

An Explanation In Support Of Neuro-Symbolic Language Models for Scaling Algorithmic Reasoning

Terry Tong¹ Yu Feng¹ Surbhi Goel¹ Dan Roth¹

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. Comparing NL reasoning and solver-based pipelines directly is ill-posed: they differ simultaneously in representation space and execution mechanism. 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. Across 44 different algorithmic tasks and 6 models, we observe that there is statistically significant gap ($p < 0.001$) between code and natural language reasoning ($> 25\%$). We find that code representations scale better, with $4.01 \times$ odds of correctness compared to natural language. Under causal intervention experiments, we identify natural language reasoning as a projection of deeper underlying algorithmic representations. Using this insight, we leverage Bayesian Inference and the Blackwell Dominance Principle to prove that the code execution route achieves lower Bayes Risk than natural language reasoning route. These results inform the design of compositional AI systems, providing principled guidance on when to use tool-augmented versus monolithic reasoning for algorithmic tasks. Our framework offers a unified perspective on the tool-use versus direct-reasoning tradeoff.

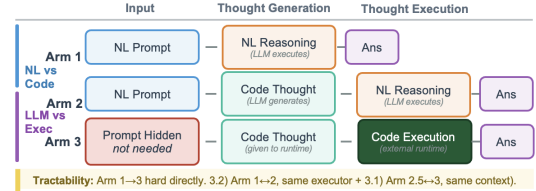


Figure 1. Three-arm Bayesian framework. We decompose algorithmic reasoning into: (1) *Translation* into NL or code, and (2) *Execution* via LLM or solver. This yields three arms: **Arm 1** (NL + LLM), **Arm 2** (Code + LLM simulation), **Arm 3** (Code + solver). Prior work compares only Arm 1 vs. Arm 3, confounding translation and execution. Our Arm 2 isolates these factors. Shaded nodes: observed; white: latent.

1. Introduction

Large language models (LLMs) have demonstrated increasingly strong natural-language (NL) reasoning capabilities (Wei et al., 2022). In parallel, LLM-based agentic systems advocate tool use, where LLMs invoke external solvers to support reasoning and execution (Gao et al., 2023). Recent works (Lyu et al., 2023; Pan et al., 2023) suggest that the **solver route**, translating problems into solver-executable representations and delegating execution, often outperforms the **direct route**, reasoning end-to-end in NL, on logic- and algorithmic-style complex reasoning tasks. However, for algorithmic reasoning alone, there is still no systematic analysis comparing the two routes, clarifying when and why LLMs perform better via the solver route versus the direct route.

A principled direct comparison is challenging since the routes operate over different representation spaces, i.e., NL traces versus solver-executable programs, and rely on different execution mechanisms, which prevents step-by-step alignment. Specifically, sample-complexity comparisons (Bai et al., 2023) are ill-posed here because the two routes learn fundamentally different objects, so there is no common formal target and metric to compare. Computational-complexity arguments (Merrill & Sabharwal, 2023) are also not a clean discriminator here because the two routes incur fundamentally different execution-dependent costs. We therefore compare the two routes via statistical difficulty, using optimal achievable end-task Bayes risk (Xie et al., 2021).

In this paper, we propose a three-route Bayesian inference

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

Evaluation Arm Prompts

Arm 1: Natural Language (NL)	Arm 2: Code Simulation (Sim)	Arm 3: Code Execution (Code)
Prompt: "Solve the following algorithmic problem: {problem} YOU ARE NEVER ALLOWED TO USE CODE. FOLLOW THE FORMAT CAREFULLY."	Prompt: "Solve the following algorithmic problem: {problem} FOLLOW THE FORMAT CAREFULLY. You MUST: • code: Python solution() function • simulation: NL trace"	Prompt: Uses code from Arm 2 Execute solution() from Arm 2 in Python runtime to get answer.
Example: Q: "Compute: 123 + 456" numerals = [123, 456] 123 + 456 = 579 "Answer": "579"	Example: Q: "Compute: 123 + 456" def solution(): return 123 + 456 "simulation": "123+456=579" "Answer": "579"	Example: Code from Arm 2: def solution(): return 123 + 456 >>> solution() 579

Figure 2. **Prompt templates for three-arm evaluation.** **Arm 1 (NL):** LLM reasons in natural language only, code forbidden. **Arm 2 (Sim):** LLM generates Python `solution()` then simulates execution in NL. **Arm 3 (Code):** Same code executed in Python runtime. This isolates execution mechanism while controlling translation.

