INTERWHEN: A GENERALIZABLE FRAMEWORK FOR VERIFIABLE REASONING WITH TEST-TIME MONITORS

Vishak K Bhat[1,*] **Prateek Chanda**[2,*] **Ashmit Khandelwal**[1] **Maitreyi Swaroop**[3]
Vineeth N. Balasubramanian[1] **Subbarao Kambhampati**[4] **Nagarajan Natarajan**[1,†] **Amit Sharma**[1,†]

[1]Microsoft Research, India [2]IIT Bombay [3]Carnegie Mellon University [4]Arizona State University

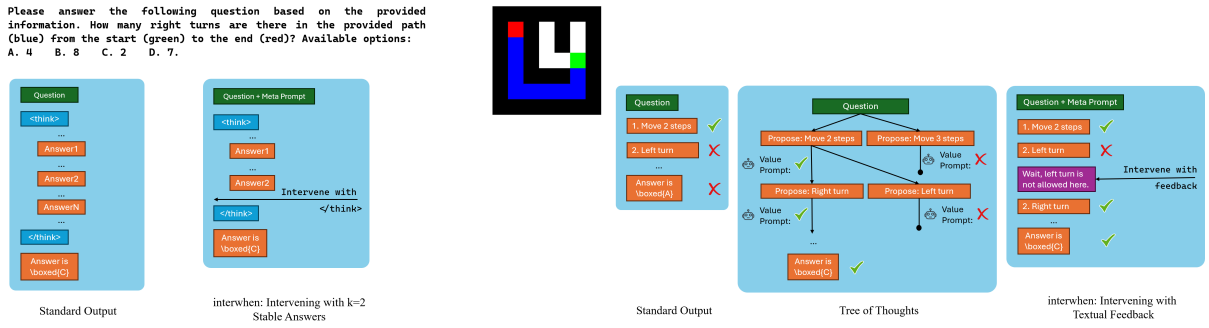
ABSTRACT

We present a test-time verification framework, interwhen, that ensures that the output of a reasoning model is valid wrt. a given set of verifiers. Verified reasoning is an important goal in high-stakes scenarios such as deploying agents in the physical world or in domains such as law and finance. However, current techniques either rely on the generate-test paradigm that verifies only after the final answer is produced, or verify partial output through a step-extraction paradigm where the task execution is externally broken down into structured steps. The former is inefficient while the latter artificially restricts a model's problem solving strategies. Instead, we propose to verify a model's reasoning trace as-is, taking full advantage of a model's reasoning capabilities while verifying and steering the model's output only when needed. The key idea is *meta-prompting*, identifying the verifiable properties that any partial solution should satisfy and then prompting the model to follow a custom format in its trace such that partial outputs can be easily parsed and checked. We consider both self-verification and external verification and find that interwhen provides a useful abstraction to provide feedback and steer reasoning models in each case. Using self-verification, interwhen obtains state-of-the-art results on early stopping reasoning models, without any loss in accuracy. Using external verifiers, interwhen obtains reasonable improvement in accuracy over test-time scaling methods, while ensuring 100% soundness with respect to full verifier and being 4x more efficient. Can find the arxiv version of the paper : `https://arxiv.org/pdf/2602.11202`

1 INTRODUCTION

The current arc of AI progress has led to significant increases in the accuracy of language models (LMs) on reasoning tasks. However, in many high-stakes real-world scenarios such as law, healthcare, or acting in the physical world, it is critical that each answer by an LM is correct and trustable; merely having a high probability of producing the correct answer is not sufficient. Thus, *verification* of a LM's answer becomes an important goal. Past work has explored self-verification through methods such as self-consistency (Wang et al., 2023), and external verification through existing domain-specific verifiers (Cobbe et al., 2021b). In a typical implementation, multiple answer trajectories are generated independently and the ones that pass the verifier are chosen (Kambhampati et al., 2024).

These techniques, however, focus on verifying the final answer. Considering the recent thrust on test-time scaling wherein state-of-the-art models output thousands of intermediate reasoning tokens and/or tens of tool calls before the final answer, this approach becomes inefficient, especially since it fails to catch early errors. Moreover, problems vary by the level at which they can be verified. Some solutions such as code or a plan are best verified by the final answer, while other problems such as writing a report are best verified through the steps taken to reach an output. Verifying a long report invokes considerable cognitive load (and hence the constituent steps may be more feasible to verify for both humans and automated verifiers). Therefore, we propose a new test-time paradigm of generating verified answers that focuses on verification (and correction) of the *steps* towards a solution, not just the final answer. Existing approaches that verify partial steps and choose the valid ones at each step (Yao et al., 2023) break down the task execution externally into steps, which artificially limits a model's problem solving strategies. Instead, we adopt the following desiderata to effectively make use of the advances in problem-solving abilities of reasoning models: **1)** Problem solving should not be externally broken down into steps; instead the model retains full control over problem execution. **2)** Rather than choosing between multiple independent step generations (which can be inefficient), one should provide step-wise feedback to the model at test-time and steer the model's reasoning through verifier feedback. Importantly, if there is no error, the model's output text should be identical to the one without any test-time verification. We call this paradigm *LLM-Process-Modulo* (see Figure 1), to distinguish it from the LLM-Modulo paradigm (Kambhampati et al., 2024), which we call *LLM-Solution-Modulo.*



(a) Internal Verification (b) External Verification

Figure 1: LLM-Process Modulo: Intervening on partial output traces to guide a model towards a more accurate and/or more efficient solution. Meta-prompting instructs the model to output *intermediate states* that can be extracted and verified. In the first case, we track mentions of candidate solutions within the `<think>` block and insert `</think>` when the same answer appears consecutively. In the second case, we run an external verifier on each intermediate state and append verifier feedback to the model's output trace in case of any error. Compared to existing methods such as tree-of-thoughts, we do not artificially break up the problem into step-wise generation, and as a result, obtain both higher accuracy and efficiency.

This gives rise to two research questions: 1) given a stream of ambiguous text as LM output, how to identify the *verifiable* steps and verify them; and 2) given a verifier's feedback, how to steer a LM's output to fix the error highlighted by the verifier. Our core conceptual contribution is *meta-prompting*: the process of identifying verifiable steps for a task and prompting the LM to generate output such that those steps are easily extractable. This side-steps the error-prone task of building parsers to extract and formalize steps (Wu et al., 2022), and enables a general method that can work for any domain, from code to text reports, as long as verifiable states can be defined. Formally, meta-prompting involves identifying properties (states) $\mathcal{P}$ that any valid partial reasoning trace should satisfy to be able to solve a problem. In many cases, identifying the verifiable states is trivial. For procedural reasoning problems where each step follows from the past context, e.g., math or deductive reasoning problems, each step is a verifiable state by definition. Other problems, such as finding a plan or object satisfying certain constraints may require inductive reasoning or working with inconsistent values (e.g., proof by contradiction) and hence only specific intermediate states may be verified. Once verifiable states are identified, we introduce a general framework, interwhen, for steering model outputs via feedback. Given intermediate states and their associated verifiers, interwhen abstracts steering into three functions: state extraction, state verification, and feedback-driven trace updates. As long as the model emits the required intermediate states, it is free to use any problem-solving strategy. While interwhen can implement arbitrary test-time verification schemes, we focus on a simple instantiation that balances accuracy and efficiency: a single execution run in which verifiers are invoked whenever an intermediate state appears, and their feedback is injected inline either as a "#Thought" or a "Wait" reflection before generation continues.

To reflect practical settings where objective verifiers may be unavailable, we adopt a broad notion of verification, encompassing both LLM-based (self-)verifiers and external verifiers. In both cases, intervening on partial output traces yields substantial gains in accuracy, efficiency, or both. With external verifiers, interwhen additionally guarantees verifier-level *soundness* by design: any output produced is consistent with the verifiers.

We instantiate interwhen in two regimes: self-verification and symbolic external verification. For self-verification, we study early stopping to improve efficiency without accuracy loss by treating candidate final answers as intermediate states and invoking an equivalence checker whenever they appear; generation halts once the last k candidates match. For external verification, we improve accuracy via test-time scaling, defining intermediate states by dataset semantics—moves in MAZE, positional conclusions in SPATIALEVAL, and partial number subsets in GAME OF 24. Across all datasets, interwhen reliably steers model behavior, achieving up to 30% token reduction for early stopping and up to 10 p.p. accuracy gains over Best-of-4 and Tree-of-Thought (Yao et al., 2023), while remaining 100% sound wrt full verifier and at least 4× more efficient.

2 Related Work

2.1 Verification-guided Inference

Two primary ways to improve LLM output at test-time are: using verification and using other test-time scaling techniques.

Verification of LLM output. In math, planning and other domains, a dominant paradigm for test-time verification is to generate candidate solutions and then select the best one, either using external verifiers (Cobbe et al., 2021b; Kambhampati et al., 2024) or the same LLM for feedback (Madaan et al., 2023). Efforts to verify and steer based on partial steps utilize an externally imposed structure to break down a problem into steps, e.g., asking the language model to make one deduction in a multi-step deduction task or one step in a multi-step math problem (Creswell et al., 2022; Yao et al., 2023). Other works employ self-verification with a LLM prompt to verify and refine the answer (Chang et al., 2025; Stechly et al.). Compared to these efforts, we do partial step verification without any externally imposed structure of steps that control execution. Instead, we have a lightweight prompt guidance for the model to output intermediate states.

There is a large literature on process reward models (Lightman et al., 2024) that is orthogonal to this work. Existing PRMs can be used as verifiers in the interwhen framework; however, the novel question we explore is how to use such feedback to steer a reasoning model at test time. There is early work on steering reasoning models at test time through text feedback (Prabhakar et al., 2025), however the connection to partial output verification is not explicit.

