CoTs as Probabilistic Programs: A Programmatic View of Thinking Step-by-Step in Language Models

Kyle Richardson[1,2*] **Yu Feng**[3*] **Poorva Garg**[4]
Junyan Cheng[5] **Guy Van den Broeck**[4] **Dan Roth**[3,6]
[1]University of Tübingen [2]Allen Institute for AI [3]University of Pennsylvania
[4]University of California, Los Angeles [5]Dartmouth College [6]Oracle AI

Abstract

Chain-of-thought (CoT) prompting has fundamentally changed how language models are used, evaluated, and developed. CoT now touches many areas of contemporary research beyond prompting, from advanced test-time inference and model tuning to interpretability, yet much of this work treats CoT traces in task-specific ways, without a shared formal account of what they are or how they compose and support inference. We argue that probabilistic programs — ordinary programs extended with stochastic choices and probabilistic conditioning — provide a natural framework for modeling CoT reasoning. To demonstrate this, we derive Copper, a small discrete probabilistic programming language for CoT in which reasoning steps are stochastic choices, branches encode dependencies, trace likelihoods score executions, and answers are return values. Even with a minimal reachability-based semantics, Copper enables a first-principles analysis of trace likelihood, exposing why likelihood alone can be a misleading model of reasoning. Motivated by this limitation, we add differentiable operators for composing scores and feedback over CoT traces. The resulting framework links several uses of CoT — scoring, evidence aggregation, feedback propagation, model tuning, and step-level sensitivity analysis — within a single probabilistic program. Through case studies, we show how this approach yields novel test-time inference strategies and new analysis techniques, offering a promising direction for programming-language-based approaches to language-model reasoning.

1 Introduction

Since the seminal work of Wei et al. (2022), chain-of-thought (CoT) prompting has become central to how language models are used, evaluated, and improved. CoT traces now appear in **prompting**, in the **training** of large reasoning models (Guo et al., 2025; Yeo et al., 2025; Chen et al., 2025a), in advanced **test-time inference** for deep research agents (Huang et al., 2025), and as a scaffold for **interpretability** analysis (Turpin et al., 2023; Lanham et al., 2023; Stechly et al., 2024). However, these uses often treat traces in task-specific ways, without a shared account of what traces denote, how their steps compose, or what inferences they support. This makes it difficult to carry insights from one area (e.g., interpretability analysis) into another (e.g., the training of reasoning models) and naturally raises the question: *How can we better unify these different areas of research?*

We argue that CoT traces can be naturally modeled as simple probabilistic programs (PPs): ordinary programs extended with stochastic choices and probabilistic conditioning (Gordon et al., 2014; Katoen et al., 2015; De Raedt & Kimmig, 2015). For example, to represent the reasoning trace for the prompt in Figure 1**(A)**, we can use the PP in **(B)**. Here reasoning steps become stochastic assignments

*Equal contribution. Work done while KR was at AI2. Code available at `https://github.com/TuLLI-AI/Copper`.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

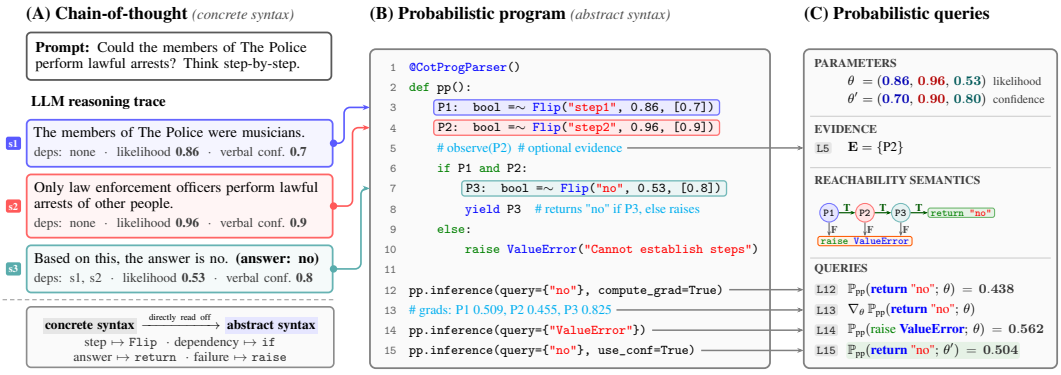


Figure 1: **Copper models CoT traces as probabilistic programs.** (**A**) An example query adapted from Geva et al. (2021b) and raw reasoning trace, interpreted directly as a program in Copper (**B**). Given step-weight parameters θ (e.g., parametrized by LLM **step likelihoods** or **verbal confidences**), reachability semantics yields exact answer/failure probabilities (**lines 12, 14**), supporting conditioning (**line 5**), alternative weights θ' (**line 15**), and sensitivity analysis via step gradients (**line 13**).

(via `Flip(.)`, a standard PP operator for defining random variables), dependencies among steps are encoded as control flow, evidence is incorporated using `observe(.)` (another standard operator for variable conditioning) and answers are program outcomes. Questions about a trace then become structured probabilistic queries and inferences over the corresponding program: answer probabilities used in learning or inference are computed as differentiable marginals, while conditional probabilities and sensitivities reveal how individual reasoning steps affect answer probabilities.

The main goal of this paper is to provide a technical outline of how such simple PPs – built from independent Bernoulli choices composed through control flow – can serve as useful proxy models of LLM computations that help unify different uses of CoT and provide new formal insights. As part of this, we study CoT through a language-design lens, asking: *What would a (probabilistic) programming language for CoT need to represent traces, reason over them and derive new inference and analysis strategies?*

Accordingly, we derive Copper, a small discrete probabilistic programming language in the style of Dries et al. (2015); Holtzen et al. (2020). As illustrated in Figure 2, Copper treats CoT steps as primitives and supports two levels of design: the *program level*, which specifies how steps compose via different program structures, and the *program inference level*, which specifies the queries computed over the resulting programs. At the former level, we show formally how standard trace likelihood and other common CoT computations can be expressed as simple conjunctive PPs with branching. This not only provides an exact semantic characterization of CoT computation, but also exposes important limitations of likelihood-based reasoning – which rules out certain intuitive prompting strategies – and facilitates deriving new likelihood bounds through principled program transformations.

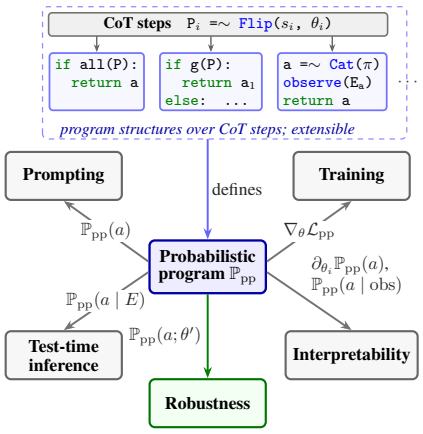


Figure 2: **Modeling and querying CoT.** At the *program level*, CoT steps compose into different reasoning structures. At the *program inference level*, existing uses of CoT become queries over these programs, while the same formulation enables new forms of inference, such as robustness testing (bottom).

Building on Copper and these formal insights, we present empirical case studies that explore both levels of design using commercial and open-weight models tested on a suite of binary and multiple-choice reasoning tasks (Clark et al., 2019; Geva et al., 2021a; Suzgun et al., 2022). At the *program level*, we derive *Bayesian multi-answer CoT*, a new prompting strategy that generates multiple candidate answers in a single model call and scores them through branching and Bayesian-style step aggregation similar to Feng et al. (2025). This offers an alternative to costly search-based inference (Yao et al., 2023; Besta et al., 2024) while exposing multiple reasoning paths for analysis.

At the *inference level*, we develop robustness tests inspired by formal verification (Kwiatkowska et al., 2007), asking how much program weights can change before the predicted answer changes. We explore different perturbations: replacing weights with scores from other models or verbal confidences (Kadavath et al., 2022; Xiong et al., 2024), and using gradients to locally estimate the smallest prediction-changing perturbation. We find that our new prompting strategy generally yields larger estimated robustness margins than standard likelihood-based inference, with higher-margin predictions more often correct – illustrating how program design and analysis work together to derive new reasoning strategies and uncover their behavioral properties.

Taken all together, our paper makes the following **contributions**: We introduce probabilistic programs as a framework for CoT, deriving a small discrete language with a semantics for representing traces, and querying their behavior. We use this framework to analyze trace likelihood and derive new inference and robustness-testing strategies. As a *Concept & Feasibility* contribution, this work advances a broader research agenda for programming-language-based approaches to LLM reasoning, connecting different areas of CoT research.

2 Related Work

Our work takes inspiration from the literature on CoT and its variants that couple CoT with advanced search techniques (Yao et al., 2023; Mo & Xin, 2024; Besta et al., 2024; Bi et al., 2024; Yang et al., 2024), external tools (Ye et al., 2023; Gao et al., 2023; Wu et al., 2025) and other forms of test-time scaling (Khot et al., 2022; Wang et al., 2022; Feng et al., 2025; Cheng et al., 2026). In contrast, our work focuses on a general formalism for CoT rather than a specific inference strategy, similar in spirit to Dohan et al. (2022). Our technical framework draws on probabilistic programming (Pfeffer, 2010; Gordon et al., 2014; Katoen et al., 2015; Van de Meent et al., 2018), building most directly on discrete probabilistic programming (Holtzen et al., 2020; Saad et al., 2021) where programs define distributions over finite executions via random choices, branching, conditioning, and return values. Our approach is also semantically closely related to probabilistic logic programming (De Raedt et al., 2007; Manhaeve et al., 2018). Exact inference in this setting is performed by reducing probabilistic queries to weighted model counting over compiled logical representations (Chavira & Darwiche, 2008; Darwiche & Marquis, 2002; Fierens et al., 2015). As such, our focus on exact inference differs from other approaches that mix probabilistic programming with approximate inference and LLMs such as Wong et al. (2023); Lew et al. (2023b) (for related, see Garg et al. (2026)). See more comparisons in Appendix B.

3 A simple language for CoT

CoT prompting elicits intermediate natural-language reasoning steps before a final answer. For an input query or prompt Q, we write a **CoT trace** as $\mathbf{T} = (T_1, \ldots, T_m)$, where each T_i is an intermediate thought sampled from a language model, $T_i \sim \mathbb{P}_{\mathcal{M}}(\cdot \mid \mathbf{T}_{<i}, Q)$, and the final answer A is sampled from $A \sim \mathbb{P}_{\mathcal{M}}(\cdot \mid \mathbf{T}, Q)$. A common probabilistic view of CoT is the **latent-trace model** (Dohan et al., 2022; Xiang et al., 2025), where answer probabilities marginalize over possible reasoning traces: $\mathbb{P}_{\text{CoT}}(A \mid Q) = \sum_{\mathbf{T}} \mathbb{P}_{\mathcal{M}}(A, \mathbf{T} \mid Q)$. Our goal is to make the structure of a given trace $\mathbf{T}$ explicit: how its steps T_j compose, how uncertainty is assigned to them, and how they support an answer. Accordingly, we define Copper, a simple Python-like probabilistic programming language for traces $\mathbf{T}$. We distinguish generating a trace from interpreting it as a program: thoughts T_j become Boolean random variables, while program structure specifies how these thoughts determine reasoning outcomes. We next introduce our language, then express trace likelihood $\mathbb{P}_{\mathcal{M}}(A, \mathbf{T} \mid Q)$ and novel variants as simple programs (§ 3.1) and analyze limitations of likelihood-based reasoning (§ 3.2).

