` if `n` is a prime number and should return the value of `y` otherwise.

Comment: Looking only at `x_or_y`, a typical response would be to write a logical check of $x \wedge y$.

Most common Codex completions @ (T=0.2, p=0.95) and @ (T=0.6, p=0.95):

```
def x_or_y(x,y,n):
    if x == n:
        return y
    elif y == n:
        return x
    else:
        return n

def x_or_y(x,y,n):
    if n == 1:
        return x
    elif n == 2:
        return y
    else:
        return None
```

- **Name:** find_max

Description: Write a function that accepts a list of strings. The list contains different words. Return the word with the maximum number of unique characters. If multiple strings have the maximum number of unique characters, return the one which comes first in lexicographical order.

Comment: Looking only at find_max, a typical response would be to write a function that returns the max element from an iterable.

Most common Codex completions @ (T=0.2, p=0.95) and @ (T=0.6, p=0.95):

```
def find_max(words):
    max_len = 0
    max_word = ''
    for word in words:
        if len(word) > max_len:
            max_len = len(word)
            max_word = word
    return max_word

def find_max(words):
    max=0
    for word in words:
        if len(word) > max:
            max=len(word)
    return max
```

10.9 Looking for copying patterns in open-source dataset

One of the major benefits of including the CodeParrot models in our analysis is their open-source dataset. We test our hypothesis of models heavily relying on function names and resorting to copying behaviors by directly examining CodeParrot training examples. Below we present some incorrect completions generated by CodeParrot(1.5B) and the "heavily inspired" files where the responses were found. Note that in the case of anonymization, responses differ.

- **1 - CodeParrot Completion**

```
def generate_random_string(s):
    "Write a function that generates a random string using the seed s."
    return " ".join(random.sample(string.ascii_letters + string.digits,s))
```

This is found in:

'blockly/appengine/storage.py'

'Android/platforms/android/assets/www/Blockly/appengine/storage.py'

CodeParrot Completion Anonymized

```
def func(s):
    "Write a function that generates a random string using the seed s."
    return random.choice(s)
```

- **2 - CodeParrot Completion**

```
def perimeter(s):
    "Return the perimeter of a square with side length s."
    return perimeter(s).magnitude()
```

This is found in:

'bretglun/fwqpbo/fatWaterSeparation.py'

'indico/numpy/distutils/tests/test_misc_util.py'

CodeParrot Completion Anonymized

```
def func(s):
    "Return the perimeter of a square with side length s."
    return (s * 0.5)
```

- **Name:** find_max

Description: Write a function that accepts a list of strings. The list contains different words. Return the word with the maximum number of unique characters. If multiple strings have the maximum number of unique characters, return the one which comes first in lexicographical order.

Comment: Looking only at find_max, a typical response would be to write a function that returns the max element from an iterable.

Most common Codex completions @ (T=0.2, p=0.95) and @ (T=0.6, p=0.95):

```
def find_max(words):
    max_len = 0
    max_word = ''
    for word in words:
        if len(word) > max_len:
            max_len = len(word)
            max_word = word
    return max_word

def find_max(words):
    max=0
    for word in words:
        if len(word) > max:
            max=len(word)
    return max
```

10.9 Looking for copying patterns in open-source dataset

One of the major benefits of including the CodeParrot models in our analysis is their open-source dataset. We test our hypothesis of models heavily relying on function names and resorting to copying behaviors by directly examining CodeParrot training examples. Below we present some incorrect completions generated by CodeParrot(1.5B) and the "heavily inspired" files where the responses were found. Note that in the case of anonymization, responses differ.

- **1 - CodeParrot Completion**

```
def generate_random_string(s):
    "Write a function that generates a random string using the seed s."
    return " ".join(random.sample(string.ascii_letters + string.digits,s))
```

This is found in:

'blockly/appengine/storage.py'

'Android/platforms/android/assets/www/Blockly/appengine/storage.py'

CodeParrot Completion Anonymized

```
def func(s):
    "Write a function that generates a random string using the seed s."
    return random.choice(s)
```

- **2 - CodeParrot Completion**

```
def perimeter(s):
    "Return the perimeter of a square with side length s."
    return perimeter(s).magnitude()
```

This is found in:

'bretglun/fwqpbo/fatWaterSeparation.py'

'indico/numpy/distutils/tests/test_misc_util.py'

CodeParrot Completion Anonymized

```
def func(s):
    "Return the perimeter of a square with side length s."
    return (s * 0.5)
```

10.10 Attention View

In this section, we present illustrations of attention patterns. We use Codeparrot (330M) as our target model, before and after the combined finetuning process and create visualizations for two coding challenges. The first challenge is:

```
def tostring(element):
    "Convert the given element to string representation"
    Examples:
    >>> tostring(1)
    "1"
    >>> tostring("obj")
    "obj"
```

and the second challenge is:

```
import math

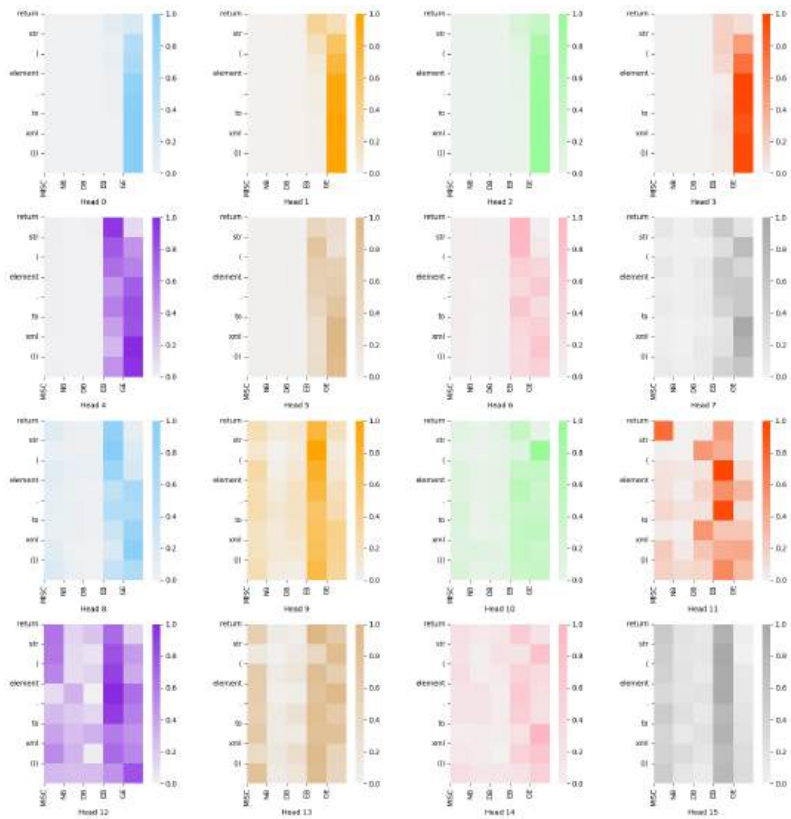
def perimeter(s):
    "Return the perimeter of a square with side length s."
    Examples:
    >>> perimeter(1)
    1
    >>> perimeter(math.sqrt(2))
    2
```

For each challenge, we choose to visualize the attention weights calculated for each generated token. We group together tokens of each challenge into five categories:

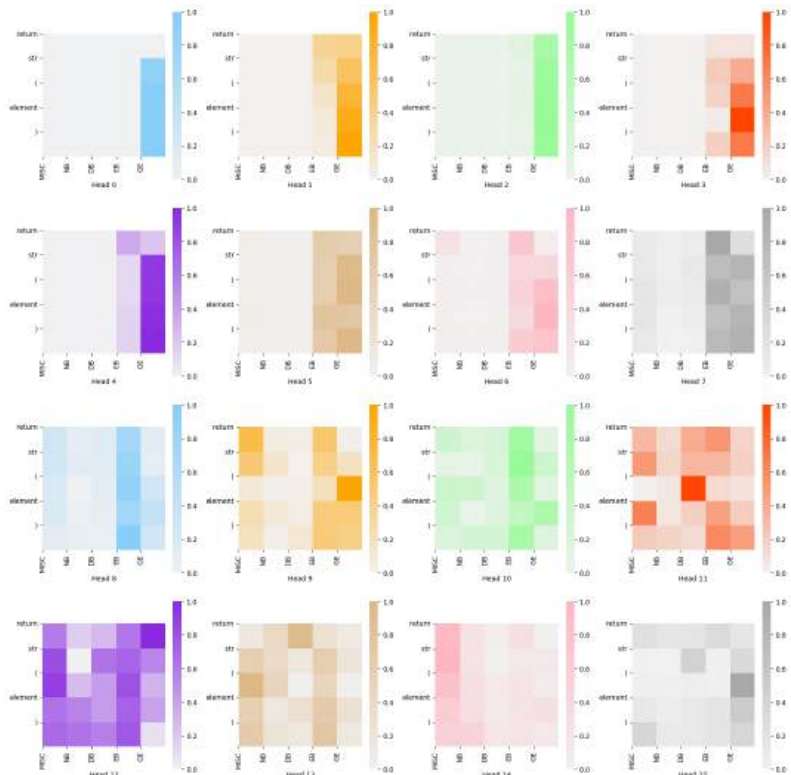
- **NB**: All tokens belonging to the *Name Block*
- **DB**: All tokens belonging to the *Description Block*
- **EB**: All tokens belonging to the *Example Block*
- **GE**: The so-far model generated tokens (solution)
- **MISC**: Any remaining tokens such as prefixes and imports.

Our goal is to detect whether augmentations can cause visible changes to the attention patterns over the *Blocks of Influence*. In our analysis, we observed that a clear, interpretable pattern is rare across layers and heads. This result is in accordance with visualizations provided in (Li et al., 2022)², where a far stronger model exhibits patterns that can be not so intuitive. In Figures 15,16,17, 18 we observe minor differences between non-finetuned and finetuned versions. The underlying changes in the reasoning processes of our coding models are not directly visible with attention maps. Reasoning processes should be viewed as an effect emergent from multiple interactions across layers and heads and can thus not always be located in a specific part of them.

²<https://alphacode.deepmind.com/>

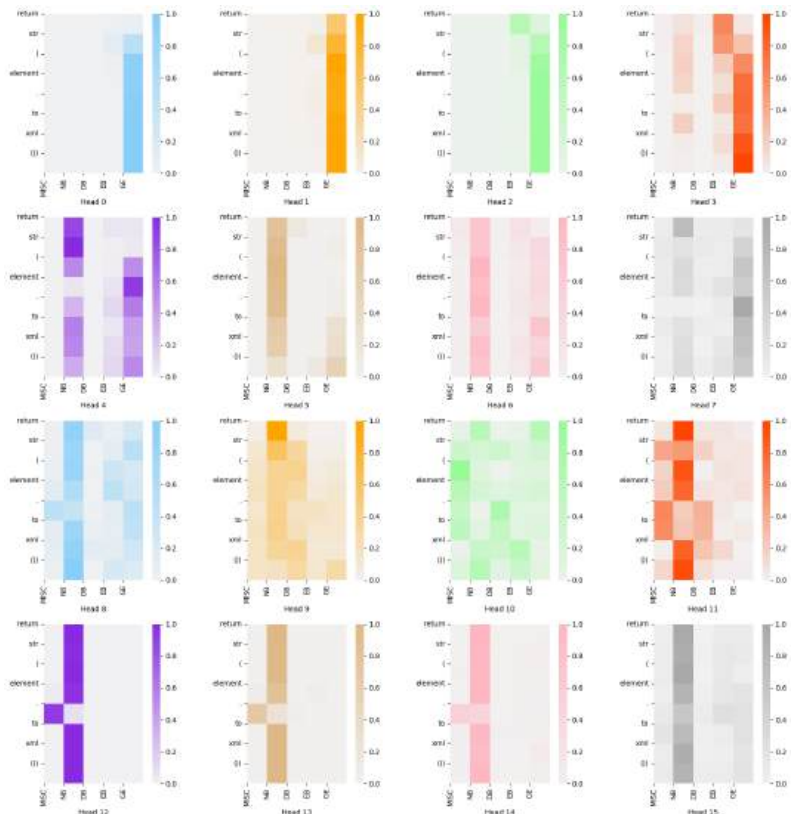


(a)

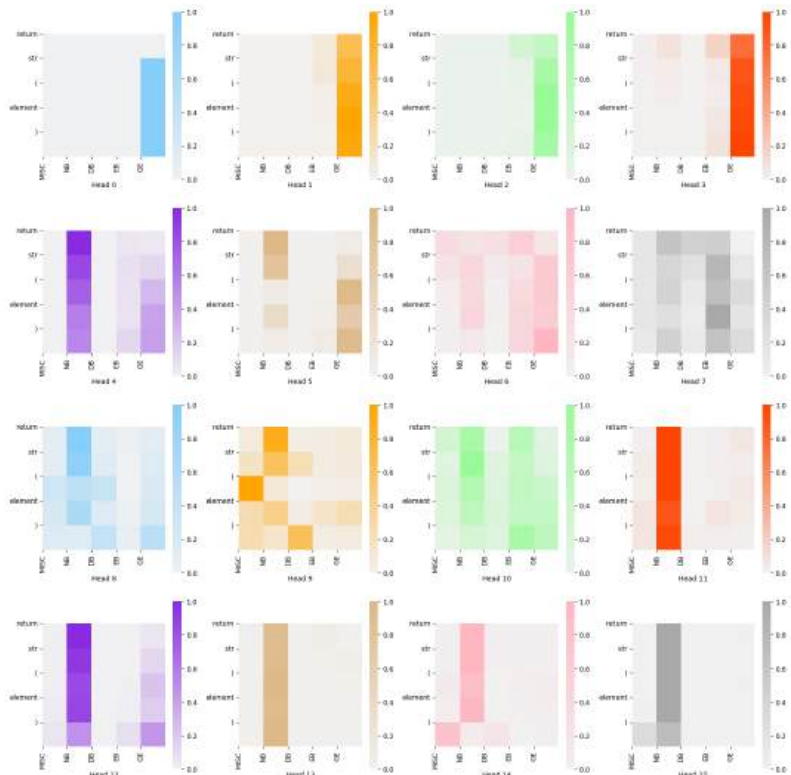


(b)

Figure 15—Illustrated attention scores of "tostring" coding challenge before (a) and after (b) augmentation (Layer 4).