framework that makes this comparison tractable by introducing an additional intermediate **simulation route**, where the model performs the same translation but simulates execution in NL, other than the **direct route** and **solver route**, and verbalizes the representation using Chain-of-Thought (Wei et al., 2022) as shown in Fig. 1. Using this framework, we characterize when and why algorithmic reasoning favors the solver route, and show that solver-based pipelines are generally easier for a broad class of tasks. This framework also enables a tractable theoretical comparison of the routes, showing that the simulation route outperforms the direct route. Finally, we empirically demonstrate that the solver route substantially outperforms the simulation route, highlighting additional gains from reliable external execution. Using our framework, we consistently observe a three-route ordering across algorithmic reasoning tasks: the solver route performs best, followed by the simulation route, and then the direct route (Figures 3 and 4). Moreover, the solver route’s advantage widens as task difficulty increases, with $4.01 \times$ odds of getting a correct answer for code over natural language reasoning. We evaluate our framework on CLRS30 (Veličković et al., 2022), NP-Hard-Eval (Fan et al., 2023), and a custom suite of algorithmic problems with controllable difficulty (addition, multiplication, LCS, rod cutting, knapsack, and ILP variants: assignment, production, and partition) across a broad range of models, spanning weaker open-source LLMs (e.g., Mistral (Jiang et al., 2023), LLaMA (Touvron et al., 2023), Qwen (Yang et al., 2024)) and stronger closed-source systems (e.g., OpenAI (Achiam et al., 2023), Gemini (Team et al., 2023), Claude (Anthropic, 2024)). We find that, averaged over tasks and models, the solver route achieves XX% accuracy, outperforming the simulation route (XX%) and the direct route (XX%) with statistical significance (XX).

Theoretically, we formalize the three reasoning routes as statistical experiments in the sense of Blackwell. We show that, under mild regularity assumptions, natural language reasoning traces can be viewed as a garbling of code-based

representations, inducing a Blackwell ordering in which code is at least as informative as natural language for downstream decision-making. This implies that code-based reasoning achieves weakly lower Bayes risk for algorithmic tasks. We further model execution as an information channel and show that delegating execution to an external solver corresponds to a deterministic channel, while LLM-based simulation introduces additional stochastic noise. Together, these results establish a principled ordering—natural language \leq code simulation $<$ code execution—providing an information-theoretic explanation for the empirical advantages of solver-based pipelines.

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. **Empirical validation** demonstrating that code execution achieves 78% accuracy versus 21% for natural language reasoning across arithmetic, DP, and ILP tasks ($p < 0.05$).
3. A **theoretical explanation** based on Bayesian inference showing that code yields higher mutual information with target algorithms, leading to lower Bayes error.

2. 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 $\text{Acc}(\text{Arm 2}) \leq \text{Acc}(\text{Arm 3})$.

2.1. Methodology

Arm 1 \leq Arm 2 Setup. Similarly, given test input X_i corresponding to instance i , we prompt the LLM to reason about it using natural language in Arm 1, mapping $(X_i) \rightarrow Y_i^{(\text{NL})}$. Likewise, Arm 2 can be written as using the same

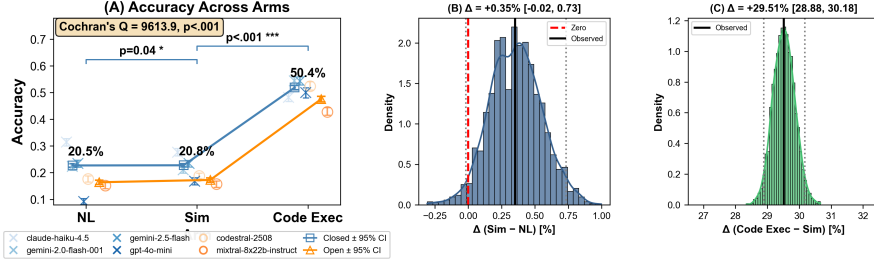


Figure 3. **Accuracy vs. task difficulty.** Difficulty τ controls problem complexity. **Arm 3** (code execution) maintains $>80\%$ accuracy while **Arm 1** (NL) and **Arm 2** (simulation) degrade rapidly. Results averaged over 5 models and 3 seeds; shaded: 95% CI.

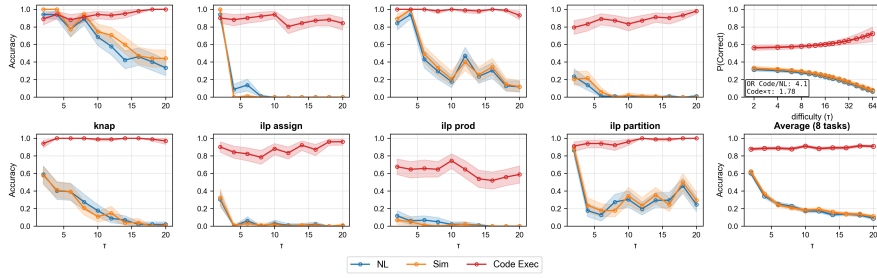


Figure 4. **Per-task accuracy breakdown.** Each panel shows accuracy vs. τ for arithmetic (add, mul), DP (lcs, rod, knap), and ILP tasks. **Arm 3** (Code Exec) consistently outperforms **Arm 2** (Sim) and **Arm 1** (NL). GLMM panel shows code maintains stability while NL degrades. Rightmost: average over 8 tasks.