Syntax and Program Level. Figure 3a summarizes the syntax of Copper, whose basic primitives are CoT steps. Each step or thought T_i in $\mathbf{T}$ is represented by a Boolean random variable P_i, sampled using `Flip(.)` with probability `<p>` $= \theta_i$. Step dependencies are encoded through ordinary *if-then* control flow over Boolean conditions involving step variables. Reasoning errors are represented as `ValueError` exceptions, background constraints through `observe(.)`, and answers as return values. `Categorical(.)` represents mutually exclusive alternatives, while `Factor(.)` combines step scores into new Bernoulli variables (introduced below). A Copper program is any valid composition of these constructs, rather than a fixed template; its structure – which can be modified to include additional

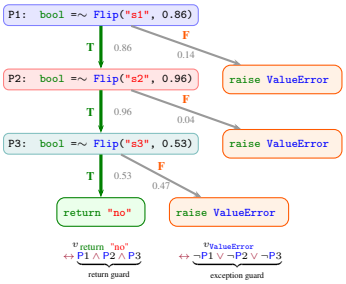
(a) **Core syntax of Copper and grounding in CoT**

CoT Concept	Syntax	Program Vars
Steps are probabilistic Booleans	`P: bool =~ Flip(<str>, <p>)`	$\mathcal{V}_{\text{dec}}$
Step dependencies are controls	`if <bool condition>:` `  <new fact or answer branch>`	guards(v)

Reasoning Errors are Exceptions	`if <bool condition>:`
	`  <new fact or answer branch>`
	`else:`
	`  raise ValueError(<error>)`
		$\mathcal{V}_{\neg\text{dec}}$

Answers are returns	`return <answer output>`	$\mathcal{V}_{\neg\text{dec}}$
Conditions are observes	`observe(<bool condition>)`	evidence
Chains/answers can be unique	`A1,..,An =~ Categorical(<p1, ..>)`	$\mathcal{V}_{\text{dec}}$
Scores can be aggregated	`P =~ Factor(<str>, `$f_m(p_1,\ldots,p_m)$`)`	$\mathcal{V}_{\text{dec}}$

(b) **Reachability semantics**



(c) **Boolean program encoding**

$$\mathsf{F} := \bigwedge_{v \in \mathcal{V}_{\neg\text{dec}}} \left(v \leftrightarrow \bigvee_{g \in \text{guards}(v)} g \right) \quad \mathsf{E} := \bigwedge_{e \in \text{evidence}} e$$

(d) **Probabilistic query computation**

$$\mathbb{P}_{\mathsf{F}}(q \mid \mathsf{E}; \theta) = \frac{\text{WMC}(\mathsf{F} \wedge \mathsf{E} \wedge q;\, \theta)}{\text{WMC}(\mathsf{F} \wedge \mathsf{E};\, \theta)}$$

Figure 3: **Syntax and semantics of Copper. (a)** Core language constructs. **(b) Reachability semantics**: each path corresponds to a program execution, with path weights determined by local decision probabilities; guards specify the Boolean conditions under which evidence-conditioned program points are reached. **(c)** Boolean encoding of the program and evidence formulas. **(d)** Exact conditional query probabilities computed by weighted model counting (WMC).

constraints or properties – reflects the reasoning steps and dependencies produced by the model, often following a structure dictated in the prompt Q.

Importantly, Copper is agnostic to the source and interpretation of θ: step weights may be derived from LLM logits or verbal confidences (as we show in Figure 1 and consider below), or external models (e.g., reward models), without necessarily representing probabilities of step truth.

Semantics and Inference Level. A Copper program defines a distribution over executions σ, which assign values to stochastic decision variables, such as P1, P2, and P3 in Figure 1. These assignments determine the branches taken and program events reached. Probabilistic queries therefore reduce to **reachability**: computing the probability of reaching a return value, exception, or intermediate point. We characterize reachability using **guards** (Dijkstra, 1975), Boolean formulas specifying when program events occur (Figure 3b). Let $\mathcal{V}_{\text{dec}}$ denote the Boolean decision variables and $\mathcal{V}_{\neg\text{dec}}$ the deterministic program-event variables. An execution $\sigma : \mathcal{V}_{\text{dec}} \to \{\mathsf{true}, \mathsf{false}\}$ uniquely determines each event variable v: it is true exactly when at least one guard in guards(v) is satisfied.

Formally, we define the **program formula** F and **evidence formula** E in Figure 3c. A query q is any Boolean condition over program variables, such as $q =$ `return "no"` and $q =$ `ValueError`. Its probability, conditioned on optional evidence, is computed in Figure 3d by **weighted model counting** (WMC) (Chavira & Darwiche, 2008; Fierens et al., 2015):

$$\text{WMC}(\phi; \theta) := \sum_{\sigma \in \text{Exec}(\phi)} \prod_{\mathsf{P} \in \mathcal{V}_{\text{dec}}} w_{\mathsf{P}}(\sigma(\mathsf{P}))$$

where Exec(ϕ) contains the Boolean decision assignments whose guard-determined extensions satisfy ϕ, and $w_{\mathsf{P}}(\mathsf{true}) = \theta_{\mathsf{P}}, w_{\mathsf{P}}(\mathsf{false}) = 1 - \theta_{\mathsf{P}}$ for P `=~ Flip(·, `θ_{P}`)` (similarly for categorical variables). Our implementation follows compilation-based approaches to PP (Holtzen et al., 2020), which translate programs into tractable circuits (Darwiche, 2011, 2021) that permit exact WMC inference (See Section C for a formal definition description of Copper).

3.1 What can we do with this simple language?

Formal reasoning about CoT. Although Copper is minimal – effectively an IMP-style language (Winskel, 1993) with probabilistic primitives – it recovers familiar CoT computations as ordinary program structures and queries. Table 1 gives two programs that represent k CoT traces $\mathbf{T}^{a_i} = (s_{i,1}, \ldots, s_{i,n_i})$ via branching, each including its final answer a_i. Each trace is represented by a sequence of flip assignments $\mathsf{P}_{i,j}$ `=~ Flip(`$s_{i,j}, \theta_{i,j}$`)`, one per step. As a first choice of weights, we

will assume a **LM likelihood parameterization** where each $\theta_{i,j} = \mathbb{P}_{\mathcal{M}}(s_{i,j} \mid s_{i,<j}, Q)$. Under this parameterization, the independent-branch program F_a with $k = 1$ recovers ordinary trace likelihood:

Proposition 1 (*CoT likelihood*)**.** *Under the LM likelihood parameterization,* F_a *with* $k = 1$ *satisfies*

$$\mathbb{P}_{\mathsf{F}_a}(\neg\texttt{ValueError}; \theta) = \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_1} \mid Q).$$

This gives likelihood an explicit conjunctive semantics whose negative logarithm recovers the standard training loss, showing the close link between inference and learning (Xu et al., 2018). For $k > 1$, branching determines how individual trace probabilities combine: independent branches model alternative routes to a successful return, while categorical branches represent mutually exclusive trace outcomes and the LLM's total probability mass for the chains, recovering a finite latent-trace approximation when all branches return the same answer.

Table 1: Program structures for modeling standard CoT computations.

Independent branching F_a

```
# trace 1
P_{1,1}   =~ Flip(s_{1,1}, θ_{1,1})...
P_{1,n_1} =~ Flip(s_{1,n_1}, θ_{1,n_1}) ...
# trace k
P_{k,1}   =~ Flip(s_{k,1}, θ_{k,1})...
P_{k,n_k} =~ Flip(s_{k,n_k}, θ_{k,n_k})

if P_{1,1} and ... and P_{1,n_1}:
    return a_1
elif P_{k,1} and ... and P_{k,n_k}:
    return a_k
else:
    raise ValueError
```

Categorical branching F_c

```
# trace 1
P_{1,1}   =~ Flip(s_{1,1}, θ_{1,1}) ...
P_{1,n_1} =~ Flip(s_{1,n_1}, θ_{1,n_1}) ...
# trace k
P_{k,1}   =~ Flip(s_{k,1}, θ_{k,1}) ...
P_{k,n_k} =~ Flip(s_{k,n_k}, θ_{k,n_k})

A_1, ..., A_k, A_¬ =~ Categorical(
    prod(θ_{1,:}), ..., prod(θ_{k,:}),
    1 - ∑_i prod(θ_{i,:}))
if A_1: return a_1 ...
elif A_k: return a_k
else: raise ValueError
```

Proposition 2 (*Multi-trace likelihood*)**.** *Under the LM likelihood parameterization, for* $k > 1$ *traces:* ***(i, noisy-or)*** *independent branches give* $\mathbb{P}_{\mathsf{F}_a}(\neg\texttt{ValueError}; \theta) = 1 - \prod_{i=1}^{k}(1 - \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_i} \mid Q))$*;* ***(ii, total mass)*** *for distinct, mutually exclusive complete traces,* $\mathbb{P}_{\mathsf{F}_c}(\neg\texttt{ValueError}; \theta) = \sum_{i=1}^{k} \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_i} \mid Q)$*;* ***(iii, latent trace)*** *if additionally all* $a_i = a$*, this sum equals* $\mathbb{P}_{\mathsf{F}_c}(\text{return } a; \theta)$ *and gives a finite approximation to the latent-trace marginal* $\mathbb{P}_{\text{CoT}}(a \mid Q)$*.*

These correspondences are elementary, but their role is foundational: *standard CoT likelihoods and latent-trace marginals arise directly from the same reachability semantics used for arbitrary program queries*. The programming formulation makes assumptions about traces explicit and formally analyzable, while the schemata in Table 1 provide declarative templates for constructing richer reasoning strategies. Self-consistency (Wang et al., 2022), for example, approximates the latent-trace marginal by sampling traces and aggregating them by answer. Best-of-N, verifier-guided, and process-reward methods (Cobbe et al., 2021; Lightman et al., 2023; Setlur et al., 2024) add answer-, trace-, or step-level scores to select or reweight traces, which in our framework correspond to different parameterizations, queries, or extensions of these templates.

Importantly, these probabilistic programs factorize over independent Boolean choices and hence serve as simple proxies for the full LLM distribution. Their exact correspondence with trace likelihoods nevertheless lets us derive bounds by relaxing reachability conditions. For example, in Table 2, F_d relaxes F_a by requiring that *at least* d steps hold (**line 3**, denoted by `sum(P) >= d`), which captures disjunction ($d = 1$) and conjunction ($d = n$). Building on Richardson et al. (2026), we observe that increasing d tightens these bounds.

Table 2: A relaxation of the trace likelihood program F_a ($k = 1$).

d-of-n program F_d

```
1 P_1 =~ Flip(s_1, θ_1) ...
2 P_n =~ Flip(s_n, θ_n)
3 if sum(P) >= d :
4     return a
5 else:
6     raise ValueError
```

Proposition 3 (*Novel likelihood bounds*)**.** *Under the LM likelihood parameterization, for a trace and answer* $\mathbf{T}^a$ *with* n *steps:*

$$\underbrace{\mathbb{P}_{\mathsf{F}_{d=1}}(\texttt{return a}; \theta)}_{\vee} \geq \mathbb{P}_{\mathsf{F}_{d=2}}(\texttt{return a}; \theta) \geq \cdots \geq \underbrace{\mathbb{P}_{\mathsf{F}_{d=n}}(\texttt{return a}; \theta)}_{\wedge} = \mathbb{P}_{\mathcal{M}}(\mathbf{T}^a \mid Q).$$

Though also simple, this result demonstrates a different role for our probabilistic programming approach: *deriving new inference targets through principled program transformations.*

3.2 Understanding the limitations of likelihood

A sensible but bad prompting strategy: likelihood-driven multi-answer CoT. Advanced test-time reasoning methods often rely on search, external solvers, or reward models (Yao et al., 2023, 2022;

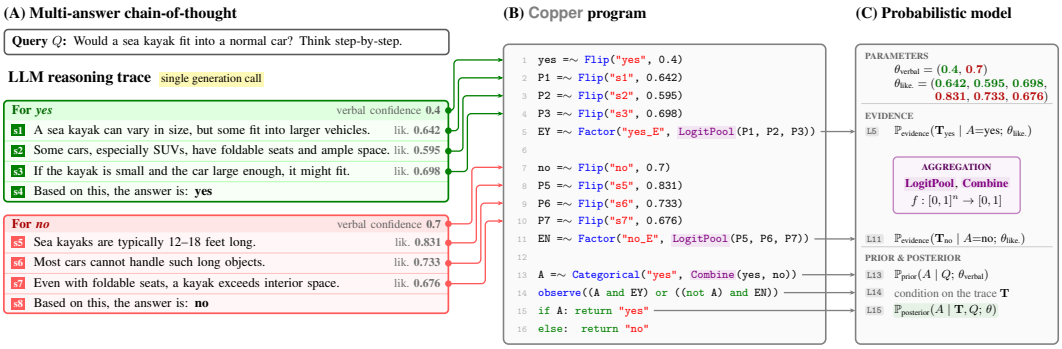


Figure 4: **Bayesian multi-answer CoT**, a prompting strategy derived from our framework that elicits multiple candidate answers and supporting reasoning paths. **(A)** An example `gpt-4o` trace produced in a single forward call. **(B)** The corresponding Copper program and its underlying probabilistic model **(C)**, which uses differentiable operators to aggregate step-level evidence using a mixture of LM likelihood signals ($\theta_{\text{likelihood}}$) and verbal probabilistic feedback (θ_{verbal}).

Gao et al., 2023; Zhong et al., 2026), typically requiring many additional model calls. We ask instead: *Can better program design yield richer reasoning from fewer model calls by structuring and using the model's own likelihood signals more effectively?* Here we use our program formulation to investigate this possibility and identify limitations of likelihood that rule out certain reasoning strategies.

We investigate **multi-answer CoT**, which instructs the model (below) to generate multiple candidate reasoning paths and answers in a single call, resulting in a branching program like discussed above.

Instruction: Please answer each question thinking step-by-step. Rather than giving a single answer, however, I want you to express different possible reasoning paths. Number your steps and identify step dependencies.
Query: Would a sea kayak fit into a normal car?
Example Trace *(Llama-3.3-70B-turbo-instruct)* **1.** A sea kayak is typically around 14-18 feet long **2.** Most cars have a maximum interior length of around 6-8 feet **3.** Based on 1 and 2, the answer is: no (ref. 1, 2) (final answer: no) **4.** Some cars have roof racks that can carry long items like kayaks **5.** Alternatively, based on 4, the answer is: yes (ref. 4) (final answer: yes)
Step likelihoods (length normalized) (*for no*) P1 0.839 P2 0.99 P3 0.951. (*for yes*) P4 0.921 P5 0.92
Program sketch: `if P1 and P2 and P3: return "no" elif P4 and P5: return "yes"`

Such an approach has the advantage of allowing candidate paths to share, contrast, or revise steps explicitly, without independent model calls or additional reward models. While the resulting text leaves answer selection unspecified, our PP formulation provides a natural selection rule: rank answers by the marginal probabilities of their supporting conjunctions and guards. Yet this favors *yes* (0.847 versus 0.790), despite the *no* path having higher mean log step support. Conjunction penalizes additional steps even when their probabilities are length-normalized, exposing a structural limitation of likelihood-based scoring that we call the length confound.