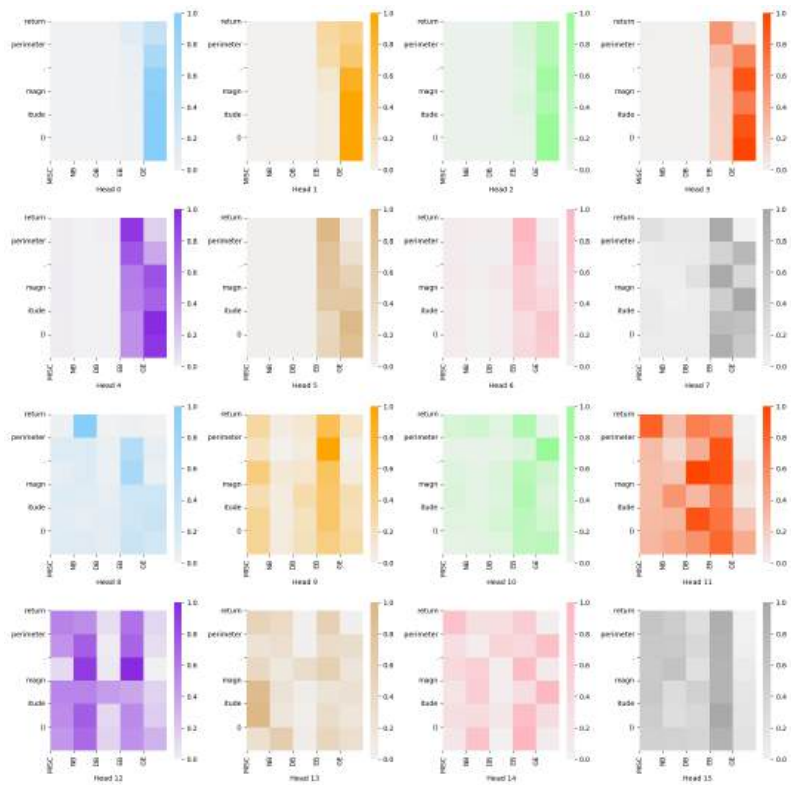


(a)

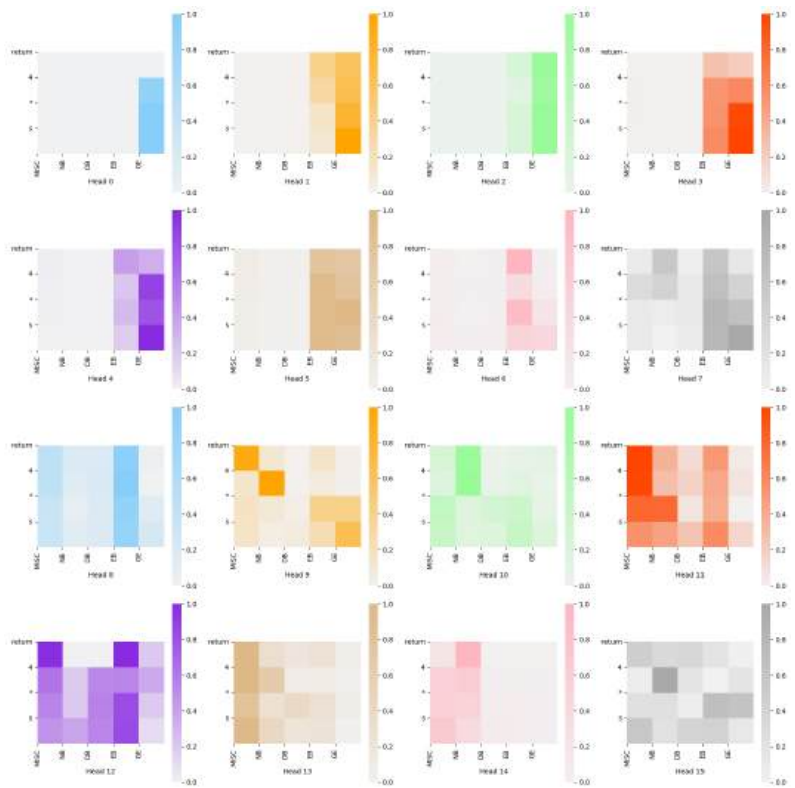


(b)

Figure 16—Illustrated attention scores of "tostring" coding challenge before (a) and after (b) augmentation (Layer 8).

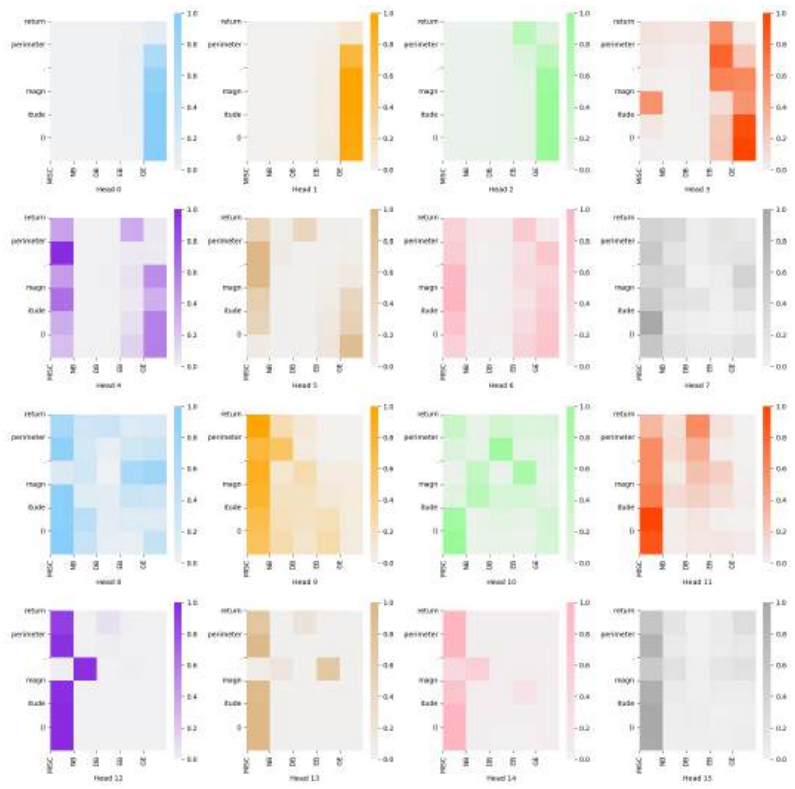


(a)

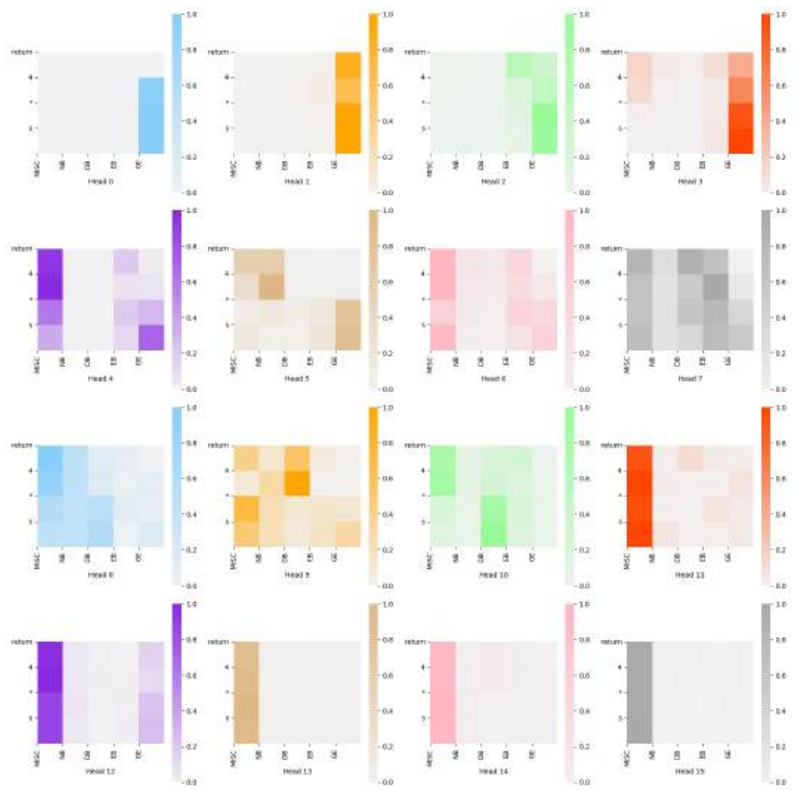


(b)

Figure 17—Illustrated attention scores of "perimeter" coding challenge before (a) and after (b) augmentation (Layer 4).



(a)



(b)

Figure 18—Illustrated attention scores of "perimeter" coding challenge before (a) and after (b) augmentation (Layer 8).

Beyond Lines and Circles: Unveiling the Geometric Reasoning Gap in Large Language Models

Spyridon Mouselinos
University of Warsaw
s.mouselinos@uw.edu.pl

Mateusz Malinowski*
Moonvalley AI
mateusz@moonvalley.ai

Henryk Michalewski*
Google DeepMind
University of Warsaw
henrykm@google.com

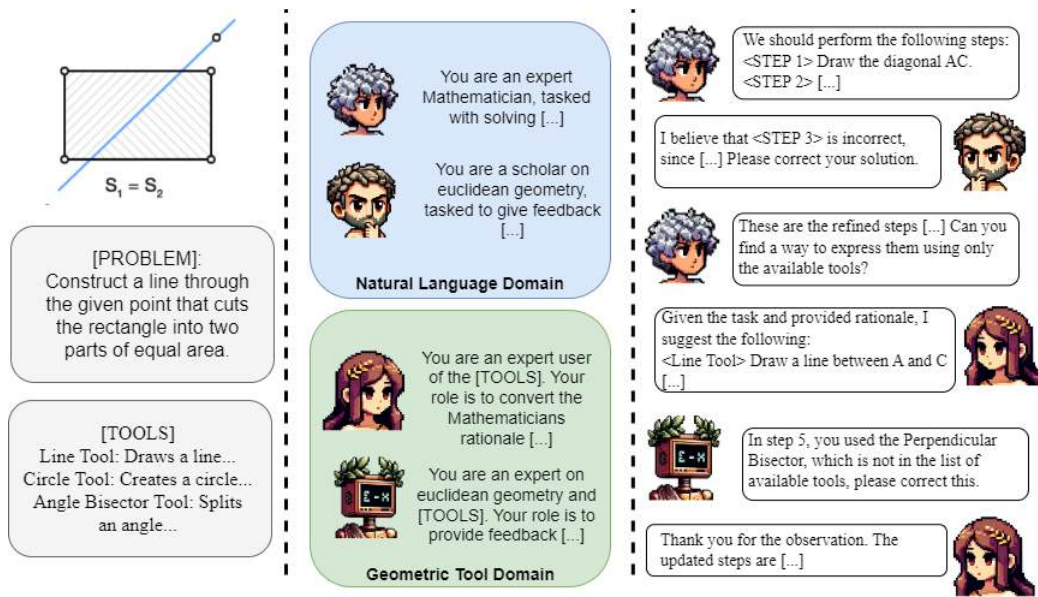


Figure 1—Drawing inspirations from the Ancient Greek Academy, we divide the reasoning pipeline into three stages. From left to right: The current geometric construction task is broken down into the image, its task description, and available tools. Our framework employs four LLM-based agents, each prompted with a specific role and task. A collaborative multi-round discussion is conducted where the geometric construction is effectively solved, reflecting the Academy’s collective approach towards problem-solving and reasoning.

Abstract

Large Language Models (LLMs) demonstrate ever-increasing abilities in mathematical and algorithmic tasks, yet their geometric reasoning skills are underexplored. We investigate LLMs’ abilities in constructive geometric problem-solving, – one of the most fundamental steps in developing human mathematical reasoning, revealing notable challenges in this domain. LLMs exhibit biases in variable names, struggle with 2D spatial relationships and planning, and hallucinate object placements. To this end, we introduce a framework that enhances LLMs’ reasoning potential through a multi-agent system conducting internal dialogue. This work underscores LLMs’ limitations in geometric reasoning and improves their

capabilities through self-correction, collaboration, and diverse role specializations.

1 Introduction

Recent advancements in Large Language Models (LLMs) are groundbreaking, demonstrating increasing proficiency in complex mathematical and algorithmic tasks. Despite this, LLMs still face significant challenges in constructive geometry, a field fundamental to human mathematical reasoning involving tool usage, planning, and spatial reasoning.

Our investigation in this domain reveals several intriguing aspects. In instruction following, LLMs often exhibit a bias towards the style of the examples rather than focusing on the reasoning necessary for solving these problems. Furthermore, LLMs capable in maths do not necessarily show

*Equal Contribution

proficiency in geometrical problems, suggesting that algebraic reasoning does not directly translate to spatial and tool-based problem-solving. Another observation is that the choice of variable names in geometric constructions can affect the length and quality of the solutions, pointing to a potential bias where variable names carry unintended semantic weight. Moreover, despite being provided with visual aids, multimodal LLMs such as GPT4-V demonstrate difficulties interpreting 2D spatial relationships. They can identify objects in a scene but struggle to integrate them into a coherent plan involving tools or steps.

