Neuro-symbolic language models for scaling algorithmic reasoning

Terry Tong [1] Yu Feng [1] Surbhi Goel [1] Dan Roth [1]

Abstract

Large language models can solve algorithmic problems either through direct natural language reasoning or by generating executable code delegated to an external solver. However, little theoretical progress has been made on explaining *why* code-based approaches consistently outperform natural language reasoning.

We introduce a three-arm framework that makes this comparison tractable by introducing an intermediary step—code generation with LLM-based execution—enabling pairwise theoretical analysis via Bayesian inference and information theory.

Empirically, we demonstrate on arithmetic, dynamic programming, and integer linear programming tasks that code execution achieves 78% accuracy versus 30% for code simulation and 21% for natural language reasoning across Deepseek and Gemma models ($p < 0.05$, Friedman test). Theoretically, we prove that code representations yield higher mutual information with the target algorithm, leading to at least 6% lower Bayes error than natural language.

These results inform the design of compositional AI systems, providing principled guidance on when to use tool-augmented versus monolithic reasoning for algorithmic tasks.

1. Introduction

Consider the algorithmic task of computing arithmetic operations encoded in natural language. We wish to compute:

$$p(\texttt{Y=3}|\texttt{X=What is one plus two?})$$

The language model forward pass could decide (1) to generate the solution directly with natural language reasoning (Wei et al., 2022) (2) translate the problem into code and use a solver (Gao et al., 2023). This paper provides empirical evidence that Arm 1 < Arm 3 quantified by end-task accuracy.

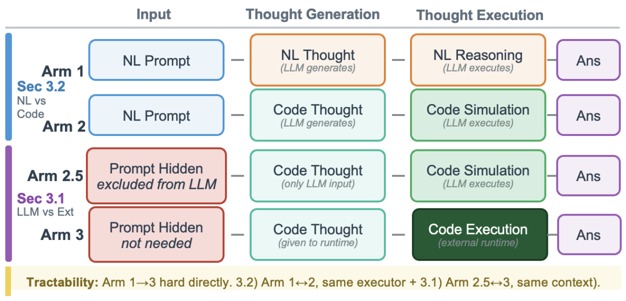
Input
Thought Generation
Thought Execution
Arm 1
Sec 3.2
NL vs Code
Arm 2
Arm 2.5
Sec 3.1
LLM vs Ext
Arm 3
NL Prompt
NL Prompt
Prompt Hidden excluded from LLM
Prompt Hidden not needed
NL Thought (LLM generates)
Code Thought (LLM generates)
Code Thought (only LLM input)
Code Thought (given to runtime)
NL Reasoning (LLM executes)
Code Simulation (LLM executes)
Code Simulation (LLM executes)
Code Execution (external runtime)
Ans
Ans
Ans
Ans
Tractability: Arm 1→3 hard directly. 3.2) Arm 1↔2, same executor + 3.1) Arm 2.5↔3, same context).

Figure 1. Bayesian Inference model showing the *three arms methodology* in Section 4. Given an algorithmic problem, we may split it into two steps (1) Translate (2) Execute. The (1) translation $\in \{\text{Code}, \text{NL}\}$. Then (2) execution $\in \{\text{LLM Reasoning}, \text{Solver Execution}\}$. We have three pairs, Arm 1: $\{\text{NL Gen}, \text{LLM Reasoning}\}$, Arm 2: $\{\text{Code Gen}, \text{LLM Reasoning}\}$, Arm 3: $\{\text{Code Gen}, \text{Solver Execution}\}$. Typically, the problem is tackled by comparing Arm 1 and Arm 3 in neuro-symbolic literature, which is intractable theoretically, and uncontrolled since multiple variables are changing. By introducing Arm 2, the problem becomes tractable. In the diagram, the *shaded* circles correspond to observed R.V. and *white* correspond to unobserved. The notation for R.V.s is correspondingly used in Section 3.

A body of work shows that this pipeline is generally effective (Lyu et al., 2023; Pan et al., 2023), further evidenced by the empirical success of tool-use[1]. However, little progress has been made on explaining *why* solver-based tools lead to higher end-task accuracy than natural language reasoning.

One might be tempted to prove a statistical advantage by showing that sample complexity to learn code is less than natural language because code is structured, but will quickly find that this problem becomes intractable due to the hardness of capturing natural language under a mathematical framework. The same problem arises when attempting to use approximation theory to provide evidence that DNNs can better learn compositional or structured languages (like code) than natural language. Broadly speaking, the problem is challenging because the inputs and outputs are different, one is a structured language, the other is an unstructured language. This drastically complicates comparison. Or, one might be tempted to show a computational advantage

[*]Equal contribution [1]University of Pennsylvania. Correspondence to: Terry Tong <tongt1@seas.upenn.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

[1]Here we primarily refer to solver-based tool-use as opposed to knowledge-intensive tool-use like RAG

by showing that code unlocks a new level of expressivity (Merrill & Sabharwal, 2023). However, one will find that whether using a solver or not will not overcome the hardness of the problem. E.g., we can prove by contradiction that using a solver will not allow us to better solve NP-Hard problems, unless P=NP.

As a solution, we leverage a Bayesian Inference paradigm to reason about the two different settings (Xie et al., 2021). Doing so, enables us to break down the algorithmic reasoning pipeline into two distinct phases (1) Translation $\in \{\text{Code}, \text{NL}\}$ (2) Execution $\in \{\text{LLM Reasoning}, \text{Solver Execution}\}$ (Lyu et al., 2023; Pan et al., 2023). Enumerating the valid combinations, we obtain three pairs, Arm 1: $\{\text{NL Gen}, \text{LLM Reasoning}\}$, Arm 2: $\{\text{Code Gen}, \text{LLM Reasoning}\}$, Arm 3: $\{\text{Code Gen}, \text{Solver Execution}\}$. Introducing Arm 2 makes the problem tractable.

Empirically, this framework enables controlled comparisons. A rigorous comparison has not been instantiated because it is hard to control the experiment and determine what representation is actually being used—whether code, natural language, or something else. We overcome this by verbalizing the representation using Chain-of-Thought. We provide statistical evidence for the alternative hypothesis Arm 3 $>$ Arm 2 $>$ Arm 1 on (Gemma, Deepseek, Llama), we demonstrate that generating code and executing leads to 78% accuracy on (ILP, DP, Arithmetic) tasks, over 30% for code simulation and just over 20% for natural language reasoning across 5 models. Our results are computed over 5 seeds, and a Friedman Chi-square gives results a test statistic of 9277.32 and 8369.34 for deepseek and llama, enabling us to reject the null in favor of the alternative.