Proposition 4 (*Length confound*). *Let t_1, t_2 be two CoT chains with lengths n_1, n_2, log-likelihoods $\ell_1, \ell_2 < 0$, and mean log-likelihoods $\bar{\ell}_i = \ell_i / n_i$. If $\bar{\ell}_1 > \bar{\ell}_2$, then likelihood prefers t_1 over t_2 only under the following condition:*

$$\ell_1 > \ell_2 \iff \frac{n_1}{n_2} < \frac{|\bar{\ell}_2|}{|\bar{\ell}_1|}.$$

While related to ordinary length biases in sequence modeling, this confound arises at the *program level*: multiplying even token-length-normalized step probabilities penalizes longer paths. The proposition characterizes exactly the tradeoff: *higher mean log step support yields higher likelihood only when* $n_1 < n_2 |\bar{\ell}_2| / |\bar{\ell}_1|$. Crucially, rearranging steps into branches or dependencies does not remove this penalty from conjunctively scored paths, limiting what structural rearrangement or program transformations alone can achieve. Our threshold relaxations in Prop. 3 yield alternative scores and likelihood bounds, but do not ensure comparability across path lengths.

Step aggregation through Factor Returning to our language-design question, this motivates more flexible score aggregation. We therefore develop the `Factor` construct listed in Figure 3a (see example program in Figure. 4). Given any differentiable function $f_m : [0,1]^m \to [0,1]$, it defines a new

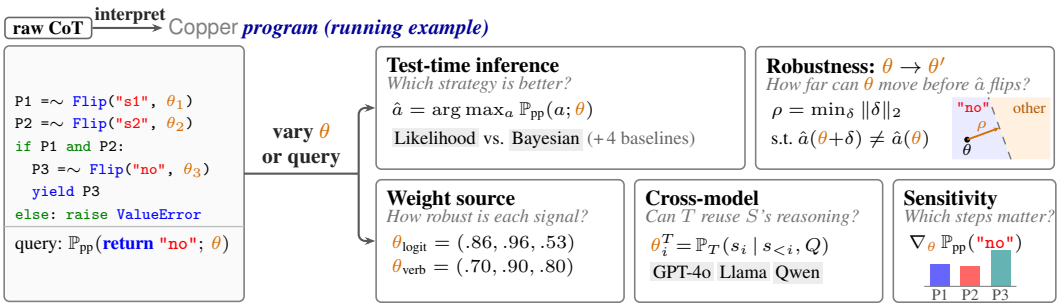


Figure 5: **High-level overview of case studies.** A raw CoT is interpreted as a Copper program whose structure is then held fixed. Each analysis is a probabilistic query over this program that varies only its step weights θ or the query itself. *Comparing CoT strategies:* **test-time inference** compares the likelihood (from § 3.1) and Bayesian (Figure 4) programs against standard prompting baselines, and **robustness** measures the smallest weight change ρ that flips the predicted answer. *Analyzing a single CoT:* **weight source** compares logit and verbalized step weights, **cross-model** scoring tests whether one model can reuse another model's reasoning trace, and **sensitivity** identifies the steps the answer depends on most (here bars show step gradients in the example program).

Boolean variable that is true with probability $f_m(p_1, \ldots, p_m)$. This permits arbitrary differentiable aggregation of step probabilities beyond sum-product composition. Importantly, this lifts Copper from a purely probabilistic PL to a differentiable PL over LLM outputs (Lew et al., 2023a; Tjoa et al., 2025), which facilitates defining more complex probabilistic models and computation graphs over CoT traces, as we discuss next.

4 Case Studies in Designing and Testing CoT Programs

We present case studies illustrating the two levels of design in Figure 2. At the *program level*, we derive a new prompting strategy, *Bayesian multi-answer CoT* (§ 4.1) that explores the multi-answer CoT strategy discussed previously. At the *program inference level*, we compare the resulting programs with standard likelihood programs from § 3.1 and through parameter perturbations, alternative weight assignments, and sensitivity queries.

4.1 Program design level: deriving new prompting strategies

Figure 4 illustrates Bayesian multi-answer CoT. Like the multi-answer strategy discussed above, it elicits multiple candidate answers and reasoning paths in a single model call. Unlike ordinary CoT, which assumes a structure $(T_1, ..., T_n, a)$, it reverses the generation order to $(a, T_1, ..., T_n)$: the LLM is first asked to produce candidate answers a and initial confidence estimates, then generate reasoning steps $\mathbf{T}$ conditioned on each answer. Formally, this design uses a Bayesian-style factorization:

$$\mathbb{P}_{\text{post}}(A \mid \mathbf{T}, Q) \propto \mathbb{P}_{\text{evidence}}(\mathbf{T} \mid A, Q)\mathbb{P}_{\text{prior}}(A \mid Q).$$

Initial answer confidences define the priors, while step weights – either token-length-normalized LM conditional **step likelihoods** derived from **logits**, or model-reported probabilities (**verbal confidences**) – are combined according to the reasoning dependencies to define evidence scores. Operationally, the model is instructed to produce extra structure in its output and assign numbers to each CoT step, generate verbal confidences (`verbal conf.`) directly after each line and identity step dependencies (`deps`) and answers (`answers:`) (as illustrated in Figure 1). While Copper supports different aggregation strategies, we experiment with a specific aggregation operator LogitPool, inspired by Feng et al. (2025), which averages step log-odds and maps the result back to a probability. The resulting program selects an answer through exact conditional inference, making both the prior and evidence contributions explicit (see Table 7 for further program parameters and details).

4.2 Program inference level: testing and analyzing CoT

Having defined the programs, we now use inference to test and analyze their behavior. Figure 5 illustrates these analyses using the running example from Figure 1. We compare predictive accuracy

and robustness across CoT strategies, then hold individual program structures fixed to examine alternative weight sources, cross-model trace reuse, and step-level sensitivity.

Experimental setup. We evaluate three commercial and open-weight models: GPT-4o (OpenAI et al., 2024), Llama-3.3-70B-Instruct, and Qwen-2.5-7B-Instruct[2] on a suite of commonsense reasoning tasks consisting of binary QA tasks BoolQ (Clark et al., 2019) and StrategyQA (Geva et al., 2021a), as well as three multiple-choice BIG-Bench Hard tasks (Suzgun et al., 2022): `Date Understanding`, `Geometric Shapes`, and `Salient Translation Error Detection`, selected for difficulty and diversity. Dataset statistics and implementation details appear in Appendix Table 8 and § E.3; further testing protocols are described below.

Test-time Inference Analysis. *How do predictions obtained through program inference compare with standard prompting?* Table 3 compares both programs with prompting baselines, including search-based Tree-of-Thoughts (Yao et al., 2023). Both improve on direct prompting across all three models. Bayesian multi-answer CoT approaches standard CoT accuracy and exceeds Tree-of-Thoughts on Qwen, though larger gaps remain on GPT-4o and Llama. Using a single generation call, it demonstrates the feasibility of an alternative to repeated search calls. The explicit program structure also supports error diagnosis, distinguishing incorrect elicited evidence from evidence overridden by answer priors, which we further discuss in § E.4. These results highlight an initial tradeoff and caveat: *while the programs enable structured analysis, they can incur an accuracy cost relative to standard prompting (e.g., owing to the need for the model to follow a more complex instruction involving producing a CoT with verbal confidences, answer annotations and step numbers).*

Table 3: Task-averaged accuracy (%) across models and prompting/ inference methods.

Task	Direct	CoT	Combined CoT	Tree-of-Thoughts	Likelihood	Bayesian
Qwen-7B	59.60	65.70	35.81	39.90	65.25	64.62
GPT-4o	76.89	81.87	78.68	83.49	80.75	77.22
Llama-3.3-70B	74.82	82.48	80.08	81.03	78.68	75.10

Robustness Analysis. *How stable are predictions under changes to reasoning weights?* Robustness is critical for trustworthy reasoning, especially in high-stakes domains where small changes on a CoT should not flip the final decision (Sclar et al., 2024; Chen et al., 2024). Beyond accuracy, the program formulation lets us test how much step probabilities or answer priors must change before a prediction flips. Inspired by formal verification (Kwiatkowska et al., 2007), we define the **robustness margin** ρ as the smallest weight perturbation θ' that changes the prediction, i.e.,

$$\rho(\theta) \;=\; \inf_{\theta' \in \mathbb{R}^d} \big\|\theta' - \theta\big\|_2 \quad \text{s.t.} \quad \operatorname{sign}\big(G(\theta')\big) \;\neq\; \operatorname{sign}\big(G(\theta)\big),$$

where G is the score gap between the two diferent answers or program outcomes. We use program gradients to obtain a first-order estimate of this margin for both likelihood and Bayesian multi-answer programs (see full details in § E.5).

Table 4 shows that **Bayesian multi-answer CoT has larger estimated robustness margins** than the likelihood program in five of six model–task pairs, for both mean and median margins, including after normalization by parameter count. The exception is Qwen on StrategyQA. Among Bayesian programs, Llama has the largest margins, followed by GPT-4o and Qwen. We also examine whether stability is informative about correctness. For each model and task, we sort examples by ρ_{norm} and compare accuracy on the 10% with the largest margins against the 10% with the smallest (Figure 8). **High-margin Bayesian predictions are consistently more accurate**, whereas likelihood-program margins sometimes exhibit the reverse pattern. These findings distinguish stability from accuracy: *Bayesian programs can have larger margins despite lower overall accuracy, while their margins help identify which predictions are more likely to be correct.*

Verbalized versus Logit-Based Weights. *How does the choice of confidence signal affect program predictions?* Verbalized confidences and logit-derived probabilities provide different ways to parameterize reasoning (Kadavath et al., 2022; Xiong et al., 2024). Holding the Bayesian program's reasoning structure fixed, we independently vary the sources of step weights and answer priors

[2]https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct and https://huggingface.co/Qwen/Qwen2.5-7B-Instruct.

Table 4: Accuracy and robustness margins of the likelihood and Bayesian multi-answer CoT programs on BoolQ and StrategyQA. ρ estimates the smallest L_2 perturbation to program parameters needed to change the predicted answer, using a first-order approximation. $\rho_{\text{norm}} = \rho/\sqrt{n}$ normalizes by the number of parameters n, expressing the estimated perturbation as a root-mean-square change per parameter. Median and mean summarize per-example margins within each model–dataset pair; larger margins indicate greater local robustness. See §E.5 for details.

Model	**Accuracy(%)**		**Median ρ**		**Mean ρ**		**Median ρ_{norm}**		**Mean ρ_{norm}**	
	Likelihood	Bayes	Likelihood	Bayes	Likelihood	Bayes	Likelihood	Bayes	Likelihood	Bayes
BoolQ										
Qwen-7B	74.34	**83.94**	0.0919	**0.1843**	0.1130	**0.1799**	0.0430	**0.0692**	0.0493	**0.0639**
GPT-4o	86.40	**91.80**	0.0853	**0.1973**	0.0993	**0.2082**	0.0387	**0.0677**	0.0440	**0.0714**
Llama-70B	88.82	**91.20**	0.1083	**0.3131**	0.1003	**0.3001**	0.0447	**0.1063**	0.0428	**0.1030**
StrategyQA										
Qwen-7B	**74.71**	66.12	**0.1956**	0.1195	**0.2231**	0.1434	**0.0853**	0.0412	**0.0917**	0.0479
GPT-4o	**81.03**	76.71	0.1180	**0.2135**	0.1661	**0.2213**	0.0512	**0.0696**	0.0669	**0.0721**
Llama-70B	**79.69**	72.49	0.0895	**0.2504**	0.0812	**0.2407**	0.0355	**0.0788**	0.0332	**0.0778**

Table 5: Dataset-size-weighted average accuracy (%) for verbalized and logit-based Bayesian program. Verbalized answer priors drive the strongest performance.

Model	Pure Verbalized	Logit Step + Verb Prior	Verb Step + Logit Prior	Pure Logits
GPT-4o	**78.19**	77.97	73.40	73.40
Qwen-2.5-7B	65.03	**65.38**	50.88	51.16
Llama-3.3-70B	**76.71**	76.42	70.58	69.95

between θ_{verb} and θ_{logit}. This isolates their effects on predictions within a shared framework of exact probabilistic inference.