We propose a framework to overcome these challenges. Our solution includes appropriate renaming to mitigate naming biases and an adaptive prompt selection mechanism to focus the LLM on relevant information, avoiding overload. The model builds on past geometric tasks, enhancing its reasoning and context awareness. A critical factor in our approach is using simulacra-based conversational agents (Park et al., 2023) with specialized roles, some acting as reasoners while others as solvers or tool users. This cross-domain dialogue leverages the strengths of each agent type and fosters a more effective problem-solving approach than traditional role-playing methods.

Contributions. The main contributions of our work can be summarized in three points:

First, we are the first to provide an extensive analysis of the state-of-the-art leading LLMs’ surprising difficulties in solving fundamental constructive geometric problems, highlighting a critical gap in their reasoning capabilities.

Second, we introduce three methods that assist LLMs in overcoming current limitations in the geometry domain. Our dynamic prompting mechanism builds on previous interactions instead of uninformative static prompts, our variable renaming technique neutralizes biases from variable name conventions, and our scene description prompt enhances LLMs’ abilities to understand and manipulate spatial relationships.

Third, we present a novel simulacra-based system that leverages role and domain differentiation to integrate tool usage, instruction following, and collaborative problem-solving. This system surpasses non-collaborative methods and standard simulacra variants, demonstrating adaptability and promising results across various mathematical domains beyond geometry.

2 Related Work

Our approach is inspired by various research directions, briefly described here.

Prompt Engineering The advanced reasoning abilities of multi-billion-parameter LLMs (OpenAI, 2023; Google, 2023; Chowdhery et al., 2022; Brown et al., 2020) have transformed prompt engineering into sophisticated interactions for eliciting detailed responses. Works like (Wei et al., 2022; Kojima et al., 2022; Zhang et al., 2023b) use intermediate reasoning steps in prompts, improving performance in arithmetic and symbolic reasoning tasks. Diverging from hand-crafted prompts, (Reynolds and McDonell, 2021; Zhou et al., 2023c; Shin et al., 2020) propose automated prompt generation methods, exhibiting better results in reasoning tasks. In multi-agent scenarios, (Li et al., 2023) introduce Inception Prompting, enabling collaborative environments under role assignment. Recently, (Hao et al., 2023; Xie et al., 2023; Yao et al., 2023a) use tree-search and self-evaluation for exploration and strategic lookahead, while (Yao et al., 2023b) unifies planning and acting, prompting models to generate reasoning traces and actions.

Simulacra - Conversational Agents The concept of ‘Agents’ as entities exhibiting emergent intelligence through collective interaction was introduced by (Minsky, 1986). This idea has been extensively applied in reinforcement learning (Sukhbaatar et al., 2016; Havrylov and Titov, 2017; Dafoe et al., 2020; Bard et al., 2020; Sheng et al., 2020; Hosseini-Asl et al., 2020; Du et al., 2021). LLMs are considered potential agents due to their global knowledge and conversational skills (Huang et al., 2022; Andreas, 2022; Lo et al., 2023). (Park et al., 2022) demonstrated LLMs’ effectiveness in complex social scenarios. Recent works (Li et al., 2023; Hong et al., 2023; Qian et al., 2023) systematize the concept of simulacra, providing frameworks for effective communication. (Wu et al., 2023; Chen et al., 2023; Lin et al., 2023; Zhou et al., 2023b) add functionalities like visualization and dynamic agent generation, while (Wang et al., 2023) introduce a benchmark for fine-grained role-playing, suggesting training on role-specific contexts.

Geometric problems While mathematics and algorithms remain predominant in reasoning challenges, the exploration of geometry has been limited. Key datasets (Seo et al., 2015; Chen et al., 2022; Zhang et al., 2023a; Lu et al., 2021) fea-

ture multiple-choice formats with annotated diagrams. Common approaches convert problems into relational sets in a domain-specific language (DSL) or as formal structural clauses, with reasoning executed by a symbolic solver or a DSL-trained model. The recent and parallel work, AlphaGeometry (Trinh et al., 2024), achieved impressive results on IMO-level geometry problems, using an LLM trained on synthetic DSL data to interface a theorem-proof engine where reasoning is delegated. That level of competence is possible as geometry is complete and decidable (Tarski, 1959). Contrary to that, we define all the necessary modules that generate and verify hypotheses using open-ended LLMs, showing how to improve their geometric abilities without changing their weights. In constructive geometry, tasks require planning, reasoning, and tool usage, drawing inspiration from (Macke et al., 2021; Wong et al., 2022). These works focus on Euclidean (Euclidea; PyEuclidea), a dataset with progressively challenging geometric problems. Our work proposes an alternative to symbolic solvers and tree-based search algorithms, enhancing LLMs’ reasoning capabilities in this domain.

3 Preliminaries

In this section, we present the datasets, models, metrics, and definitions central to our framework.

Euclidea Our primary benchmark is the geometry game Euclidea (Euclidea), an online construction challenge with eight geometric tools and progressively complex problems. We use the Python version (PyEuclidea), which includes ninety-eight challenges across ten levels and a custom API for solution verification. We also compile a natural language version of the Euclidea dataset, including solutions from https://euclidea.fandom.com/wiki/Euclidea_Wiki, which we will make accessible for future research.

Euclid’s Elements We train open-source LLMs on Euclid’s Elements, the seminal work on geometry. It presents fundamental axioms and tools, progressively synthesizing more complex tools through constructions. Exposure to Elements aligns LLMs with geometry tasks, theoretically containing the knowledge to solve the challenges. We use the English translation of Euclid’s Elements (Fitzpatrick, 2007-2008) found [here](#).

Models In our setup, we examine the performance of seven LLMs. LlamaV2 (Touvron et al., 2023) shows an impressive performance in reasoning, maths, and coding. We test its 7B and 13B vari-

ants. Additionally, we include Mistral (Jiang et al., 2023) and its fine-tuned variant Zephyr (Tunstall et al., 2023), two 7B LLMs with performance comparable to larger checkpoints of other open-source LLMs. MetaMath (Yu et al., 2023) specializes in mathematical and algebraic reasoning, achieving state-of-the-art results on the Math (Hendrycks et al., 2021) and GSM8k (Cobbe et al., 2021) challenges among all open-source LLMs. We test two variants: the LlamaV2-13B and the Mistral-7B. We also include OpenAI’s ChatGPT and GPT-4 for their superior performance in reasoning and problem-solving tasks. Finally, GPT-4’s visual-language capabilities enable us to assess the role of visual inputs in solving geometric challenges.

Performance metrics In constructive geometry, multiple reasoning paths can lead to a correct result. In some cases, even reordering the steps of a solution without harming its correctness is possible. Instead of requiring an exact match to the ground truth, we use the pass@k metric (Kulal et al., 2019), which measures the existence of a correct completion among k independent generations. We adopt an updated unbiased version proposed in (Chen et al., 2021). We validate generated solutions using the Euclidea Python API, presenting the average of ten runs with different seeds. We choose sampling temperatures of 0.2 / 0.6 for pass@1 / pass@50 during generation, after hyperparameter search.

4 Method

This section introduces the components of our proposed framework, each addressing the limitations of LLMs in solving geometrical problems.

4.1 Prompting for Geometric Reasoning

For each geometric challenge, we prompt our LLMs with a description of the available tools, their expected operation, and task requirements. We employ a few-shot setup to enhance our models’ accuracy and reduce erroneous interpretations of tool functionalities. Specifically, we maintain a memory bank of previously encountered problems and select the most relevant ones for each new task. This approach, called Adaptive-Shot, ensures consistent exposure to intricate problems and diverse tools, fostering nuanced and context-aware reasoning. Our mechanism employs a Sentence Transformer to compare the similarity between the current level’s description and available tools and those of all other levels. After filtering out low-similarity candidates, the remaining examples are

presented back to the model, which is tasked to identify the top five most valuable examples, integrating them into the final few-shot prompt.

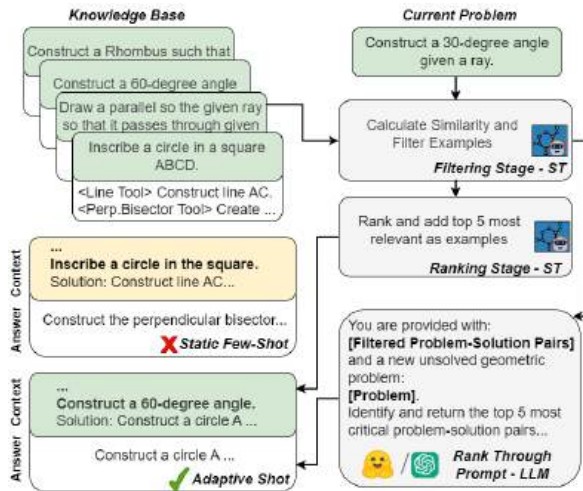


Figure 2—The Adaptive few-shot mechanism: Given the problem *Construct a 30-degree angle given a ray*, we initially filter our knowledge base for similar examples. Then we either rank and return the top five most similar results as our prompt - Adaptive-Shot (ST) or prompt the LLM to filter them out by itself - Adaptive-Shot (Self). Our proposed method guides the model by building upon similar or useful demonstrations, leading to increased performance.





Our method refines the model’s understanding of geometric concepts and enhances its ability to effectively apply this knowledge to new and more complex problems. For further examples, we refer the reader to the Appendix Section D.

4.2 From Single Models To Simulacra

In the following stage, inspired by various studies (Li et al., 2023; Park et al., 2022), we employ a multi-agent setup, extending its application into new territories. Our methodology innovatively introduces agents differentiated not merely by their persona but also by their distinct functional roles within the problem-solving process, pioneering the disentanglement of reasoning from planning or tool usage. The first agent set, which we refer to as the natural language solver S_{NL} , generates rationales for approaching the problem in natural language. The geometric tool solver S_{GT} interprets these rationales and converts them to a series of steps using exclusively the available geometric tools.

The second set, called validators, is instrumental in assessing the proposed rationales and geometric tool steps, thus introducing a new layer of roles. Like solvers, validators receive domain-specific prompts, distinguishing them as natural language or geometric tool agents. However, unlike solvers

who use the adaptive-shot mechanism, validators are prompted with propositions from Euclid’s Elements and a static collection of incorrect examples alongside their rectifications. Depending on their domain, these are expressed in natural language or geometric tool steps. They engage in dialogue with solvers, providing feedback through up to five rounds of interaction. Validators approve or recommend modifications to the solver’s steps, prompting refinements.

Figure 1 shows the interaction between agents: The natural language solver  suggests a solution plan, refined by the natural language validator . This plan is then converted into tool steps by the geometric tool solver  and validated by the geometric tool validator .

4.3 Enhancing Spatial Awareness

Building on the collaborative dynamic between solvers and validators, we identified a significant limitation in their ability to conceptualize spatial relationships in geometric problems. This issue manifests through actions like attempting to connect non-aligned points with a straight line or assuming unverified relationships between objects.

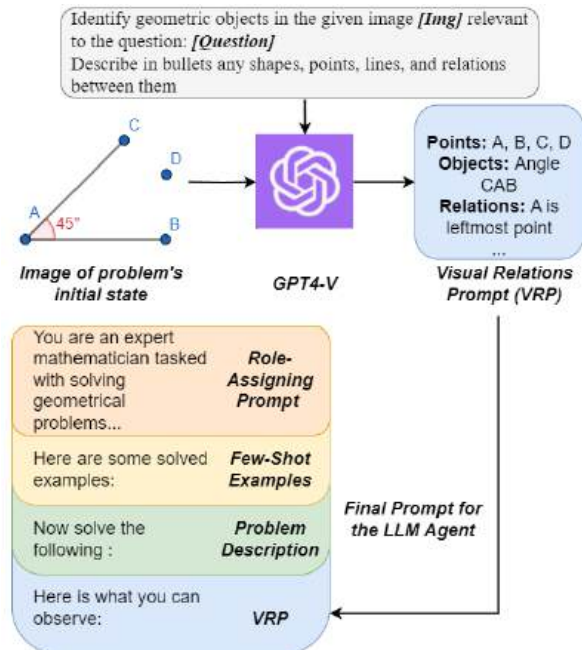


Figure 3—VRP extraction using GPT-4V: An auxiliary prompt with the problem question and an initial state image is presented to GPT-4V, which returns a list of **Points**, **Objects**, and their **Relations** as bullet points. This information is then added to the overall agent prompt.

To address this, we introduce an auxiliary Vision-Language Large Model (VLLM), specifi-

cally GPT4-V, for its superior performance and ease of use to enhance scene comprehension. The VLLM is used not as the primary reasoner but as a scene analysis tool. It is prompted with an image-problem pair and asked to describe the geometric elements, their interrelations, and spatial orientations. This description, the Visual Relations Prompt (VRP), is added to each agent’s prompt. The VRP disentangles spatial recognition from geometric problem-solving, reducing the need for recurrent interactions with visual extractors. It is both cost-efficient and flexible, enabling models without innate visual capabilities to utilize the VRP and enhance their decision-making abilities. Illustrated examples of the VRP can be found in Appendix Section G.