LLM to first $(X_i) \rightarrow C_i$ and then $(X_i, C_i) \rightarrow Y_i^{(\text{Sim})}$. Example prompts are shown in Figure 2. We control the prompts to be as similar as possible, with the Arm 2 simply concatenating a section that says it should generate code and simulate it.

Arm 2 < Arm 3 Setup. Let the original problem statement corresponding to instance i be X_i , and let C_i be the corresponding code generated for instance i . Let tuple (X_i, C_i) be fixed inputs in our experimental design. Let Arm 2 denote the output $Y_i^{(\text{Sim})}$ which is mapped to by an LLM $(X_i, C_i) \rightarrow Y_i^{(\text{Sim})}$, and let Arm 3 denote the output $Y_i^{(\text{Exec})}$ mapped to by an external python runtime $(X_i, C_i) \rightarrow Y_i^{(\text{Exec})}$ ¹. Example prompts are shown in Figure 2.

Arm 1 ≤ Arm 2 < Arm 3 Setup. For each problem instance i , we observe bernoulli outcomes $(Y_i^{(\text{NL})}, Y_i^{(\text{Sim})}, Y_i^{(\text{Exec})})$. We first test the overall arm effect using Cochran’s Q test, evaluating the global null:

$$H_0 = E[Y_i^{(\text{NL})}] = E[Y_i^{(\text{Sim})}] = E[Y_i^{(\text{Exec})}]$$

i.e. that all arms have marginal success probabilities. Rejection of this null indicates at least one arm differs in accuracy. Condition on rejecting the global null, following our framework, we break down the tests into Arm 1 and Arm 2 $(Y_i^{(\text{NL})}, Y_i^{(\text{Sim})})$, and Arm 2 and Arm 3 $(Y_i^{(\text{Sim})}, Y_i^{(\text{Exec})})$. For these paired bernoulli outcomes, we run the McNemar

¹ X_i is ignored by executor, but we include it for notational symmetry

test under the null that:

$$\begin{aligned} H_0 : \Pr(Y^{(\text{Sim})} = 1, Y^{(\text{Exec})} = 0) \\ = \Pr(Y^{(\text{Sim})} = 0, Y^{(\text{Exec})} = 1) \end{aligned}$$

that is, when pairs disagree, each one ($\text{Sim} > \text{NL}$) and ($\text{NL} < \text{Sim}$) occur equally as often. We test an analogous null for Arm 2 and Arm 3. To quantify effect size, we take paired accuracies

$$\Delta_{\text{Sim-NL}} = \text{Acc}(\text{Sim}) - \text{Acc}(\text{NL})$$

and estimate its sampling distribution and 95% confidence interval via a cluster bootstrap, resampling over instances i with preserved pair outcomes $(Y_i^{(\text{NL})}, Y_i^{(\text{Sim})})$. We run the same procedure for Arm 2 and Arm 3.

Since we apply multiple statistical tests, we add Holm-Bonferroni corrections to control family wise error rate at $\alpha = 5\%$.

Finally, to analyze how task difficulty interacts with different arms, we run a generalized linear mixed-effects model (GLMM) with a logistic link.

$$Y_i = \alpha + \beta_{\text{arm}_i} + \gamma \tau_i + \delta_{\text{arm}_i} \tau_i + \varepsilon_i, \quad \varepsilon_i \sim \text{Bernoulli noise}$$

We model arm and task hardness as fixed effects, along with their interaction, with problem instance and seed as random effects.

Data and Models. We use the CLRS 30 Benchmark ($n=500$), NPHardEval Benchmark ($n=540$), and a custom fine-grained evaluation suite ($n=540$), across three seeds. We define our own task suite—Arithmetic, Dynamic Programming, Integer Linear Programming (ILP)—to modulate hardness with parameter τ . For arithmetic, τ controls

digit length; for DP, it controls table dimensionality; for ILP, it controls the constraint matrix size. We assume our algorithms are representative enough of the distribution. We select frontier models (Haiku 4.5, GPT-4o, Gemini 2.0 Flash) as well as open-source models (Mixtral, Codestral). Since we require structured output, we filter out models that give $>50\%$ JSON Parse Error, since this is indicative of instruction-following failures, rather than lack of coding fidelity.

Prompting to generate Code and Reasoning Traces. We prompt the LLM in Arm 1 to never use any code in its reasoning, and to give a structured output of the rationale and answer to the algorithmic problem Figure 2. Holding all other parts of the prompt the same, we replace the section instructing the model not to use code with another asking it to generate code first before reasoning through it. Models have access to native python packages, and numpy, pandas, scipy, PuLP, and pytorch. For Arm 3, we take the generated function (no prompt), and execute it in a python3 runtime. We ask model’s to output a structured json following a corresponding schema for that arm, and example is show in Figure 2.