Table 5 compares the four combinations of step-weight and answer-prior sources (full results in Appendix Table 10). **Verbalized answer priors yield the highest aggregate accuracy across all three models**, while changing the step-weight source has a smaller effect. Moreover, **logit-based priors often concentrate predictions on a single answer label** on binary tasks; verbalized priors better track the gold answer distribution (see details in Appendix Figure 9). Together, these results show how *controlled program reweighting can identify which confidence signals drive predictive performance and label bias.*

Cross-model Trace Reuse. *Can one model reuse another model's reasoning traces?* This question is relevant to CoT distillation (Magister et al., 2023; Chen et al., 2025b) and on-policy distillation (Agarwal et al., 2024; Zhao et al., 2026), where traces provide training or supervision signals. Copper offers an inference-time probe by separating trace generation from scoring. Using the pure-logit Bayesian variant, we hold a source model S's trace fixed (e.g., one produced by gpt-4o), recompute its weights from a target model T's teacher-forced probabilities on the same text (e.g., llama or qwen), and rerun exact inference. This isolates the effect of changing the scoring model while preserving the reasoning structure, testing whether a CoT can serve as a reusable reasoning artifact across models.

Table 6 shows reuse in both directions. **Weaker models can benefit from stronger-model traces**: for example, Qwen gains 5.7 percentage points when scoring GPT-4o traces and 4.2 points when scoring Llama traces, relative to scoring its own traces. **Stronger models can also extract signal from weaker-model traces**: Llama achieves 65.3% on Qwen's traces, compared with Qwen's own 51.2%, although this remains below Llama's performance on its own traces. These results *support treating CoTs as reusable reasoning artifacts whose utility depends on both their source and the model that scores them.* These results and the programming framework also suggest new directions for distillation and data selection: *step-level scores and sensitivities could help to more effectively identify which traces—and which individual steps—provide useful supervision for a target model.*

Interpretability and Sensitivity Analysis. *Which reasoning steps is a prediction most sensitive to?* Copper answers this through gradients of the final answer marginal with respect to step weights. The Bayesian program analyzes candidate-answer chains jointly, exposing how changes to supporting and

Table 6: Cross-model teacher-forcing results (%). Baseline report each model evaluated on its own generated CoTs. Cross-model T on S denotes target model T scoring source model S's CoTs. Micro Mean denotes dataset-size-weighted average accuracy across tasks.

	Self CoTs			**Cross-model TF**			
Task	GPT	Qwen (Q)	Llama (L)	L on Q	L on GPT	Q on GPT	Q on L
BoolQ	89.67	69.45	90.77	90.55	90.55	69.01	69.01
StrategyQA	70.71	45.66	67.63	68.02	67.63	45.66	45.66
Date	74.71	51.18	62.35	47.06	68.82	74.71	61.18
Geometry	38.36	25.34	27.40	30.14	29.45	30.82	33.56
Salient	64.89	38.17	64.12	29.77	61.07	64.89	63.36
Micro Mean	**73.40**	**51.16**	**69.95**	**65.31**	**70.58**	**56.86**	**55.38**

opposing evidence affect its prediction. Appendix Figure 10 illustrates the parsed multi-answer trace (A), its weights under different scoring models (B), and the resulting sensitivity profiles (C). These provide step-level signals for inspecting program behavior and potentially guiding supervision or distillation.

Different models yield different sensitivities over the same multi-answer CoT. In the illustrative STRATEGYQA-0503 example, we hold the CoT fixed and compare gradients of $P(\text{yes})$. Within the yes-supporting chain, GPT-4o is most sensitive to P2 ($+0.108$), whereas Qwen-2.5-7B-Instruct and Llama-3.3-70B-Instruct are most sensitive to P3 ($+0.149$ and $+0.156$, respectively). Within the no-supporting chain, GPT-4o is most sensitive to P6 (-0.215), while Qwen and Llama peak at P8 (-0.121 and -0.243, respectively). By analyzing both answer chains within one Bayesian program, we can identify which supporting and opposing steps the final prediction is most sensitive to. Importantly: *these sensitivities show how the same CoT can support different program-level explanations of a prediction, depending on the model used to score its steps.*

5 Conclusion

We introduced Copper, a probabilistic programming language for representing CoT traces and reasoning over them through exact, differentiable inference. By separating program design from inference queries, the framework makes modeling assumptions explicit and connects different uses of CoT within a shared semantics. This perspective recovers standard likelihood computations, exposes their limitations, and yields new likelihood bounds. Our case studies demonstrate how it supports both deriving reasoning strategies and testing their robustness, reusability, and step-level sensitivities.

These programs remain simplified proxies for LLM computations, but make reasoning procedures explicit, testable, and amenable to principled refinement. We see Copper as a first step toward a broader language-design agenda for LLM reasoning, in which such formal models support both the development of new reasoning strategies and systematic analysis of their behavior. Future work could use program-level feedback and sensitivity signals to better guide supervision and data selection for learning, while extending the language with tool calls, iteration, and richer state and memory constructs to model more complex reasoning processes.

Acknowledgments

Yu Feng was supported in part by ONR contracts N00014-23-1-2364 and N00014-23-1-2417 and NSF grant IIS-2135581. This work was also funded in part by the DARPA CLARA program under Agreement No. HR0011269E224, and gifts from Cisco Research, Qualcomm, and Amazon.

References

Agarwal, R., Vieillard, N., Zhou, Y., Stanczyk, P., Garea, S. R., Geist, M., and Bachem, O. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

Ahmed, K., Teso, S., Chang, K.-W., Van den Broeck, G., and Vergari, A. Semantic probabilistic layers for neuro-symbolic learning. *Advances in Neural Information Processing Systems*, 35: 29944–29959, 2022.

Ahmed, K., Chang, K.-W., and Broeck, G. V. d. Controllable generation via locally constrained resampling. *arXiv preprint arXiv:2410.13111*, 2024.

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024.

Beurer-Kellner, L., Fischer, M., and Vechev, M. Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1946–1969, 2023.

Bi, Z., Han, K., Liu, C., Tang, Y., and Wang, Y. Forest-of-thought: Scaling test-time compute for enhancing llm reasoning. *arXiv preprint arXiv:2412.09078*, 2024.

Blondel, M. and Roulet, V. The elements of differentiable programming. *arXiv preprint arXiv:2403.14606*, 2024.

Calanzone, D., Teso, S., and Vergari, A. Logically consistent language models via neuro-symbolic integration. *arXiv preprint arXiv:2409.13724*, 2024.

Chavira, M. and Darwiche, A. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

Chen, Q., Qin, L., Liu, J., Peng, D., Guan, J., Wang, P., Hu, M., Zhou, Y., Gao, T., and Che, W. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*, 2025a.

Chen, X., Sun, Z., Wenjin, G., Zhang, M., Chen, Y., Sun, Y., Su, H., Pan, Y., Klakow, D., Li, W., and Shen, X. Unveiling the key factors for distilling chain-of-thought reasoning. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 15094–15119, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.782. URL https://aclanthology.org/2025.findings-acl.782/.

Chen, Z., Ma, J., Zhang, X., Hao, N., Yan, A., Nourbakhsh, A., Yang, X., McAuley, J., Petzold, L. R., and Wang, W. Y. A survey on large language models for critical societal domains: Finance, healthcare, and law. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=upAWnMgpnH. Survey Certification.

Cheng, J., Richardson, K., and Chin, P. Analytica: Soft propositional reasoning for robust and scalable llm-driven analysis. *Proceedings of ICLR*, 2026.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Clark, K. L. Negation as failure. In *Logic and data bases*, pp. 293–322. Springer, 1977.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Cozman, F. G. and Mauá, D. D. The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference. *International Journal of Approximate Reasoning*, 125:218–239, 2020.

Darwiche, A. Sdd: A new canonical representation of propositional knowledge bases. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, pp. 819, 2011.

Darwiche, A. Tractable boolean and arithmetic circuits. In *Neuro-Symbolic Artificial Intelligence: The State of the Art*, pp. 146–172. IOS Press, 2021.

Darwiche, A. and Marquis, P. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

De Raedt, L. and Kimmig, A. Probabilistic (logic) programming concepts. *Machine Learning*, 100: 5–47, 2015.

De Raedt, L., Kimmig, A., and Toivonen, H. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of IJCAI*, pp. 2462–2467, 2007.

De Raedt, L., Kersting, K., Natarajan, S., and Poole, D. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning*, 10(2):1–189, 2016.

Dijkstra, E. W. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

Dohan, D., Xu, W., Lewkowycz, A., Austin, J., Bieber, D., Lopes, R. G., Wu, Y., Michalewski, H., Saurous, R. A., Sohl-Dickstein, J., et al. Language model cascades. *arXiv preprint arXiv:2207.10342*, 2022.

Dries, A., Kimmig, A., Meert, W., Renkens, J., Van den Broeck, G., Vlasselaer, J., and De Raedt, L. Problog2: Probabilistic logic programming. In *Joint european conference on machine learning and knowledge discovery in databases*, pp. 312–315. Springer, 2015.

Feng, Y., Zhou, B., Lin, W., and Roth, D. Bird: A trustworthy bayesian inference framework for large language models. *Proceedings of ICLR*, 2025.

Feng, Y., Weir, N., Bostrom, K., Bayless, S., Cassel, D., Chaudhary, S., Kiesl-Reiter, B., and Rangwala, H. Vericot: Neuro-symbolic chain-of-thought validation via logical consistency checks. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=zHuV3Vatov`.

Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., and De Raedt, L. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

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, P., Geh, R. L., Israel, D., Millstein, T., Richardson, K., and Broeck, G. V. d. Probabilistic programs of thought. *arXiv preprint arXiv:2604.17290*, 2026.

Geva, M., Khashabi, D., Segal, E., Khot, T., Roth, D., and Berant, J. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021a. doi: 10.1162/tacl_a_00370. URL `https://aclanthology.org/2021.tacl-1.21/`.

Geva, M., Khashabi, D., Segal, E., Khot, T., Roth, D., and Berant, J. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021b.

Gordon, A. D., Henzinger, T. A., Nori, A. V., and Rajamani, S. K. Probabilistic programming. In *Future of software engineering proceedings*, pp. 167–181. 2014.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Holtzen, S., Millstein, T., and den Broeck, G. V. Symbolic exact inference for discrete probabilistic programs, 2019. URL https://arxiv.org/abs/1904.02079.

Holtzen, S., Van den Broeck, G., and Millstein, T. Scaling exact inference for discrete probabilistic programs. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–31, 2020.

Huang, Y., Chen, Y., Zhang, H., Li, K., Zhou, H., Fang, M., Yang, L., Li, X., Shang, L., Xu, S., et al. Deep research agents: A systematic examination and roadmap. *arXiv preprint arXiv:2506.18096*, 2025.

Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., Johnston, S., El-Showk, S., Jones, A., Elhage, N., Hume, T., Chen, A., Bai, Y., Bowman, S., Fort, S., Ganguli, D., Hernandez, D., Jacobson, J., Kernion, J., Kravec, S., Lovitt, L., Ndousse, K., Olsson, C., Ringer, S., Amodei, D., Brown, T., Clark, J., Joseph, N., Mann, B., McCandlish, S., Olah, C., and Kaplan, J. Language models (mostly) know what they know, 2022. URL https://arxiv.org/abs/2207.05221.

Katoen, J.-P., Gretz, F., Jansen, N., Kaminski, B. L., and Olmedo, F. Understanding probabilistic programs. In *Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015, Proceedings*, pp. 15–32. Springer, 2015.

Khot, T., Trivedi, H., Finlayson, M., Fu, Y., Richardson, K., Clark, P., and Sabharwal, A. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

Kwiatkowska, M., Norman, G., and Parker, D. Stochastic model checking. In *Formal Methods for Performance Evaluation*, pp. 220–270. Springer, 2007.

Lanham, T., Chen, A., Radhakrishnan, A., Steiner, B., Denison, C., Hernandez, D., Li, D., Durmus, E., Hubinger, E., Kernion, J., et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

Lew, A. K., Huot, M., Staton, S., and Mansinghka, V. K. Adev: Sound automatic differentiation of expected values of probabilistic programs. *Proceedings of the ACM on Programming Languages*, 7(POPL):121–153, January 2023a. ISSN 2475-1421. doi: 10.1145/3571198. URL http://dx.doi.org/10.1145/3571198.

Lew, A. K., Zhi-Xuan, T., Grand, G., and Mansinghka, V. K. Sequential monte carlo steering of large language models using probabilistic programs. *arXiv preprint arXiv:2306.03081*, 2023b.

Li, Z., Huang, J., and Naik, M. Scallop: A language for neurosymbolic programming. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1463–1487, 2023.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Liu, A., Liu, X., Zhao, D., Niepert, M., Liang, Y., and Broeck, G. V. d. Tractable transformers for flexible conditional generation. *arXiv preprint arXiv:2502.07616*, 2025.

Magister, L. C., Mallinson, J., Adamek, J., Malmi, E., and Severyn, A. Teaching small language models to reason. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1773–1781, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.151. URL https://aclanthology.org/2023.acl-short.151/.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

Mo, S. and Xin, M. Tree of uncertain thoughts reasoning for large language models. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 12742–12746. IEEE, 2024.