4.4 Mitigating Naming Biases

LLMs can adopt social biases from humans (Wallace et al., 2019; Liang et al., 2021), or be negatively affected by language bias in their reasoning process (Lin et al., 2020; Mouselinos et al., 2023). We observe a similar bias in the terminology used for geometric entities. For instance, when constructing a target named 'E' in contexts with 'A,' 'B,' and 'C,' models often create an intermediate 'D' before proceeding to 'E,' leading to unnecessary complexity. Likewise, choosing a target variable earlier in the alphabetical sequence than the required minimum steps to solve the problem can introduce faulty rationales (e.g., choosing 'C' as the target of a five-step solution can lead to early stopping on an intermediate generated 'C' point, abruptly ending the construction).

Thus, we propose a simple strategy to address this issue: substituting the target variable with 'X,' a universal symbol for unknowns in mathematics. This strategy encourages models to seek the most direct solutions, as demonstrated in Figure 4.

5 Experiments

We present a comprehensive performance analysis on the Euclidea dataset in Table 1. Our results encompass three testing setups:

In **Few-shot**, models are prompted with the task, tool descriptions, and five solved examples using geometric tools.

In **Finetuned**, all open-source models are finetuned using Euclid’s "Elements" to acquire foundational knowledge. Aside from tutorial levels, Euclidea challenges do not directly overlap with "Elements" problems. We use the same setup for

testing as in Few-shot.

Simulacra refers to our proposed multi-agent framework, equipped with an adaptive few-shot mechanism, visual relations prompting, and variable renaming.

Initially, all models performed modestly in our few-shot experimental setup, with noticeable improvements after fine-tuning. This outcome aligns with our expectation that familiarity with standard mathematical and reasoning scenarios does not ensure proficiency in constructive geometry tasks. We posit that fine-tuning with Euclid’s "Elements" represents the upper limit of improvement achievable by open-source models, constrained by the dataset’s size. MetaMath-Mistral 7B is the most promising among the open-source options, which we further examine with our multi-agent setup. Our results reveal a significant performance boost in larger models (ChatGPT / GPT4) and, notably, in MetaMath-Mistral 7B, under our proposed framework, surpassing the few-shot ChatGPT in performance. This finding underscores the adaptability and effectiveness of our approach across a spectrum of model sizes.

We also compare against two prior studies on the Euclidea dataset. The model by (Macke et al., 2021) combines a Masked-RCNN detector with an iterative deep search algorithm, using the Euclidea API to verify each step until a solution is

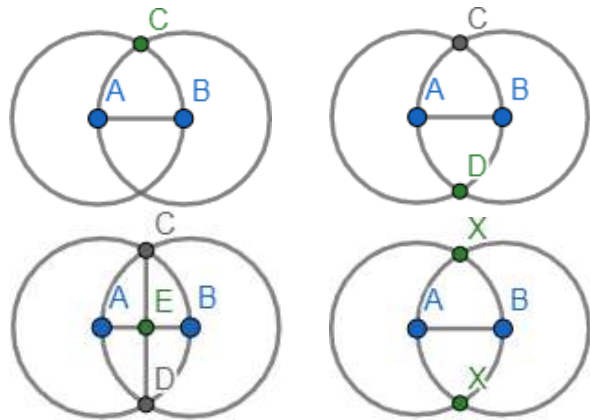


Figure 4—Visualized GPT-4 reasoning paths for the problem: "Find a point [Target] that is equidistant from given points A and B." Four illustrations depict different reasoning paths based on the target name. Naming the target **C** leads to a 3-step solution: draw circle A with radius AB, draw circle B with radius AB, and mark their intersection as C. Naming it **D** adds a fourth step, marking D as the final answer. Naming it **E** introduces variables C and D first, resulting in a 5-step solution with E on the perpendicular bisector of AB. For **X**, the model reverts to a 3-step process, marking any intersection arbitrarily and offering multiple solutions for X.

found within a pre-defined depth limit. Euclid-Net (Wong et al., 2022) uses "Deep visual reasoning with backtracking," where a neural network assists in step selection. These models rely on precise visual component extractors and real-time feedback during their trial-and-error process. To assess the possible benefits of real-time feedback, we introduce Feedback mode (+FB), where validators have access to ground truth answers instead of relying solely on their internal knowledge but are restricted to simply informing solvers about the correctness of their steps, avoiding any solution leak. Our findings show only a slight performance improvement, indicating that solvers struggle to develop complex solutions even with ground truth hints, highlighting the nuanced complexities of applying LLMs to constructive geometry.

Method	pass@1	pass@50
Few-Shot		
LlamaV2 (7B)	3.1 (\pm 0.5)	4.4 (\pm 0.7)
LlamaV2 (13B)	4.4 (\pm 0.4)	7.5 (\pm 0.8)
MetaMath (13B)	4.7 (\pm 0.4)	8.1 (\pm 0.8)
Mistral (7B)	5.1 (\pm 0.6)	8.7 (\pm 1.1)
Zephyr- α (7B)	5.3 (\pm 0.5)	8.9 (\pm 1.1)
MetaMath-Mistral (7B)	8.9 (\pm 0.7)	13.4 (\pm 1.2)
ChatGPT	11.7 (\pm 1.1)	18.6 (\pm 1.5)
GPT4	21.2 (\pm 1.3)	38.3 (\pm 1.4)
GPT4-V	22.8 (\pm 1.2)	38.5 (\pm 1.4)
Finetuned		
LlamaV2 (7B)	3.7 (\pm 0.6)	5.1 (\pm 0.7)
LlamaV2 (13B)	4.9 (\pm 0.4)	8.7 (\pm 0.8)
MetaMath (13B)	5.3 (\pm 0.5)	9.2 (\pm 1.0)
Mistral (7B)	6.9 (\pm 0.7)	9.7 (\pm 1.1)
Zephyr- α (7B)	6.6 (\pm 0.6)	9.5 (\pm 1.2)
MetaMath-Mistral(7B)	9.4 (\pm 0.9)	16.2 (\pm 1.3)
Ours - Simulacra		
MetaMath-Mistral (7B)	14.9 (\pm 1.3)	21.1 (\pm 1.7)
ChatGPT	32.3 (\pm 1.3)	61.4 (\pm 2.1)
GPT4	38.9 (\pm 1.1)	67.7 (\pm 2.2)
GPT4-V	37.1 (\pm 1.4)	65.9 (\pm 2.0)
FB + MetaMath-Mistral (7B)	15.1 (\pm 1.5)	21.4 (\pm 1.6)
FB + ChatGPT	35.6 (\pm 1.7)	63.5 (\pm 2.2)
FB + GPT4	41.2 (\pm 1.6)	71.2 (\pm 2.0)
FB + GPT4-V	40.3 (\pm 1.7)	70.6 (\pm 2.3)
Supervised Visual Component + Exhaustive Search		
Method	Accuracy	
(Macke et al., 2021) (LOO-levels)	44.1	
(Macke et al., 2021) (LOO-packs)	45.5	
(Wong et al., 2022) Euclid-Net	75.5	

Table 1: Results on Euclidean. Ours refers to the $S_{NL} - S_{GT}$ with VRP, variable renaming, and Adaptive Shot (Self). LOO stands for "Leave-One-Out": The model is either trained on the rest of the levels in the same pack (LOO-level) or the rest of the packs in the dataset (LOO-packs). GPT4-V refers to the multimodal use of GPT4 without the use of VRP in the Simulacra experiments.

6 Ablation Studies

This section presents ablation studies that underpin our model’s development, as detailed in the methods section. These studies highlight the iterative refinement and integration of model elements, addressing limitations observed in LLMs during geometric problem-solving.

6.1 Hallucinations and Context Dependence

In our exploration, we initially tested LLMs in a zero-shot manner using only tool descriptions as context, which often resulted in hallucinated tool functionalities. To mitigate this, we transitioned to a few-shot setup, providing solved examples to demonstrate proper tool usage. This adjustment reduced the incidence of tool hallucinations as models benefited from clear demonstrations. However, a new problem emerged: models began replicating entire reasoning processes and were heavily influenced by the step sequences in the few-shot examples. We hypothesized that this behavior stemmed from the instruction-following training nature of LLMs, where they are predisposed to mimic styles and patterns seen in the prompts. In our case, models were not mimicking styles but inadvertently replicating entire reasoning processes.

Our adaptive shot method solved these challenges. Unlike random or static few-shot examples, our adaptive shot method ensures that the most valuable demonstrations are selected for the models at each step. This dynamic approach significantly reduced hallucinations and repetitive reasoning patterns. We also compared Adaptive-Shot (Self) with a simpler variant, Adaptive-Shot (ST), where the Sentence Transformer selected the examples after the filtering stage. The Adaptive-Shot (Self) method proved slightly more effective, albeit with the trade-off of more API calls than the Adaptive-Shot (ST), showcasing that our prompting method exceeds the Sentence Transformer’s contextual abilities.

The results, as shown in Table 2, validate the performance benefits of our proposed method, demonstrating its superiority in addressing the limitations observed in earlier setups.

6.2 Effectiveness of Domain and Role Division

Interestingly, LLMs proved quite successful using geometric tools when prompted to generate ideas—not specific steps. Although they made convenient assumptions, their overall reasoning was still accurate. This led us to question the root cause

of the discrepancy between knowing the solution as a plan in natural language and failing to execute it accurately using strict and abstract geometric tools.

We tested whether we benefit from those worlds’ synergy: an LLM could focus on generating ideas in natural language, and another could specialize in transforming these ideas into a series of geometric tool steps. By employing such a multi-agent setup, as seen in Table 3, we observed a significant performance gain $-13.6 \rightarrow 21.5$ pass@1 – when comparing a single agent (S_{GT}) operating directly with geometric tools against a duo of collaborative agents with differentiated domains ($S_{NL} - S_{GT}$). The introduction of validators, the second set of agents in our framework, plays a crucial role in assessing the proposed rationales and geometric tool steps. Their continuous feedback loop with solvers, correcting superficial mistakes and preventing errors, leads to a further performance boost of $22.2 \rightarrow 28.1$ pass@1 in our proposed multi-agent configuration. Our approach of disentangling reasoning and execution facilitates efficient dialogue between agents. Each agent entirely focuses on its domain, transmitting only the necessary information to the next step while being receptive to feedback. This method enables more accurate and reliable problem-solving compared to traditional multi-agent techniques.

Method	pass@1	pass@50
Zero-Shot	5.9 (± 1.9)	9.6 (± 2.7)
Few-Shot (Tutorial)	7.2 (± 1.4)	15.9 (± 3.1)
Few-Shot (Alpha)	11.4 (± 1.8)	18.6 (± 3.2)
Few-Shot (Beta)	12.7 (± 2.2)	20.8 (± 3.6)
Adaptive-Shot (ST)	13.3 (± 1.8)	21.2 (± 3.0)
Adaptive-Shot (Self)	13.6 (± 1.8)	21.5 (± 2.7)

Table 2: Effectiveness of in-context examples: ChatGPT 3.5-Turbo on Alpha and Beta levels. Our adaptive method showcases its ability to employ useful examples from the already seen levels, leading to increased performance than static, hard-coded alternatives.

Configuration	Agents	Domains	pass@1	pass@50
S_{GT}	1	1	13.6 (± 1.8)	21.5 (± 2.7)
SV_{GT}	2	1	17.9 (± 1.2)	34.9 (± 2.3)
$S_{NL} - S_{GT}$	2	2	22.2 (± 1.3)	46.7 (± 2.0)
$SV_{NL} - SV_{GT}$	4	2	28.1 (± 1.1)	53.5 (± 2.4)

Table 3: ChatGPT 3.5-Turbo on Alpha and Beta levels. **S**: Solver, **V**: Validator, **NL**: Natural language, **GT**: Geometric tools. Our proposed multi-agent setup with role and domain differentiation showcases significant performance gains against a typical multi-agent setup $S_{NL} - S_{GT}$ vs S_{GT} . This holds also true with the introduction of validators that further boost the overall performance of the framework $SV_{NL} - SV_{GT}$ vs SV_{GT} .

6.3 Visual Aids in Spatial Reasoning

Another noteworthy finding was that LLMs often created new geometric objects without acknowledging their overlap with existing ones. Moreover, they occasionally suggested steps that violated geometric rules or led to repetitive movements. We hypothesized that LLMs’ difficulty with geometric reasoning in 2D spaces stems from a lack of exposure to such setups, typically operating in a unidimensional, left-to-right manner. This raised the question: Could introducing a visual signal bridge this reasoning gap?

We prompted GPT4-V with simple freehand sketches of geometric objects to explore this. The model successfully identified these, including subtle aspects like right angles indicated by small corner squares. In this way, we first established that the model can indeed understand geometric scenes. Continuing with a more complex test, we presented GPT4-V with an image/problem pair and asked for the first solution step. We then deliberately performed an incorrect step, drew it on the image, and presented it back to the model. The model often validated these erroneous steps in this setup, suggesting a disconnect between scene understanding and geometric reasoning.

The findings in Table 4 illustrate this observation: Comparing GPT4 with its multimodal variant, GPT4-V, revealed a marginal improvement, suggesting that visual signals assist during initial scene understanding. Building on these insights, we compared our proposed VRP method, VRP-GPT4, to the multimodal approach GPT4-V and observed an additional performance boost. Here lies the key advantage of our VRP method: Not only does it match, if not slightly surpass, the effectiveness of the multimodal approach, but it enables the transfer of scene understanding benefits to non-visually capable models like ChatGPT, which significantly improved when enhanced with VRP.