2.2. Experiments

Arm 1 \leq Arm 2 $<$ Arm 3 Results. For Arm 1 = Arm 2 = Arm 3, we strongly reject the global null hypothesis ($p < 0.001$ after controlling for FWER).

Our post-hoc paired tests Arm 2 $<$ Arm 3, strongly reject the null hypothesis too ($p < 0.001$). We observe the paired accuracy gap is strictly large and positive, excluding zero in 95% bootstrap confidence interval. Observing the distribution, the result is not driven by outliers, given the tightness of the distribution, indicating advantages over instance pairs and not just the average. This means that LLM execution via natural language reasoning of a piece of code has statistically significantly worse performances than code execution under the representative benchmarks and suites of task.

For post-hoc paired test Arm 1 \leq Arm 2, despite running 30,000+ samples total and rejecting the null ($p=0.04$), our cluster bootstraps’ 95% confidence interval overlaps with 0.0, indicating that we cannot conclude that code simulation performs differently than natural language under our test benchmarks and sampling experimental design. This leads us to conclude that code and natural language are approximately the same.

Observing Figure 3, we see that the dominance trends for Arm 2 $<$ Arm 3 and Arm 2 \simeq Arm 3 holds across both open and closed models.

Advantages of Arm 3 emerge as tasks get harder. Observing Figure 3, we see that across 8 different algorithmic tasks, code execution strongly outperforms NL both on average and individually. Using the GLMM, we extrapolate the data on a log scale and observe that code and NL correctness probabilities diverges as tasks get harder, code remains

stable whereas NL diverges to 0. Code has $4.1\times$ odds of getting the answer correct compared to natural language.

3. Evaluating Translated NL and NL Distributional Similarity.

One key hypothesis is that natural language reasoning follows a deeper algorithmic computation encoded in its representations. If this is the case, it would make sense to surface the code and delegate the execution to an external runtime, rather than simulating noisy execution. We test whether natural language reasoning generated by information contained in the code alone can be distributionally similar to natural language reasoning generated from the prompt alone in Arm 1. Then, we test whether they are functionally similar.

3.1. Distributional Similarity between Translated NL and original NL

We show that the distribution of traces produced directly from the reasoning model can be approximated by post-processing the code with a fixed task-independent transformation.

Evaluation Setup For each evaluation task x and corresponding code c , we construct two samples from CoT we observed in experiment 1. We leverage the distribution of Arm 1 reasoning traces $p_{\text{NL}}(\cdot | x)$ and a matched distribution generated by a translator T (GPT-4o with 10 in-context examples, Figure 5a) $p_{\text{Tr}}(\cdot | c)$:

$$\begin{aligned} (x, z_{\text{NL}}), \quad z_{\text{NL}} &\sim p_{\text{NL}}(\cdot | x), \quad \text{labelled Native.} \\ (x, \hat{z}_{\text{NL}}), \quad \hat{z}_{\text{NL}} &\sim p_{\text{Tr}}(\cdot | c), \quad \text{labelled Translated.} \end{aligned}$$

We then formulate a binary classification task in which a powerful zero-shot judge model (Claude Opus 4.0, Gemini 2.5 Pro, Grok 4.1 Fast) is given a problem instance X , and a single reasoning trace Z , and prompted to predict whether z was generated by Arm 1, or the translated Arm 2 (Figure 5b). This setup corresponds to discriminating between the joint distributions

$$p(x)p_{\text{NL}}(z | x) \text{ and } p(x)p_{\text{Tr}}(z | x)$$

We sample problem instances x randomly from a pool of 30,000 CoT examples from before spread across different models, seeds, tasks, hardness. Fixing these, we obtain paired outcomes corresponding to an x . We ensure that label counts between \hat{z}_{NL} and z_{NL} are balanced on the test set, and tasks (21 tasks from CLRS 30 Benchmark) are disjoint from any in-context prompting examples.

As a control, we report the judge’s discriminative power by asking it to classify between raw code and the Native NL reasoning, where high accuracy would indicate that performance results on the main evaluation is not due to an underpowered judge.