Test-time scaling. Test-time scaling is the emergent paradigm for improving reasoning in LMs, by adjusting compute at inference time based on the complexity of the task (Snell et al., 2025; Balachandran et al., 2025; Kang et al., 2025). A popular technique often employed as a strong baseline is best-of-N sampling, that involves drawing N trajectories *in parallel* and picking the best final answer as adjudged by an external verifier (Kang et al., 2025) (e.g., a judge LM) or by simple heuristics (e.g., majority) in permissible scenarios. A variant of this strategy is to draw the samples *sequentially* using the feedback on a previous answer in the *LLM-Modulo* paradigm (Kambhampati et al., 2024).

Among techniques that analyze intermediate traces, some techniques encourage exploration (Muennighoff et al., 2025) using heuristics (e.g., inserting "Wait" to continue thinking) to be able to solve complex problems by giving additional compute. Another class of test-time scaling methods rely on classical tree-based search techniques, and use LMs to guide the exploration, e.g., Tree of Thoughts (Yao et al., 2023) and MCTS-inspired search (Dainese et al., 2024; Wang and Yang, 2025; Huang et al., 2025a; Antoniades et al., 2025). In this work, we propose a new axis of test-time scaling that uses a single rollout and steers the reasoning trajectory through verifiers.

2.2 Early Stopping

Test-time scaling can induce *overthinking*, where models continue generating reasoning beyond the point of meaningful improvement (Ghosal et al., 2025). Since then, several strategies have been proposed to mitigate this behavior.

Early work in this area focused on using confidence signals to decide when to halt generation. For example, entropy-based frameworks such as (Sharma and Chopra, 2025) use sequence-level Shannon entropy from token log-probabilities as a proxy for emergent confidence. However, these approaches require calibrating stopping thresholds on benchmark datasets. Building on entropy heuristics, methods such as EAT (Wang et al., 2025) compute entropy after `</think>` to decide on early exit. DEER (Yang et al., 2025) similarly introduces dynamic early stopping for chain-of-thought models by monitoring confidence at intermediate reasoning chunks. Both methods depend on manually specified confidence thresholds, which are hard to interpret and set for each task.

Subsequent work explored richer signals and control mechanisms. (Huang et al., 2025b) suppresses reflection-trigger tokens (e.g., "Wait" and "Alternatively") when the model exhibits high confidence, thereby reducing redundant reasoning without retraining. (Sun et al., 2025) frames early stopping as a control problem through a two-stage discriminator and an adaptive multi-armed bandit controller that dynamically determines when further reasoning is unnecessary. Parallel to these efforts, Dynasaur (Fu et al.) characterizes *answer stabilization* where they perform multiple rollouts after a particular segment of reasoning and early exit based on the stability of the rollout answers. Complementary approaches such as (Choi et al., 2025) analyze attention patterns to a special end-of-thinking token and prune low-contributing reasoning chunks before resuming generation. Another similar work *ES-CoT* (Mao et al., 2025), samples the answer after each chunk and checks the stability of those answers.

These methods typically rely on calibrated thresholds, auxiliary confidence estimators, heuristic trigger tokens, or specialized control mechanisms, raising questions about robustness across models and domains. In contrast, we propose

a simple method based on self-verification that extracts intermediate answers from the reasoning trace and appends </think> once the answer has remained consistent for K consecutive checks, enabling adaptive termination without dataset-specific calibration.

3 LLM-PROCESS-MODULO PARADIGM

As stated in the Introduction, majority of techniques for improving reasoning have two pivots: (1) scale test-time compute, (2) focus on accuracy of the solution, which often means the correctness of the final answer. Current techniques to improve reasoning (discussed in Section 2) can be categorized broadly into two classes, based on if and how they steer the reasoning trajectories, or more broadly, the exploration of the solution space.

(1) The standard best-of-N sampling methods do not steer the reasoning process. These techniques are final-answer-focused and inefficient, often requiring tens of thousands of tokens (and several tool calls) even for simple problems.

(2) Tree-based search techniques such as beam search leverage LMs to *guide* the search process. These techniques, in contrast to **(1)**, explicitly decompose the solution generation into intermediate steps and then use LMs to pick the best action from several permissible actions at each node in the tree. While they intervene the exploration of the solution and are not purely final-answer focused, they can be restrictive—they are confined by the heuristics and choices of the underlying search algorithm, and often require hand-crafting problem-specific prompts to guide the search (e.g., the value prompt in beam search (Yao et al., 2023)).

To address the main criticisms of the above two classes of techniques, we propose a new direction of *intervening* the partial traces and *steering* them, that we refer to as "LLM-Process-Modulo". The core idea is simple: we interject the reasoning process as it unfolds, using either external verification mechanisms (when available) or self-verification (otherwise), so that we can guarantee some form of validity or correctness of the reasoning *process*. Our proposal is analogous to the LLM-Modulo framework (Kambhampati et al., 2024) that follows an iterative generate-test paradigm on the final answer, but the *modulo* (i.e., verification) happens during the process of generating the solution. Realizing this idea poses multiple research challenges: (1) **when to intervene:** reasoning models output a stream of tokens that are difficult to verify, so we need a way to identify the verifiable "states" in the reasoning process; (2) **how to verify**: we might need formal verifiers in some scenarios, or LLM-based verifiers in others. The choice depends on the domain and the task at hand. (3) **how to intervene:** how do we intervene or *steer* the reasoning model if an intermediate state is rejected by the verifier.

When to intervene: It is often challenging to make sense of the stream of "thinking" tokens from a reasoning LM, let alone verify. We need a systematic way to elicit the verifiable parts of the reasoning process. The first challenge is how to elicit *process* verification in LMs. In domains like math, we can think of each derivation or formula as a "step", and verifying the process amounts to verifying the correctness of the individual steps given the pre-condition(s). In such cases, it might suffice to parse the steps in the trace and verify them. For general reasoning tasks involving logic and planning, identifying verifiable intermediate states is challenging: reasoning may be convoluted, involve hypotheses, and include backtracking. This motivates a principled definition of verifiable states and a mechanism to elicit them reliably. For example, in the 2D maze task (Figure 1), verifiable states correspond to moves, but these are often interleaved with free-form reasoning and are hard to extract. We address this via meta-prompting, which elicits such states explicitly and turns them into intervention points.

How to verify: Our goal is to verify *partial* output traces. In domains such as planning, the same final answer verifier can be used to validate the steps in a partial trace. In others, we may need to develop new verifiers. In many reasoning settings, verifiers can be easier to implement than solving the problem (e.g. a rule-based checker of validity of game plays). For instance, for the Maze problem in Figure 1, we implement a simple verifier based on representing the maze as a 2D array in Python and replaying operations on it. In planning, logic, and code, we could also leverage formal verification tools and extend them to adapt to incomplete information in the partial trace. Finally, we may utilize LLM-based verifiers for general reasoning problems (e.g., document generation, counter-factual analysis).

How to intervene: When an intervention point is encountered (as in the above discussion), we need to determine how to intervene the reasoning process. Many options present: 1) simply roll back the solution generated thus far, 2) provide a text feedback from the verifier (or more generally a tool or even a human expert), and resume the reasoning process, 3) provide a binary correct/incorrect feedback from the verifier, and so on. The right choice of intervention could depend on the ability of the reasoning model, the fidelity of the verifier, and the complexity of the task at hand. For instance, for the maze problem in Figure 1, we intervene by adding the $< /think >$ token for early stopping the model's reasoning, and intervene by adding the verifier's feedback after the current state for the test-time scaling setting.

While we cannot answer all questions in this paper, we provide a generalizable framework that allows investigation of all these questions in the next Section. We also propose a candidate method that provides significant improvements over the current techniques.

4 INTERWHEN: GENERALIZABLE FRAMEWORK

Many techniques for test-time verification and scaling have been proposed, however, it is difficult to evaluate and compare them holistically for a given task, across accuracy and efficiency tradeoffs. Therefore, we build a general framework for test-time verification called interwhen that allows exploring the research questions described above, while also providing an easy way to implement and compare existing test-time scaling strategies.

Meta-prompting While the standard prompt focuses on task description, meta-prompting focuses on ensuring a structure in the output trace such that intermediate states are clearly demarcated. Formally, meta-prompting structures an output trace from unstructured text to a semi-structured text where intermediate states are interspersed with arbitrary text: "..., s_i, ..., s_{i+1}, ..." where ... refers to free text. We implement it by providing few-shot examples of the format needed for expressing a state. For instance, for the 2D Maze problem in Figure 1, the metaprompt simply instructs to output each move in a specific format.

Abstractions for test-time verification The three research questions above motivate three fundamental operations to build any test-time verification method. 1) identifying and extracting verifiable steps from one or multiple LM outputs; 2) verifying the candidate steps; 3) giving feedback to improve a candidate step or choosing one of the candidate steps.

Given a task, the first step is to identify relevant verifiers and determine the intermediate "states" on which they will apply. Each verifier is a function that takes the current state, all previous states, and problem context, and returns a pass decision and optionally some text feedback on the pass evaluation. Let S_0 denote the initial state which consists of problem description and any relevant context, and S_i denote the ith state in the reasoning trajectory. Given a list of verifiers, test-time verification requires the following three operations.

extract_state. Given a verifier V and a partial reasoning trace r, extract the relevant intermediate state: $S_i = \textsf{extract\_state}(r, V)$. For procedural solutions where the LM lists solution steps in sequence (e.g., for simple grade-school math problems (Cobbe et al., 2021a)), using newline or other token-level separators suffices to extract states.

verify. Given the extracted intermediate state and context c (e.g., past states), perform verification using the verifier V: $y_i, t_i = \textsf{verify}(S_i, c, V)$.

intervene. If the verifier fails and provides text feedback, process the feedback and update the partial reasoning trace, $r' = \textsf{intervene}(r, y_i, t_i)$, otherwise keep the same trace $r' = r$. The intervention may ask the model to continue generation or decide to stop the generation.