6.4 Impact of Geometry Nomenclature

LLMs mirror the human convention of alphabetical naming in mathematical contexts. The choice of target variables later in the alphabetical order leads to longer and more inaccurate solutions. This is a byproduct of their training on human-generated texts, where entities in algebraic or geometric contexts typically adhere to an alphabetical naming convention -labeling a triangle as ABC rather than EOA. Similarly, variables associated with the as-

signment of solutions, like X, would theoretically condition the LLM to find its value, possibly assigning it to any given constructed object. We designed an experiment with 20 geometric problems to empirically validate this hypothesis, each requiring 3 to 5 solution steps. In these problems, we manipulated the target variable in three distinct ways: maintaining the original name (+0), replacing it with the letter X (+X), and renaming it using a letter 1, 2, or 3 positions further in the alphabetical sequence (+1, +2, +3). As depicted in Table 5, models perform worse when the target variable is shifted by one or two letters in the alphabet. Interestingly, this tendency diminishes when the target is more than three letters away and is further reduced with the substitution of 'X,' underscoring its effectiveness as a neutral, bias-mitigating variable.

Domain	Method	pass@1	pass@50
Language	ChatGPT	13.6 (\pm 1.8)	21.5 (\pm 2.7)
	GPT4	23.9 (\pm 0.9)	44.8 (\pm 1.6)
	ChatGPT*	28.1 (\pm 1.2)	53.5 (\pm 1.7)
	GPT4*	33.7 (\pm 1.0)	62.0 (\pm 1.3)
Multimodal	GPT4-V	24.2 (\pm 1.4)	45.1 (\pm 1.7)
	VRP-ChatGPT	19.4 (\pm 1.1)	37.1 (\pm 1.3)
	VRP-ChatGPT*	34.5 (\pm 0.8)	59.2 (\pm 1.2)
	VRP-GPT4	25.3 (\pm 0.9)	48.6 (\pm 1.4)
	VRP-GPT4*	38.8 (\pm 0.9)	64.6 (\pm 1.2)

Table 4: Multimodal prompt effectiveness. Experiments on the Alpha and Beta pack levels. The (*) symbol refers to an $SV_{NL} - SV_{GT}$ multi-agent configuration.

	ChatGPT	ChatGPT*	VRP-ChatGPT*
+0	10.7 (\pm 2.1)	48.1 (\pm 2.3)	60.7 (\pm 2.9)
+1	10.1 (\pm 2.4)	46.3 (\pm 2.1)	57.9 (\pm 2.5)
+2	10.1 (\pm 2.4)	47.5 (\pm 1.9)	57.9 (\pm 3.0)
+3	10.6 (\pm 1.2)	47.9 (\pm 1.7)	59.2 (\pm 2.2)
+X	10.5 (\pm 1.7)	48.6 (\pm 1.9)	61.1 (\pm 2.2)
	GPT4	GPT4*	VRP-GPT4*
+0	35.2 (\pm 2.5)	52.9 (\pm 2.1)	65.1 (\pm 2.2)
+1	32.8 (\pm 2.6)	50.4 (\pm 2.2)	63.4 (\pm 3.2)
+2	31.6 (\pm 2.8)	50.7 (\pm 2.7)	63.6 (\pm 2.9)
+3	32.4 (\pm 1.1)	51.8 (\pm 1.4)	64.7 (\pm 2.1)
+X	37.5 (\pm 1.3)	53.3 (\pm 1.5)	66.2 (\pm 2.4)

Table 5: Qualitative results on the effect of variable renaming. Results refer to the pass@50 metric. The star symbol (*) refers to $SV_{NL} - SV_{GT}$ configuration.

6.5 Generalisation to different datasets

While our focus has been primarily on geometric problems, our framework is designed with adaptability in mind. Transitioning to a different toolset from the Euclidean tools, currently represented in our framework as text-based Name: Operation lists, requires minimal adjustment. This flexible design can theoretically introduce any new toolset by simply integrating it into the domain-specific agents' persona prompts with slight modifications to align

with their unique capabilities. To this end, we tested its performance against three datasets involving mathematical reasoning: GSM8K, SVAMP, and the Geometry split from the MATH dataset. The toolset consisted of all basic math operations (+, -, *, /) and a Python interpreter tool that could sequentially execute any steps suggested before its call. We were inspired by (Zhao et al., 2023), who identified diverse reasoning patterns with Chain of Thought (COT) (Wei et al., 2022) versus Program-Aid (PAL) (Gao et al., 2023) methodologies. While COT is recognized for its creativity and flexibility in devising solutions, PAL is noted for its enhanced accuracy in numerical computations. This differentiation of domains was an ideal fit for our framework: A pair of Solver-Validator agents (SV_{NL}) initially planned the solution of a math problem in the natural language domain. The produced rationale is passed to another pair (SV_{MPT}), which utilizes the set of tools to formulate precise solutions. Finally, we employed the Adaptive Few-Shot mechanism with examples from each dataset's training split. Results in Table 6 show our multi-agent setup performing close to state-of-the-art methods despite not being primarily designed for these tasks. The MetaMath-Mistral model also saw significant performance boosts using our method, even in non-geometric setups.

Method	Model	GSM8K	SVAMP	Geometry
Ours	MM-Mist	84.3	79.7	21.6
	ChatGPT	88.4	86.1	40.2
	GPT4	96.9	95.8	56.3
MetaMath (Yu et al., 2023)	MM-Mist	77.7	75.8	18.4
MS (SC, K=15) (Zhao et al., 2023)	ChatGPT	89.2	85.2	N/A
	GPT4	96.8	95.8	N/A
PHP (SC, K=40) (Zheng et al., 2023)	ChatGPT	85.1	83.1	25.4
	GPT4	95.5	91.9	41.9
CSV (K=1 / K=16) (Zhou et al., 2023a)	GPT4-Code	92.9 / 97.0	N/A	54.0 / 64.9

Table 6: Results of the multi-agent framework on mathematical datasets. MM-Mist: MetaMath-Mistral 7B. N/A refers to not reported results.

7 Conclusions

Our study highlights the challenges LLMs face in constructive geometry, including limited skill transfer from other mathematical domains, inadequacies of typical prompting techniques, and a lack of 2D spatial reasoning. We identify that existing LLMs struggle with geometric tasks without inductive solid biases like theorem provers (Trinh et al., 2024). Furthermore, our research demonstrates that a multi-agent system with role and domain spe-

cializations can effectively address geometric problems and perform equally well in other domains. We hope our work serves as a foundation for developing training systems that operate in multi-agent settings, which is crucial for domains requiring deep, specific, and accurate cognitive processing.

8 Limitations

We hope to attract more attention to this domain and look forward to seeing future work improving our multi-agent setup or shifting the training paradigm of newer-generation LLMs to foster deeper reasoning in mathematical and geometric domains. However, we are aware of the potential risk associated with our work, specifically the cost of experiments involving closed-source LLMs behind APIs. These costs primarily stem from extensive communication rounds and large context sizes, which can lead to substantial expenses. Researchers planning to replicate or extend our work must consider this, as it could significantly impact their research budgets and timelines. Furthermore, we acknowledge the possible limitations of applying our method to domains where planning, tool usage, and/or validation might not be critical. Simpler methods could be proven cheaper and better in these domains.

Finally, we do not identify any ethical considerations associated with our proposed ideas and suggested methods or any possible malicious or unintended harmful uses.

References

- Jacob Andreas. 2022. [Language models as agent models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5769–5779, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. 2020. [The hanabi challenge: A new frontier for ai research](#). *Artificial Intelligence*, 280:103216.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. [Language models are few-shot learners](#). *CoRR*, abs/2005.14165.
- Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. 2023. [Autoagents: A framework for automatic agent generation](#). *Preprint*, arXiv:2309.17288.
- Jiaqi Chen, Tong Li, Jinghui Qin, Pan Lu, Liang Lin, Chongyu Chen, and Xiaodan Liang. 2022. [Unigeo: Unifying geometry logical reasoning via reformulating mathematical expression](#). In *The 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. [Evaluating large language models trained on code](#). *arXiv preprint arXiv:2107.03374*.
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. [Palm: Scaling language modeling with pathways](#). *arXiv preprint arXiv:2204.02311*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. [Training verifiers to solve math word problems](#). *arXiv preprint arXiv:2110.14168*.
- Allan Dafoe, Edward Hughes, Yoram Bachrach, Tantum Collins, Kevin R. McKee, Joel Z. Leibo, Kate Larson, and Thore Graepel. 2020. [Open problems in cooperative ai](#). *Preprint*, arXiv:2012.08630.
- Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. [Qlora: Efficient finetuning of quantized llms](#). *arXiv preprint arXiv:2305.14314*.
- Yali Du, Bo Liu, Vincent Moens, Ziqi Liu, Zhicheng Ren, Jun Wang, Xu Chen, and Haifeng Zhang. 2021. [Learning correlated communication topology in multi-agent reinforcement learning](#). In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '21*, page 456–464, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.
- Euclidea. [Euclidea game](#).
- Richard Fitzpatrick. 2007–2008. [Euclid’s Elements of Geometry](#). I.L. Heiberg, Ed. & Trans.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. [Pal: Program-aided language models](#). In *Proceedings of the 40th International Conference on Machine Learning*, pages 10764–10799. PMLR.

- Google. 2023. [Bard](#).
- Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. [Reasoning with language model is planning with world model](#). *ArXiv*, abs/2305.14992.
- Serhii Havrylov and Ivan Titov. 2017. [Emergence of language with multi-agent games: Learning to communicate with sequence of symbols](#). In *5th International Conference on Learning Representations (ICLR 17, workshop track)*. 5th International Conference on Learning Representations, ICLR 2017 ; Conference date: 24-04-2017 Through 26-04-2017.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2023. [Metagpt: Meta programming for a multi-agent collaborative framework](#). *Preprint*, arXiv:2308.00352.
- Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. 2020. [A simple language model for task-oriented dialogue](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 20179–20191. Curran Associates, Inc.
- Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. [Inner monologue: Embodied reasoning through planning with language models](#). *Preprint*, arXiv:2207.05608.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. 2023. [Mistral 7b](#). *Preprint*, arXiv:2310.06825.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. [Large language models are zero-shot reasoners](#). *ArXiv*, abs/2205.11916.
- Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. 2019. [Spoc: Search-based pseudocode to code](#). In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. [Camel: Communicative agents for "mind" exploration of large language model society](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2021. Towards understanding and mitigating social biases in language models. In *International Conference on Machine Learning*, pages 6565–6576. PMLR.
- Bill Yuchen Lin, Seyeon Lee, Rahul Khanna, and Xiang Ren. 2020. [Birds have four legs?! NumerSense: Probing Numerical Commonsense Knowledge of Pre-Trained Language Models](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6862–6868, Online. Association for Computational Linguistics.
- Jiaju Lin, Haoran Zhao, Aochi Zhang, Yiting Wu, Huqiyue Ping, and Qin Chen. 2023. [Agentsims: An open-source sandbox for large language model evaluation](#). *Preprint*, arXiv:2308.04026.
- Yat Long Lo, Christian Schroeder de Witt, Samuel Sokota, Jakob Nicolaus Foerster, and Shimon Whiteson. 2023. [Cheap talk discovery and utilization in multi-agent reinforcement learning](#). *Preprint*, arXiv:2303.10733.
- Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. 2021. [Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning](#). In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6774–6786, Online. Association for Computational Linguistics.
- J. Macke, J. Sedlar, M. Olsak, J. Urban, and J. Sivic. 2021. Learning to solve geometric construction problems from images.
- Marvin Minsky. 1986. *The Society of Mind*. Simon & Schuster, Inc., USA.
- Spyridon Mouselinos, Mateusz Malinowski, and Henryk Michalewski. 2023. [A simple, yet effective approach to finding biases in code generation](#). In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11299–11329, Toronto, Canada. Association for Computational Linguistics.
- OpenAI. 2023. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. [Training language models to follow instructions with human feedback](#). *Preprint*, arXiv:2203.02155.

- Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.
- Joon Sung Park, Lindsay Popowski, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2022. [Social simulacra: Creating populated prototypes for social computing systems](#).
- Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. [Are NLP models really able to solve simple math word problems?](#) In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.
- PyEuclidean. [PyEuclidean](#).
- Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Yufan Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. [Communicative agents for software development](#). Preprint, arXiv:2307.07924.
- Laria Reynolds and Kyle McDonell. 2021. [Prompt programming for large language models: Beyond the few-shot paradigm](#). In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI EA '21, New York, NY, USA. Association for Computing Machinery.
- Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. [Solving geometry problems: Combining text and diagram interpretation](#). In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1466–1476, Lisbon, Portugal. Association for Computational Linguistics.
- Junjie Sheng, Xiangfeng Wang, Bo Jin, Junchi Yan, Wenhao Li, Tsung-Hui Chang, Jun Wang, and Hongyuan Zha. 2020. [Learning structured communication for multi-agent reinforcement learning](#).
- Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. [AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.
- Sainbayar Sukhbaatar, arthur szlam, and Rob Fergus. 2016. [Learning multiagent communication with backpropagation](#). In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.
- Alfred Tarski. 1959. What is elementary geometry? In *Studies in Logic and the Foundations of Mathematics*, volume 27, pages 16–29. Elsevier.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiohu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. [Llama 2: Open foundation and fine-tuned chat models](#). Preprint, arXiv:2307.09288.
- Trieu Trinh, Yuhuai Wu, Quoc Le, He He, and Thang Luong. 2024. [Solving olympiad geometry without human demonstrations](#). *Nature*.
- Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Cl  mentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. 2023. [Zephyr: Direct distillation of lm alignment](#). Preprint, arXiv:2310.16944.
- Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. [Universal adversarial triggers for attacking and analyzing NLP](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162, Hong Kong, China. Association for Computational Linguistics.
- Zekun Moore Wang, Zhongyuan Peng, Haoran Que, Jiaheng Liu, Wangchunshu Zhou, Yuhan Wu, Hongcheng Guo, Ruitong Gan, Zehao Ni, Man Zhang, Zhaoxiang Zhang, Wanli Ouyang, Ke Xu, Wenhao Chen, Jie Fu, and Junran Peng. 2023. [Rolellm: Benchmarking, eliciting, and enhancing role-playing abilities of large language models](#). *arXiv preprint arXiv: 2310.00746*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. [Chain of thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*.
- Man Fai Wong, Xintong Qi, and Chee Wei Tan. 2022. [Euclidnet: Deep visual reasoning for constructible problems in geometry](#). *Adv. Artif. Intell. Mach. Learn.*, 3:839–853.

- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. [Autogen: Enabling next-gen llm applications via multi-agent conversation](#). *Preprint*, arXiv:2308.08155.
- Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. 2023. [Self-evaluation guided beam search for reasoning](#). *Preprint*, arXiv:2305.00633.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. [Tree of thoughts: Deliberate problem solving with large language models](#). *ArXiv*, abs/2305.10601.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023b. [React: Synergizing reasoning and acting in language models](#). In *The Eleventh International Conference on Learning Representations*.
- Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. [Metamath: Bootstrap your own mathematical questions for large language models](#). *arXiv preprint arXiv:2309.12284*.
- Ming-Liang Zhang, Fei Yin, and Cheng-Lin Liu. 2023a. [A multi-modal neural geometric solver with textual clauses parsed from diagram](#). In *International Joint Conference on Artificial Intelligence*.
- Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2023b. [Automatic chain of thought prompting in large language models](#). In *The Eleventh International Conference on Learning Representations*.
- Xu Zhao, Yuxi Xie, Kenji Kawaguchi, Junxian He, and Qizhe Xie. 2023. [Automatic model selection with large language models for reasoning](#). *Preprint*, arXiv:2305.14333.
- Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. 2023. [Progressive-hint prompting improves reasoning in large language models](#). *arXiv preprint arXiv:2304.09797*.
- Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023a. [Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification](#). *Preprint*, arXiv:2308.07921.
- Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. 2023b. [Agents: An open-source framework for autonomous language agents](#). *Preprint*, arXiv:2309.07870.
- Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023c. [Large language models are human-level prompt engineers](#). In *The Eleventh International Conference on Learning Representations*.

A Information on Models and Datasets

Model Name	Link	LICENSE
LlamaV2 (Touvron et al., 2023)	https://github.com/facebookresearch/llama	LLAMA 2 COMMUNITY LICENSE AGREEMENT
MetaMath (Yu et al., 2023)	https://github.com/meta-math/MetaMath	Apache License 2.0
Mistral (Jiang et al., 2023)	https://github.com/mistralai/mistral-src	Apache License 2.0
Zephyr- α (Tunstall et al., 2023)	https://huggingface.co/HuggingFaceH4/zephyr-7b-alpha	MIT
MetaMath-Mistral (Yu et al., 2023)	https://github.com/meta-math/MetaMath	Apache License 2.0
ChatGPT (Ouyang et al., 2022)	OpenAI - API usage of <i>gpt-3.5-turbo-16k</i>	N/A
GPT4 (OpenAI, 2023)	OpenAI - API usage of <i>gpt-4-32k</i>	N/A
GPT4-V (OpenAI, 2023)	OpenAI - API usage of <i>gpt-4-vision-preview</i>	N/A

Table 7: URL and Licenses of used Large Language Models / APIs.

Dataset Name	Link	LICENSE
Euclid-Elements (Fitzpatrick, 2007-2008)	https://farside.ph.utexas.edu/books/Euclid/Elements.pdf	CC BY-NC-SA 4.0
Python Port of Euclidea (PyEuclidea)	https://github.com/mirefek/py_euclidea	MIT
Euclidea Wiki Page	https://euclidea.fandom.com/wiki/Euclidea_Wiki	CC BY-SA
GSM8k (Cobbe et al., 2021)	https://github.com/openai/grade-school-math	MIT
SVAMP (Patel et al., 2021)	https://github.com/arkilpatel/SVAMP	MIT
Geometry / MATH (Hendrycks et al., 2021)	https://github.com/hendrycks/math/	MIT

Table 8: URL and Licenses of used Datasets.

The licenses associated with the models and datasets used in this work are consistent with their intended use in academic and research contexts. Specifically, the LLAMA 2 Community License Agreement and the Apache License 2.0 are permissive licenses that allow for extensive use in research and academic settings, provided that proper attribution is given. Similarly, the MIT License, the CC BY-NC-SA 4.0, and the CC BY-SA also support usage in academic work, allowing for modification and distribution under certain conditions. The OpenAI API usage adheres to OpenAI’s terms of service, which permits their application in research.

B Information on Experimental Setup

Our experimental setup consisted of 1x NVIDIA A100 GPU. Regarding the fine-tuning results on Euclid’s Elements of Table 1, we trained all LLMs using the bitsandbytes library (<https://github.com/TimDettmers/bitsandbytes>) and 4-bit quantization with the QLoRA technique (Dettmers et al., 2023). The Euclid-Elements dataset we created consists of approximately 1 million characters, or equivalently 290k thousand ChatGPT / GPT4 tokens.

Below, the reader can find the full hyperparameter list:

Hyperparameter	Value
Training Epochs	3
Batch Size	32
Accumulation Steps	4
Learning Rate	2e-5
Warmup Ratio	0.03
Scheduler	Cosine
Max Gradient Norm	0.3
Weight Decay	0.001
Lora Alpha	16
Lora Dropout	0.1
Lora R	64
Use 4bit	True
BnB_4bit_compute_dtype	Float16
BnB_4bit_quant_type	NF4
Gradient Checkpointing	True
Optimizer	Paged_adamw_32bit

Table 9: Hyperparameters and their values

Regarding the training objective, we used the typical causal language modeling loss. Moreover, we used

a validation split of 10% sampled uniformly across different book chapters. We monitored the validation loss every 500 steps as our metric for early stopping.

Regarding the API calls to OpenAI models, gpt3.5-turbo-16k was used for the ChatGPT experiments, gpt4-32k was used for the GPT4 experiments, and the extraction of Visual Relation Prompts, The endpoint called gpt4-vision-preview was used. Our API calls were subject to throttling limits, and waiting loops were introduced to avoid service interruptions. We conducted most of our ablation studies and early experiments with ChatGPT to avoid massive waiting times and reduce the high experimental cost. The total experiment time was approximately 500 hours, and our total costs were around 2000 USD.

Finally, regarding the choice of 0.2 and 0.6 temperature values for pass@1 and pass@50, we experimented with different combinations of values for solvers on the Alpha pack of the Euclidea dataset and a 100 problem subset of GSM8K. For pass@1, values between 0.2 and 0.3 worked best, while for pass@50, values between 0.6 and 0.8 also worked great. We finalized our decision on the values 0.2 and 0.6 for all solver experiments. The temperature of validator agents was 0.8 in all cases, based on the idea that we wanted more expressiveness and different reasoning pathways to be considered when validating a solution. We found that setting the temperature too low (0-0.2) led validators to accept more solutions verbatim, defying their purpose.

C Tools and problems from Euclidea dataset

There are 10 tools available in Euclidea, although not all of them can be used on every level. Tools become available progressively as the difficulty increases. Here is a list of all available tools:

1. **Move Tool:** Moves a geometric object.
2. **Point Tool:** Marks a point and labels it.
3. **Line Tool:** Draws a line between two points or a ray from a starting point.
4. **Circle Tool:** Constructs a circle using a specific point as the center.
5. **Perpendicular Bisector Tool:** Creates the perpendicular bisector of a segment between two points.
6. **Perpendicular Tool:** Draws a line perpendicular to a given one at a specific point.
7. **Angle Bisector Tool:** Creates a line that bisects a given angle.
8. **Parallel Tool:** Draws a line parallel to a given line or segment.
9. **Compass Tool:** Uses a compass to construct a circle with a radius equal to a given segment.
10. **Intersect Tool:** Marks the intersection between two geometric objects.

Here are some sample problems from different difficulty levels:

Given the rectangle ABCD with $AB > AD$. Inscribe a rhombus in the rectangle so that they share a diagonal.

Available Tools: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Intersect Tool]

Solution:

Perpendicular Bisector Tool: Construct the perpendicular bisector of AC, intersecting AB at E and CD at F.

Line Tool: Construct line AF.

Line Tool: Construct line CE.

Problem: Rhombus in Rectangle - Pack: Alpha

Given the side AB. Construct a rhombus with the given side and an angle of 45° in a vertex.

Available Tools: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Perpendicular Tool, Intersect Tool, Angle Bisector Tool]

Solution:

Perpendicular Tool: Construct the perpendicular to AB from A; let C be a distinct arbitrary point on that perpendicular.

Angle Bisector Tool: Construct the angle bisector of BAC.

Circle Tool: Construct the circle with center A and radius AB, intersecting the angle bisector at D and line AC at E with E on the side of A opposite from C.

Perpendicular Tool: Construct the perpendicular to AC through D.

Line Tool: Construct line BE.

Problem: Lozenge - Pack: Gamma

Let A be the vertex. Construct two rays that divide the given angle of 54 degrees into three equal parts.

Available Tools: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Intersect Tool, Parallel Tool, Compass Tool]

Solution:

Circle Tool: Construct a circle with center B on either line and radius AB. Circle B intersects with line AB at point C, and the other line at point D.

Circle Tool: Construct a circle with center D and radius AD. Circle D intersects with circle B at point E.

Line Tool: Draw line AE. This is the first angle trisector.

Circle Tool: Draw circle with center E and radius EC. Circle E intersects with circle B at point F.

Line Tool: Draw line AF. This is the second angle trisector.

Problem: Angle of 54 - Pack: Theta

D Static Versus Adaptive Few-Shot

In this section, we will provide a more detailed overview of our adaptive few-shot mechanism.

Initially, we collect a set of previously solved problems alongside their solutions, which we refer to as our knowledge base. This set can be acquired in multiple ways: In the case of our GSM8K / SVAMP / Geometry-Math experiments of Table 6, we use all problems belonging to the provided training split.

In the case of the Euclidea experiments of Table 1, we begin with a small set of five handcrafted geometric challenges. Euclidea’s problems vary in difficulty and are grouped into increasingly difficult level packs. Since we are not provided with a training split, we add all problems belonging to previously seen packs to our knowledge base during the solution of a level. In this way, we continuously increase the size of our base and the availability of more diverse and complex techniques that our agents can utilize.

The second stage involves using a Sentence Transformer to reduce the size of our knowledge base. For this, we compare the cosine similarity scores of our current problem and the problems in our knowledge base and keep entries with scores over 0.5 or the top fifteen, whichever leads to fewer examples.

The third and final stage involves using the Sentence Transformer again, which chooses the top five most similar examples to build the final few-shot prompt. This procedure is referred to as Adaptive-Shot (ST) in our experiments. An alternative to this is to use the examples from the second step and prompt our solver-LLM to filter the top five most relevant examples by itself. We call this alternative Adaptive-Shot (Self). Note here that the second step of the procedure can be skipped during Adaptive-Shot (ST) since instead of filtering, we can directly return the top five results. However, it is necessary for the Adaptive-Shot (Self) method, where we need to respect the context size limits of the tested LLMs.

Below, we present an example of three different few-shot prompts: A static-handcrafted version, an adaptive prompt compiled by a Sentence Transformer (Adaptive ST), and finally, an adaptive prompt filtered by the model itself (Adaptive Self). Our model is ChatGPT, and the problem at hand is: *Construct the circumcircle of a given triangle ABC.*

Construct the circumcircle of a given triangle ABC.

Problem

[Line Tool, Circle Tool, Intersection Tool, Perpendicular Bisector Tool]

Available Tools

[...]

Example 1:

Description: Construct an equilateral triangle given one side. Let AB be the given side.

Tool List: [Line Tool, Circle Tool, Intersection Tool, Perpendicular Bisector Tool]

Solution:

Circle Tool: Construct a circle with center A and radius AB.

Circle Tool: Construct a circle with center B and radius AB, intersecting circle A at point C.

Line Tool: Construct lines AC and BC. Triangle ABC is equilateral.

Example 2:

Description: Construct a right angle at a point on a line. Given point A on a ray.

Tool List: [Line Tool, Circle Tool, Intersection Tool, Perpendicular Bisector Tool]

Solution:

Circle Tool: Construct a circle with center A and an arbitrary radius, intersecting the ray at points B and C.

Perpendicular Bisector Tool: Construct the perpendicular bisector of BC.

Example 3:

Description: Bisect a given segment. Let AB be the given segment.

Tool List: [Line Tool, Circle Tool, Intersection Tool]

Solution:

Circle Tool: Construct a circle with center A and radius AB.

Circle Tool: Construct a circle with center B and radius AB, intersecting circle A at points C and D.

Line Tool: Construct line CD, intersecting AB at point E. Point E is the midpoint of AB.

Example 4:

Description: Construct a 45-degree angle given a right angle. Let angle XYZ be a right angle with vertex Y.

Tool List: [Line Tool, Circle Tool, Intersection Tool, Perpendicular Bisector Tool]

Solution:

Circle Tool: Construct a circle with center Y and an arbitrary radius, intersecting XY at point A and YZ at point B.

Perpendicular Bisector Tool: Construct the perpendicular bisector of AB, intersecting XY at point C and YZ at point D.

Line Tool: Construct lines YC and YD. Angles XYZ and YYD are 45 degrees.

Example 5:

Description: Construct the perpendicular from a point to a line. Given point A and line l.