Results. Across 2000 samples and 3 judge models (6000 samples total), we find that the accuracy of the prompt is

<p>(a) Translator Prompt</p> <p>You are given code that solves an algorithmic problem. Reason through the problem step-by-step using natural language and arrive at the answer. Do NOT translate the code mechanically.</p> <p>GUIDELINES</p> <ul style="list-style-type: none"> • Think like a human (exploratory reasoning) • Be conversational ("Let me check...", "I notice") • Skip obvious steps, focus on insights (WHY) • Use natural structure (paragraphs over lists) <p>10 IN-CONTEXT EXAMPLES</p> <p>Example: Topological Sort Input: Adjacency matrix A = [[0,1,0,...],...] Output: "Node 3 has in-degree 0... Answer is 3." (+ 9 more: KMP, Bridges, LCS, Bellman-Ford, ...)</p> <p>TEST INPUT</p> <pre>def solution(): # [Code to translate]</pre>	<p>(b) Discriminator Prompt</p> <p>You are analyzing an explanation of how to solve an algorithmic problem.</p> <p>TASK: Determine whether this was written by someone solving naturally ("Native NL") or translating/simulating code ("Translated").</p> <p>INPUT FORMAT</p> <p>PROBLEM: {Algorithmic problem description}</p> <p>EXPLANATION: {Reasoning trace to classify}</p> <p>OUTPUT FORMAT</p> <p>PREDICTION: [NATIVE or TRANSLATED] CONFIDENCE: [HIGH, MEDIUM, or LOW] REASONING: [1-2 sentence justification]</p>
--	--

Figure 5. Translation and discrimination prompts. (a) **Translator:** Converts code to NL reasoning step-by-step, mimicking native reasoning. (b) **Discriminator:** Judge models classify traces as “Native NL” or “Translated”. Used in Section 3.

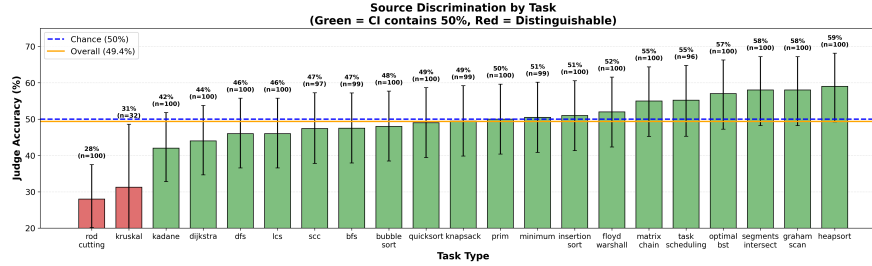


Figure 6. **Discrimination results.** Judge models classify traces as Native or Translated. Accuracy is near chance (49.4%, 95% CI [47.2%, 51.5%]), indicating translated NL is distributionally similar to native NL. Control: 79% accuracy on code vs. NL confirms calibrated judges.

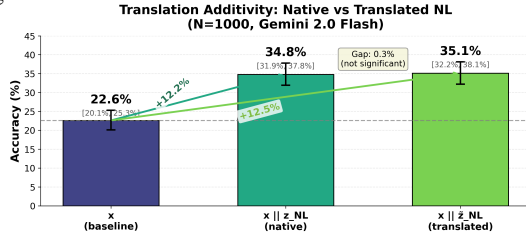


Figure 7. **Functional equivalence.** Overlapping 95% CIs between Native and Translated conditions indicate code-to-NL translation preserves functional information for correct answers.

$49.4\% \in [47.2\%, 51.5\%]$ Wilson CI (Figure 6), the control accuracy is 79.0% so the discriminators are calibrated. This holds within-model and within-task with a few exceptions being rod-cutting 1D dynamic programming and kruskal’s MST algorithm problems. These findings are surprising since one might expect models to have different reasoning patterns, but our results explain that there exists a post-processing model that can yield the same reasoning process under the right prompting conditions.

3.2. Functional Similarity between Translated NL and original NL

We wish to verify whether the translated natural language reasoning from the experiment before has the same functionality as the original natural language reasoning produced in Arm 1.

Evaluation Setup. Given a task instance x , we prompt the

target language model three ways.

1. Baseline: x (question only)
2. Arm 1: $x || z_{NL}$
3. Translated Arm 2: $x || \hat{z}_{NL}$

where \hat{z}_{NL} is obtained by translating code to natural language. If the translated NL loses information relative to the original NL, then conditioning on \hat{z}_{NL} should yield worse performance than conditioning on z_{NL} . We run on 1000 samples across 3 models (Claude Opus 4.0, Gemini 2.5 Pro, Grok 4.1 Fast) and report results on held-out tasks disjoint from the ICL example tasks. The same prompt is used for the translator in the previous experiment as the one used here. Crucially, the translator models we use are the same as the models that generated the original code x , unlike the experiment from before.

Results. We fail to reject the null hypothesis, and see overlapping 95% confidence intervals for Native and Translated (Figure 7). This leads us to believe that post-processed code and natural language reasoning have similar functionality quantified by end-task accuracy. Since we fixed the natural language reasoners here (same model), and only vary whether we feed in the original prompt and the generated code (with prompt masked), we infer that code and the original prompt share similar information, i.e. code does not lose much information. We also conclude that the algorithmic representations in code are simulated by the natural language reasoning.