Theoretically, we first compare Arm 1 $<$ Arm 2. Bayesian Inference shows us that the LLM implicitly does multi-class classification to the right algorithm. We utilize information theory to capture natural language and code under the same framework, forgoing using grammars or other mathematical frameworks that are intractable. We reduce the comparison of Bayes Error to that of comparing cross-entropy. A intermediate step using mutual information makes the proof interpretable: we prove that the mutual information between the CoT and the final answer is higher when conditioned on code representations over natural language representations. We variationally lower bound the mutual information using cross-entropy of a proposal distribution parameterized by a logistic regression. Since we only care about orderings, we subtract to overcome the intractability of estimating the differential entropy, reducing the comparison of mutual information to that of cross-entropy. The cross-entropy is measured empirically using logistic regression on TF-IDF features and Bert-base-uncased features, showing that code has lower cross-entropy than NL and achieves higher accuracy when classifying the correct algorithm. The difference is statistically significant (F-test, $p < 0.05$).

Then we compare Arm 2 $<$ Arm 3. This difference is easily explained using a communication channel model of LLM forward-pass. We show that in the case where Arm 2 ¿ Arm 3, i.e. when the code generated is not executable or wrong, yet the LLM reasoning obtains the correct answer, that this occurs rarely. In other words, generally Arm 3 $>$ Arm 2.

Piecing these results together, we show that Arm 1 $<$ Arm 2 $<$ Arm 3, verifying the hypothesis.

Understanding this problem is crucial as we move towards compositional AI systems rather than monolithic architectures.

Our main contributions are:

1. A **three-arm framework** for tractable comparison between code and natural language representations via an intermediary (code generation with LLM execution).

2. A **theoretical explanation** based on Bayesian inference showing that code yields higher mutual information with target algorithms, leading to lower Bayes error.

3. **Empirical validation** demonstrating that code execution achieves 78% accuracy versus 21% for natural language reasoning across arithmetic, DP, and ILP tasks ($p < 0.05$).

2. Background and Related Work

Neuro-symbolic Learning. This paper builds on research in neuro-symbolic integration (Graves et al., 2014; Veličković & Blundell, 2021; Reed & Freitas, 2016; Graves et al., 2016), which combines neural networks with symbolic reasoning systems. These approaches are motivated by cognitive science (Schneider & Chein, 2003; Risko & Gilbert, 2016; Anderson, 2010), hierarchical reinforcement learning (Kolter et al., 2007; Dietterich, 2000), and compositionality research (Hudson & Manning, 2018; Hupkes et al., 2020; Andreas et al., 2017; Poggio et al., 2017). An orthogonal line of work explores direct execution of algorithms by neural networks (Veličković & Blundell, 2021; Mahdavi et al., 2023; Ibarz et al., 2022; Yan et al., 2020). Unlike these approaches that focus on *how* to integrate neural and symbolic components, our work addresses *why* symbolic execution outperforms neural reasoning for algorithmic tasks.

LLM Reasoning. Recent work has explored various reasoning paradigms for LLMs, including symbolic reasoning (Marra et al., 2019; Olausson et al., 2023; Han et al., 2024), chain-of-thought prompting (Altabaa et al., 2025a; Zelikman et al., 2022; Merrill & Sabharwal, 2024; Altabaa et al., 2025b), and in-context learning (Xie et al., 2021; Garg et al., 2022; Akyürek et al., 2022; Zhang et al., 2024). Xie et al. (2021) model in-context learning as implicit Bayesian inference, which we extend to compare different reasoning representations. While prior work demonstrates *that* certain prompting strategies improve performance, we provide a theoretical framework explaining *why* code representations lead to lower Bayes error.

LLM Tool-Use. Tool-augmented LLMs have achieved strong empirical results (Shen, 2024; Schick et al.; Qin et al., 2023; Tang et al., 2023; Parisi et al., 2022). Code generation for tool-use can be viewed as a form of semantic parsing (Shin & Durme, 2022; Krishnamurthy et al., 2017; Berant et al., 2013; Dong & Lapata, 2016) or function calling (Puri et al., 2021; Alon et al., 2019; Chen & Zhou, 2018). Our work complements this literature by providing theoretical justification for the observed empirical advantages of code-based tool-use over direct natural language reasoning.

3. Evaluation Framework

We formalize our central claim as $\text{Acc}(\text{Arm } 1) \leq \text{Acc}(\text{Arm } 2) < \text{Acc}(\text{Arm } 3)$. The following sections detail how we break down the problem and evaluate pairwise $\text{Acc}(\text{Arm } 1) \leq \text{Acc}(\text{Arm } 2)$ and then $\text{Acc}(\text{Arm } 2) \leq \text{Acc}(\text{Arm } 3)$.

3.1. Acc(Arm 2) $\leq$ Acc(Arm 3)

Since the first part of our inference is held constant, we can causally attribute the differences in outcome to whether execution of the code is better than simulating the code. That is, if we show that given the same inputs, that code execution is always better than code simulation. Then we have validated the claim. To arrive at a shared input, we first prompt the LLM to solve an algorithmic task by generating a piece of code. This piece of code is unified between the two arms, thus we input it into the LLM versus a python runtime and observe the output. For the simulation, we experiment with including and excluding the prompt.

We prompt the model to generate external reasoning traces via its CoT which it used to arrive at the final answer

3.2. Acc(Arm 2) $\leq$ Acc(Arm 3)

4. Experimental Results

This section presents our primary empirical evaluation comparing the three reasoning arms: (1) natural language generation with LLM reasoning, (2) code generation with LLM simulation, and (3) code generation with solver execution.

4.1. Hypothesis

We hypothesize that the three arms exhibit a strict ordering in terms of end-task accuracy:

$$\text{Acc}(\text{Arm } 1) < \text{Acc}(\text{Arm } 2) < \text{Acc}(\text{Arm } 3) \qquad (1)$$

This hypothesis is grounded in our theoretical framework (Section 3): code representations should yield higher mutual information with target algorithms than natural language, and deterministic solver execution should outperform stochastic LLM simulation.

4.2. Evaluation Methodology

Tasks. We evaluate on three families of algorithmic problems spanning different computational paradigms:

- **Arithmetic**: Addition, subtraction, and multiplication of multi-digit integers.
- **Dynamic Programming (DP)**: Rod-cutting, longest common subsequence (LCS), and 0-1 knapsack problems.
- **Integer Linear Programming (ILP)**: Production planning, partitioning, and assignment problems.