Tool List: [Line Tool, Circle Tool, Intersection Tool, Perpendicular Bisector Tool]

Solution:

Circle Tool: Construct a circle with center A intersecting line l at points B and C.

Problem Solution [ChatGPT]:

Circle Tool: Construct a circle at point A with radius equal to the distance between point A and any of the other two vertices, let's say point B.

Circle Tool: Construct a circle at point B with radius equal to AB.

Intersection Tool: Use the Intersection Tool to find the two intersection points between the two circles. These points are the circumcircle's endpoints.

Line Tool: Connect the two intersection points to form the circumcircle of triangle ABC.

Static Prompt X

[...]

Example 1:

Description: Inscribe a circle in the square.

Tool List: [Line Tool, Circle Tool, Perpendicular Bisector Tool]

Solution:

Line Tool: Construct line AC.

Perpendicular Bisector Tool: Construct the perpendicular bisector of AB, intersecting AB at E and AC at F.

Circle Tool: Construct a circle with center E and radius EF.

Example 2:

Description: Construct an equilateral triangle that is circumscribed about the circle O with radius R and contains the given point A on its circumference.

Tool List: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Perpendicular Tool]

Solution:

Line Tool: Construct line OA, intersecting the circle at B

Perpendicular Tool: Construct the perpendicular to OA from A

Circle Tool: Construct the circle with center B and radius OB, intersecting the circle at C and D, and line OA at E

Line Tool: Construct line CE

Line Tool: Construct line DE

Example 3:

Description: Construct the midpoint of the segment that connects the midpoints of the diagonals of the quadrilateral.

Tool List: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Perpendicular Tool]

Solution:

Construct the perpendicular bisector of AB, intersecting AB at E

Construct the perpendicular bisector of CD, intersecting CD at F

Construct the perpendicular bisector of EF

Construct line EF

Example 4:

Description: Inscribe an equilateral triangle in the circle using the given point as a vertex. The center of the circle is not given.

Tool List: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Perpendicular Tool]

Solution:

Circle Tool: Construct a circle with center A and an arbitrary radius, intersecting the original circle at B and C

Circle Tool: Construct the circle with center B and radius AB, intersecting circle A at D and E

Perpendicular Bisector Tool: Construct the perpendicular bisector of CD, intersecting the original circle at F

Perpendicular Bisector Tool: Construct the perpendicular bisector of CE, intersecting the original circle at G

Line Tool: Construct line FG

Example 5:

Description: Circumscribe a square about the circle O. Two of its sides should be parallel to the given line.

Tool List: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Perpendicular Tool]

Solution:

Perpendicular Tool: Construct the perpendicular to the line through O, intersecting the circle at A and B

Perpendicular Tool: Construct the perpendicular to AB from A

Perpendicular Tool: Construct the perpendicular to AB from B

Circle Tool: Construct circle with center A and radius OA, intersecting the perpendicular to AB from A at C and D

Perpendicular Tool: Construct the perpendicular to CD from C

Perpendicular Tool: Construct the perpendicular to CD from D

Problem Solution [ChatGPT]:

Perpendicular Bisector Tool: Construct the perpendicular bisector of side AB. Label the intersection with AB as D.

Perpendicular Bisector Tool: Construct the perpendicular bisector of side BC. Label the intersection with BC as E.

Perpendicular Bisector Tool: Construct the perpendicular bisector of side AC. Label the intersection with AC as F.

Intersection Tool: Label the intersection of the perpendicular bisectors as O.

Circle Tool: Create a circle with center O and radius OA.

Adaptive ST ✓

[...]

Example 1:

Description: Construct a segment connecting the sides of the angle to get a triangle whose perpendicular bisectors are intersected in the point O.

Tool List: [Line Tool, Circle Tool, Perpendicular Bisector Tool]

Solution:

Circle Tool: Construct the circle with center O and radius OA, intersecting the sides of the angle at B and C

Line Tool: Construct line BC

Example 2:

Description: Construct an equilateral triangle that is circumscribed about the circle O with radius R and contains the given point A on its circumference.

Tool List: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Perpendicular Tool]

Solution:

Line Tool: Construct line OA, intersecting the circle at B

Perpendicular Tool: Construct the perpendicular to OA from A

Circle Tool: Construct the circle with center B and radius OB, intersecting the circle at C and D, and line OA at E

Line Tool: Construct line CE

Line Tool: Construct line DE

Example 3:

Description: Construct the midpoint of the segment that connects the midpoints of the diagonals of the quadrilateral.

Tool List: [Line Tool, Circle Tool, Intersection Tool, Perpendicular Bisector Tool, Angle Bisector Tool, Perpendicular Tool]

Solution:

Perpendicular Bisector Tool: Construct the perpendicular bisector of AB, intersecting AB at E

Perpendicular Bisector Tool: Construct the perpendicular bisector of CD, intersecting CD at F

Perpendicular Bisector Tool: Construct the perpendicular bisector of EF Line Tool: Construct line EF

Example 4:

Description: Inscribe an equilateral triangle in the circle using the given point as a vertex. The center of the circle is not given.

Tool List: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Perpendicular Tool]

Solution:

Circle Tool: Construct a circle with center A and an arbitrary radius, intersecting the original circle at B and C

Circle Tool: Construct the circle with center B and radius AB, intersecting circle A at D and E

Perpendicular Bisector Tool: Construct the perpendicular bisector of CD, intersecting the original circle at F

Perpendicular Bisector Tool: Construct the perpendicular bisector of CE, intersecting the original circle at G

Line Tool: Construct line FG

Example 5:

Description: Construct a segment connecting the sides of an angle to get a triangle whose orthocenter is in the given point O.

Tool List: [Line Tool, Circle Tool, Perpendicular Bisector Tool, Perpendicular Tool]

Solution:

Perpendicular Bisector Tool: Construct the perpendicular to one side R1 of the angle through O, intersecting the second side at A

Perpendicular Bisector Tool: Construct the perpendicular to the second side R2 of the angle through O, intersecting the first side at B

Line Tool: Construct line AB

Problem Solution [ChatGPT]:

Perpendicular Bisector Tool: Construct the perpendicular bisector of side AB. Label the intersection with AB as D.

Perpendicular Bisector Tool: Construct the perpendicular bisector of side BC. Label the intersection with BC as E.

Circle Tool: Create a circle with center D and radius AD.

Adaptive Self ✓

We observe that the static prompt leads to an incorrect solution, with the model misinterpreting the meaning of a circumcircle, and blindly performing step sequences similar to those presented in the prompt.

In the case of Adaptive ST, the prompt consists of examples more closely related to the task at hand, leading to a correct solution. However, the model still suggests unnecessary steps, which can be attributed to style-mimicking tendencies introduced by instruction tuning.

Finally, when the model is tasked to filter the examples by itself - Adaptive Self, we observe an overlap in the selected examples with the Sentence Transformer method. Here, the model suggests a correct sequence of steps, free from any unnecessary or repetitive tool usage.

E Target Variable Name Bias

Below, we present the completions obtained from GPT4 when presented with the problem: "Find a point that is equidistant from given points A and B.", as presented visually in Figure 4.

Given two points, A and B, find a point [Target] so that its distance from A is equal to its distance from B.	Problem
[Line Tool, Circle Tool, Intersection Tool, Perpendicular Bisector Tool]	Available Tools
<p>Circle Tool: Construct a circle with center A and radius AB.</p> <p>Circle Tool: Construct another circle with center B and radius AB.</p> <p>Intersect Tool: Identify the points where the two circles intersect. These intersection points are equidistant from A and B. Let's call one of these intersection points C.</p> <p>The point C, found at the intersection of the two circles, satisfies the condition that its distance from A is equal to its distance from B.</p>	Target = C
<p>Circle Tool: Construct a circle with center A and radius AB.</p> <p>Circle Tool: Construct a circle with center B and radius AB.</p> <p>Intersect Tool: Find the intersection points of the two circles, let's call them C and D. These points are equidistant from A and B.</p> <p>By following these steps, point D will be at an equal distance from both points A and B.</p>	Target = D
<p>Circle Tool: Use the circle tool to draw a circle with center A and radius AB.</p> <p>Circle Tool: Use the circle tool again to draw a circle with center B and radius AB.</p> <p>Intersect Tool: The two circles will intersect at two points, which we can call point C and point D.</p> <p>Line Tool: Draw the line CD.</p> <p>Choose any point on line CD (other than C or D) and label it point E.</p>	Target = E
<p>Circle Tool: Construct a circle with center A and radius AB.</p> <p>Circle Tool: Construct a circle with center B and radius AB.</p> <p>Intersection Tool: The intersection points of the two circles represent the points that are equidistant from A and B. These intersection points are the solutions for point X.</p> <p>In this case, there will be two solutions for X, each lying on the circumference of the circles where they intersect.</p>	Target = X

Table 10: Completions for different name choices of the target variable.

F Baselines

We introduce two baselines to analyze the complexity of the geometric construction problem. In the first baseline, which we will call "Longest Common Solution" (LCS), we identify the top five longest common sequences of steps between ground truth solutions. Then, for each given problem, we uniformly sample from these sequences, adjusting the variables of each step and the tool usage to the current task. If a sampled step does not apply to the current state of the problem, the sampled sequence is discarded, and a new one is sampled instead. This baseline corresponds to the success rate of an agent who memorized a set of fixed solution steps and applied them to each given problem.

Additionally, we propose a second baseline named 'N-Gram Rollouts' (N-Gram), which begins by creating a database of uni-, bi-, and tri-grams derived from the tools used in ground-truth answers. For each problem, our method involves a two-phase iterative process. Initially, we choose either a single tool ($n = 1$) or a sequence of tools ($n > 1$) from our database. Following this, we select the geometric variables upon which these tools will be applied. To facilitate this, we maintain a memory initially populated with variables given in the problem statement. It is important to note that tool application varies in complexity, with some tools requiring a single variable (e.g., constructing a ray from point A) and others necessitating two (e.g., drawing a line between points A and B). For each tool or sequence of tools selected, we sample the required number of variables from memory, with recent variables weighted more heavily than older ones, following an exponential decay schema. Any new variables a tool generates (such as a new point) are added to this memory. This process is repeated until a predefined number of steps is reached.

Method	Correct tool sequence	Fully correct solution
LCS	0.17	0.02
1-Gram	0.14	0.11
2-Gram	0.12	0.09
3-Gram	0.07	0.06

Table 11: Performance of Baselines. Results refer to pass@1 estimated after 1000 trials.

According to Table 11, finding an entirely correct solution randomly is almost 0.1% probable. Note here that finding the correct tool sequence score is significantly higher. However, this is expected since many problems share common starting steps (e.g., drawing a line or a circle).

G Visual Relations Prompt

Here are two examples presenting the benefits of VRP inclusion. In Example 1, the models are devoid of visual input and seem to struggle to find a correct solution to the problem. On the contrary, in example 2, the models are now equipped with the VRP and complete the challenge successfully.

Example 1:

Given the rectangle, ABCD, and the point E located over the side AB at a random position, construct a line through E that cuts the rectangle into two parts of equal area. Problem
[Line Tool, Circle Tool, Perpendicular Bisector Tool, Angle Bisector Tool, Intersect Tool] Available Tools
Line Tool: Construct line EF, parallel to AB. Perpendicular Bisector Tool: Construct the perpendicular bisector of line EF. Intersect Tool: Mark the intersection of the perpendicular bisector and line AB as point G. Line Tool: Construct line EG. ✘ MetaMath-Mistral
Perpendicular Bisector: Construct the perpendicular bisector of AB. This bisector will cut the rectangle into two equal parts. Intersect Tool: Extend this perpendicular bisector line to intersect side CD. Label this intersection point as F. Draw a line through point E and point F. This line will cut the rectangle into two parts of equal area. ✘ ChatGPT
Line Tool: Construct Line AE. Perpendicular Bisector: Construct the perpendicular bisector of CD. Intersect Tool: Mark the intersection of the perpendicular bisector with CD and mark it as H. This point H is the midpoint of CD. Line Tool: Draw a line from point E to point H. ✘ GPT4

Example 2:



Image

Given the rectangle, ABCD, and point E located over the side AB at a random position, construct a line through E that cuts the rectangle into two parts of equal area.

Problem

[Line Tool, Circle Tool, Perpendicular Bisector Tool, Angle Bisector Tool, Intersect Tool]

Available Tools

Here is the full description of all the relations between geometric objects:

Shapes:

Rectangle: ABCD

Points:

A: Top-left corner of the rectangle.

B: Top-right corner of the rectangle.

C: Bottom-right corner of the rectangle.

D: Bottom-left corner of the rectangle.

E: Point outside of the rectangle.

Lines:

AB: Top horizontal line of the rectangle.

BC: Right vertical line of the rectangle.

CD: Bottom horizontal line of the rectangle.

DA: Left vertical line of the rectangle.

Relations:

AB is parallel to CD.

AD is parallel to BC.

The length of AB is equal to the length of CD.

The length of AD is equal to the length of BC.

E is an isolated point, not connected to any lines of the rectangle ABCD.

Visual Relations Prompt

Perpendicular Bisector Tool: Construct the perpendicular bisector of the line AB.

Perpendicular Bisector Tool: Construct the perpendicular bisector of the line CD.

Perpendicular Bisector Tool: Construct the perpendicular bisector of the line BC.

Perpendicular Bisector Tool: Construct the perpendicular bisector of the line DA.

Intersect Tool: Find the intersection of the perpendicular bisectors of AB and CD and name it F.

Intersect Tool: Find the intersection of the perpendicular bisectors of BC and DA and name it G.