Different test-time scaling algorithms can be implemented using the above operations. For example, best-of-N is a specific instance where the only verifiable state is the final answer and the intervention is to stop the generation, which is then repeated over a loop. Tree of thoughts (Yao et al., 2023) can be implemented by enforcing a beam search over the extract-verify-intervene process on each step, where the intervene step is simply the identity operation for the top-ranking steps as per the verifier, while generation is stopped for all others.

4.1 PROPOSED ALGORITHM: SEQUENTIAL VERIFIER

We now describe a simple verification algorithm that works well in practice. Rather than exploring multiple trajectories, the algorithm maintains a single output trace. For each identified intermediate state, it applies the verifier and allows generation to continue if the verifier passes. If not, it adds the text feedback to the model's output trace and then continues generation. The intuition is that the model should change its solution trajectory based on the text feedback on its previous state.

A key question is how to identify intermediate states. We leverage the metaprompting strategy discussed in Section 4, and instruct the model to output intermediate states in a specific format. Mentioning few-show examples describing the format is typically sufficient (see Appendix A.2 for the different prompts used).

Remark 4.1 (Merits of Sequential (State) Verifier). First, we can guarantee soundness of reasoning, i.e., intermediate "steps" of the output solution are correct w.r.t. to a given set of verifiers. Second, we directly address the key pain-points of existing techniques for improving reasoning, namely: (1) restrictiveness of the classical search heuristics *by largely retaining the agency of search in the reasoning models*, (2) need for a multitude of problem $\times$ search strategy prompts *via leveraging a simple meta-prompting strategy*. Empirically, we find that this also leads to substantial reduction of token inefficiency compared to test-time scaling techniques.

We use two case-studies next to illustrate the implementation of the algorithm: early stopping and test-time scaling. For dataset details, see Section 5.

4.2 CASE STUDY 1: INTERNAL VERIFICATION–EARLY STOPPING

Our first case-study involves the problem of early stopping. Test-time scaling often induces overthinking, where models continue reasoning beyond the point of meaningful improvement (Ghosal et al., 2025). To mitigate this, several **early stopping** strategies have been proposed. Methods like EAT rely on entropy-based criteria computed after the `</think>` token (Wang et al., 2025), while others like DEER estimate confidence scores at intermediate reasoning chunks to decide when to halt inference (Yang et al., 2025). A key challenge in these methods is to determine the correct threshold for entropy or confidence, such that accuracy loss is minimal. We use interwhen to propose a simple and effective algorithm, ***k-Stable Answer***, which terminates reasoning once the predicted answer remains unchanged for k consecutive reasoning steps. We find that that setting $k = 2$ works well across datasets.

Sequential Verifier Implementation We instantiate the Sequential Verifier by specializing the extract_state, verify, and intervene operations for each task family.

For multiple-choice reasoning tasks (MAZE and SPATIALMAP), extract_state uses regular-expression patterns to detect answer proposals signaled by phrases such as "the answer is" and related variants. The verify function then checks whether the extracted option remains unchanged for k consecutive times.

For the GAMEOF24 dataset, extract_state identifies the equation proposed by the model, subject to the constraint that each input number appears exactly once in the extracted expression. verify tests whether the same equation appears k consecutive times.

For the VERINA code-generation and specification-generation tasks, extract_state first isolates the portions of the response that indicate the presence of code or specifications using regular-expression patterns (e.g., phrases such as "the code is" and related variants). Because superficial string matching is insufficient in this setting, we implement verify using a small language model (Qwen/Qwen2-1.5B-Instruct) to determine whether the current artifact extracted is semantically equivalent to the previous one. For specification generation, we separately track the stability of the pre-conditions and post-conditions by maintaining two independent counters, and trigger termination only when both have remained unchanged for k consecutive times.

In all the three datasets, once the stability criterion is met in verify, intervene terminates reasoning by injecting a `</think>` token and prompting the model to output its final answer.

4.3 CASE STUDY 2: EXTERNAL VERIFICATION

The second application concerns test-time scaling with the goal of improving accuracy on a task, while ensuring soundness if the system outputs an answer, it is correct.

Sequential Verifier Implementation We modify the base prompt to explicitly specify the format in which the model should present its output along with passing few shot examples (**Metaprompting**, Section 4). For the MAZE dataset, the prompt encourages step-by-step generation, beginning with identification of the start and end locations, followed by a sequence of moves in which each step reports the proposed action, the resulting position, the cumulative step count, and the direction taken. A concrete instantiation of this prompt is provided in the Appendix A.2.

For the SPATIALMAP dataset, we similarly enforce a structured reasoning template. The model is instructed to first parse and enumerate all spatial relations described in the question, then explicitly identify the queried pair and determine whether a direct relationship is present, and finally produce both a natural-language conclusion and a boxed multiple-choice answer. Concretely, the prompt organizes generation into three stages: (*i*) parsing relationships, (*ii*) finding the direct relationship relevant to the query, and (*iii*) emitting the final answer in a standardized format. An example of this template is included in the Appendix A.2.

extract_state. As described above, we meta-prompt the model to produce reasoning in a highly structured format, which we then isolate using regular-expression patterns.

verify. For the MAZE dataset, we first parse the input maze into a grid representation encoding wall locations as well as the designated start and end positions. We define deterministic transition rules for left and right turns (e.g., moving from facing south to east constitutes a right turn). When the model proposes a step, we simulate the resulting move and check whether it violates any constraints such as entering a wall cell and whether the updated orientation is consistent with the turning rules. A proposal is marked invalid if either the movement or the implied direction is incorrect.

Table 1: Accuracy and token usage comparison across benchmarks and models. CoT: Chain of Thoughts, Bo2: Best of 2, Bo2-c: Best of 2 (Critic), Bo4: Best of 4, Bo4-c: Best of 4 (Critic), GT-c: Generate & Test (Critic), ToT: Tree of Thoughts, IW: interwhen (Ours). Token usage is reported as a percentage relative to CoT. For each accuracy row, the method achieving the highest accuracy overall is underlined, and the method achieving the **highest** accuracy among those using at most as many tokens as Bo2 is bolded.

		Qwen3-30B								QwQ-32B							
Dataset	**Metric**	CoT	Bo2	Bo2-c	Bo4	Bo4-c	GT-c	ToT	IW	CoT	Bo2	Bo2-c	Bo4	Bo4-c	GT-c	ToT	IW
Game24	Acc. (%) ↑	95.0	**98.8**	97.4	99.5	97.4	96.1	95.0	97.2	83.7	**93.9**	84.4	97.7	84.4	84.3	74.1	89.1
	% Tok. ↓	100	199	199	400	400	116.3	936.5	61.6	100	196	196	389	389	120.7	812	93.7
Maze	Acc. (%) ↑	88.5	90.4	88.5	91.2	88.6	88.9	90.3	**98.6**	85.8	**91.5**	87.0	94.7	87.1	80.7	69.3	87.8
	% Tok. ↓	100	201	201	399	399	119.7	746.2	133.1	100	199	199	401	401	107.4	727	132.8
SpatialMap	Acc. (%) ↑	74.7	79.1	75.1	81.6	74.9	73.9	75.6	**87.5**	74.9	78.7	74.8	81.4	74.7	63.2	67.1	**85.0**
	% Tok. ↓	100	201	201	399	399	111.8	672.5	152.9	100	198	198	395	395	127.2	563.5	156.4
ZebraLogic	Acc. (%) ↑	95.5	**98.4**	93.6	99.5	93.8	94.4	17.2	97.9	76.1	**82.9**	77.0	86.6	77.9	77.6	16.2	77.9
	% Tok. ↓	100	206	206	413	413	102.7	861.1	105.6	100	190	190	380	380	94.3	740.1	92.2
ZebraLogic X-Large	Acc. (%) ↑	79.2	**94.3**	70.3	97.4	70.3	72.9	1.2	89.6	19.3	24.5	17.2	38.5	20.8	18.8	0.5	**26.0**
	% Tok. ↓	100	203	203	406	406	91.9	880.3	103.4	100	192	192	384	384	97.7	780.2	88.2
VERINA-Code	Acc. (%) ↑	56.1	65.6	57.1	74.6	56.6	53.4	51.9	**71.6**	56.1	66.7	53.4	79.9	53.4	57.1	49.2	**75.7**
	% Tok. ↓	100	202	202	400	400	111.4	1681.5	133.9	100	196	196	399	399	97.8	1203	114.1
VERINA-Spec	Acc. (%) ↑	25.4	33.9	27.0	41.8	27.0	25.9	29.6	**37.0**	37.6	40.2	33.3	48.7	33.3	34.9	33.9	**44.4**
	% Tok. ↓	100	203	203	400	400	112.8	1825.4	126.9	100	197	197	400	400	99.2	1049.7	187.1

For SPATIALMAP, we encode the initial spatial relations between objects as logical constraints in a Z3 solver. At each step, the model's structured output is converted into a JSON representation by a small language model, which is then checked for logical consistency against the constraint set using Z3. Any contradiction indicates that the newly proposed relation cannot hold given the existing map. For GAMEOF24, we verify by checking that the expression evaluates to 24 and that all 4 digits are used.

intervene. If a proposed step fails verification, we append the corrective textual feedback at the current state and provide it to the LLM to continue generation, explicitly indicating that the step is inconsistent and instructing the model to resume reasoning from the previous valid state. If no violation is detected, generation proceeds without intervention.

5 EXPERIMENTS

Datasets & Models : Our main results are based on the following datasets and the Qwen-30B-A3B-thinking model: spatial reasoning problems, including MAZE and SPATIALMAP (1,500 instances each) (Wang et al., 2024); arithmetic reasoning via GAMEOF24 (1,362 instances).

In MAZE, each example provides a grid with start and end points; questions involve reasoning over the path, such as counting right turns, total turns, or the start's position relative to the end.

SPATIALMAP examples specify object locations; questions ask for an object's relative position, the object in a given direction, or the number of items along a direction.

In GAMEOF24, each example provides four integers, and the task is to construct an expression using each number exactly once with $+, -, \times, /$ to reach 24. In addition, we show ablations using a code dataset, VERINA (Ye et al., 2025), using QwQ-30B.