OpenAI, :, Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., Mądry, A., Baker-Whitcomb, A., Beutel, A., Borzunov, A., Carney, A., Chow, A., Kirillov, A., Nichol, A., Paino, A., Renzin, A., Passos, A. T., Kirillov, A., Christakis, A., Conneau, A., Kamali, A., Jabri, A., Moyer, A., Tam, A., Crookes, A., Tootoochian, A., Tootoonchian, A., Kumar, A., Vallone, A., Karpathy, A., Braunstein, A., Cann, A., Codispoti, A., Galu, A., Kondrich, A., Tulloch, A., Mishchenko, A., Baek, A., Jiang, A., Pelisse, A., Woodford, A., Gosalia, A., Dhar, A., Pantuliano, A., Nayak, A., Oliver, A., Zoph, B., Ghorbani, B., Leimberger, B., Rossen, B., Sokolowsky, B., Wang, B., Zweig, B., Hoover, B., Samic, B., McGrew, B., Spero, B., Giertler, B., Cheng, B., Lightcap, B., Walkin, B., Quinn, B., Guarraci, B., Hsu, B., Kellogg, B., Eastman, B., Lugaresi, C., Wainwright, C., Bassin, C., Hudson, C., Chu, C., Nelson, C., Li, C., Shern, C. J., Conger, C., Barette, C., Voss, C., Ding, C., Lu, C., Zhang, C., Beaumont, C., Hallacy, C., Koch, C., Gibson, C., Kim, C., Choi, C., McLeavey, C., Hesse, C., Fischer, C., Winter, C., Czarnecki, C., Jarvis, C., Wei, C., Koumouzelis, C., Sherburn, D., Kappler, D., Levin, D., Levy, D., Carr, D., Farhi, D., Mely, D., Robinson, D., Sasaki, D., Jin, D., Valladares, D., Tsipras, D., Li, D., Nguyen, D. P., Findlay, D., Oiwoh, E., Wong, E., Asdar, E., Proehl, E., Yang, E., Antonow, E., Kramer, E., Peterson, E., Sigler, E., Wallace, E., Brevdo, E., Mays, E., Khorasani, F., Such, F. P., Raso, F., Zhang, F., von Lohmann, F., Sulit, F., Goh, G., Oden, G., Salmon, G., Starace, G., Brockman, G., Salman, H., Bao, H., Hu, H., Wong, H., Wang, H., Schmidt, H., Whitney, H., Jun, H., Kirchner, H., de Oliveira Pinto, H. P., Ren, H., Chang, H., Chung, H. W., Kivlichan, I., O'Connell, I., O'Connell, I., Osband, I., Silber, I., Sohl, I., Okuyucu, I., Lan, I., Kostrikov, I., Sutskever, I., Kanitscheider, I., Gulrajani, I., Coxon, J., Menick, J., Pachocki, J., Aung, J., Betker, J., Crooks, J., Lennon, J., Kiros, J., Leike, J., Park, J., Kwon, J., Phang, J., Teplitz, J., Wei, J., Wolfe, J., Chen, J., Harris, J., Varavva, J., Lee, J. G., Shieh, J., Lin, J., Yu, J., Weng, J., Tang, J., Yu, J., Jang, J., Candela, J. Q., Beutler, J., Landers, J., Parish, J., Heidecke, J., Schulman, J., Lachman, J., McKay, J., Uesato, J., Ward, J., Kim, J. W., Huizinga, J., Sitkin, J., Kraaijeveld, J., Gross, J., Kaplan, J., Snyder, J., Achiam, J., Jiao, J., Lee, J., Zhuang, J., Harriman, J., Fricke, K., Hayashi, K., Singhal, K., Shi, K., Karthik, K., Wood, K., Rimbach, K., Hsu, K., Nguyen, K., Gu-Lemberg, K., Button, K., Liu, K., Howe, K., Muthukumar, K., Luther, K., Ahmad, L., Kai, L., Itow, L., Workman, L., Pathak, L., Chen, L., Jing, L., Guy, L., Fedus, L., Zhou, L., Mamitsuka, L., Weng, L., McCallum, L., Held, L., Ouyang, L., Feuvrier, L., Zhang, L., Kondraciuk, L., Kaiser, L., Hewitt, L., Metz, L., Doshi, L., Aflak, M., Simens, M., Boyd, M., Thompson, M., Dukhan, M., Chen, M., Gray, M., Hudnall, M., Zhang, M., Aljubeh, M., Litwin, M., Zeng, M., Johnson, M., Shetty, M., Gupta, M., Shah, M., Yatbaz, M., Yang, M. J., Zhong, M., Glaese, M., Chen, M., Janner, M., Lampe, M., Petrov, M., Wu, M., Wang, M., Fradin, M., Pokrass, M., Castro, M., de Castro, M. O. T., Pavlov, M., Brundage, M., Wang, M., Khan, M., Murati, M., Bavarian, M., Lin, M., Yesildal, M., Soto, N., Gimelshein, N., Cone, N., Staudacher, N., Summers, N., LaFontaine, N., Chowdhury, N., Ryder, N., Stathas, N., Turley, N., Tezak, N., Felix, N., Kudige, N., Keskar, N., Deutsch, N., Bundick, N., Puckett, N., Nachum, O., Okelola, O., Boiko, O., Murk, O., Jaffe, O., Watkins, O., Godement, O., Campbell-Moore, O., Chao, P., McMillan, P., Belov, P., Su, P., Bak, P., Bakkum, P., Deng, P., Dolan, P., Hoeschele, P., Welinder, P., Tillet, P., Pronin, P., Tillet, P., Dhariwal, P., Yuan, Q., Dias, R., Lim, R., Arora, R., Troll, R., Lin, R., Lopes, R. G., Puri, R., Miyara, R., Leike, R., Gaubert, R., Zamani, R., Wang, R., Donnelly, R., Honsby, R., Smith, R., Sahai, R., Ramchandani, R., Huet, R., Carmichael, R., Zellers, R., Chen, R., Chen, R., Nigmatullin, R., Cheu, R., Jain, S., Altman, S., Schoenholz, S., Toizer, S., Miserendino, S., Agarwal, S., Culver, S., Ethersmith, S., Gray, S., Grove, S., Metzger, S., Hermani, S., Jain, S., Zhao, S., Wu, S., Jomoto, S., Wu, S., Shuaiqi, Xia, Phene, S., Papay, S., Narayanan, S., Coffey, S., Lee, S., Hall, S., Balaji, S., Broda, T., Stramer, T., Xu, T., Gogineni, T., Christianson, T., Sanders, T., Patwardhan, T., Cunninghman, T., Degry, T., Dimson, T., Raoux, T., Shadwell, T., Zheng, T., Underwood, T., Markov, T., Sherbakov, T., Rubin, T., Stasi, T., Kaftan, T., Heywood, T., Peterson, T., Walters, T., Eloundou, T., Qi, V., Moeller, V., Monaco, V., Kuo, V., Fomenko, V., Chang, W., Zheng, W., Zhou, W., Manassra, W., Sheu, W., Zaremba, W., Patil, Y., Qian, Y., Kim, Y., Cheng, Y., Zhang, Y., He, Y., Zhang, Y., Jin, Y., Dai, Y., and Malkov, Y. Gpt-4o system card, 2024. URL `https://arxiv.org/abs/2410.21276`.

Pfeffer, A. Practical probabilistic programming. In *International Conference on Inductive Logic Programming*, pp. 2–3. Springer, 2010.

Quan, X., Valentino, M., Dennis, L. A., and Freitas, A. Verification and refinement of natural language explanations through LLM-symbolic theorem proving. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 2933–2958, Miami, Florida, USA, November 2024. Association for Computational

Linguistics. doi: 10.18653/v1/2024.emnlp-main.172. URL https://aclanthology.org/2024.emnlp-main.172/.

Richardson, K., Srikumar, V., and Sabharwal, A. Understanding the logic of direct preference alignment through logic. *Proceedings of ICML*, 2025.

Richardson, K., Anderson, C., Balakrishnan, P., Ban, T., Ladia, D., Gupta, A., and Hudspeth, M. From token probabilities to semantic constraints: Towards declarative probabilistic evaluation of language models. In *Proceedings of EMNLP*, 2026.

Saad, F. A., Rinard, M. C., and Mansinghka, V. K. Sppl: probabilistic programming with fast exact symbolic inference. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pp. 804–819, 2021.

Sclar, M., Choi, Y., Tsvetkov, Y., and Suhr, A. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=RIu5lyNXjT.

Setlur, A., Nagpal, C., Fisch, A., Geng, X., Eisenstein, J., Agarwal, R., Agarwal, A., Berant, J., and Kumar, A. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

Stechly, K., Valmeekam, K., and Kambhampati, S. Chain of thoughtlessness? an analysis of cot in planning. *Advances in Neural Information Processing Systems*, 37:29106–29141, 2024.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., and Wei, J. Challenging big-bench tasks and whether chain-of-thought can solve them, 2022. URL https://arxiv.org/abs/2210.09261.

Tjoa, R., Garg, P., Goldstein, H., Millstein, T., Pierce, B. C., and Van den Broeck, G. Tuning random generators: Property-based testing as probabilistic programming. *Proceedings of the ACM on Programming Languages*, 9(OOPSLA2):894–921, October 2025. ISSN 2475-1421. doi: 10.1145/3763082. URL http://dx.doi.org/10.1145/3763082.

Turpin, M., Michael, J., Perez, E., and Bowman, S. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

Van de Meent, J.-W., Paige, B., Yang, H., and Wood, F. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

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.

Winskel, G. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

Wong, L., Grand, G., Lew, A. K., Goodman, N. D., Mansinghka, V. K., Andreas, J., and Tenenbaum, J. B. From word models to world models: Translating from natural language to the probabilistic language of thought. *arXiv preprint arXiv:2306.12672*, 2023.

Wu, M., Zhu, T., Han, H., Zhang, X., Shao, W., and Chen, W. Chain-of-tools: Utilizing massive unseen tools in the cot reasoning of frozen language models. *arXiv preprint arXiv:2503.16779*, 2025.

Xiang, V., Snell, C., Gandhi, K., Albalak, A., Singh, A., Blagden, C., Phung, D., Rafailov, R., Lile, N., Mahan, D., et al. Towards system 2 reasoning in llms: Learning how to think with meta chain-of-thought. *arXiv preprint arXiv:2501.04682*, 2025.

Xiong, M., Hu, Z., Lu, X., Li, Y., Fu, J., He, J., and Hooi, B. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. *Proceedings of ICLR*, 2024.

Xu, J., Zhang, Z., Friedman, T., Liang, Y., and Broeck, G. A Semantic Loss Function for Deep Learning with Symbolic Knowledge. In *Proceedings of ICML*, pp. 5498–5507, 2018.

Yang, L., Yu, Z., Zhang, T., Cao, S., Xu, M., Zhang, W., Gonzalez, J. E., and Cui, B. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544, 2024.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Ye, X., Chen, Q., Dillig, I., and Durrett, G. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36:45548–45580, 2023.

Yeo, E., Tong, Y., Niu, M., Neubig, G., and Yue, X. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*, 2025.

Yidou-Weng, G., Wang, B., and Van den Broeck, G. Trace back from the future: A probabilistic reasoning approach to controllable language generation. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Zhang, H., Dang, M., Peng, N., and Van den Broeck, G. Tractable control for autoregressive language generation. In *International Conference on Machine Learning*, pp. 40932–40945. PMLR, 2023.

Zhao, S., Xie, Z., Liu, M., Huang, J., Pang, G., Chen, F., and Grover, A. Self-distilled reasoner: On-policy self-distillation for large language models, 2026. URL `https://arxiv.org/abs/2601.18734`.

Zhong, W., Liu, C., Wu, Y., Tan, B., Sun, C., Wang, Y., Liu, X., and Kuang, K. P2s: Probabilistic process supervision for general-domain reasoning question answering. *arXiv preprint arXiv:2601.20649*, 2026.

A Limitations and Future Work

Our approach has the following limitations. Copper depends on reliably parsing reasoning steps and dependencies from model outputs, and its programs remain simplified proxies: exact inference within a program does not establish that it faithfully captures the LLM's reasoning or that its weights measure step correctness. Differentiable aggregation introduces additional modeling choices, while exact inference may become expensive as program structure grows.

Our case studies establish feasibility across selected models and tasks rather than broad empirical superiority. Bayesian multi-answer CoT sometimes sacrifices accuracy, and its robustness margins are local, first-order estimates sensitive to the chosen parameterization; they do not certify robustness to textual or semantic changes. Cross-model reuse and sensitivity analyses suggest directions for distillation and supervision, but we do not evaluate their training benefits.

B Extended Related Work

CoT and Variants Our work takes inspiration from the literature on CoT and its variants that couple CoT with advanced search techniques (Yao et al., 2023; Mo & Xin, 2024; Besta et al., 2024; Bi et al., 2024; Yang et al., 2024), external tools (Ye et al., 2023; Gao et al., 2023; Wu et al., 2025) and other forms of test-time scaling (Khot et al., 2022; Wang et al., 2022; Feng et al., 2025; Cheng et al., 2026). In contrast, our work focuses not on a specific inference strategy but on a general formalism for CoT. As such, our work is closest to Dohan et al. (2022), who propose a related graphical model view of CoT; however, their work is primarily conceptual, whereas we define an explicit language for CoT with a formal semantics and use it to derive new formal and empirical results. We also follow other work such as Beurer-Kellner et al. (2023); Quan et al. (2024); Feng et al. (2026) that attempts to ground LLM inference in programming.