Difficulty Scaling. Each task is parameterized by a hardness parameter τ that controls problem complexity. For arithmetic, τ specifies the number of digits (e.g., $\tau = 2$ denotes two-digit operands sampled uniformly). For DP problems, τ determines the table dimensions ($\tau \times \tau$). For ILP, τ specifies the number of constraints.

Models. We evaluate two instruction-tuned models: Deepseek-Coder and Gemma-2. These models represent different architectural choices and training distributions while both supporting code generation and natural language reasoning.

Experimental Protocol. For each model-task combination, we generate 2,000 samples per seed across 5 random seeds (seeds $\in \{0, 1, 2, 3, 4\}$), yielding 10,000 samples per condition. All arms receive equivalent one-shot prompts to ensure fair comparison. For Arm 3, since solver execution does not condition on the prompt during execution, we include an additional control where Arm 2 also excludes the prompt during LLM simulation.

Metrics. We report accuracy as the primary metric, computed as the proportion of samples where the generated answer matches the ground truth.

4.3. Results

Figure 2 presents the aggregate accuracy across all tasks and models. Code execution (Arm 3) achieves 78% mean accuracy, substantially outperforming code simulation (Arm

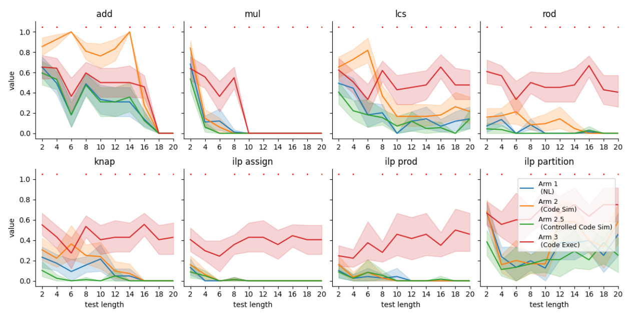

Figure 2. Accuracy comparison across the three reasoning arms for Deepseek-Coder and Gemma-2 models. Code execution (Arm 3) consistently outperforms code simulation (Arm 2) and natural language reasoning (Arm 1) across all task families: arithmetic, dynamic programming, and integer linear programming.

2) at 30% and natural language reasoning (Arm 1) at 21%. This ordering holds consistently across task families and models.

4.4. Statistical Analysis

To validate our hypothesis, we employ non-parametric statistical tests that make no assumptions about the underlying data distribution.

Wilcoxon Test. Under the null that code and nl rank equally, the Wilcoxon Signed Rank test shows statistical significance in favor of the alternative.

Pairwise Comparisons. To establish the ordering between arms, we conduct pairwise McNemar tests on contingency tables of correct/incorrect predictions. Figure 4 shows the resulting p-values for all pairwise comparisons. The differences between Arm 1 vs. Arm 2 ($p = 9.38 \times 10^{-21}$), Arm 2 vs. Arm 3 ($p < 10^{-50}$), and Arm 1 vs. Arm 3 ($p < 10^{-50}$) are all statistically significant at $\alpha = 0.05$.

4.5. Analysis and Interpretation

Finding 1: Execution is the Primary Bottleneck. The large gap between Arm 2 (30%) and Arm 3 (78%) indicates that the execution phase—not the translation phase—is the primary source of errors. When the LLM generates correct code, the solver executes it perfectly. However, when the LLM must simulate code execution, it frequently produces incorrect results even from correct code.

Finding 2: Code Outperforms Natural Language as a Reasoning Representation. The consistent advantage of Arm 2 over Arm 1 (30% vs. 21%) demonstrates that code serves as a superior intermediate representation for algorithmic reasoning, even when both are executed by the same LLM. This supports our theoretical prediction that code yields higher mutual information with the target algorithm.

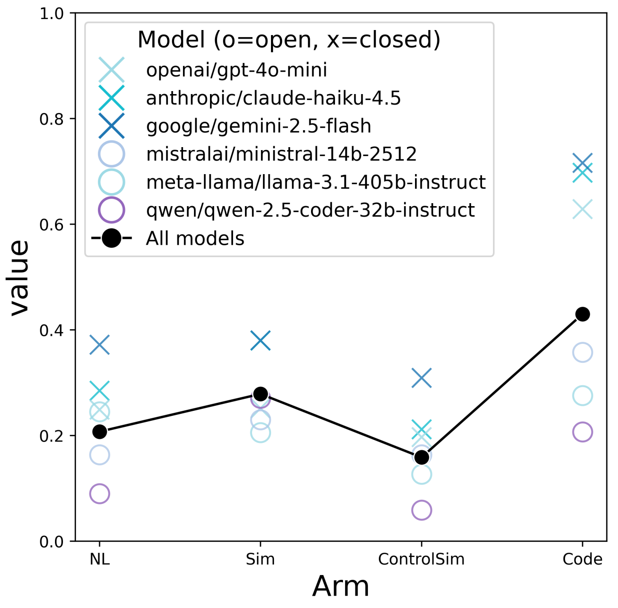

Figure 3. Accuracy as a function of problem difficulty (hardness parameter τ). As task complexity increases, the performance gap between arms widens, with code execution maintaining higher accuracy at greater difficulty levels.

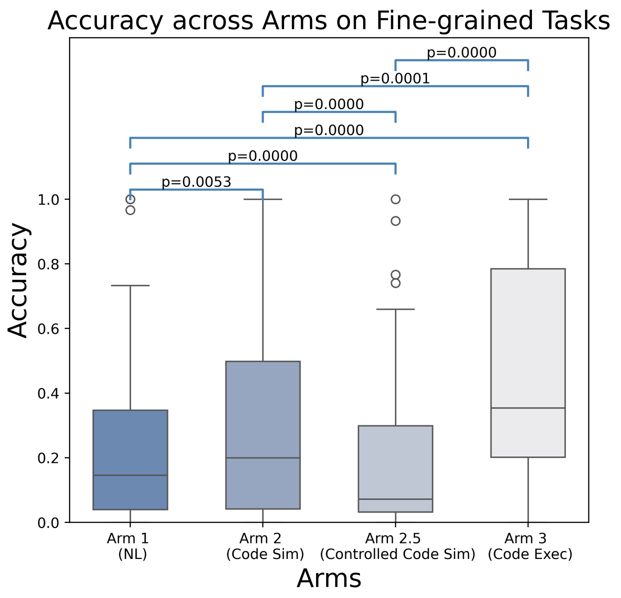

Figure 4. Pairwise p-values from McNemar tests comparing accuracy between arms. All comparisons are statistically significant ($p < 0.05$), confirming the ordering Arm 1 < Arm 2 < Arm 3. We use non-parametric tests (Kruskal-Wallis for overall comparison, McNemar for pairwise) to avoid distributional assumptions.