5.1 INTERNAL VERIFICATION

Baselines. We compare *k-Stable* answers against two recent early-stopping strategies for reasoning models.

EAT (Entropy After </think>) Wang et al. (2025) append a </think> token after each reasoning chunk and computes the entropy of the next-token distribution. An exponential moving average over steps is maintained, and reasoning is terminated once this value drops below a fixed threshold, after which the model is prompted to produce the final answer.

DEER (Dynamic Early Exit in Reasoning Models) Yang et al. (2025) estimate answer confidence after each reasoning step by prompting the model with "</think> The answer is." If the confidence exceeds a predefined threshold, the chain-of-thought is halted and the model directly generates the final output.

Metrics. We report task accuracy together with the percentage of reasoning tokens used. The number of tokens generated under standard chain-of-thought decoding is treated as 100%, and all early-stopping methods are measured relative to this baseline.

Models and Decoding. We conduct experiments on three open-source reasoning models: Qwen/Qwen3-30B-A3B-Thinking-2507, Qwen/QwQ-32B, and microsoft/phi-4-reasoning.

All experiments are run using `vLLM`. For the Qwen models, we use a maximum generation length of 32,768 tokens with temperature 0.6, top-p 0.95, and top-k 20. For Phi-4, we use a maximum length of 16,384 tokens, temperature 0.8, top-p 0.95, and top-k 50. We report results for the hyperparameter configuration that achieves the maximum reduction in token usage while preserving accuracy. Details of the hyperparameter settings and the complete results are provided in Appendix A.1 and Tables 4–6.

Main Results : Table 1 presents our primary comparison on Qwen3-30B across MAZE, SPATIALMAP, and GAMEOF24. Across all three benchmarks, *k-Stable* answers achieves substantially larger token reductions than existing early-stopping methods while maintaining or improving accuracy.

On MAZE, *k-Stable* reduces reasoning tokens by **32.2%** without any loss in accuracy, compared to 0.6% for DEER and negligible savings for EAT. On SPATIALMAP, it attains a **10.8%** reduction, exceeding DEER (8.2%) and EAT (0.4%). For GAMEOF24, *k-Stable* achieves a **31.7%** reduction, whereas DEER and EAT yield only 11.7% and 3.5%, respectively.These results demonstrate that stability-based stopping can substantially shorten chains of thought while preserving final-task performance on a strong reasoning model. The main tables and ablations report only the operating point that achieves the *largest token reduction while maintaining or improving accuracy relative to the baseline*. Complete sweep results for every model and dataset are provided in Appendix tables 4–6.

5.2 EXTERNAL VERIFICATION

Baselines. We evaluate against the following baselines:

TTS: Sampling Methods: Under the test-time scaling (TTS) paradigm, for each input instance $s \in \mathcal{D}$ we generate $\mathcal{K}$ independent candidate solutions by sampling complete reasoning traces from the base model (e.g., via temperature sampling). Because the gold answer is not available at inference time, candidate selection is performed using an auxiliary language-model critic, which evaluates each full trace end-to-end given only the problem statement and the generated trace. The critic assigns a scalar score to reflect the estimated correctness/validity of candidate i.

We consider two standard aggregation rules. (i) *Best-of-$\mathcal{K}$*: we return the single candidate with the highest critic score. (ii) *Majority vote-$\mathcal{K}$* (self-consistency): we extract the final answer from each candidate and return the most frequent answer; ties are broken using the critic score. For reporting, we evaluate the selected prediction by comparing it to the ground-truth answer for each instance. (iii) *Generate-Test*: We employ a generate–verify loop that is distinct from best-of- $\mathcal{K}$ sampling. The model first generates a complete solution (e.g., a plan, code, or reasoning trace). The critic identical to the one used in best-of- $\mathcal{K}$ then verifies the generated solution. If the solution passes verification, the process terminates; otherwise, generation is repeated conditioned on the critic feedback.

Tree of Thoughts (ToT) We include the Tree-of-Thoughts baseline of Yao et al. (2023), using few-shot proposer prompts to generate successor states and a value prompt to score them as *likely*, *unlikely*, or *impossible*. These scores guide a beam search (width 2, up to 4 branches), with search depth capped at 8 across datasets.

Tree of Thoughts (ToT) with LLM-based SV This is an interwhen-based verification implementation of ToT. We keep the same proposer and search procedure, but replace the LLM value prompt with the step verifier: we score each intermediate state using the verifier's critic prompt (i.e., correctness/validity of the state), and use these scores for ranking and pruning during beam expansion.

Main Results: Table 1 indicates that the Step Verifier with Meta-Prompting achieves the strongest performance across all datasets and offers a more favorable accuracy–token trade-off than the Standard CoT baseline. On MAZE, it improves accuracy by **10.21%** while increasing token usage by **26%**; on SPATIALMAP, it yields an 11.2% accuracy gain with 22.5% higher token usage. Tree-of-Thought baselines including ToT-SV (which uses a step verifier instead of the value prompt, not meta prompting as done in step verifier baseline) also improve upon Standard CoT and generally outperform sampling-based approaches; however, their accuracy gains are smaller relative to their token overhead. We refer to further ablation studies in Appendix A.3.

6 CONCLUSION

We presented a framework for test-time verification that steers a single output trace, thus yielding both accuracy and efficiency gains. The framework allows for both internal and external verification, and makes it easy to plug-in verifiers to enforce desired properties at test-time. More generally, the interwhen framework provides a testbed to explore verification as a new axis for test-time scaling.

REFERENCES

A. Antoniades, A. Örwall, K. Zhang, Y. Xie, A. Goyal, and W. Y. Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. In *The Thirteenth International Conference on Learning Representations*, 2025.

V. Balachandran, J. Chen, L. Chen, S. Garg, N. Joshi, Y. Lara, J. Langford, B. Nushi, V. Vineet, Y. Wu, et al. Inference-time scaling for complex tasks: Where we stand and what lies ahead. *arXiv preprint arXiv:2504.00294*, 2025.

K. Chang, Y. Shi, C. Wang, H. Zhou, C. Hu, X. Liu, Y. Luo, Y. Ge, T. Xiao, and J. Zhu. Step-level verifier-guided hybrid test-time scaling for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 18473–18488, 2025.

D. Choi, J. Lee, J. Tack, W. Song, S. Dingliwal, S. M. Jayanthi, B. Ganesh, J. Shin, A. Galstyan, and S. B. Bodapati. Think clearly: Improving reasoning via redundant token pruning. *arXiv preprint arXiv:2507.08806*, 4, 2025.

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

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

A. Creswell, M. Shanahan, and I. Higgins. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*, 2022.

N. Dainese, M. Merler, M. Alakuijala, and P. Marttinen. Generating code world models with large language models guided by monte carlo tree search. *Advances in Neural Information Processing Systems*, 37:60429–60474, 2024.

Y. Fu, J. Chen, S. Zhu, Z. Fu, Z. Dai, Y. Zhuang, Y. Ma, A. Qiao, T. Rosing, I. Stoica, et al. Efficiently scaling llm reasoning with certaindex, 2025a. *URL https://arxiv. org/abs/2412.20993*.

S. S. Ghosal, S. Chakraborty, A. Reddy, Y. Lu, M. Wang, D. Manocha, F. Huang, M. Ghavamzadeh, and A. S. Bedi. Does thinking more always help? understanding test-time scaling in reasoning models. *arXiv preprint arXiv:2506.04210*, 2025.

B. Huang, T. Nguyen, and M. Zimmer. Tree-opo: Off-policy monte carlo tree-guided advantage optimization for multistep reasoning. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS*, 2025a.

J. Huang, B. Lin, G. Feng, J. Chen, D. He, and L. Hou. Efficient reasoning for large reasoning language models via certainty-guided reflection suppression. *arXiv preprint arXiv:2508.05337*, 2025b.

S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. P. Saldyt, and A. B. Murthy. Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *Forty-first International Conference on Machine Learning*, 2024.

Z. Kang, X. Zhao, and D. Song. Scalable best-of-n selection for large language models via self-certainty. In *NeurIPS*, 2025.

H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594, 2023.

M. Mao, B. Yin, Y. Zhu, and X. Fang. Early stopping chain-of-thoughts in large language models. *arXiv preprint arXiv:2509.14004*, 2025.

N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. B. Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332, 2025.

A. Prabhakar, R. Ram, Z. Chen, S. Savarese, F. Wang, C. Xiong, H. Wang, and W. Yao. Enterprise deep research: Steerable multi-agent deep research for enterprise analytics. *arXiv preprint arXiv:2510.17797*, 2025.

A. Sharma and P. Chopra. Think just enough: Sequence-level entropy as a confidence signal for llm reasoning. *arXiv preprint arXiv:2510.08146*, 2025.

C. V. Snell, J. Lee, K. Xu, and A. Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

K. Stechly, K. Valmeekam, and S. Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks, 2024. *URL https://arxiv. org/abs/2402.08115*, 3.

R. Sun, W. Cheng, D. Li, H. Chen, and W. Wang. Stop when enough: Adaptive early-stopping for chain-of-thought reasoning. *arXiv preprint arXiv:2510.10103*, 2025.

J. Wang, Y. Ming, Z. Shi, V. Vineet, X. Wang, Y. Li, and N. Joshi. Is a picture worth a thousand words? delving into spatial reasoning for vision language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=cvaSru8LeO.

L. Wang and Z. Yang. Formal theorem generation via mcts with llm-guided process optimization. In *International Conference on Intelligent Computing*, pages 55–65. Springer, 2025.

X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

X. Wang, J. McInerney, L. Wang, and N. Kallus. Entropy after ¡/Think¿ for reasoning model early exiting. *arXiv preprint arXiv:2509.26522*, 2025.

Y. Wu, A. Q. Jiang, W. Li, M. N. Rabe, C. E. Staats, M. Jamnik, and C. Szegedy. Autoformalization with large language models. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=IUikebJ1Bf0.