4. Statistical and Information Theoretic

Foundations of Algorithmic Reasoning

(Blackwell, 1953) Here, we prove that $Arm1 \simeq Arm2 < Arm3$ leveraging a similar breakdown framework as our experiments. We utilize information theory and statistical decision theory to sidestep representing the ambiguity of natural language and differences between NL and Code in a mathematical framework. Using the intuitions we gain in our experiments, we prove that Arm 2 is at least as good as Arm 1, and that Arm 3 is always better than Arm 2.

4.1. Arm 1 \simeq Arm 2

Our empirical results on experiment 3 leads us to hypothesize that natural language reasoning is a post-processing (garbling) of code (Blackwell, 1953) under a noisy channel paradigm of inference, leading code reasoning to be at least as good as NL reasoning quantified by Bayes Risk.

Setup. Let $X \sim p(x)$ denote the task instance (problem + inputs), drawn from a representative test distribution. Let \mathcal{Y} be an output space (e.g., answer strings), and let $\ell : \mathcal{Y} \times X \rightarrow [0, 1]$ be a bounded and measurable loss function (0–1 binary loss). In Arm 1 let the generative and bayesian inference process be the latent variable model:

$$\begin{aligned} \text{Arm1} : X &\xrightarrow{p_{NL}(Z_{NL}|x)} Z_{NL} \xrightarrow{\delta_{NL}(y|x, Z_{NL})} Y_{NL} \\ P(Y_{NL}|X=x) &:= \int \underbrace{\delta_{NL}(y|x, z)}_{\text{Execute}} \underbrace{p_{NL}(z|x)}_{\text{Generate}} dz \end{aligned}$$

With the addition of a fixed LLM translator T (independent of x), let the generative and inference process for Arm 2.5 with translator be:

$$\begin{aligned} \text{Arm2.5} : X &\xrightarrow{p_C(Z_C|x)} Z_C \xrightarrow{T(\hat{Z}_{NL}|Z_C)} \hat{Z}_{NL} \xrightarrow{\delta_{NL}(y|x, \hat{Z}_{NL})} \hat{Y}_{Tr} \\ P(\hat{Y}_{Tr}|X=x) &:= \int \underbrace{\delta_{NL}(y|x, \hat{z})}_{\text{Execute}} \underbrace{T(\hat{z}|z)}_{\text{Translate}} \underbrace{p_{Code}(dz|x)}_{\text{Generate}} dz \end{aligned}$$

Let the original Arm 2 be:

$$\begin{aligned} \text{Arm2} : X &\xrightarrow{p_C(Z_C|x)} Z_C \xrightarrow{\delta_{Sim}(y|x, Z_C)} Y_{Sim} \\ P(Y_{Sim}|X=x) &:= \int \underbrace{\delta_{Sim}(y|x, z)}_{\text{Execute}} \underbrace{p_{Code}(dz|x)}_{\text{Generate}} dz \end{aligned}$$

We show that for some negligible ε , the Bayes Risk $R^*(Z)$ for code is less than for natural language.

$$\begin{aligned} R^*(Z) &:= \inf_{\delta} \mathbb{E}[\ell(Y, X)] \\ R^*(Z_{Sim}) &\leq R^*(Z_{NL}) + O(\varepsilon) \end{aligned}$$

ε is the maximum change in expected loss when replacing the true distribution by the approximated distribution, defined below in Assumption 2. Empirically, we showed that this value was small in experiment 2.

Assumption 1. We assume there exists a (near Bayes optimal) translator T mapping code to natural language reasoning $Z_{Code} \rightarrow \hat{Z}_{NL}$ independent of X .

Assumption 2. We assume that the original NL reasoning chain of thought is close to the translated NL on average. Let p_{NL} be the Arm 1 channel and $p_{translated}(\cdot | x)$ be the translated NL channel. Assume an average conditional TV bound:

$$\mathbb{E}_{X \sim p} [d_{TV}(p_{NL}(\cdot | X), p_{translated}(\cdot | X))] \leq \varepsilon,$$

where

$$d_{TV}(P, Q) = \sup_B |P(B) - Q(B)|.$$

In other words, averaged over task instances, the NL trace produced by Arm 1 is close in distribution to the NL traces obtained by translating the code trace (Arm 2) using the translator T . We empirically verify this in experiment 2.