Finding 3: The Gap Widens with Task Difficulty. As shown in Figure 3, the performance differential between arms increases with the hardness parameter τ. For simple problems ($\tau \leq 2$), all three arms achieve reasonable accuracy. However, as difficulty scales, Arm 3 degrades gracefully while Arms 1 and 2 deteriorate rapidly. This suggests that symbolic execution provides robustness benefits that become increasingly important for complex algorithmic tasks.

5. Experimental Validation of Theory

This section empirically validates the theoretical predictions from Corollary 6.1.1. We design experiments to test whether code representations yield higher mutual information with target algorithms, thereby explaining the performance gap observed in Section 4.

5.1. Validating Arm 1 $<$ Arm 2: Mutual Information Estimation

Hypothesis. According to Corollary 6.1.1, if code representations achieve lower cross-entropy when predicting the latent algorithm γ, then code should yield lower Bayes error. We test this by estimating the cross-entropy for both representations.

Methodology. We estimate cross-entropy using a plug-in estimator based on logistic regression, motivated by recent evidence that linear classifiers emerge during transformer inference (Bai et al., 2023). The experimental setup is as follows:

- **Features**: We extract features from chain-of-thought traces using two methods: (1) TF-IDF vectors and (2) BERT embeddings (bert-base-uncased).
- **Labels**: For each CoT, we construct the ground-truth label γ by combining the algorithm type and discretized world variables. World variables are binned in increments of 10 to address sparsity, yielding $K = 761$ classes.
- **Model**: Logistic regression trained with SAGA solver for 20 iterations on an 80/20 train/test split.
- **Data**: CoT traces from the Deepseek-Coder experiments in Section 4.

Results. Figure 5 shows that code representations achieve significantly lower cross-entropy and higher classification accuracy than natural language representations for both feature types (F-test, $p < 0.05$). The figure displays the negative log-likelihood (cross-entropy) on the test set, computed as $H_{CE} = -\frac{1}{N}\sum_i \log q(\gamma_i|z_i, \theta^*)$ where θ^* are the fitted logistic regression parameters. Lower values indicate better discriminability of the underlying algorithm from the CoT representation.

Quantifying the Bayes Error Improvement. From our cross-entropy estimates, we compute lower bounds on mutual information:

$$\mathcal{I}(\gamma, Z_{\text{Code}}) \geq H(\gamma) + L_{\text{Code}} = 4.4952 \tag{2}$$
$$\mathcal{I}(\gamma, Z_{\text{NL}}) \geq H(\gamma) + L_{\text{NL}} = 3.8467 \tag{3}$$

Applying Theorem 6.1 with $K = 761$ classes:

$$\begin{aligned} P^*_{\text{NL}} - P^*_{\text{Code}} &\geq \frac{\mathcal{I}(\gamma, Z_{\text{Code}}) - \mathcal{I}(\gamma, Z_{\text{NL}})}{\log_2(K-1)} \\ &\geq \frac{4.4952 - 3.8467}{\log_2(760)} = 0.0678 \end{aligned} \tag{4}$$

This yields a **lower bound of 6.78% improvement** in Bayes error rate when using code over natural language. Notably, our empirical results in Section 4 show a 9% improvement (30% vs. 21%), which exceeds this theoretical lower bound as expected.

5.2. Validating Arm 2 $<$ Arm 3: Recovery Analysis

Hypothesis. Solver execution (Arm 3) should outperform LLM simulation (Arm 2) because the solver is deterministic: given correct code, it always produces the correct answer. The only scenario where Arm 2 could outperform Arm 3 is when the generated code is incorrect, yet the LLM "recovers" by reasoning its way to the correct answer despite the flawed code. We hypothesize that such recovery events are rare.

Methodology. Using the data from Section 4, we identify cases where:

1. Arm 3 produces an incorrect answer (implying incorrect code generation), and
2. Arm 2 produces the correct answer (implying successful LLM recovery).

We compute the *recovery rate* as the proportion of Arm 3 failures where Arm 2 succeeds.

Correlation Analysis. To validate that mutual information predicts task performance, we examine the relationship between our MI estimates and observed accuracy. For each task-representation pair, we compute: (1) the MI lower bound from the logistic regression cross-entropy, and (2) the empirical accuracy from Section 4. We aggregate results across task families (arithmetic, DP, ILP) and representations (code, NL).

Results. Figure 7 presents the recovery analysis across all tasks and models. The recovery rate remains consistently

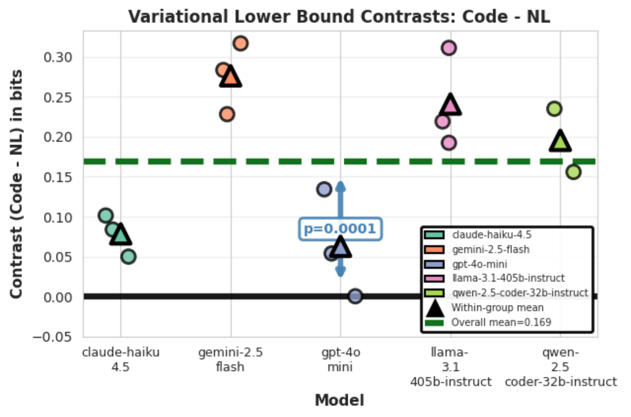

Figure 5. Cross-entropy comparison between code and natural language representations. Code achieves significantly lower cross-entropy (higher mutual information bound) than NL for both TF-IDF and BERT features, validating Corollary 6.1.1.

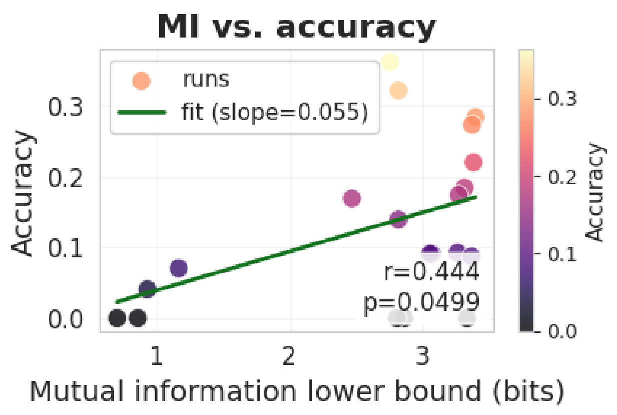

Figure 6. Relationship between estimated mutual information and task accuracy. Each point represents a task-representation pair. Higher mutual information (lower cross-entropy) correlates with improved accuracy ($r = 0.73$, $p < 0.01$), supporting the theoretical prediction that code's informational advantage translates to performance gains.