Line Tool: Draw a line from E to F. ✓

MetaMath-Mistral

Line Tool: Draw a diagonal of the rectangle AC.

Perpendicular Bisector Tool: Construct the perpendicular bisector of AC.

Intersect Tool: Label the intersection point of the perpendicular bisector and AC as M.

Line Tool: Draw a line segment from point E to point M. This line, EM, will cut the rectangle ABCD into two parts of equal area since it passes through the midpoint of the diagonal AC. ✓

ChatGPT

Line Tool: Draw a diagonal of the rectangle AC.

Perpendicular Bisector Tool: Construct the perpendicular bisector of AC.

Intersect Tool: Label the intersection point of the perpendicular bisector and AC as M.

Line Tool: Draw a line segment from point E to point M. This line, EM, will cut the rectangle ABCD into two parts of equal area since it passes through the midpoint of the diagonal AC. ✓

GPT4

When tasked with tackling the same geometrical problem, our three agents (MetaMath-Mistral / ChatGPT / GPT4) resort to a "convenient" placement of point E outside rectangle ABCD. All of them try to utilize the fact that the perpendicular bisector of any edge of a rectangle cuts the rectangle into two equal areas. Then, a series of erroneous steps are suggested, trying to connect point E with this line.

After introducing the VRP, we observed that all agents managed to correct their responses. Placing geometric objects in the prompt helps them better understand the problem's nature. Agents no longer resort to steps that suggest splitting the rectangle area into two equal parts, ignoring point E, but rather respect the particular spatial circumstances the problem presents. Note here that the initial solutions are valid if the random position of point E happens to align with the midpoint of any side. The initial tendency to produce such solutions can be attributed to the fact that the splitting of a rectangle must have been observed more frequently in their training data through the use of diagonals or lines connecting midpoints.

H Multi-Agent Dialogue Examples

Illustration of our proposed $SV_{NL} - SV_{GT}$ multi-agent setup. We present the agent dialogue leading to a successful solution. The use of a single ChatGPT instance is not sufficient to solve this particular problem. However, the combined power of multiple agents can.

According to our proposed method, we employ 4 agents:

1. Natural Language Solver  (NL Solver)
2. Natural Language Validator  (NL Validator)
3. Geometric Tools Solver  (GT Solver)
4. Geometric Tools Validator  (GT Validator)

The four different agents interact with each other from top to bottom and from left to right.

Given the square ABCD, inscribe a circle in the square.

Problem

[Line Tool, Circle Tool, Perpendicular Bisector Tool, Intersect Tool]

Available Tools

[...] To inscribe a circle in the square ABCD means to draw a circle such that all four sides of the square are tangent to the circle. The circle should touch each side of the square at exactly one point. I suggest the following steps:

<STEP>1: Draw the square ABCD.

<STEP>2: Draw the diagonal AC of the square. The diagonal AC will pass through the center of the circle.

<STEP>3: Construct the perpendicular bisector of the diagonal AC. This line will intersect the diagonal at the center of the circle.

<STEP>4: Draw a line segment from the center of the circle to one of the vertices of the square (e.g., A or C). This line will be the radius of the circle.

<STEP>5: Use the compass to measure the length of the line segment drawn in the previous step.

<STEP>6: Set the compass to the measured length and draw a circle with the center at the intersection of the diagonal AC and the perpendicular bisector.

NL Solver



According to the task and the provided rationale, I am confident regarding the validity of the following steps:

<STEP>1: Correct.

<STEP>2: Correct.

<STEP>3: Correct.

Please try to correct the remaining steps.

NL Validator



[...]<STEP>4: Draw a line segment from the center of the circle to one of the points where the circle is tangent to the square. This line segment will be the radius of the circle.

Following these corrected steps, we will successfully inscribe a circle in the square ABCD.

NL Solver



Given the suggested expert rationale and the available tools I suggest the following steps:

<Line Tool>1: Draw a line segment from point A to point B.

<Line Tool>2: Draw a line segment from point A to point C.

<Perpendicular Bisector Tool>3: Draw the perpendicular bisector of AB.

<Intersect Tool>4: Mark the intersection point of the bisector and AB as E.

<Intersect Tool>5: Mark the intersection of the perpendicular bisector and the diagonal AC as O.

<Circle Tool>6: Create a circle with center O and radius AO.

GT Solver



According to the steps given, the available tools and the suggested rationale, I am confident of the validity of the following steps:

<Line Tool>1: Correct.

<Line Tool>2: Correct.

<Perpendicular Bisector Tool>3: Correct.

<Intersect Tool>4: Correct.

<Intersect Tool>5: Correct.

Please try to correct the solution according to my suggestions.

GT Validator



[...] <Circle Tool>6: Create a circle with center O and radius OE. ✓

GT Solver



ChatGPT Simulaca

Below, we present another example where we showcase the benefits of incorporating variable renaming in multi-agent scenarios. Simulacra are not immune to the variable name bias: In the example below, the variables **A**, **B**, and **C** are used. **A** and **B** mark the two initial points of the problem, while **C** stands as the target to be found. Initially, the **NL Solver** comes up with a promising 5-step rationale. However, the **NL Validator** corrects the proposed final step as a mistake. Since **A**, **B**, and **C** are consecutive letters in the alphabet, and **C** has already appeared in the fourth step, the validator marks the final step as redundant. The new variable **D** generated at the fifth step is after **C** (in terms of alphabetical ordering), and thus the problem must have been already solved. This error is propagated to the GT agents, leading to an eventually wrong final response.

Let $|AB| = 1$. Construct a point C on the line AB such that the length of AC is equal to $\sqrt{2}$. Do not use arbitrary numbers but only existing lengths and sizes in your solution

Problem

[Line Tool, Circle Tool, Perpendicular Bisector Tool, Perpendicular Tool, Angle Bisector Tool, Intersect Tool]

Available Tools

[...] To create a line segment whose length is the square root of 2, we should utilize an existing segment of length 1, here AB. This requires forming a right-angled triangle where one side measures 1, and determining the hypotenuse's length which will be the square root of 2. I suggest the following steps:

<STEP>1: Draw a straight line segment AB of any length. This will serve as our unit length.

<STEP>2: At point B, construct a perpendicular line to AB.

<STEP>3: From point B, draw a line segment BC on the perpendicular line with the same length as AB.

<STEP>4: Connect points A and C to form a right triangle ABC. The length of AC, the hypotenuse, is the square root of 2.

<STEP>5: Finally, mark the point D on the line AB such that $AD = AC$. This will give us the line segment AD of length square root of 2. ✓

NL Solver



According to the task and the provided rationale, I am confident regarding the validity of the following steps:

<STEP>1: Correct.

<STEP>2: Correct.

<STEP>3: Correct.

<STEP>4: Correct.

Please try to correct the solution according to my suggestions.

NL Validator



I suggest we should re-write the steps so that all suggested changes are incorporated.

[...] <STEP>5: Point C is the required point on the line segment extended from AB such that the length AC equals the square root of 2. ✗

NL Solver



Given the suggested expert rationale and the available tools I suggest the following steps:

<Circle Tool>1: Use the circle tool to draw a circle with center A and radius AB.

<Perpendicular Tool>2: Draw a perpendicular line to AB at point B using the Perpendicular Tool.

<Circle Tool>3: Draw a circle with center B and radius BA.

<Intersect Tool>4: Use the Intersection Tool to find point C, which is the intersection of the circle centered at B and the line perpendicular to AB.

<Line Tool>5: Draw a line from A to C using the Line Tool. ✗

GT Solver



According to the steps given, the available tools and the suggested rationale, I am confident of the validity of the following steps:

<Circle Tool>1: Correct.

<Perpendicular Tool>2: Correct.

<Circle Tool>3: Correct.

<Intersect Tool>4: Correct.

<Line Tool>5: Correct.

Great job!

GT Validator



Thank you!

GT Solver



GPT4 Simulacra

Now, let us rename the target variable and present the problem to our multi-agent setup again. Instead of the consecutive letter C, we swap the final target with a far-away (in terms of alphabetical ordering) as well as a common mathematical symbol X. We observe that the validator now approves the suggested rationale, leading to a successful generation.

Let $|AB| = 1$. Construct a point X on the line AB such that the length of AX is equal to $\sqrt{2}$. Do not use arbitrary numbers but only existing lengths and sizes in your solution.

Target Variable Renaming

[...] Based on the problem description, we are tasked with constructing a point on a given line segment such that the length of the segment from the initial point to the new point equals the square root of 2 without using explicit numerical values. Here's how we can approach this:

<STEP>1: Draw a straight line segment AB of any length (this will be our unit length as we are not to use numbers).

<STEP>2: Construct a circle with center at A and radius AB.

<STEP>3: Draw a line perpendicular to AB at point B. This line intersects the circle A at a new point D.

<STEP>4: Draw a line segment AD. The length of AD is the square root of 2 (according to Pythagoras' theorem).

<STEP>5: Take point X such that AX = AD. Now AX is equal to the square root of 2. ✓

NL Solver



According to the task and the provided rationale, I am confident regarding the validity of the following steps:

<STEP>1: Correct.

<STEP>2: Correct.

<STEP>3: Correct.

<STEP>4: Correct.

<STEP>5: Correct.

Your solution is valid!

NL Validator



Thank you for your help!

NL Solver



Given the suggested expert rationale and the available tools I suggest the following steps:

<Line Tool>1: Use the line tool to draw the line AB.

<Circle Tool>2: Draw a circle with center B and radius BA.

<Perpendicular Tool>3: Draw a perpendicular line at point B.

<Intersect Tool>4: Use the Intersection Tool to find point D, which is the intersection of the circle centered at B and the line perpendicular to AB.

<Line Tool>5: Draw the line AD.

<Line Tool>6: Draw a line from A to X. ✗

GT Solver



According to the steps given, the available tools and the suggested rationale, I am confident of the validity of the following steps:

<Line Tool>1: Correct.

<Circle Tool>2: Correct.

<Perpendicular Tool>3: Correct.

<Intersect Tool>4: Correct.

<Line Tool>5: Correct.

Please try to correct the solution according to my suggestions.

GT Validator



[...] <Circle Tool>6: Draw a circle with center A and radius AD.

<Intersect Tool>7: Mark the point of intersection of the circle with the line AB as X. X is the desired point. ✓

GT Solver



GPT4 Simulacra

I Human Study On Euclidea Dataset

While there is no concrete data on human performance - assuming an adequate level of geometric knowledge background - on the Euclidea challenge, [Macke et al. \(2021\)](#) describe these challenges as formidable even for well-prepared mathematicians, including International Mathematics Olympiad (IMO) participants, whereas [Wong et al. \(2022\)](#) characterizes the dataset as notably difficult. To validate these claims, we conducted an empirical study involving ten individuals with mathematical or computer science backgrounds, of graduate level, and experience with geometrical problem solving. The participants collaborated voluntarily and without any financial compensation.

Given the progression of the original Euclidea game, which requires solving preceding problems to advance, we decided to sample 20 problems across all packs. The outcomes revealed a success rate ranging from 20% to 35% - 5 to 7 correctly solved problems - with the average performance at 30.5%, indicating the high level of challenge presented by the tested geometry tasks. This finding underscores the domain's complexity. Table 12 contains the problems that were presented to participants, while Figure 5 showcases an example. Here is the list of the problems presented to the participants:

Name	Pack
Angle of 60	Alpha
Rhombus in Rectangle	Alpha
Angle of 30	Beta
Circle in Rhombus	Beta
Three equal segments v1	Gamma
Center of Quadrilateral	Gamma
Square Root of 3	Delta
Equilateral Triangle in Circle	Delta
Square in Square	Epsilon
Regular Hexagon	Epsilon
Given Angle Bisector	Zeta
Symmetry of Four Lines	Zeta
Annulus	Eta
Angle of 75	Eta
Egyptian Triangle	Theta
Torricelli Point	Theta
Harmonic Mean of Trapezoid Bases	Iota
Minimum Perimeter v2	Iota
Rotation 60	Kappa
Inner Tangent	Kappa

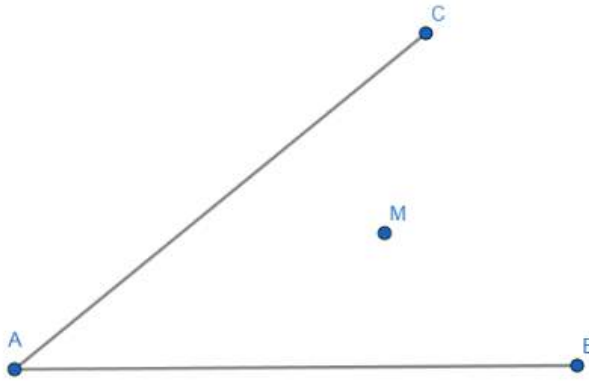
Table 12: Euclidea problems included in the human study.

Here is what a question in the questionnaire looked like:

Difficulty 3

Available Tools: Line, Circle, Intersection, Perp.Bisector, Angle Bisector

Given an angle ABC and a point M inside it, find points D on BA and E on BC and construct segments DM and ME such that $BD = DM = ME$.



Your answer

Figure 5—Example of a given problem in the human study: On top, the available tools are provided to the participant. Below, an image containing the annotated initial setup of the problem is provided. The participant is then asked to use the provided tools and respond with their solution.

J On the use of AI assistants

We disclose that AI writing assistant tools were used during the creation of this work. However, these tools were strictly used to refine, summarize, and check the accuracy of grammar and syntax. No AI assistant was involved in generating ideas or constructing arguments to support our perspectives or claims. All original thoughts and analyses are entirely our own.