Proof. Our high level strategy is to use Blackwell’s simulation principle, and show that NL reasoning is just code generation plus a noisy translation, thus a code-based agent can simulate exactly what the NL agent would do by averaging over that noise internally. In other words, under Assumptions 1–2, for the bounded loss $\ell \in [0, 1]$,

$$R^*(Z_{Code}) \leq R^*(Z_{NL}) + O(\varepsilon).$$

Step 1: Simulate NL from code via translation. Here, we first show that for any NL-based policy, there exists a code-based policy that induces the same input-output behaviour. Here we first translate the input problem into CoT, then execute the CoT via LLM reasoning, similar to what we do in the Arm 2 experiments. We abuse notation here, and $z = z_{Code}$. Define code simulation to be an implicit translation then reason in the original simulation branch:

$$\delta_{Sim}(y | x, z) := \int \underbrace{\delta_{NL}(y | x, \hat{z})}_{\text{Execute}} \underbrace{T(\hat{z} | z)}_{\text{Translate}} dz.$$

Then, we can apply the law of iterated expectation on Arm 2 to get Arm 2.5:

$$\begin{aligned} P(Y_{Sim}|X=x) &:= \int \underbrace{\delta_{Sim}(y|x, z)}_{\text{Execute}} \underbrace{p_{Code}(dz|x)}_{\text{Generate}} dz \\ &= \int \int \underbrace{[\delta_{NL}(y|x, \hat{z})]}_{\text{Execute}} \underbrace{T(\hat{z}|z)}_{\text{Translate}} \underbrace{p_{Code}(dz|x)}_{\text{Generate}} dz \\ &= P(\hat{Y}_{Tr}|X=x) \end{aligned}$$

The joint distributions (X, Y_{Sim}) and $(X, \hat{Y}_{translated})$ are the same. Thus,

$$\mathbb{E}[\ell(Y_{Sim}, X)] = \mathbb{E}[\ell(\hat{Y}_{translated}, X)].$$

Step 2: Substitute translated NL and original NL via TV lemma. Here we show that if two signals look similar,

they perform similarly. Even if translated NL is not exactly native NL, bounded loss decision problems cannot exploit small distributional differences, a property of the continuity property of Bayes Risk.

Lemma 4.0.1 (TV Lemma). *Let $X \sim p(x)$. Let $Z \mid X = x \sim P_x$ and $Z' \mid X = x \sim Q_x$. Let $g(x, z) \in [0, 1]$ be measurable. Then*

$$\mathbb{E}[g(X, Z)] - \mathbb{E}[g(X, Z')] \leq \mathbb{E}_X[d_{TV}(P_X, Q_X)].$$

For each x and trace z , define $g(x, z) := \mathbb{E}_{y|x,z}[\ell(y, x)]$. Then $g(x, z) \in [0, 1]$. Note that

$$\begin{aligned} \mathbb{E}_{(Y_{NL}, X)}[\ell(Y_{NL}, X)] &= \mathbb{E}_{(X, Z_{NL})}[g(X, Z_{NL})], \\ \mathbb{E}_{(\hat{Y}_{Tr}, X)}[\ell(\hat{Y}_{Tr}, X)] &= \mathbb{E}_{(X, \hat{Z}_{NL})}[g(X, \hat{Z}_{NL})]. \end{aligned}$$

Applying the TV lemma with $P_x = p_{NL}(\cdot \mid x)$ and $Q_x = p_{translated}(\cdot \mid x)$:

$$\begin{aligned} |\mathbb{E}[\ell(Y_{NL}, X)] - \mathbb{E}[\ell(\hat{Y}_{Tr}, X)]| &= |\mathbb{E}[g(X, Z_{NL})] - \mathbb{E}[g(X, \hat{Z}_{NL})]| \\ &\leq \mathbb{E}_X[d_{TV}(p_{NL}(\cdot \mid X), p_{translated}(\cdot \mid X))] \leq \varepsilon. \end{aligned}$$

Therefore, rearranging gives

$$\begin{aligned} \mathbb{E}[\ell(\hat{Y}_{Tr}, X)] &\leq \mathbb{E}[\ell(Y_{NL}, X)] + \varepsilon \\ \mathbb{E}[\ell(Y_{Sim}, X)] &= \mathbb{E}[\ell(\hat{Y}_{Tr}, X)] \leq \mathbb{E}[\ell(Y_{NL}, X)] + \varepsilon. \end{aligned}$$

We add superscripts to denote the decision rule (LLM executor) that was used. Then, since the inequality holds for arbitrary LLM executors δ_{NL} (including the best), we get:

$$\inf_{(\delta_{Sim}, \delta_{NL})} \mathbb{E}[\ell(Y_{Sim}^{(\delta_{Sim})}, X)] \leq \inf_{(\delta_{Sim}, \delta_{NL})} \mathbb{E}[\ell(Y_{NL}^{(\delta_{NL})}, X)] + \varepsilon.$$

Therefore, ignoring infimums over independent variables δ_{NL} on the left and δ_{Sim} on the right, we get exactly the Bayes Risk defined before.

$$R^*(Z_{Sim}) \leq R^*(Z_{NL}) + \varepsilon$$

□

4.2. Arm 2 < Arm 3

We prove that deterministic execution in Arm 3 yields strictly lower Bayes Risk.