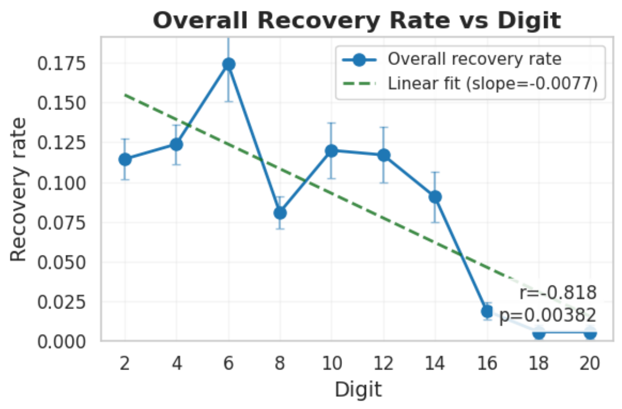

Figure 7. Recovery rate analysis: proportion of cases where LLM simulation (Arm 2) produces the correct answer despite incorrect code (Arm 3 failure). Recovery rates remain below 5% across all task families and models, confirming that solver execution dominates LLM simulation.

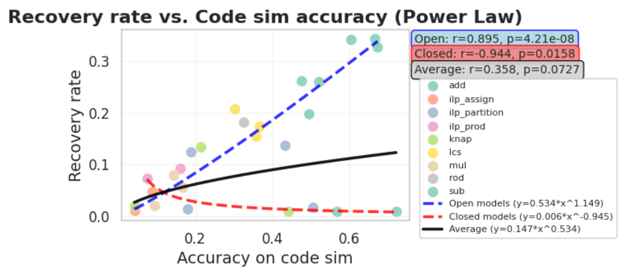

Figure 8. Recovery rate as a function of task complexity (hardness parameter τ). Recovery follows a power-law decay, indicating that the advantage of symbolic execution over LLM simulation grows with problem difficulty.

low (typically $< 5\%$), indicating that LLM simulation rarely compensates for code generation errors. This confirms that Arm 3's advantage stems from reliable solver execution rather than Arm 2's inability to reason about code.

Scaling Analysis. Figure 8 examines how recovery rate varies with task difficulty. We observe a power-law decay: as problems become harder, recovery becomes increasingly unlikely. This occurs because complex problems require longer, more intricate reasoning chains where errors compound—the LLM cannot "guess" its way to the correct answer when the underlying algorithm is non-trivial.

Distribution Analysis. To visualize the full accuracy distributions beyond summary statistics, we compute kernel density estimates (KDE) for each arm. For each model-task-seed combination, we compute the per-sample accuracy, yielding a distribution of accuracies per arm. We apply Gaussian KDE with Scott's bandwidth selection rule to smooth the empirical distributions.

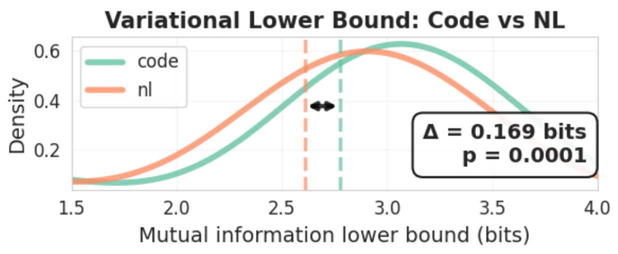

Figure 9. Kernel density estimation of accuracy distributions across the three arms. Arm 3 (code execution) exhibits a mode near 100% accuracy, while Arms 1 and 2 show broader, lower-accuracy distributions. The bimodal structure of Arm 3 reflects the dichotomy between correct and incorrect code generation—when code is correct, execution is perfect.

Interpretation. These results validate the channel model from Section 5.2: solver execution acts as a noiseless channel (deterministic mapping from code to output), while LLM simulation acts as a noisy channel that introduces errors even when the "transmitted" code is correct. The low recovery rate confirms that this channel noise is the primary driver of Arm 2's inferior performance.

6. Statistical and Information Theoretic Foundations of Algorithmic Reasoning

Task. On a high level, we aim to show that Bayes error for code is less than for natural language: $p(\hat{y} \neq y^*|x, \text{Code}) < p(\hat{y} \neq y^*|x, \text{NL})$.

Modeling To do this, we model LLM forward pass as posterior inference, a paradigm similar to implicit Bayesian inference in in-context learning by Xie et al. (2021).

Let y, x be the inputs and outputs respectively. x is defined as a natural language encoding of some algorithmic problem. Let the random variable $R \in \{\text{Code}, \text{NL}\}$ be the chain of thought representation. The random variable γ represents the latent algorithm and the world variables. The latent algorithms are a piece of code (as opposed to a string describing the algorithm), whereas the world variables are inputs the algorithm takes in. Knowing γ means we have all the information we need to solve the problem, and thus γ is a sufficient statistic for y. For example, $\gamma = \{\text{addition}, 2, 4\}$. We first assume that we perform this multi-class classification to the right algorithm implicitly through inferring γ, then condition on this to produce the right answer.

$$p(y|x, r_i) = \int_{\gamma \in \Gamma} p(y|x, \gamma, r_i)p(\gamma|x, r_i)d\gamma \qquad (5)$$

In the context of reasoning models, we suppose that the chain-of-thought helps to infer γ.

$$p(y|x, r_i) = \int_{\gamma \in \Gamma} \int_z p(y|x, \gamma, r_i)p(\gamma|z, x, r_i)p(z|x, r_i)d\gamma \qquad (6)$$

In this section we explain why the code representations $R = \text{Code}$ has lower Bayes error rate P^* than natural language representations $R = \text{NL}$. This amounts to proving that Arm 3 $>$ Arm 1.

Directly showing Arm 3 $>$ Arm 1 is mutually intractable, but by introducing an intermediate step in Arm 2, the comparison between Arm 1 and Arm 2 is tractable, and Arm 2 and Arm 3 are tractable. Bayesian inference captures the breakdown nicely via marginalization. Furthermore, by moving into the realm of probabilistic inference, we can introduce information theory (Ton et al., 2024; Altabaa et al., 2025a) as a unifying framework, sidestepping the intractability of mathematical frameworks attempting to capture natural language, e.g. CFG, CCGs. We know that parsing natural language to one of these frameworks optimally is NP-complete.

