Thought-Level Beam Search for Reasoning

Lijie Yang
Princeton University
ly3223@princeton.edu

Hongyin Luo
MIT CSAIL
hyluo@mit.edu

Jiawei Zhao
Meta AI
jwzhao@meta.com

Tri Dao*
Princeton University
tridao@princeton.edu

Ravi Netravali*
Princeton University
rnetravali@cs.princeton.edu

Abstract

Test-time compute scaling is a primary driver of performance in large reasoning models (LRMs), but extreme inefficiency bounds current approaches, shifting the critical question from *how much* compute to spend, to *where* to allocate it. We formalize test-time reasoning as a constrained compute allocation problem over partial trajectories. Under a fixed hardware budget, existing paradigms fail to actively allocate the compute to the most promising partial progress: traditional parallel sampling treats traces independently and induces severe memory bottlenecks, while subtractive pruning starves hardware and fails to actively and sufficiently shift the output distribution. To overcome this dichotomy, we introduce Gambit, an inference algorithm that executes *thought-level beam search*. By periodically pruning unpromising trajectories and immediately branching from high-quality prefixes, Gambit dynamically concentrates compute onto the most promising reasoning traces via a light-weight scorer probing hidden states while maintaining continuous high hardware utilization. Extensive evaluations across multiple models and benchmarks demonstrate that Gambit strictly dominates existing baselines. Under identical hardware constraints, our method yields up to a +6.7% absolute accuracy gain on HMMT-24 and +3.3% on AIME-25 over pruning baselines, delivers $>2\times$ higher throughput on trace completion, and reduces total token consumption by up to 68.5% relative to standard parallel sampling [1].

1 Introduction

Large reasoning models (LRMs) have recently demonstrated impressive capabilities on challenging mathematical and scientific tasks by generating long chains of thought during inference (Anthropic, 2025; OpenAI, 2025a;b; DeepSeek-AI, 2025; Wei et al., 2023; Team, 2025). A key driver of these accuracy gains is **test-time compute scaling**, or allocating additional compute during inference, typically through parallel sampling of reasoning traces (Muennighoff et al., 2025; Kojima et al., 2023). Self-consistency, which aggregates answers from multiple independently sampled traces, is the canonical implementation of this strategy (Wang et al., 2023).

However, the current paradigm is reaching diminishing returns due to extreme inefficiency. As shown in Figure 1, parallel sampling requires generating hundreds of long reasoning traces, the vast majority of which can lead to incorrect answers. Specifically, one NVIDIA B300, the state-of-the-art hardware, needs several hours to complete 512 traces for a single AIME-2025 problem on vLLM (Kwon et al., 2023) with Qwen3-8B model (Team, 2025) while still failing on the hardest instances. As a result, it becomes necessary to ensure that the scaled inference can effectively lead to performance improvements. In other words, the central question for the field has shifted from ***how much*** compute to spend to ***where*** to allocate it.

At its core, test-time reasoning can be viewed as a problem of allocating compute over a population of partial trajectories under strict hardware constraints. Making this allocation

*Equal advising.
[1]Code is available at https://github.com/Dao-AILab/Gambit.git.

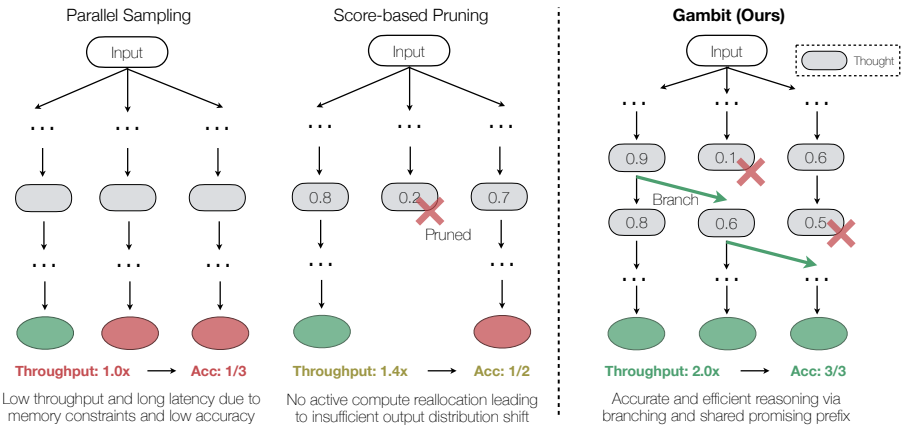

Figure 1: **Reasoning topologies for test-time compute allocation. (Left)** Parallel sampling treats traces as independent trials, leading to wrong answers in majority. **(Center)** Score-based pruning terminates unpromising paths but leaves the freed capacity idle, failing to actively shift the sampling distribution. **(Right)** Gambit actively reallocates compute: when low-scoring traces are pruned, it immediately branches from high-quality prefixes to amplify correct reasoning and saturate the compute budget.

effective requires resolving three tightly coupled challenges:(1) *checkpointing partial trajectories worth continuing*, (2) *allocating additional compute to amplify them*, and (3) *doing so while keeping the hardware fully utilized*.