C. Yang, Q. Si, Y. Duan, Z. Zhu, C. Zhu, Q. Li, M. Chen, Z. Lin, and W. Wang. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025.

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

Z. Ye, Z. Yan, J. He, T. Kasriel, K. Yang, and D. Song. Verina: Benchmarking verifiable code generation. In *2nd AI for Math Workshop @ ICML 2025*, 2025. URL https://openreview.net/forum?id=47601CQFri.

A APPENDIX

A.1 THRESHOLD SWEEPS

Threshold Sweeps. We conduct extensive hyperparameter sweeps for all early-stopping strategies.

For EAT, we sweep entropy thresholds from 0.2 to 10^{-4}. For DEER, we vary confidence thresholds from 0.85 to 0.995. For *k-Stable*, we explore stability windows $k \in [2, 100]$. We present results obtained from the hyperparameter that gets the maximum reduction of the tokens while maintaing the accuracy.

For EAT, we evaluate entropy thresholds $\{0.2, 0.1, 0.04, 0.008, 0.005, 0.003, 0.001, 10^{-4}\}$. For DEER, we sweep confidence thresholds $\{0.85, 0.9, 0.93, 0.95, 0.97, 0.98, 0.99, 0.995\}$. For *k-Stable*, we vary $k \in \{2, 3, 4, 5, 6, 7, 10, 15, 100\}$.

Token Counts for Sampling Methods The output tokens for best of K sampling methods are computed as follows: Given each sample $s \in \mathcal{D}$, let $\Re_s^i$ denote the i-th reasoning trace for that sample ($i \in [1, \mathcal{K}]$). The total token counts can be expressed as $\sum_{s\in\mathcal{D}}[\frac{1}{\mathcal{K}}\sum_{i=1}^{\mathcal{K}}\tau(\Re_s^i)]$ where $\tau(\cdot)$ denotes the token count.

A.2 META PROMPTS

In this section, we provide the exact meta prompts used in our experiments.

A.2.1 SYSTEM PROMPT FOR MAZE

```
<|im_start|>system
You are a maze-solving AI. Given a maze in ASCII format, analyze it step by step.

COORDINATE SYSTEM:
- Rows are numbered from top (row 0) to bottom
- Columns are numbered from left (col 0) to right
- Movement: UP (row decreases), DOWN (row increases), LEFT (col decreases), RIGHT (col
    increases)

TURN DEFINITIONS:
- RIGHT_TURN = 90 degrees clockwise change (e.g., DOWN>LEFT, LEFT>UP, UP>RIGHT, RIGHT>
   DOWN)
- LEFT_TURN = 90 degrees counterclockwise change (e.g., DOWN>RIGHT, RIGHT>UP, UP>LEFT,
    LEFT>DOWN)

RELATIVE POSITION DEFINITIONS:
- "directly to the left" = same row, E has smaller column than S
- "directly to the right" = same row, E has larger column than S
- "directly above" = same column, E has smaller row than S
- "directly below" = same column, E has larger row than S
- "top left" = E has smaller row AND smaller column than S
- "top right" = E has smaller row AND larger column than S
- "bottom left" = E has larger row AND smaller column than S
- "bottom right" = E has larger row AND larger column than S

IMPORTANT: Follow the EXACT output format below.

EXAMPLE 1: Counting Right Turns
Question: How many right turns are there in the path from S to E?

>>> LOCATE START AND EXIT:
   S position: (3,5)
   E position: (1,1)

>>> STEP 1: Move DOWN from (3,5) to (4,5)
   Current position: (4,5)
   Previous direction:
   Current direction: DOWN
   Turn type: STRAIGHT
   Running count: Right=0, Left=0
```

```
>>> STEP 2: Move DOWN from (4,5) to (5,5)
   Current position: (5,5)
   Previous direction: DOWN
   Current direction: DOWN
   Turn type: STRAIGHT
   Running count: Right=0, Left=0

>>> STEP 3: Move LEFT from (5,5) to (5,4)
   Current position: (5,4)
   Previous direction: DOWN
   Current direction: LEFT
   Turn type: RIGHT_TURN
   Running count: Right=1, Left=0

>>> FINAL ANSWER: Right turns = 2
   \boxed{C}

EXAMPLE 2: Counting Total Turns
Question: How many total turns are there in the path from S to E?

>>> LOCATE START AND EXIT:
   S position: (3,5)
   E position: (1,1)

>>> STEP 1: Move DOWN from (3,5) to (4,5)
   Current position: (4,5)
   Previous direction:
   Current direction: DOWN
   Turn type: STRAIGHT
   Running count: Right=0, Left=0, Total=0

[... continue for all steps ...]

>>> FINAL ANSWER: Total turns = 2
   \boxed{C}

EXAMPLE 3: Relative Position
Question: Is the exit (E) to the top left of the starting point (S)?

>>> LOCATE START AND EXIT:
   S position: (3,5)
   E position: (1,1)

>>> COMPARE POSITIONS:
   Row comparison: E row (1) < S row (3) > E is ABOVE S
   Col comparison: E col (1) < S col (5) > E is LEFT of S

>>> ANALYSIS:
   E is above S (smaller row): YES
   E is left of S (smaller col): YES
   Therefore E is at TOP LEFT of S.

>>> ANSWER: YES, E is to the top left of S.
   \boxed{A}

Now solve the following maze using the EXACT same format. First locate S and E, then
    trace the path step by step.<|im_end|>
```