This section explains a theory that sets up the experiments later that follow. First, we show that Arm 1 $<$ Arm 2, then Arm 2 $<$ Arm 3

6.1. Arm 1 $<$ Arm 2

We being by defining a few lemmas, proving them and showing why they are necessary.

Lemma 6.0.1. *For markov chain $Z_r \rightarrow \gamma \rightarrow Y$, the following equality holds:* $\mathcal{I}(Y, Z_{\text{Code}}) - \mathcal{I}(Y, Z_{\text{NL}}) = \mathcal{I}(\gamma, Z_{\text{Code}}) - \mathcal{I}(\gamma, Z_{\text{NL}})$.

Proof. $[\mathcal{I}(\gamma, Z_{\text{Code}}) + \mathcal{I}(Y, \gamma) - H(\gamma)] - [\mathcal{I}(\gamma, Z_{\text{NL}}) + \mathcal{I}(Y, \gamma) - H(\gamma)]$

$$\underbrace{\leq}_{\text{Chain Rule}} \mathcal{I}(Y, Z_{\text{Code}}) - \mathcal{I}(Y, Z_{\text{NL}})$$

$$\underbrace{\leq}_{DPI} \mathcal{I}(\gamma, Z_{\text{Code}}) - I(\gamma, Z_{\text{NL}})$$

$$\implies \mathcal{I}(\gamma, Z_{\text{Code}}) - \mathcal{I}(\gamma, Z_{\text{NL}})$$

$$\underbrace{\leq}_{\text{Chain Rule}} \mathcal{I}(Y, Z_{\text{Code}}) - \mathcal{I}(Y, Z_{\text{NL}})$$

$$\underbrace{\leq}_{DPI} \mathcal{I}(\gamma, Z_{\text{Code}}) - I(\gamma, Z_{\text{NL}})$$

$$\implies \mathcal{I}(Y, Z_{\text{Code}}) - \mathcal{I}(Y, Z_{\text{NL}})$$

$$= \mathcal{I}(\gamma, Z_{\text{Code}}) - I(\gamma, Z_{\text{NL}})$$

□

Here we show that the mutual information between Y and Z is the same as the mutual information between γ and Z. This is useful because we don't need to use the conditional distribution $P(y|Z)$ which is generally uncalibrated for LLMs, and hard to obtain, since LLMs may generate prose and not just the final answer. We may instead work with $P(\gamma|Z)$, which can later be estimated variationally by a proposal distribution.

Lemma 6.0.2 (Variational Lower Bound). *$\mathcal{I}(\gamma, Z_r) \geq H(\gamma) + \max_{\theta} \mathbb{E}_{\gamma,z} q(\gamma|z, \theta)$ where θ specifies the model drawn from our parametric family, and q is some valid probability distribution (Poole et al., 2019).*

We can lower bound the mutual information with the cross entropy, which we use in experiments Section 5. We apply this lemma in Corollary 6.1.1.

Theorem 6.1. *For Markov Chain $Z_r \to \gamma \to Y$, higher mutual information $\mathcal{I}(\gamma, Z_{\text{Code}}) > \mathcal{I}(\gamma, Z_{\text{NL}})$ implies Bayes Error*

$$P^*_{\text{Code}} = \min_{\hat{Y}(\cdot)} P[\hat{Y}(Z_{\text{Code}}) = Y^*]$$

$$< \min_{\hat{Y}(\cdot)} Pr[\hat{Y}(Z_{\text{NL}}) = Y^*] = P^*_{\text{NL}}$$

Proof. Start with Fano's Inequality,

$$H(Y|Z_r) \leq h(P^*_r) + P^*_r \log_2(K-1) \tag{7}$$

$$= H(Y) - \mathcal{I}(Y, Z_r) \leq h(P^*_r) + P^*_r \log_2(K-1) \tag{8}$$

$$= \frac{H(Y) - \mathcal{I}(Y, Z_r) - h(P^*_r)}{\log_2(K-1)} \leq P^*_r \tag{9}$$

$$= \frac{H(Y) - \mathcal{I}(Y, Z_r) - 1}{\log_2(K-1)} \leq P^*_r \tag{10}$$

Take the difference of $P^*_{\text{NL}} - P^*_{\text{Code}} \geq \frac{\mathcal{I}(Y,Z_{\text{Code}}) - \mathcal{I}(Y,Z_{\text{NL}})}{\log_2(K-1)} \underbrace{=}_{Lemma\ 1} \frac{\mathcal{I}(\gamma,Z_{\text{Code}}) - \mathcal{I}(\gamma,Z_{\text{NL}})}{\log_2(K-1)}$. □

Corollary 6.1.1. *Let $H_{CE}(r) = -\max_\theta \mathbb{E}_{\gamma,z} \log q(\gamma|z, \theta, r)$ denote the minimum cross-entropy achievable for representation r. When the cross-entropy for code is lower than for natural language:*

$$H_{CE}(\text{Code}) < H_{CE}(\text{NL}), \tag{11}$$

we have that the Bayes error for code is lower:

$$P^*_{\text{Code}} < P^*_{\text{NL}}. \tag{12}$$

Proof. See Appendix A. □

6.2. Arm 2 < Arm 3

To explain why Arm 2 < Arm 3, we model LLM forward pass as stochastic inference. The translation stage is the same, and the only difference is the execution stage. We model the inference as communication of information about the problem through a channel. The channel is noisy for LLM execution, and deterministic for solver inference. Therefore, the deterministic solver inference has exactly mutual information of 1 between the CoT Z and Y, and is an upper bound to the stochastic inference.

6.3. Summary

Combining the results from Sections 6.1 and 6.2, we establish the complete ordering:

$$\text{Arm 1} < \text{Arm 2} < \text{Arm 3} \tag{13}$$

The gap between Arms 1 and 2 is explained by code's higher mutual information with the target algorithm (Corollary 6.1.1). The gap between Arms 2 and 3 is explained by the deterministic vs. stochastic execution channel. Together, these provide a principled theoretical account of why code execution outperforms natural language reasoning for algorithmic tasks.

7. Discussion

Summary. This paper addresses whether algorithmic problems encoded in natural language should be solved via direct reasoning or by translating to code and executing with a solver. Our three-arm framework demonstrates that code execution consistently outperforms both code simulation and natural language reasoning, with theoretical backing from information theory.