Probabilistic and Differential Programming Our technical framework draws on probabilistic programming (Pfeffer, 2010; Gordon et al., 2014; Katoen et al., 2015; Van de Meent et al., 2018), statistical relational learning (De Raedt et al., 2016), and differentiable programming (Blondel & Roulet, 2024). We build our approach and language most directly on discrete probabilistic programming (Holtzen et al., 2020; Saad et al., 2021), where programs define distributions over finite executions via random choices, branching, conditioning, and return values. Exact inference in this setting can be performed by reducing probabilistic queries to weighted model counting over compiled logical representations (Chavira & Darwiche, 2008; Darwiche & Marquis, 2002). As such, our focus on exact inference differs from other approaches that mix probabilistic programming with approximate inference and LLMs such as Wong et al. (2023); Lew et al. (2023b).

While we use imperative-style program representations, our work work takes inspiration from probabilistic logic programming and other declarative-style approaches to probabilistic programming (De Raedt et al., 2007; Dries et al., 2015; Manhaeve et al., 2018; Li et al., 2023; Cozman & Mauá, 2020). Indeed, we note that the Boolean nature of our programs and our program encoding in Figure 3c operationalizes a Clark-style completion semantics (Clark, 1977), however our language differs in having more complex differentiable operators.

Tractable Probabilistic Modeling for LLMs Finally, our work takes inspiration from work on using tractable probabilistic models for LLM modeling problems (Ahmed et al., 2024; Zhang et al., 2023; Liu et al., 2025; Yidou-Weng et al., 2025; Garg et al., 2026), and other approaches involving tractable neuro-symbolic inference (Xu et al., 2018; Ahmed et al., 2022; Richardson et al., 2025; Calanzone et al., 2024). In contrast to this work, much of which centers around constrained decoding problems, we focus on applications of probabilistic modeling of CoT.

C Syntax of Copper and Compilation to Weighted Boolean Formula

Figure 6 provides the formal syntax of Copper and Figure 7 provides the formal compilation rules to weighted Boolean formulae. To perform inference over the chain-of-thought programs represented by Copper, given a set of defined variables, we compile it to a weighted Boolean formula. Formally, the compilation judgement for Copper programs have the form $\Gamma, \Sigma \vdash p \rightsquigarrow (w, g, e)$. Here, Γ refers to a set of variables, Σ refers to a mapping from strings to variables, weights w refers to a mapping from variables to probabilities, guards g refers to a mapping from tuples of string and deterministic

$$
\begin{array}{rlll}
\textit{Values} & v & ::= & \texttt{T} \mid \texttt{F} \mid x \\
\textit{Boolean Expressions} & b & ::= & v \mid v \vee b \mid v \wedge b \mid \neg b \\
\textit{Statements} & s & ::= & x =\!\sim \texttt{Flip}(\theta) \\
 & & \mid & x_1, \ldots x_n =\!\sim \texttt{Categorical}(\theta_1, \theta_2, \ldots, \theta_n) \\
 & & \mid & x =\!\sim \texttt{Factor}(f_m(\theta_1, \ldots, \theta_m)) \mid \texttt{observe}(b) \\
 & & \mid & \texttt{if}\ b\ \texttt{then}\ p\ \texttt{else}\ p \mid \textbf{return}\ t \mid \textbf{raise}\ t \\
\textit{Program} & p & ::= & s \mid s; p
\end{array}
$$

Figure 6: The syntax of Copper. The metavariable x ranges over variable names, θ ranges over real numbers in the range $[0, 1]$, t ranges over strings and f_m ranges over differentiable functions $[0, 1]^m \to [0, 1]$

$$\Gamma, \Sigma \vdash x =\!\sim \texttt{Flip}(\theta) \rightsquigarrow (\{x : \theta, \neg x : 1 - \theta\}, \emptyset, \texttt{true})$$

$$\frac{\sum_{i=1}^{n} \theta_i = 1}{\begin{array}{l}\Gamma, \Sigma \vdash x_1, \ldots, x_n \sim \texttt{Categorical}(\theta_1, \ldots, \theta_n) \\ \rightsquigarrow (\{x_1 : \theta_1, \ldots, x_n : \theta_n, \neg x_1 : 1, \ldots, \neg x_n : 1\},\ \emptyset,\ \texttt{exact_one}(x_1, \ldots, x_n))\end{array}}$$

$$\Gamma, \Sigma \vdash x =\!\sim \texttt{Factor}(f_m(\theta_1, \ldots \theta_m)) \rightsquigarrow (\{x : f_m(\theta_1, \ldots \theta_m), \neg x : 1 - f_m(\theta_1, \ldots \theta_m)\}, \emptyset, \texttt{true})$$

$$\frac{\exists x\ \ (t, x) \in \Sigma}{\Gamma, \Sigma \vdash \textbf{return}\ t \rightsquigarrow (\emptyset, \{(t, x) : \texttt{true}\}, \texttt{true})} \qquad \frac{\nexists x\ \ (t, x) \in \Sigma \qquad \texttt{fresh}\ x}{\Gamma, \Sigma \vdash \textbf{return}\ t \rightsquigarrow (\emptyset, \{(t, x) : \texttt{true}\}, \texttt{true})}$$

$$\frac{\exists x\ \ (t, x) \in \Sigma}{\Gamma, \Sigma \vdash \textbf{raise}\ t \rightsquigarrow (\emptyset, \{(t, x) : \texttt{true}\}, \texttt{true})} \qquad \frac{\nexists x\ \ (t, x) \in \Sigma \qquad \texttt{fresh}\ x}{\Gamma, \Sigma \vdash \textbf{raise}\ t \rightsquigarrow (\emptyset, \{(t, x) : \texttt{true}\}, \texttt{true})}$$

$$\frac{\Gamma, \Sigma \vDash b}{\Gamma, \Sigma \vdash \texttt{observe}(b) \rightsquigarrow (\emptyset, \emptyset, b)}$$

$$\frac{\Gamma \vDash b \qquad \Gamma, \Sigma \vdash p_1 \rightsquigarrow (w_1, g_1, e_1) \qquad \Gamma, \Sigma \cup dom(g_1) \vdash p_2 \rightsquigarrow (w_2, g_2, e_2)}{\Gamma, \Sigma \vdash \texttt{if}\ b\ \texttt{then}\ p_1\ \texttt{else}\ p_2 \rightsquigarrow (w_1 \cup w_2, (b \wedge g_1) \sqcup (\neg b \wedge g_2), e_1 \wedge e_2)}$$

$$\frac{\Gamma, \Sigma \vdash s \rightsquigarrow (w_1, g_1, e_1) \qquad \Gamma \cup dom(w_1), \Sigma \cup dom(g_1) \vdash p \rightsquigarrow (w_2, g_2, e_2)}{\Gamma, \Sigma \vdash s; p \rightsquigarrow (w_1 \cup w_2, g_1 \sqcup g_2, e_1 \wedge e_2)}$$

Figure 7: Compilation of Copper programs to weighted Boolean formulas.

variables to Boolean formulae, evidence e refers to a Boolean formulae on which the probability distribution is to be conditioned. These compilation rules closely follow the rules provided in existing work for compiling discrete PPLs to weighted Boolean formulae (Holtzen et al., 2019). In fact, it is simpler for Copper as it assumes that all variables are assigned distributions only once and all observations are global.

Before we describe the compilation judgement, we first define few helper functions.

- For a mapping w, we define its domain $dom(w) := \{v \mid \exists \theta\ (v, \theta) \in w\}$.

- Given Boolean variables, $x_1, x_2, \ldots x_n$, we define the constraint of exactly one of them being true as $\texttt{exact_one}(x_1, x_2, \ldots x_n)$

$$\texttt{exact_one}(x_1, x_2, \ldots x_n) := \begin{cases} x_1 & n = 1 \\ (x_1 \wedge \bigwedge_{i=2}^{n} \neg x_i) \\ \vee(\neg x_1 \wedge \texttt{exact_one}(x_2, \ldots x_n)) & n \neq 1 \end{cases}$$

- If for all variables x occurring in a Boolean formula b, $x \in \Gamma$, then we say that b is well defined in the context Γ, i.e. $\Gamma \vDash b$.
- Given a mapping g from variables to Boolean formulas, and a Boolean formula b, the conjunction $b \wedge g := \{((t, x), b \wedge f) \mid ((t, x), f) \in g\}$.
- Given two mappings g_1 and g_2, their union is defined as

$$(g_1 \sqcup g_2)((t, x)) := \begin{cases} g_1((t, x)) \vee g_2((t, x)) & (t, x) \in dom(g_1) \cap dom(g_2), \\ g_1((t, x)) & (t, x) \in dom(g_1) \setminus dom(g_2), \\ g_2((t, x)) & (t, x) \in dom(g_2) \setminus dom(g_1). \end{cases}$$

- The construct $\texttt{fresh}\ \ x$ retrieves a fresh variable name x.

Now with the compilation rules, if $\emptyset, \emptyset \vdash p \rightsquigarrow (w, g, e)$, then $\nu_{\neg\text{dec}} = dom(g)$, $guards(v) = g(v)$.

D Formal Results

Proposition 1 (*CoT likelihood*)**.** *Under the LM likelihood parameterization,* F_a *with* $k = 1$ *satisfies*

$$\mathbb{P}_{\mathsf{F}_a}(\neg\texttt{ValueError}; \theta) = \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_1} \mid Q).$$

Proof. Let the following formula be the guard formula for reaching answer a_1:

$$G_1 = \bigwedge_{j=1}^{n_1} \mathsf{P}_{1,j}$$

which is satisfied exactly when $\texttt{ValueError}$ is not encountered. Hence, under the semantics of the $\texttt{Flip}$ we have:

$$\mathbb{P}_{\mathsf{F}_a}(\neg\texttt{ValueError}; \theta) = \mathbb{P}_{\mathsf{F}_a}(G_1; \theta) = \prod_{j=1}^{n_1} \theta_{1,j}.$$

Substituting the assumed model conditionals and applying the chain rule then yields

$$\prod_{j=1}^{n_1} \theta_{1,j} = \prod_{j=1}^{n_1} \mathbb{P}_{\mathcal{M}}(s_{1,j} \mid s_{1,<j}, Q) = \mathbb{P}_{\mathcal{M}}(s_{1,1}, \ldots, s_{1,n_1} \mid Q) = \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_1} \mid Q).$$

which establishes the correspondence. □

Proposition 2 (*Multi-trace likelihood*)**.** *Under the LM likelihood parameterization, for* $k > 1$ *traces:* ***(i, noisy-or)*** *independent branches give* $\mathbb{P}_{\mathsf{F}_a}(\neg\texttt{ValueError}; \theta) = 1 - \prod_{i=1}^{k}(1 - \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_i} \mid Q))$*;* ***(ii, total mass)*** *for distinct, mutually exclusive complete traces,* $\mathbb{P}_{\mathsf{F}_c}(\neg\texttt{ValueError}; \theta) = \sum_{i=1}^{k} \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_i} \mid Q)$*;* ***(iii, latent trace)*** *if additionally all* $a_i = a$*, this sum equals* $\mathbb{P}_{\mathsf{F}_c}(\text{return } a; \theta)$ *and gives a finite approximation to the latent-trace marginal* $\mathbb{P}_{\text{CoT}}(a \mid Q)$*.*

Proof. As above, under the LM likelihood parameterization, the chain rule gives, for each trace i, $\prod_{j=1}^{n_i} \theta_{i,j} = \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_i} \mid Q)$.

(i, noisy-or). In F_a, let $G_i = \bigwedge_{j=1}^{n_i} \mathsf{P}_{i,j}$ be the success condition for trace i. The program avoids `ValueError` exactly when at least one G_i holds. Since these guards involve disjoint sets of independent decision variables, they are independent. Hence

$$\mathbb{P}_{\mathsf{F}_a}(\neg\texttt{ValueError};\theta) = \mathbb{P}_{\mathsf{F}_a}\left(\bigvee_{i=1}^{k} G_i;\theta\right) = 1 - \prod_{i=1}^{k}\left(1 - \mathbb{P}_{\mathsf{F}_a}(G_i;\theta)\right)$$
$$= 1 - \prod_{i=1}^{k}\left(1 - \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_i} \mid Q)\right).$$

The ordering of the branches determines which answer is returned when several guards hold, but does not affect the probability of avoiding `ValueError`.