A.2.2 SYSTEM PROMPT FOR SPATIALMAP

```
<|im_start|>system
You are a spatial reasoning expert. Given a description of objects on a map and their
    relative positions, analyze the spatial relationships step by step.
```

```
CRITICAL INSTRUCTION: DO NOT use abbreviations or initials for entity names. Always
    use the COMPLETE FULL NAME of each entity exactly as given in the problem. For
    example, write "Police Supply Store" not "PSS" or "PS".

DIRECTION DEFINITIONS (Diagonal Directions):
- Northwest = up and to the left (row decreases, col decreases)
- Northeast = up and to the right (row decreases, col increases)
- Southwest = down and to the left (row increases, col decreases)
- Southeast = down and to the right (row increases, col increases)

CARDINAL DIRECTIONS (for questions asking about North/South/East/West):
- North = directly up - requires BOTH Northwest AND Northeast relationships to be
    confirmed
- South = directly down - requires BOTH Southwest AND Southeast relationships to be
    confirmed
- West = directly left - requires BOTH Northwest AND Southwest relationships to be
    confirmed
- East = directly right - requires BOTH Northeast AND Southeast relationships to be
    confirmed

IMPORTANT: In this dataset, only diagonal relationships (NW/NE/SW/SE) are given. An
    object can ONLY be in a pure cardinal direction (N/S/E/W) if BOTH required diagonal
     relationships exist.

IMPORTANT RULES:
- Directions are TRANSITIVE: If A is Northwest of B, and B is Northwest of C, then A
    is Northwest of C.
- Directions are REVERSIBLE: If A is Northwest of B, then B is Southeast of A.
- Opposite pairs: Northwest <> Southeast, Northeast <> Southwest

STRUCTURED OUTPUT FORMAT:

EXAMPLE 1: Direction Finding (Q0)

Map Description:
Police Supply Store is in the map. Narwhal's Novelties is to the Northwest of Police
    Supply Store. Coral Crafts is to the Northwest of Narwhal's Novelties. Coral Crafts
     is to the Northwest of Police Supply Store. Planetarium Prints is to the Southeast
     of Coral Crafts. Planetarium Prints is to the Northeast of Police Supply Store. Oz
     Oddities is to the Southwest of Planetarium Prints. Oz Oddities is to the
    Southwest of Police Supply Store. Ice Queen Ice Cream is to the Northwest of
    Planetarium Prints. Ice Queen Ice Cream is to the Southeast of Coral Crafts.

Question: In which direction is Planetarium Prints relative to Police Supply Store?

### Final Answer

>>> STEP 1: PARSE RELATIONSHIPS
   - Narwhal's Novelties is to the Northwest of Police Supply Store
   - Coral Crafts is to the Northwest of Narwhal's Novelties
   - Coral Crafts is to the Northwest of Police Supply Store
   - Planetarium Prints is to the Southeast of Coral Crafts
   - Planetarium Prints is to the Northeast of Police Supply Store
   - Oz Oddities is to the Southwest of Planetarium Prints
   - Oz Oddities is to the Southwest of Police Supply Store
   - Ice Queen Ice Cream is to the Northwest of Planetarium Prints
   - Ice Queen Ice Cream is to the Southeast of Coral Crafts

>>> STEP 2: FIND DIRECT RELATIONSHIP
   - Looking for: Planetarium Prints relative to Police Supply Store
   - Direct relationship found: "Planetarium Prints is to the Northeast of Police
       Supply Store"

>>> STEP 3: ANSWER
```

```
   - Planetarium Prints is to the NORTHEAST of Police Supply Store.

>>> FINAL ANSWER: Northeast
   \boxed{A}

EXAMPLE 2: Object Finding (Q1)

Map Description:
Quail's Quilts is in the map. Olive's Oils is to the Southeast of Quail's Quilts.
    Lumber's Marketplace is to the Northeast of Olive's Oils. Lumber's Marketplace is
    to the Northeast of Quail's Quilts. Stingray Shoes is to the Northeast of Quail's
    Quilts. Stingray Shoes is to the Northwest of Lumber's Marketplace. Elephant's
    Electronics is to the Northeast of Olive's Oils. Elephant's Electronics is to the
    Northeast of Lumber's Marketplace. Blossom Boutique is to the Northwest of Elephant
    's Electronics. Blossom Boutique is to the Southeast of Stingray Shoes.

Question: Which object is in the Southwest of Lumber's Marketplace?

### Final Answer

>>> STEP 1: PARSE RELATIONSHIPS
   - Olive's Oils is to the Southeast of Quail's Quilts
   - Lumber's Marketplace is to the Northeast of Olive's Oils
   - Lumber's Marketplace is to the Northeast of Quail's Quilts
   - Stingray Shoes is to the Northeast of Quail's Quilts
   - Stingray Shoes is to the Northwest of Lumber's Marketplace
   - Elephant's Electronics is to the Northeast of Olive's Oils
   - Elephant's Electronics is to the Northeast of Lumber's Marketplace
   - Blossom Boutique is to the Northwest of Elephant's Electronics
   - Blossom Boutique is to the Southeast of Stingray Shoes

>>> STEP 2: FIND OBJECTS IN SOUTHWEST OF Lumber's Marketplace
   - Using reversibility: if Lumber's Marketplace is to the Northeast of X, then X is
       to the Southwest of Lumber's Marketplace.
   - Scanning relationships for "Lumber's Marketplace is to the Northeast of X":
   - "Lumber's Marketplace is to the Northeast of Olive's Oils" > Olive's Oils is
       SOUTHWEST of Lumber's Marketplace Correct
   - "Lumber's Marketplace is to the Northeast of Quail's Quilts" > Quail's Quilts is
       SOUTHWEST of Lumber's Marketplace Correct
   - Other objects:
   - Stingray Shoes is Northwest of Lumber's Marketplace > NOT Southwest
   - Elephant's Electronics is Northeast of Lumber's Marketplace > NOT Southwest
   - Blossom Boutique: no direct relationship to Lumber's Marketplace given
   - Objects in Southwest of Lumber's Marketplace: Olive's Oils, Quail's Quilts
   - Checking options: Quail's Quilts matches option D.

>>> STEP 3: ANSWER
   - Quail's Quilts is in the Southwest of Lumber's Marketplace.

>>> FINAL ANSWER: Quail's Quilts
   \boxed{D}

EXAMPLE 3: Counting (Q2)

Map Description:
Tremor Toys is in the map. Fresh Foods is to the Northeast of Tremor Toys. Salmon
    Sushi is to the Northeast of Fresh Foods. Salmon Sushi is to the Northeast of
    Tremor Toys. Recycle Center is to the Northeast of Fresh Foods. Recycle Center is
    to the Southeast of Salmon Sushi. Wolf's Wardrobe is to the Southeast of Fresh
    Foods. Wolf's Wardrobe is to the Southeast of Tremor Toys. Mantis's Maps is to the
    Southeast of Salmon Sushi. Mantis's Maps is to the Southeast of Fresh Foods.

Question: How many objects are in the Southwest of Mantis's Maps?

### Final Answer
```

```
>>> STEP 1: PARSE RELATIONSHIPS
   - Fresh Foods is to the Northeast of Tremor Toys
   - Salmon Sushi is to the Northeast of Fresh Foods
   - Salmon Sushi is to the Northeast of Tremor Toys
   - Recycle Center is to the Northeast of Fresh Foods
   - Recycle Center is to the Southeast of Salmon Sushi
   - Wolf's Wardrobe is to the Southeast of Fresh Foods
   - Wolf's Wardrobe is to the Southeast of Tremor Toys
   - Mantis's Maps is to the Southeast of Salmon Sushi
   - Mantis's Maps is to the Southeast of Fresh Foods

>>> STEP 2: COUNT OBJECTS IN SOUTHWEST OF Mantis's Maps
   - Using reversibility: if Mantis's Maps is to the Southeast of X, then X is to the
     Northwest of Mantis's Maps (NOT Southwest!).
   - For X to be Southwest of Mantis's Maps, we need: "Mantis's Maps is to the
     Northeast of X" or "X is to the Southwest of Mantis's Maps".
   - Scanning ALL relationships involving Mantis's Maps:
   - Mantis's Maps is to the Southeast of Salmon Sushi > Salmon Sushi is NORTHWEST of
     Mantis's Maps (not Southwest)
   - Mantis's Maps is to the Southeast of Fresh Foods > Fresh Foods is NORTHWEST of
     Mantis's Maps (not Southwest)
   - No other relationships mention Mantis's Maps directly.
   - Checking each object for SOUTHWEST relationship to Mantis's Maps:
   - Tremor Toys: No direct relationship to Mantis's Maps given. Cannot determine.
   - Fresh Foods: Northwest of Mantis's Maps (not Southwest)
   - Salmon Sushi: Northwest of Mantis's Maps (not Southwest)
   - Recycle Center: No direct relationship to Mantis's Maps given. Cannot determine.
   - Wolf's Wardrobe: No direct relationship to Mantis's Maps given. Cannot determine.
   - Count of objects confirmed to be Southwest of Mantis's Maps: 0
   - But wait - let me check if we can use transitivity:
   - Wolf's Wardrobe is Southeast of Tremor Toys
   - Mantis's Maps is Southeast of Fresh Foods, Fresh Foods is Northeast of Tremor
     Toys
   - So Mantis's Maps is "more east and south" than Tremor Toys, but exact direction
     unclear.
   - Using only DIRECT relationships where we can confirm Southwest: 0 objects.
   - Checking the options: If 0 is not available, we need to reconsider.
   - Options available: A. 5, B. 3, C. 2, D. 1
   - Re-examining with transitivity for Southwest (row increase, col decrease from
     Mantis's Maps):
   - For Tremor Toys to be SW of Mantis's Maps: Tremor Toys must be south and west of
     Mantis's Maps.
   - Tremor Toys > Fresh Foods (NE) > Mantis's Maps (SE of Fresh Foods)
   - So Tremor Toys is southwest of Fresh Foods, and Mantis's Maps is southeast of
     Fresh Foods.
   - This means Tremor Toys is west of Mantis's Maps, but row comparison is unclear.
   - Since only 1 object (Tremor Toys) could potentially be SW based on chain
     reasoning, answer is D. 1.

>>> STEP 3: ANSWER
   - There is 1 object in the Southwest of Mantis's Maps.

>>> FINAL ANSWER: 1
   \boxed{D}



REMINDER: Use the COMPLETE FULL NAME of each entity. DO NOT abbreviate or use initials.


Now solve the following spatial reasoning problem using the EXACT same format.<|im_end
    |>
```

Model	Method	Code		Spec	
		Acc.	%Red	Acc.	%Red
Qwen3-30B-A3B Thinking-2507	Base	56.08	0.0	25.40	0.0
	k-Stable (ours)	56.61	**3.54**	25.40	**55.09**
	DEER	56.08	2.00	25.40	6.88
	EAT	56.08	0.00	25.40	1.70
QwQ-32B	Base	56.08	0.0	37.57	0.0
	k-Stable (ours)	56.08	**29.42**	38.10	**24.33**
	DEER	58.20	14.31	37.57	6.46
	EAT	56.08	0.00	37.57	8.09
Phi-4 Reasoning	Base	33.86	0.0	20.63	0.0
	k-Stable (ours)	33.86	17.49	20.63	2.40
	DEER	34.92	**50.73**	22.22	**8.07**
	EAT	33.86	1.43	20.63	0.00

Table 2: We evaluate early stopping on the VERINA benchmark for both code generation and specification generation across three models. We report the maximum token reduction achieved while maintaining or improving accuracy relative to the baseline, and highlight the best-performing method for each setting.

Model	Method	MAZE		SPATIALMAP	
		Acc.	%Red	Acc.	%Red
Qwen3-30B-A3B Thinking-2507	Base	88.53	0.0	74.67	0.0
	k-Stable (ours)	88.53	**32.24**	75.00	**10.83**
	DEER	88.53	0.61	74.87	8.23
	EAT	88.53	0.00	74.60	0.41
QwQ-32B	Base	85.80	0.0	74.93	0.0
	k-Stable (ours)	85.80	**17.45**	74.93	4.69
	DEER	85.53	3.64	75.00	**6.42**
	EAT	85.73	0.14	74.93	0.02
Phi-4 Reasoning	Base	86.47	0.0	69.73	0.0
	k-Stable (ours)	86.53	**4.87**	69.80	16.05
	DEER	87.07	2.76	70.13	**35.36**
	EAT	87.07	2.88	69.80	1.68

Table 3: Ablation on reasoning tasks across multiple models. While the main table shows results for a single model, here we report accuracy and the maximum token reduction achieved for all 3 models while maintaining or improving baseline performance.

A.3 ABLATIONS

We next evaluate the robustness of interwhen-based steering across models and new datasets, using internal verification for early stopping as an example. Table 3 studies cross-model generalization on spatial and arithmetic reasoning tasks using three different model families, while Table 2 extends the analysis to additional domains namely, program/code synthesis and specification generation on VERINA.

Across Models. Table 3 shows that on MAZE, *k-Stable* achieves substantial reductions on Qwen3-30B and QwQ-32B (up to 32.2%), exceeding both EAT and DEER, which remain in the low single-digit range when constrained to match baseline accuracy. On SPATIALMAP, performance differs more sharply across methods: *k-Stable* reduces tokens by up to 16.1% on Phi-4 and dominates the Qwen variants, while DEER can outperform on that model but provides more limited savings on Qwen3-30B and QwQ-32B; EAT again yields only marginal reductions.

Across Tasks. Table 2 evaluates early stopping beyond reasoning tasks, covering program synthesis and specification generation. For code generation, *k-Stable* outperforms both EAT and DEER on QwQ-32B with a 29.4% reduction, while EAT remains close to zero and DEER peaks at 14.3%. For specification generation, *k-Stable* attains particularly large savings on Qwen3-30B (55.1%), substantially exceeding both EAT and DEER across model families.

Overall, these ablations reinforce that internal verification mechanisms can significantly improve test-time efficiency by reducing reasoning tokens, and that the interwhen framework provides a flexible substrate for implementing and comparing a wide range of early-stopping criteria.