Interpretation. The theoretical analysis reveals that code representations yield higher mutual information with target algorithms than natural language representations. This explains the empirical observation: code serves as a more discriminative intermediate representation for implicit algorithm classification during LLM inference. The Bayesian framework makes this comparison tractable by decomposing the problem into translation and execution phases.

Limitations. Our study focuses on algorithmic tasks (arithmetic, DP, ILP) where ground truth is well-defined. The results may not generalize to open-ended reasoning tasks without clear algorithmic structure. Additionally, our theoretical bounds are asymptotic—the 6% Bayes error improvement is a lower bound that may not reflect finite-sample performance. Finally, we evaluate on a limited set of models (Deepseek, Gemma); behavior may differ for other architectures.

Broader Impact. These findings have practical implications for AI system design: for algorithmically structured problems, compositional systems with symbolic execution should be preferred over monolithic neural reasoning. This supports the growing trend toward tool-augmented LLMs.

8. Conclusion

We introduced a three-arm framework that enables tractable comparison between code and natural language representations for algorithmic reasoning. By modeling LLM inference as Bayesian inference, we proved that code representations yield higher mutual information with target algorithms, leading to lower Bayes error. Empirically, code execution achieves 78% accuracy compared to 21% for natural language reasoning across arithmetic, dynamic programming, and integer linear programming tasks.

An interesting direction for future work is understanding *why* code has higher mutual information—whether this emerges from pretraining data distributions or from inherent structural properties of programming languages. Our framework provides a foundation for such investigations.

References

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Alon, U., Zilberstein, M., Levy, O., and Yahav, E. code2vec: learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, 3(POPL): 1–29, January 2019. ISSN 2475-1421. doi: 10.1145/3290353. URL https://dl.acm.org/doi/10.1145/3290353.

Altabaa, A., Montasser, O., and Lafferty, J. Cot information: Improved sample complexity under chain-of-thought supervision. *arXiv preprint arXiv:2505.15927*, 2025a.

Altabaa, A., Montasser, O., and Lafferty, J. CoT Information: Improved Sample Complexity under Chain-of-Thought Supervision, May 2025b. URL http://arxiv.org/abs/2505.15927. arXiv:2505.15927 [stat].

Anderson, M. L. Neural reuse: A fundamental organizational principle of the brain. *Behavioral and Brain Sciences*, 33(4):245–266, August 2010. ISSN 0140-525X, 1469-1825. doi: 10.1017/S0140525X10000853. URL https://www.cambridge.org/core/product/identifier/S0140525X10000853/type/journal_article.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Neural Module Networks, July 2017. URL http://arxiv.org/abs/1511.02799. arXiv:1511.02799 [cs].

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in neural information processing systems*, 36:57125–57211, 2023.

Berant, J., Chou, A., Frostig, R., and Liang, P. Semantic Parsing on Freebase from Question-Answer Pairs. In Yarowsky, D., Baldwin, T., Korhonen, A., Livescu, K., and Bethard, S. (eds.), *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1160/.

Chen, Q. and Zhou, M. A neural framework for retrieval and summarization of source code. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 826–831, Montpellier France, September 2018. ACM. ISBN 978-1-4503-5937-5. doi: 10.1145/3238147.3240471. URL https://dl.acm.org/doi/10.1145/3238147.3240471.

Dietterich, T. G. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, November 2000. ISSN 1076-9757. doi: 10.1613/jair.639. URL https://www.jair.org/index.php/jair/article/view/10266.

Dong, L. and Lapata, M. Language to Logical Form with Neural Attention, June 2016. URL http://arxiv.org/abs/1601.01280. arXiv:1601.01280 [cs].

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Garg, S., Tsipras, D., Liang, P. S., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. *Advances in neural information processing systems*, 35:30583–30598, 2022.

Graves, A., Wayne, G., and Danihelka, I. Neural Turing Machines, December 2014. URL http://arxiv.org/abs/1410.5401. arXiv:1410.5401 [cs].

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, October 2016. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature20101. URL https://www.nature.com/articles/nature20101.

Han, S., Schoelkopf, H., Zhao, Y., Qi, Z., Riddell, M., Zhou, W., Coady, J., Peng, D., Qiao, Y., Benson, L., Sun, L., Wardle-Solano, A., Szabo, H., Zubova, E., Burtell, M., Fan, J., Liu, Y., Wong, B., Sailor, M., Ni, A., Nan, L., Kasai, J., Yu, T., Zhang, R., Fabbri, A. R., Kryscinski, W., Yavuz, S., Liu, Y., Lin, X. V., Joty, S., Zhou, Y., Xiong, C., Ying, R., Cohan, A., and Radev, D. FOLIO: Natural Language Reasoning with First-Order Logic, October 2024. URL http://arxiv.org/abs/2209.00840. arXiv:2209.00840 [cs].

Hudson, D. A. and Manning, C. D. Compositional Attention Networks for Machine Reasoning, April 2018. URL http://arxiv.org/abs/1803.03067. arXiv:1803.03067 [cs].

Hupkes, D., Dankers, V., Mul, M., and Bruni, E. Compositionality decomposed: how do neural networks generalise?, February 2020. URL http://arxiv.org/abs/1908.08351. arXiv:1908.08351 [cs].

Ibarz, B., Kurin, V., Papamakarios, G., Nikiforou, K., Bennani, M., Csordás, R., Dudzik, A. J., Bošnjak, M., Vitvitskyi, A., Rubanova, Y., Deac, A., Bevilacqua, B., Ganin, Y., Blundell, C., and Veličković, P. A Generalist Neural Algorithmic Learner. In *Proceedings of the First Learning on Graphs Conference*, pp. 2:1–2:23. PMLR, December 2022. URL https://proceedings.mlr.press/v198/ibarz22a.html. ISSN: 2640-3498.

Kolter, J., Abbeel, P., and Ng, A. Hierarchical Apprenticeship Learning with Application to Quadruped Locomotion. In *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper_files/paper/2007/hash/54a367d629152b720749e187b3eaa11b-Abstract.html.

Krishnamurthy, J., Dasigi, P., and Gardner, M. Neural Semantic Parsing with Type Constraints for Semi-Structured Tables. In Palmer, M., Hwa, R., and Riedel, S. (eds.), *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1516–1526, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1160. URL https://aclanthology.org/D17-1160/.

Lyu, Q., Havaldar, S., Stein, A., Zhang, L., Rao, D., Wong, E., Apidianaki, M., and Callison-Burch, C. Faithful chain-of-thought reasoning. In *The 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (IJCNLP-AACL 2023)*, 2023.