Setup. Let $X \sim p(x)$ be a task instance and let $Y^*(x)$ denote the corresponding ground-truth. Suppose $Z_{Sim} \sim p_{Sim}$ is the code produced by the model. Let g be a deterministic python3 runtime. Define the two arms:

$$\begin{aligned} \text{Arm2} : Y_{Sim} &\sim \delta_{Sim}(\cdot \mid X, Z_{Sim}) \\ \text{Arm2} : Y_{Exec} &:= g(X, Z_{Sim}) \end{aligned}$$

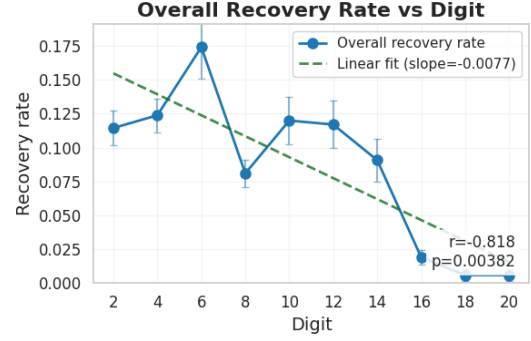


Figure 8. Recovery rate. How often Arm 2 succeeds when Arm 3 fails. Recovery rates are <5% across all tasks, confirming Arm 3’s advantage comes from reliable execution, not LLM compensation.

Proof. Under $\ell(y, x) = \mathbf{1}\{y \neq Y^*(x)\}$, the solver achieves zero risk ($R_3^* = 0$) whenever the generated code is correct. In contrast, LLM simulation incurs positive risk ($R_2^* > 0$) whenever $\Pr[Y_2 \neq Y_3] > 0$, which occurs empirically due to execution errors in mental simulation.

$$R_3^* < R_2^*$$

□

The only scenario where Arm 2 could outperform Arm 3 is when the generated code is *incorrect*, yet the LLM “recovers” by reasoning to the correct answer despite the flawed code. We empirically quantify this recovery rate below.

Recovery reduces as tasks get harder. To further reinforce this result, we rule out the possibilities of recovery as tasks get harder, eliminating any benefit of running Arm 2:

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

Figure 8 presents the recovery analysis across all tasks and models. The recovery rate remains consistently 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.

5. Related Work and Discussion

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., 2023; 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.

6. Conclusion

We introduced a three-arm Bayesian framework for comparing code and natural language representations in algorithmic reasoning. By introducing an intermediate arm—code generation with LLM-based simulation—we isolate translation effects from execution effects, enabling tractable pairwise analysis. Empirically, code execution achieves $4.01 \times$ higher odds of correctness than NL reasoning ($p < 0.001$) across 44 algorithmic tasks and 6 models. Our causal intervention experiments demonstrate that NL reasoning traces are distributionally and functionally equivalent to code-translated traces, supporting the hypothesis that NL reasoning implicitly simulates underlying algorithmic computations. Theoretically, we prove that code-based reasoning achieves lower Bayes risk via Blackwell dominance, providing an information-theoretic explanation for the empirical advantages of solver-based pipelines.

Limitations. (1) *Task scope:* Our evaluation focuses on algorithmic tasks (arithmetic, DP, ILP) with well-defined ground truth; results may not generalize to open-ended reasoning or tasks without clear algorithmic structure. (2) *Empirical approximations:* Our distributional similarity tests use finite samples and specific judge models, introducing estimation variance. (3) *Model coverage:* While we evaluate

both frontier (GPT-4o, Claude, Gemini) and open-source models (Mistral, LLaMA), we cannot guarantee findings generalize to all architectures or future models. (4) *Theoretical assumptions:* The Blackwell dominance result assumes the existence of a garbling channel from code to NL, which we validate empirically but do not prove from first principles.

References

- Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- 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].
- Anthropic. The claude 3 model family: A new standard for intelligence. *Anthropic Technical Report*, 2024.
- 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/>.
- Blackwell, D. Equivalent comparisons of experiments. *The Annals of Mathematical Statistics*, 24(2):265–272, 1953.
- 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].
- Fan, L., Hua, W., Li, L., Ling, H., Zhang, Y., and Hemphill, L. Nphardeval: Dynamic benchmark on reasoning ability of large language models via complexity classes. *arXiv preprint arXiv:2312.14890*, 2023.
- 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., Benani, 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.
- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.
- 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., Havaladar, 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. LogicIm: 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.
- 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., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, 2023.
- 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].
- Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Sorber, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- Veličković, P., Badia, A. P., Budden, D., Pascanu, R., Bannino, A., Dashevskiy, M., Hadsell, R., and Blundell, C. The clsr algorithmic reasoning benchmark. *International Conference on Machine Learning*, pp. 22084–22102, 2022.
- 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].
- Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.
- 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.