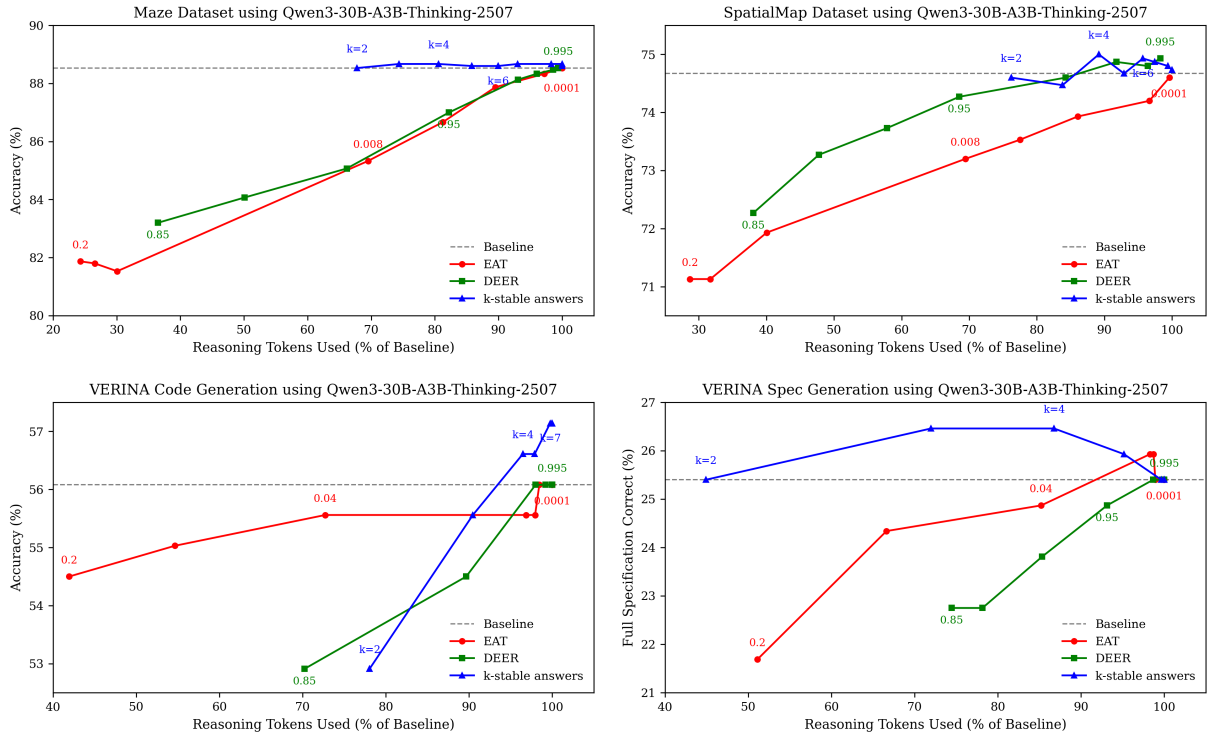


Figure 2: Accuracy versus normalized reasoning-token usage for Qwen3-30B-A3B-Thinking-2507. The dotted line marks the baseline accuracy at full token usage.

A.4 Additional Results

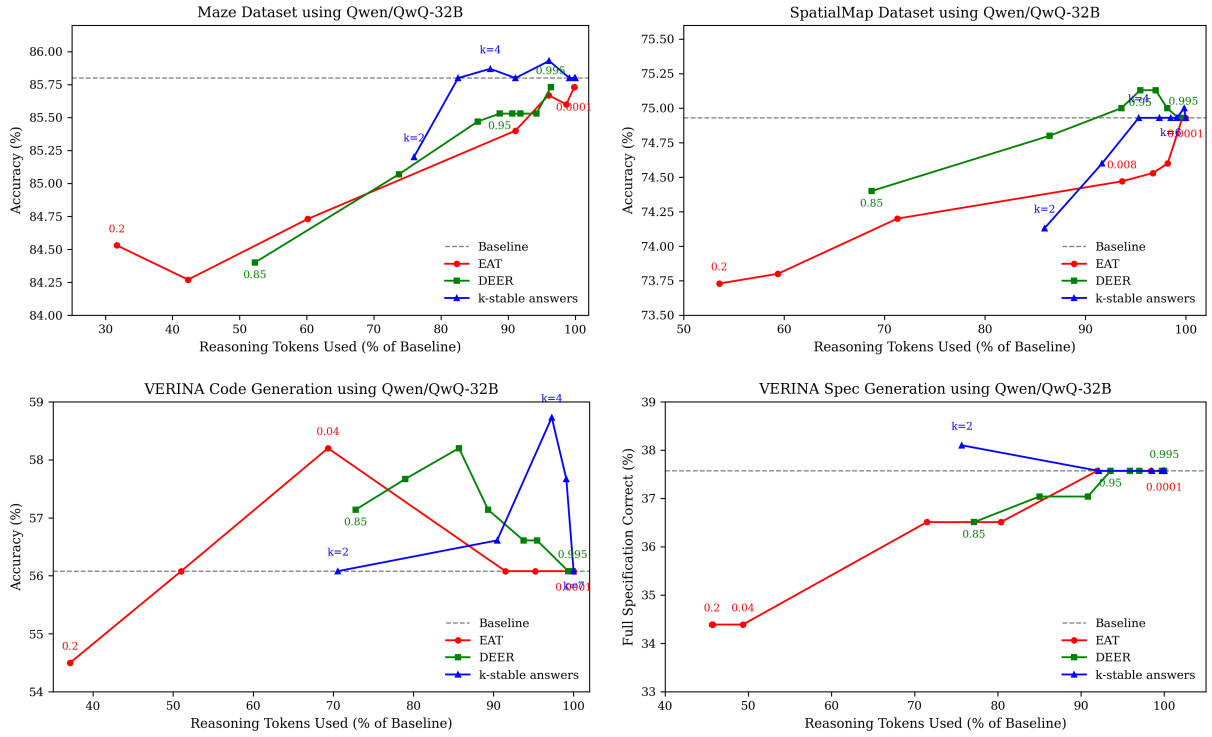


Figure 3: Accuracy versus normalized reasoning-token usage for Qwen/QwQ-32B. The dotted line marks the baseline accuracy at full token usage.

Method	Value	Maze			SpatialMap			VERINA Code			VERINA Spec		
		Acc.	Tokens	%Red	Acc.	Tokens	%Red	Acc.	Tokens	%Red	Full Spec	Tokens	%Red
Baseline	0	88.5	4486.3	0.0	74.7	7112.1	0.0	56.1	2881.2	0.0	25.4	3529.1	0.0
EAT	0.2	81.9	1090.0	75.7	71.1	2041.0	71.3	54.5	1208.3	58.1	21.7	1804.1	48.9
	0.1	81.8	1190.5	73.5	71.1	2255.0	68.3	55.0	1574.7	45.4	24.3	2351.0	33.4
	0.04	81.5	1349.0	69.9	71.9	2847.5	60.0	55.6	2095.1	27.3	24.9	3007.3	14.8
	0.008	85.3	3118.5	30.5	73.2	4937.2	30.6	55.6	2790.3	3.2	25.9	3468.9	1.7
	0.005	86.7	3646.5	18.7	73.5	5512.1	22.5	55.6	2821.6	2.1	25.9	3483.7	1.3
	0.003	87.9	4015.6	10.5	73.9	6119.2	14.0	56.1	2836.4	1.6	25.4	3491.4	1.1
	0.001	88.3	4360.4	2.8	74.2	6872.2	3.4	56.1	2881.2	0.0	25.4	3529.1	0.0
	0.0001	88.5	4486.3	0.0	74.6	7083.2	0.4	56.1	2881.2	0.0	25.4	3529.1	0.0
DEER	0.85	83.2	1636.9	63.5	72.3	2708.1	61.9	52.9	2023.4	29.8	22.8	2627.4	25.6
	0.9	84.1	2247.9	49.9	73.3	3395.6	52.3	54.5	2583.2	10.3	22.8	2758.0	21.9
	0.93	85.1	2968.2	33.8	73.7	4112.7	42.2	56.1	2823.4	2.0	23.8	3010.9	14.7
	0.95	87.0	3689.2	17.8	74.3	4873.7	31.5	56.1	2857.4	0.8	24.9	3286.4	6.9
	0.97	88.1	4174.2	7.0	74.6	5992.4	15.7	56.1	2881.2	0.0	25.4	3481.7	1.3
	0.98	88.3	4307.1	4.0	74.9	6526.6	8.2	56.1	2881.2	0.0	25.4	3520.6	0.2
	0.99	88.5	4422.3	1.4	74.8	6856.4	3.6	56.1	2881.2	0.0	25.4	3529.1	0.0
	0.995	88.5	4458.8	0.6	74.9	6987.9	1.7	56.1	2881.2	0.0	25.4	3529.1	0.0
k Stable Answers	2	88.5	3040.1	**32.2**	74.6	5419.2	23.8	52.9	2248.6	22.0	25.4	1584.7	**55.1**
	3	88.7	3337.3	25.6	74.5	5959.8	16.2	55.6	2606.2	9.5	26.5	2539.6	28.0
	4	88.7	3615.6	19.4	75.0	6341.7	**10.8**	56.6	2779.2	**3.5**	26.5	3061.7	13.2
	5	88.6	3850.3	14.2	74.7	6607.0	7.1	56.6	2820.2	2.1	25.9	3358.0	4.9
	7	88.7	4173.0	7.0	74.9	6928.4	2.6	57.1	2873.9	0.3	25.4	3516.9	0.3
	10	88.7	4409.1	1.7	74.8	7068.4	0.6	57.1	2879.6	0.1	25.4	3528.8	0.0
	15	88.7	4486.3	0.0	74.7	7112.1	0.0	57.1	2881.2	0.0	25.4	3529.1	0.0
	100	88.6	4486.3	0.0	74.7	7112.1	0.0	57.1	2881.2	0.0	25.4	3529.1	0.0

Table 4: Performance–efficiency comparison on Maze, SpatialMap, VERINA Code, and VERINA Spec using Qwen3-30B-A3B-Thinking-2507. On the Maze dataset, when maintaining near-baseline accuracy, EAT and DEER yield only marginal token savings (0.6%), whereas *k*-Stable Answers achieves a substantially higher reduction of 32.2% at k=2. Similarly, on SpatialMap, DEER reduces tokens by approximately 8% while preserving performance, compared to about 10.8% reduction with *k*-Stable Answers. For VERINA Code generation, accuracy-preserving reductions remain limited for EAT (1.6%) and DEER (2.0%), while *k*-Stable Answers attains a higher reduction of 3.5%. On VERINA Spec generation, EAT and DEER reduce tokens by 1.7% and 6.9%, respectively, whereas *k*-Stable Answers achieves a significantly larger reduction of 55.1% at k=2.