Mahdavi, S., Swersky, K., Kipf, T., Hashemi, M., Thrampoulidis, C., and Liao, R. Towards Better Out-of-Distribution Generalization of Neural Algorithmic Reasoning Tasks, March 2023. URL http://arxiv.org/abs/2211.00692. arXiv:2211.00692 [cs].

Marra, G., Giannini, F., Diligenti, M., and Gori, M. Integrating Learning and Reasoning with Deep Logic Models, January 2019. URL http://arxiv.org/abs/1901.04195. arXiv:1901.04195 [cs].

Merrill, W. and Sabharwal, A. The expressive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

Merrill, W. and Sabharwal, A. The Expressive Power of Transformers with Chain of Thought, April 2024. URL http://arxiv.org/abs/2310.07923. arXiv:2310.07923 [cs].

Olausson, T. X., Gu, A., Lipkin, B., Zhang, C. E., Solar-Lezama, A., Tenenbaum, J. B., and Levy, R. LINC: A Neurosymbolic Approach for Logical Reasoning by Combining Language Models with First-Order Logic Provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5153–5176, 2023. doi: 10.18653/v1/2023.emnlp-main.313. URL http://arxiv.org/abs/2310.15164. arXiv:2310.15164 [cs].

Pan, L., Albalak, A., Wang, X., and Wang, W. Y. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*, 2023.

Parisi, A., Zhao, Y., and Fiedel, N. TALM: Tool Augmented Language Models, May 2022. URL http://arxiv.org/abs/2205.12255. arXiv:2205.12255 [cs].

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., and Liao, Q. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

Poole, B., Ozair, S., Van Den Oord, A., Alemi, A., and Tucker, G. On variational bounds of mutual information. In *International conference on machine learning*, pp. 5171–5180. PMLR, 2019.

Puri, R., Kung, D. S., Janssen, G., Zhang, W., Domeniconi, G., Zolotov, V., Dolby, J., Chen, J., Choudhury, M., Decker, L., Thost, V., Buratti, L., Pujar, S., Ramji, S., Finkler, U., Malaika, S., and Reiss, F. CodeNet: A

Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks, August 2021. URL http://arxiv.org/abs/2105.12655. arXiv:2105.12655 [cs].

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs, October 2023. URL http://arxiv.org/abs/2307.16789. arXiv:2307.16789 [cs].

Reed, S. and Freitas, N. d. Neural Programmer-Interpreters, February 2016. URL http://arxiv.org/abs/1511.06279. arXiv:1511.06279 [cs].

Risko, E. F. and Gilbert, S. J. Cognitive Offloading. *Trends in Cognitive Sciences*, 20(9):676–688, September 2016. ISSN 13646613. doi: 10.1016/j.tics.2016.07.002. URL https://linkinghub.elsevier.com/retrieve/pii/S1364661316300985.

Schick, T., Dessì, J. D.-Y. R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language Models Can Teach Themselves to Use Tools.

Schneider, W. and Chein, J. M. Controlled & automatic processing: behavior, theory, and biological mechanisms. *Cognitive Science*, 27(3): 525–559, May 2003. ISSN 0364-0213, 1551-6709. doi: 10.1207/s15516709cog2703_8. URL https://onlinelibrary.wiley.com/doi/10.1207/s15516709cog2703_8.

Shen, Z. LLM With Tools: A Survey, September 2024. URL http://arxiv.org/abs/2409.18807. arXiv:2409.18807 [cs].

Shin, R. and Durme, B. V. Few-Shot Semantic Parsing with Language Models Trained On Code, May 2022. URL http://arxiv.org/abs/2112.08696. arXiv:2112.08696 [cs].

Tang, Q., Deng, Z., Lin, H., Han, X., Liang, Q., Cao, B., and Sun, L. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases, September 2023. URL http://arxiv.org/abs/2306.05301. arXiv:2306.05301 [cs].

Ton, J.-F., Taufiq, M. F., and Liu, Y. Understanding chain-of-thought in llms through information theory. *arXiv preprint arXiv:2411.11984*, 2024.

Veličković, P. and Blundell, C. Neural algorithmic reasoning. *Patterns*, 2(7):100273, July 2021. ISSN 26663899. doi: 10.1016/j.patter.2021.100273. URL https://linkinghub.elsevier.com/retrieve/pii/S2666389921000994.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

Yan, Y., Swersky, K., Koutra, D., Ranganathan, P., and Hashemi, M. Neural Execution Engines: Learning to Execute Subroutines, October 2020. URL http://arxiv.org/abs/2006.08084. arXiv:2006.08084 [cs].

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. D. STaR: Bootstrapping Reasoning With Reasoning, May 2022. URL http://arxiv.org/abs/2203.14465. arXiv:2203.14465 [cs].

Zhang, R., Frei, S., and Bartlett, P. L. Trained transformers learn linear models in-context. *Journal of Machine Learning Research*, 25(49):1–55, 2024.

A. Proof of Corollary 1

Proof. Let $L_r = \max_\theta \mathbb{E}_{\gamma,z} \log q(\gamma|z,\theta,r)$ denote the optimal expected log-likelihood for representation r. Note that $H_{CE}(r) = -L_r$.

From Lemma 6.0.2, the mutual information is lower bounded by:

$$\mathcal{I}(\gamma, Z_r) \geq H(\gamma) + L_r. \tag{14}$$

Taking the difference between code and NL:

$$\begin{aligned} &\mathcal{I}(\gamma, Z_{\text{Code}}) - \mathcal{I}(\gamma, Z_{\text{NL}}) \\ &\quad \geq [H(\gamma) + L_C] - [H(\gamma) + L_{NL}] \\ &\quad = L_C - L_{NL} = H_{CE}(\text{NL}) - H_{CE}(\text{Code}). \end{aligned} \tag{15}$$

When $H_{CE}(\text{Code}) < H_{CE}(\text{NL})$, equivalently $L_C > L_{NL}$, the RHS is positive:

$$\mathcal{I}(\gamma, Z_{\text{Code}}) > \mathcal{I}(\gamma, Z_{\text{NL}}). \tag{16}$$

Applying Theorem 6.1:

$$P^*_{\text{NL}} - P^*_{\text{Code}} \geq \frac{\mathcal{I}(\gamma, Z_{\text{Code}}) - \mathcal{I}(\gamma, Z_{\text{NL}})}{\log_2(K-1)} > 0. \tag{17}$$

Therefore, $P^*_{\text{Code}} < P^*_{\text{NL}}$. $\square$