(ii, total mass). In F_c, the categorical indicators $A_1, \ldots, A_k, A_\neg$ are mutually exclusive, with $\mathbb{P}_{\mathsf{F}_c}(A_i;\theta) = \prod_{j=1}^{n_i} \theta_{i,j}$. The assumed mutual exclusivity of the complete LM traces ensures that these masses sum to at most one, so the residual mass assigned to $A_\neg$ is nonnegative. The program avoids `ValueError` exactly when one of $A_1, \ldots, A_k$ holds. Therefore,

$$\mathbb{P}_{\mathsf{F}_c}(\neg\texttt{ValueError};\theta) = \mathbb{P}_{\mathsf{F}_c}\left(\bigvee_{i=1}^{k} A_i;\theta\right) = \sum_{i=1}^{k} \mathbb{P}_{\mathsf{F}_c}(A_i;\theta) = \sum_{i=1}^{k} \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_i} \mid Q).$$

(iii, latent trace). If all $a_i = a$, every non-residual categorical outcome returns a. Consequently,

$$\mathbb{P}_{\mathsf{F}_c}(\texttt{return a};\theta) = \sum_{i=1}^{k} \mathbb{P}_{\mathcal{M}}(\mathbf{T}^{a_i} \mid Q).$$

Writing each complete trace as $\mathbf{T}^{a_i} = (\mathbf{T}_i, a)$, this sum equals $\sum_{i=1}^{k} \mathbb{P}_{\mathcal{M}}(A = a, \mathbf{T} = \mathbf{T}_i \mid Q)$, the restriction of the latent-trace marginal $\mathbb{P}_{\mathrm{CoT}}(a \mid Q)$ to the selected traces. It therefore gives the claimed finite approximation. □

Proposition 3 (*Novel likelihood bounds*)**.** *Under the LM likelihood parameterization, for a trace and answer $\mathbf{T}^a$ with n steps:*

$$\underbrace{\mathbb{P}_{\mathsf{F}_{d=1}}(\texttt{return a};\theta)}_{\vee} \geq \mathbb{P}_{\mathsf{F}_{d=2}}(\texttt{return a};\theta) \geq \cdots \geq \underbrace{\mathbb{P}_{\mathsf{F}_{d=n}}(\texttt{return a};\theta)}_{\wedge} = \mathbb{P}_{\mathcal{M}}(\mathbf{T}^a \mid Q).$$

Proof. We follow the argument of Richardson et al. (2026). Let G_d be the Boolean condition `sum(P) >= d` guarding the return in F_d. If at least $d+1$ step variables are true, then at least d are true. Hence $G_n \models G_{n-1} \models \cdots \models G_1$. All programs use the same decision variables and weights, so their return events are nested. By monotonicity of probability (see Xu et al. (2018)), it follows then that:

$$\mathbb{P}_{\mathsf{F}_{d=1}}(\texttt{return a};\theta) \geq \cdots \geq \mathbb{P}_{\mathsf{F}_{d=n}}(\texttt{return a};\theta).$$

Finally, G_n requires all n step variables to be true and is therefore equivalent to their conjunction. By Proposition 1, $\mathbb{P}_{\mathsf{F}_{d=n}}(\texttt{return a};\theta) = \mathbb{P}_{\mathcal{M}}(\mathbf{T}^a \mid Q)$, establishing the result. □

Proposition 4 (*Length confound*)**.** *Let t_1, t_2 be two CoT chains with lengths n_1, n_2, log-likelihoods $\ell_1, \ell_2 < 0$, and mean log-likelihoods $\bar{\ell}_i = \ell_i / n_i$. If $\bar{\ell}_1 > \bar{\ell}_2$, then likelihood prefers t_1 over t_2 only under the following condition:*

$$\ell_1 > \ell_2 \iff \frac{n_1}{n_2} < \frac{|\bar{\ell}_2|}{|\bar{\ell}_1|}.$$

Proof. Raw likelihood prefers t_1 iff $\ell_1 > \ell_2$. Since $\bar{\ell}_i = \ell_i / n_i$, we have $\ell_i = n_i \bar{\ell}_i$, so this is equivalent to $n_1\bar{\ell}_1 > n_2\bar{\ell}_2$. Because log-likelihoods are negative, write $\bar{\ell}_i = -|\bar{\ell}_i|$. Then $-n_1|\bar{\ell}_1| > -n_2|\bar{\ell}_2|$. Multiplying by -1 then reverses the inequality: $n_1|\bar{\ell}_1| < n_2|\bar{\ell}_2|$. Dividing both sides by $n_2|\bar{\ell}_1| > 0$ gives

$$\frac{n_1}{n_2} < \frac{|\bar{\ell}_2|}{|\bar{\ell}_1|}.$$

Therefore

$$\ell_1 > \ell_2 \quad \iff \quad \frac{n_1}{n_2} < \frac{|\bar{\ell}_2|}{|\bar{\ell}_1|}.$$

□

E Experiment

E.1 Configuration axes for the Bayesian multi-answer CoT framework

We present configuration axes for the Bayesian multi-answer CoT framework in Table 7.

E.2 Data Statistics

We present data statistics in Table 8.

E.3 Implementation Details

We query Qwen-2.5-7B-Instruct and Llama-3.3-70B-Instruct using the Together API, and GPT-4o using the OpenAI API. For cross-model teacher-forcing analysis, we run inference with vLLM on a node with eight NVIDIA A6000 48GB GPUs.

E.4 Per-Task Direct-Answer Accuracy

We evaluate how Copper supports different CoT prompting strategies and compare their predictive performance with commonly used prompting methods across five tasks and three models (Table 9). We instantiate two strategies within Copper: **standard CoT prompting**, which generates a single reasoning chain represented by the **Likelihood** program, and **multi-answer CoT prompting**, which generates reasoning for each candidate answer represented by the **Bayesian** program. Both programs select the answer with the largest marginal computed through; the Bayesian program additionally incorporates an answer prior parsed from the trace.[3]

For comparison, we evaluate four prompting baselines: **Direct**, zero-shot answering without intermediate reasoning; **CoT**, a single reasoning chain followed by answer extraction (Wei et al., 2022); **Combined CoT**, one rollout that considers all candidate answers before producing a final answer; and **Tree-of-Thoughts**, branched search with self-evaluation over intermediate states (Yao et al., 2023).

Table 9 reports per-task accuracy (%) for the four prompting baselines and our two probabilistic-program variants across the five evaluation tasks and three models. The Bayesian probabilistic programs have an average accuracy gap of 4.3% relative to commonly used baselines, but provide explicit analytical structure for examining how reasoning, answer priors, and their aggregation shape the final prediction.

The Bayesian variant performs best on several binary or lower-choice settings, including Qwen on BoolQ and Salient Translation and GPT-4o on BoolQ, but is weaker on some multiple-choice tasks where the answer prior has larger influence.

To further demonstrate the diagnostic value of the explicit program structure, we analyze cases across all datasets in which plain CoT is correct but the Bayesian program is wrong, using Claude Opus 5 as an LLM judge. Within the program, the prediction is computed solely from the declared answer prior and parsed CoT evidence, making the program-level inference path fully inspectable. We identify two major, non-exhaustive failure modes:

- Incorrect elicited evidence: the multi-answer chains provide stronger evidence for an incorrect candidate, causing the program to assign it the highest posterior while revealing that the error originates in the elicited reasoning.
- Correct evidence overridden by the prior: the CoT evidence favors the gold answer, but not enough to overcome the prior, leaving the incorrect candidate with a posterior advantage of at most 5%.

E.5 Robustness Margin

Score gap. For a binary CoT program with parameter vector θ (the per-Flip weights of every step and the answer prior), let $w_{\text{yes}}(\theta)$ and $w_{\text{no}}(\theta)$ be the marginal masses returned by symbolic inference.

[3] Unless otherwise specified, we use `#calls=1`, `pool_mode=logitpool`, `answer_aggregation_fn=cal`, `statement_aggregation_fn=mean`, and `step_dependency_mode=flat`.

Dimension	Options	Description
Trace elicitation (Fig. 1A)		
`#calls`	**1** N	Number of forward LLM calls used to obtain the trace. **1**: a single call argues every candidate. N: one call per candidate.
verbal conf.	**enabled** disabled	Whether per-step "(confidence: $X\%$)" tags and per-candidate "initial confidence: $X\%$" tags are parsed from the trace. **enabled**: parsed and used as an alternative source for step weights and the answer prior. disabled: not parsed.
verbal step deps.	enabled **disabled**	Whether parsed "(ref. i, j)" premise references are elicited and used. enabled: references are parsed; only has effect when `step_dependency_mode`$\neq$`flat`. **disabled**: not elicited.
Program construction (Fig. 1B)		
`answer_aggregation_fn`	c p **cal** mean max min	Per-candidate prior `Flip` attribute fed into `Mixture`. c: verbalized "initial confidence: $X\%$" only. p: logit logprob from a direct-ranking call over the answer letters only. **cal**: calibrated combination of the verbalized and logit slots. mean/max/min: arithmetic combination of the two slots. If neither slot is populated, the prior reduces to uniform $1/N$.
`statement_aggregation_fn`	c p **mean** max min pool	Per-step `Flip` attribute used inside the pool. c: verbalized "(confidence: $X\%$)" only. p: token logprob only. **mean**/max/min: arithmetic combination of the verbalized-confidence and token-logprob slots. pool: `LogitPool` of the two slots.
`pool_mode`	**logitpool**	How per-step factors are combined into one evidence factor per candidate. **logitpool**: unweighted logit average.
`step_dependency_mode`	**flat** gated	How per-step `Flip`s condition on each other. **flat**: independent within a candidate. gated: every `Flip` conditions on the conjunction of its referenced steps.
alt. probs	**disabled** single multi-model	Optional secondary probability slot on each `Flip`, populated from a different probability source than the primary slot. **disabled**: only one slot is used. single: cross-model logits from one alternate LM. multi-model: averaged across multiple LMs.

Table 7: Configuration axes for the Bayesian multi-answer CoT framework. Trace-elicitation rows control how the LLM produces the raw text and which auxiliary annotations are extracted from it; program-construction rows control how the parsed pieces are assembled into the probabilistic program. Defaults are bolded; unless noted otherwise, all reported numbers use these defaults (`#calls=1`, `answer_aggregation_fn=cal`, `statement_aggregation_fn=mean`, `pool_mode=logitpool`, `step_dependency_mode=flat`).

Dataset	Format	Val	Test	License	Description
BoolQ	Binary	200	500	CC BY-SA 3.0	Reading-comprehension yes/no questions over short passages. We sample 200 validation and 500 test examples from the BoolQ validation split. Source: huggingface.co/datasets/google/boolq.
StrategyQA	Binary	200	687	MIT	Open-domain multi-hop yes/no questions whose answers compose implicit commonsense facts. We use the full ChilleD/StrategyQA test split for evaluation and a 200-example subsample for validation. Source: huggingface.co/datasets/ChilleD/StrategyQA.
Date Understanding	MCQ (6)	50	200	MIT	BIG-Bench Hard task. Date arithmetic / calendar reasoning over short prompts; the model picks among six candidate dates. Source: BIG-Bench-Hard.
Geometric Shapes	MCQ (11)	50	200	MIT	BIG-Bench Hard task. Given an SVG-style path, choose the resulting shape from eleven options. Source: BIG-Bench-Hard.
Salient Translation Error Detection	MCQ (6)	50	200	MIT	BIG-Bench Hard task. Identify the type of translation error among six labels: modifiers, numerical values, named entities, negation, facts, and dropped content. Source: BIG-Bench-Hard.

Table 8: Dataset statistics, licenses, and descriptions. Val splits are used for variant selection (§4.2); test splits are used for all reported accuracies.

The *score gap* is the log-ratio

$$G(\theta) \;=\; \log \frac{w_{\text{yes}}(\theta)}{w_{\text{no}}(\theta)}, \tag{1}$$

so $\operatorname{sign}(G(\theta))$ is the program's predicted answer.

Robustness margin $\rho(\theta)$. We define the (exact) robustness margin as the smallest L_2 perturbation of the parameters that flips the prediction:

$$\rho(\theta) \;=\; \inf_{\theta' \in \mathbb{R}^d} \big\|\theta' - \theta\big\|_2 \quad \text{s.t.} \quad \operatorname{sign}\big(G(\theta')\big) \;\neq\; \operatorname{sign}\big(G(\theta)\big). \tag{2}$$

Computing the constraint exactly is intractable because G is a non-linear function of θ obtained by analytical inference over the compiled SDD circuit. We therefore use the standard first-order linearization of G around θ, which gives the closed-form distance to the decision boundary $\{\theta' : G(\theta') = 0\}$:

$$\rho(\theta) \;\approx\; \frac{\big|\, G(\theta) \,\big|}{\big\|\nabla_\theta G(\theta)\big\|_2}. \tag{3}$$

The numerator is the model's current confidence (margin away from the decision boundary in log-space), while the denominator is the local slope of G: questions where the prediction is decided by a single high-leverage Flip have a large gradient norm and therefore a small ρ. The gradient is computed by backpropagating over the program inference computation, which gives $\partial \log w_r / \partial \theta_i$ for every Flip parameter θ_i and every return value $r \in \{\text{yes}, \text{no}\}$; we obtain ∇G as the elementwise difference.