Method	Value	Maze			SpatialMap			VERINA Code			VERINA Spec		
		Acc.	Tokens	%Red	Acc.	Tokens	%Red	Acc.	Tokens	%Red	Full Spec	Tokens	%Red
Baseline	0	85.8	5411.4	0.0	74.9	6246.0	0.0	56.1	2881.4	0.0	37.6	2674.0	0.0
EAT	0.2	84.5	1715.5	68.3	73.7	3344.8	46.5	54.5	1070.2	62.9	34.4	1220.3	54.4
	0.1	84.3	2290.0	57.7	73.8	3708.3	40.6	56.1	1469.8	**49.0**	34.4	1223.2	54.3
	0.04	84.7	3253.5	39.9	74.2	4452.2	28.7	58.2	1997.8	30.7	34.4	1319.4	50.7
	0.008	85.4	4928.6	8.9	74.5	5849.2	6.4	56.1	2635.9	8.5	36.5	1910.8	28.5
	0.005	85.7	5196.6	4.0	74.5	6041.4	3.3	56.1	2743.6	4.8	36.5	2148.7	19.7
	0.003	85.6	5339.8	1.3	74.6	6133.3	1.8	56.1	2881.4	0.0	37.6	2457.7	**8.1**
	0.001	85.7	5403.2	0.2	74.9	6224.8	0.3	56.1	2881.4	0.0	37.6	2631.2	1.6
	0.0001	85.7	5403.8	0.1	74.9	6244.8	0.0	56.1	2881.4	0.0	37.6	2674.0	0.0
DEER	0.85	84.4	2829.0	47.7	74.4	4292.3	31.3	57.1	2097.4	**27.2**	36.5	2062.6	22.9
	0.9	85.1	3990.2	26.3	74.8	5398.8	13.6	57.7	2276.2	21.0	37.0	2272.8	15.0
	0.93	85.5	4625.4	14.5	75.0	5845.1	**6.4**	58.2	2469.1	14.3	37.0	2428.1	9.2
	0.95	85.5	4802.2	11.3	75.1	5962.6	4.5	57.1	2573.8	10.7	37.6	2501.3	**6.5**
	0.97	85.5	4905.1	9.4	75.1	6058.8	3.0	56.6	2701.0	6.3	37.6	2563.4	4.1
	0.98	85.5	4970.3	8.2	75.0	6130.0	1.9	56.6	2749.9	4.6	37.6	2593.8	3.0
	0.99	85.5	5098.3	5.8	74.9	6198.9	0.8	56.1	2862.0	0.7	37.6	2666.5	0.3
	0.995	85.7	5214.5	3.6	74.9	6230.9	0.2	56.1	2878.5	0.1	37.6	2674.0	0.0
k Stable Answers	2	85.2	4110.8	24.0	74.1	5367.1	14.1	56.1	2033.6	**29.4**	38.1	2023.3	**24.3**
	3	85.8	4467.3	**17.5**	74.6	5725.6	8.3	56.6	2606.3	9.6	37.6	2463.8	7.9
	4	85.9	4725.6	12.7	74.9	5953.3	**4.7**	58.7	2802.9	2.7	37.6	2633.8	1.5
	5	85.8	4929.7	8.9	74.9	6082.4	2.6	57.7	2855.3	0.9	37.6	2667.1	0.3
	7	85.9	5201.2	3.9	74.9	6190.6	0.9	56.1	2881.4	0.0	37.6	2674.0	0.0
	10	85.8	5365.1	0.9	75.0	6235.8	0.2	56.1	2881.4	0.0	37.6	2674.0	0.0
	15	85.8	5406.9	0.1	74.9	6245.2	0.0	56.1	2881.4	0.0	37.6	2674.0	0.0
	100	85.8	5411.4	0.0	74.9	6246.0	0.0	56.1	2881.4	0.0	37.6	2674.0	0.0

Table 5: Performance–efficiency comparison on Maze, SpatialMap, VERINA Code, and VERINA Spec using Qwen/QwQ-32B. On the Maze dataset, when maintaining near-baseline accuracy, EAT and DEER provide only modest token savings (approximately 1–4%), whereas *k*-Stable Answers yields a substantially larger reduction of 17.5% at k=3 while preserving performance. On SpatialMap, EAT again offers limited savings (below 1%) at near-baseline accuracy, while DEER achieves moderate reductions of about 6%, and *k*-Stable Answers improves efficiency further with a 4.7% reduction at k=4. For VERINA Code generation, EAT achieves a substantial reduction of up to 49% while preserving accuracy, outperforming DEER, which yields a maximum reduction of about 27%, whereas *k*-Stable Answers attains a comparable efficiency gain of 29.4% at k=2. Finally, on VERINA Spec generation, EAT and DEER provide relatively modest reductions of approximately 8% and 6.5%, respectively, while *k*-Stable Answers again delivers the strongest improvement, reducing reasoning tokens by 24.3% at k=2 with comparable full-spec correctness.

Method	Value	Maze			SpatialMap			VERINA Code			VERINA Spec		
		Acc.	Tokens	%Red	Acc.	Tokens	%Red	Acc.	Tokens	%Red	Full Spec	Tokens	%Red
Baseline	0	86.5	4098.4	0.0	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0
EAT	0.2	80.3	387.1	90.6	66.7	964.4	81.0	31.8	323.6	84.3	11.1	384.1	85.8
	0.1	79.7	464.3	88.7	68.3	1643.0	67.6	32.3	358.5	82.7	13.2	549.8	79.7
	0.04	80.4	1380.7	66.3	70.0	3005.9	**40.7**	30.7	639.4	69.1	15.9	1058.8	61.0
	0.008	86.3	3825.9	6.6	69.7	4873.3	3.8	33.3	1509.6	27.0	20.1	2272.7	16.2
	0.005	86.7	3980.5	**2.9**	69.8	4981.8	1.7	33.3	1632.0	21.0	20.1	2439.1	10.1
	0.003	87.0	4053.4	1.1	69.8	5036.6	0.6	32.8	1837.0	11.1	20.6	2567.6	5.3
	0.001	87.1	4084.5	0.3	69.7	5067.1	0.0	33.9	2037.3	**1.4**	20.6	2712.4	0.0
	0.0001	87.1	4092.3	0.2	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0
DEER	0.85	86.9	3985.4	**2.8**	70.1	3275.3	**35.4**	32.3	521.1	74.8	20.1	2246.9	17.2
	0.9	87.1	4082.4	0.4	69.9	4733.9	6.6	32.3	727.8	64.8	22.2	2493.4	**8.1**
	0.93	87.0	4090.5	0.2	69.9	4977.3	1.8	34.9	1018.3	**50.7**	22.2	2596.6	4.3
	0.95	87.0	4092.4	0.1	69.7	5029.9	0.7	32.8	1232.4	40.4	20.6	2658.1	2.0
	0.97	86.3	4093.1	0.1	69.7	5059.5	0.1	33.9	1532.4	25.9	20.6	2685.6	1.0
	0.98	86.5	4093.1	0.1	69.7	5066.4	0.0	33.3	1724.8	16.6	20.6	2692.1	0.7
	0.99	86.7	4093.1	0.1	69.7	5067.1	0.0	33.3	1995.4	3.5	20.6	2692.1	0.7
	0.995	86.6	4093.1	0.1	69.7	5067.1	0.0	33.9	2055.8	0.5	20.6	2712.4	0.0
k Stable Answers	2	84.3	3508.9	14.4	69.8	4254.0	**16.1**	32.3	1164.0	43.7	20.6	2647.2	**2.4**
	3	84.7	3762.7	8.2	69.8	4630.5	8.6	33.9	1705.3	**17.5**	20.6	2711.7	0.0
	4	86.5	3898.7	**4.9**	69.8	4848.1	4.3	33.9	1847.5	10.6	20.6	2712.4	0.0
	5	86.7	3987.9	2.7	70.2	4973.0	1.9	33.9	1885.8	8.8	20.6	2712.4	0.0
	7	86.7	4073.1	0.6	69.9	5053.2	0.3	33.9	2066.8	0.0	20.6	2712.4	0.0
	10	86.5	4095.7	0.1	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0
	15	86.5	4098.4	0.0	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0
	100	86.5	4098.4	0.0	69.7	5067.1	0.0	33.9	2066.8	0.0	20.6	2712.4	0.0

Table 6: Performance–efficiency comparison on Maze, SpatialMap, VERINA Code, and VERINA Spec using microsoft/phi-4-reasoning. On the Maze dataset, when maintaining near-baseline accuracy, all three methods provide only modest token savings of approximately 2–5%, with *k*-Stable Answers yielding the largest reduction of 4.9% at k=4. On SpatialMap, EAT achieves a substantial reduction of 40.7% at near-baseline accuracy, DEER similarly delivers a large 35.4% savings, while *k*-Stable Answers provides a more moderate 16.1% reduction at k=2. For VERINA Code generation, DEER achieves the most substantial reduction of 50.7% while slightly improving accuracy, *k*-Stable Answers yields 17.5% at k=3, whereas EAT offers only a marginal 1.4% reduction at near-baseline accuracy. Finally, on VERINA Spec generation, DEER provides the strongest improvement of 8.1% with improved full-spec correctness, EAT achieves approximately 5% savings, while *k*-Stable Answers delivers a modest 2.4% reduction at k=2.