Task	Direct	CoT	Combined CoT	Tree-of-Thoughts	Likelihood	Bayesian
Qwen-2.5-7B-Instruct						
BoolQ	78.60	77.40	27.80	56.40	73.00	83.94
StrategyQA	62.01	72.49	37.41	23.87	71.18	66.12
Date Understanding	63.50	67.50	55.00	61.50	70.50	54.70
Geometric Shapes	16.50	37.00	27.50	31.00	47.50	35.59
Salient Translation	43.00	40.00	39.50	41.00	38.00	45.79
Micro Mean	**59.60**	**65.70**	**35.81**	**39.90**	**65.25**	**64.62**
GPT-4o						
BoolQ	91.40	90.80	87.80	90.40	85.00	91.80
StrategyQA	79.91	83.84	78.89	82.24	81.80	76.71
Date Understanding	79.00	85.00	83.50	85.00	91.50	84.50
Geometric Shapes	41.00	63.00	58.50	85.50	67.50	47.50
Salient Translation	64.00	68.50	70.50	67.00	69.00	65.00
Micro Mean	**76.89**	**81.87**	**78.68**	**83.49**	**80.75**	**77.22**
Llama-3.3-70B-Instruct						
BoolQ	92.20	91.80	90.40	90.40	88.20	91.20
StrategyQA	76.71	83.11	78.02	82.39	80.06	72.49
Date Understanding	75.00	88.50	85.50	87.00	90.00	82.50
Geometric Shapes	34.00	62.00	60.00	58.50	48.00	44.00
Salient Translation	65.50	71.50	76.00	69.50	69.50	67.50
Micro Mean	**74.82**	**82.48**	**80.08**	**81.03**	**78.68**	**75.10**

Table 9: Per-task and micro-averaged accuracy (%) across models and prompting/inference methods.

Normalized margin ρ_{norm}. $\rho(\theta)$ has units of L_2 distance in parameter space and so is not directly comparable across questions whose programs have different numbers of Flips (and hence different dimension $n = \dim \theta$). We therefore normalize by $\sqrt{n}$:

$$\rho_{\text{norm}}(\theta) \;=\; \frac{\rho(\theta)}{\sqrt{n}} \;=\; \frac{\left|\, G(\theta) \,\right|}{\sqrt{n}\,\left\|\nabla_\theta G(\theta)\right\|_2}, \tag{4}$$

i.e. the root-mean-square perturbation per parameter dimension. This makes margins comparable across questions whose programs have different numbers of Flips: a question with $\rho_{\text{norm}} = 0.1$ flips under an average perturbation of 0.1 per coordinate, regardless of how many coordinates there are.

E.6 Robustness Margins as Confidence Signals

Fig. 8 show the Accuracy of the top-10% vs. bottom-10% ρ_{norm} deciles per model $\times$ task.

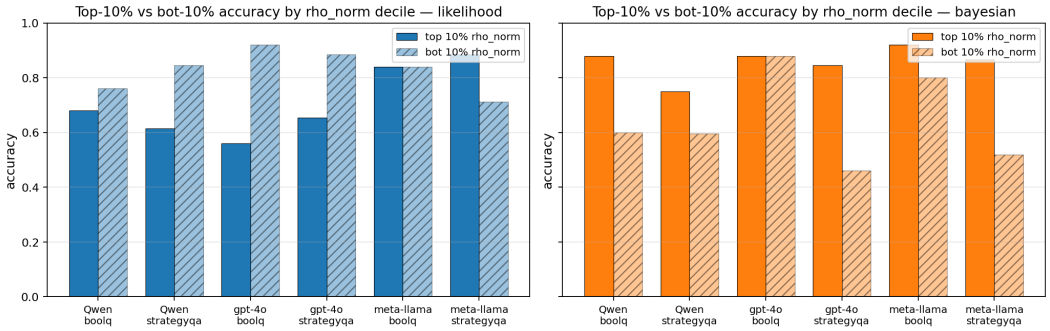


Figure 8: Accuracy of the top-10% vs. bottom-10% ρ_{norm} deciles per model $\times$ task. **Left:** likelihood program. ρ_{norm} is uncorrelated or anti-correlated with correctness. **Right:** Bayesian multi-answer program. The high-margin decile is consistently more accurate, indicating ρ_{norm} is a more calibrated confidence signal.

E.7 Verbalization–Logit Analysis

Table 10 reports task-level accuracy for each model and shows that variants using a verbalized answer prior generally outperform logit-prior variants, while changing the step-weight source has a much smaller effect.

Task	Pure Verbalized	Logit Step + Verb Prior	Verb Step + Logit Prior	Pure Logits
		GPT-4o		
BoolQ	91.65	**91.87**	89.67	89.67
StrategyQA	**75.92**	75.72	70.71	70.71
Date Understanding	**86.47**	83.53	74.71	74.71
Geometric Shapes	47.95	**49.32**	38.36	38.36
Salient Translation	63.36	63.36	**64.89**	**64.89**
		Qwen-2.5-7B-Instruct		
BoolQ	**83.96**	83.74	69.23	69.45
StrategyQA	**66.86**	66.47	45.09	45.66
Date Understanding	**58.24**	52.35	51.18	51.18
Geometric Shapes	23.97	**37.67**	25.34	25.34
Salient Translation	**46.56**	45.04	38.17	38.17
		Llama-3.3-70B-Instruct		
BoolQ	**91.43**	91.21	91.21	90.77
StrategyQA	**73.03**	72.83	68.21	67.63
Date Understanding	**82.35**	80.59	62.35	62.35
Geometric Shapes	**47.26**	**47.26**	30.14	27.40
Salient Translation	65.65	**66.41**	64.12	64.12

Table 10: Comparison of answer aggregation strategies across GPT-4o, Qwen-7B, and Llama-70B on five tasks. We report task accuracy (%) for pure verbalized answers, logit-step aggregation with verbalized priors, verbalized-step aggregation with logit priors, and pure-logit aggregation. The best result for each model-task pair is bolded.

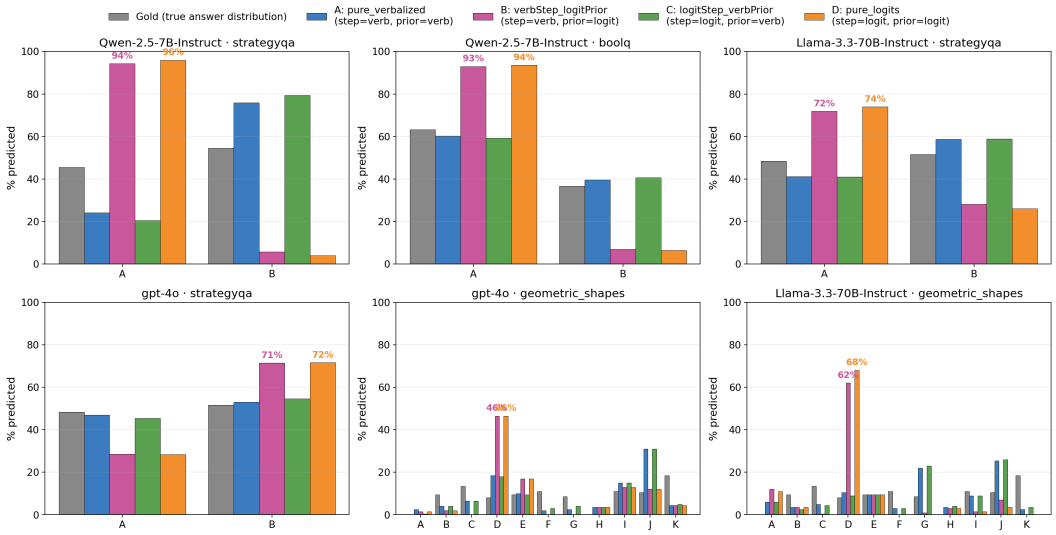


Figure 9: Predicted dataset answer distributions under different step-weight and answer-prior sources. For each model–task pair, we compare the gold dataset answer distribution with four Bayesian multi-answer CoT variants.

Figure 9 shows the predicted dataset-level answer distributions for the four Bayesian multi-answer CoT variants used in §4.2. The comparison separates the source of the step weights from the source of the answer prior. Variants A and C use verbalized answer priors, while variants B and D use logit-based answer priors. Across binary tasks, the logit-prior variants often place most probability mass on a single answer label, whereas the verbalized prior variants more closely match the empirical gold distribution.

E.8 A Qualitative Example of Sensitivity Analysis

We present a qualitative example of sensitivity analysis in Figure 10.

(A) CoT Trace (gpt-4o)

Settings | #calls = 1 | answer prior: logit only

Query: Do the James Bond and Doctor Who series have a similarity in format?
Gold: A (yes)

For A (yes) initial confidence 70%

s1 Both James Bond and Doctor Who are long-running series with a significant cultural impact.
ref. none · 90%

s2 Both series feature a central character who is often portrayed by different actors over time.
ref. none · 95%

s3 The format of both series involves standalone stories or missions, often with a recurring theme or villain.
ref. none · 85%

s4 Both series have a mix of action, adventure, and science fiction elements.
ref. none · 80%

s5 Based on 1, 2, 3, 4, the answer is: **A** (yes).
ref. 1, 2, 3, 4

For B (no) initial confidence 30%

s6 James Bond is primarily a film series, while Doctor Who is a television series.
ref. none · 95%

s7 The narrative structure of James Bond films is typically more linear and self-contained compared to the episodic and often serialized nature of Doctor Who.
ref. none · 85%

s8 Doctor Who often explores more diverse genres and themes due to its episodic nature, whereas James Bond maintains a consistent spy thriller genre.
ref. none · 80%

s9 Based on 1, 2, 3, the answer is: **B** (no).
ref. 1, 2, 3

(B) Program (three scoring models)

Tuples: [GPT-4o, Qwen-7B, Llama-70B].

Teacher-forced probability scores; answer priors use the teacher-forced logit only.

```
## answer priors (logit only)
yes_A: bool =~ Flip("yes", [0.62, 0.89, 0.65])
no_A: bool =~ Flip("no", [0.38, 0.11, 0.35])
A =~ Categorical("answer", Combine(yes_A, no_A))
## evidence for yes
P1:  bool =~ Flip("s1", [0.70, 0.28, 0.38])
P2:  bool =~ Flip("s2", [0.76, 0.34, 0.58])
P3:  bool =~ Flip("s3", [0.48, 0.17, 0.82])
P4:  bool =~ Flip("s4", [0.72, 0.49, 0.30])
yes_E =~ Factor("yes_E", LogitPool(P1, P2, P3,
  P4))
## evidence for no
P6:  bool =~ Flip("s6", [0.88, 0.58, 0.47])
P7:  bool =~ Flip("s7", [0.60, 0.40, 0.43])
P8:  bool =~ Flip("s8", [0.59, 0.27, 0.22])
no_E =~ Factor("no_E", LogitPool(P6, P7, P8))
## answer/evidence constraint
observe((A and yes_E) or ((not A) and no_E))
if A:   return "yes"
else:   return "no"
```

(C) Step sensitivity $\partial P(\textbf{yes})/\partial w$

Mode: `teacher_forcing`, answer prior = logit only. **Bold** = largest $|\partial P/\partial w|$ per judge within each evidence group.

	evidence for "yes"				*evidence for "no"*			*output*	
judge	P1	P2	P3	P4	P6	P7	P8	$P(\text{yes})$	answer
gpt-4o	$+0.093$	$+\mathbf{0.108}$	$+0.079$	$+0.097$	$-\mathbf{0.215}$	-0.095	-0.094	0.61	yes ✓
Qwen-7B	$+0.104$	$+0.094$	$+\mathbf{0.149}$	$+0.084$	-0.098	-0.099	$-\mathbf{0.121}$	0.86	yes ✓
Llama-70B	$+0.098$	$+0.095$	$+\mathbf{0.156}$	$+0.110$	-0.168	-0.170	$-\mathbf{0.243}$	0.73	yes ✓

Figure 10: Sensitivity analysis for StrategyQA-0503 ("Do the James Bond and Doctor Who series have a similarity in format?"; gold answer: **yes**). **(A)** A fixed GPT-4o multi-answer CoT with verbalized confidences and within-chain references. **(B)** Schematic program with teacher-forced scores in the order [GPT-4o, Qwen-7B, Llama-70B]. The answer priors use the teacher-forced logit only; the verbalized initial confidences in (A) are not used. **(C)** Sensitivities of $P(\text{yes})$ to program parameters, computed by automatic differentiation through the compiled program. Bold marks the largest absolute sensitivity within each parameter group for each model. Within the yes/no evidence groups, GPT-4o peaks at P2/P6, whereas Qwen-7B and Llama-70B peak at P3/P8.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All the main claims in the abstract and introduction are justified in Sections 3-5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are descibed in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The theoretical results with their assumptions and proof can be referred in Section 3, Appendix D.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We describe experiment setup in Appendix E.3 and we will release dataset and code upon acceptance.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We will release the data and code upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We describe experiment setup in Section 5 and Appendix E.2 and E.3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Our experiments are intended primarily as exploratory analyses of the potential uses of the proposed language.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We describe experiment setup in Section 5 and Appendix E.2 and E.3.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The authors have reviewed and conformed to the Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We describe experiment setup in Section 5 and Appendix E.2 and E.3.

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: All code and resources are public.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.