Checkpointing promising partial trajectories. Reasoning is inherently fragile, yet successful and failed trajectories often share rigorous intermediate prefixes before diverging (Lightman et al., 2023; Dziri et al., 2023; Song et al., 2026; Uesato et al., 2022). Capturing these states during generation is therefore critical. Existing evaluators—from costly Process Reward Models (Wang et al., 2024; Xie et al., 2024; Yao et al., 2023) to lightweight internal signals (Fu et al., 2025; Kadavath et al., 2022; Lin et al., 2022)—can identify promising steps, but without checkpointing, a single downstream error discards all prior valid progress, wasting previous compute. Naive reliance on evaluators is also brittle-early reasoning is incomplete and yields noisy, unreliable signals (Hao et al., 2023). An effective strategy must therefore combine efficient evaluation with principled control over *when* a prefix is reliable enough to checkpoint.

Allocating compute to amplify promising trajectories. Yet, successfully checkpointing a reliable prefix is only the first step. On challenging problems, correct solutions are exceedingly rare and easily overwhelmed by plausible but incorrect reasoning paths (Lee & Hockenmaier, 2025; Fu et al., 2025). Consequently, purely subtractive strategies—which attempt to improve efficiency and accuracy by merely terminating low-quality traces—are fundamentally insufficient (Liang et al., 2026; Hong et al., 2025). That is because the underlying output distribution is largely left unchanged. To overcome this, freed compute must be *actively reallocated* to expand high-quality prefixes into more continuations, fundamentally shifting the output distribution toward the correct answer.

Sustaining hardware utilization. Finally, active reallocation must respect strict GPU memory constraints. Unmanaged parallel sampling saturates memory early, inducing severe queueing delays in modern serving engines (Kwon et al., 2023; Zheng et al., 2024). Conversely, pruning-based methods (Liang et al., 2026; Fu et al., 2025; Hong et al., 2025) alleviate memory pressure but induce hardware starvation: terminated traces are discarded without replacement, leaving GPU resources increasingly idle. An optimal approach should balance these extremes, maintaining a constant level of active computation to ensure sustained hardware utilization throughout decoding.

To address these limitations, we propose **Gambit**, an inference algorithm that formulates test-time reasoning as a *thought-level beam search* over partial trajectories. Our design mirrors the three challenges above. First, we evaluate and checkpoint promising prefixes using efficient,

model-internal signals—either off-the-shelf hidden-state probes adapted from STEP (Liang et al., 2026) or a custom sequence scorer. To overcome the inherent early-stage noise of these signals, Gambit introduces a strict *warmup threshold*. This algorithmic constraint delays all scoring and checkpointing until a sufficient reasoning depth is reached, ensuring the system only checkpoints logically stable foundations. Second, we amplify these prefixes through a periodic **beam search** procedure that performs synchronized prune-and-branch operations, reallocating compute toward high-quality trajectories. Third, we enforce a *zero-sum allocation policy* that maintains a constant-size pool of active traces, ensuring sustained hardware utilization throughout generation. Furthermore, to balance exploration and exploitation and prevent greedy collapse, Gambit utilizes a decoupled memory management scheme to avoid aggressively branching over a single top trajectory. Together, these components transform test-time scaling from passive sampling into an active, system-aware search process.

To validate our approach, we deploy Gambit system atop vLLM and evaluate across a suite of challenging reasoning benchmarks (AIME, HMMT, and GPQA-Diamond). Across varying model architectures, active thought-level search strictly dominates both static and subtractive baselines. Under identical hardware constraints, Gambit improves accuracy by **+6.7%** on HMMT-24 and **+3.3%** on AIME-25 over aggressive pruning methods, while reducing total token consumption by up to **68.5%** relative to standard parallel sampling. Furthermore, Gambit sustains continuous hardware saturation, delivering over $2\times$ higher productive trace throughput with $<1\%$ system overhead.

2 Related Work

Test-Time Compute Scaling. Scaling compute at inference time is a primary driver of modern reasoning performance (OpenAI, 2025b; DeepSeek-AI, 2025; Muennighoff et al., 2025; Team et al., 2025). The popular paradigm, Self-Consistency (SC) (Wang et al., 2023), improves accuracy by aggregating traces via majority voting. However, SC scales highly inefficiently: because trajectories are sampled entirely independently, the model repeatedly allocates compute to the redundant exploration of degenerate pathways (Song et al., 2026).

Inference-Time Search. To improve sample efficacy, methods such as Tree-of-Thoughts (ToT) (Yao et al., 2023) and Monte Carlo Tree Search (MCTS) (Xie et al., 2024) frame reasoning as a structured search guided by value function or process reward models (PRMs) (Lightman et al., 2023; Wang et al., 2024). While MCTS balances exploration and exploitation, its asymmetric expansion and asynchronous rollouts clash with the synchronous, large-batch nature of modern LLM inference. Gambit bridges this gap by reformulating search as a hardware-constrained, synchronous beam allocation by maintaining a constant-size active pool via zero-sum prune-and-branch operations.

Efficient Serving and Pruning-based Approaches. Large-batch, long-context reasoning serving imposes severe KV-cache pressure (Kwon et al., 2023; Zheng et al., 2024). To mitigate this, recent works propose subtractive pruning: DeepConf (Fu et al., 2025) and STEP (Liang et al., 2026) early-terminate unpromising traces using internal model signals, while Slim-SC (Hong et al., 2025) deduplicates via similarity. However, these purely subtractive methods terminate traces without reallocating the freed compute, failing to sufficiently shift the underlying generation distribution. Gambit resolves this by active compute reallocation, dynamically amplifying the presence of correct traces in the ensemble while eliminating the hardware starvation inherent to pruning.

3 Motivation

As we discuss in Section 1, an effective test-time strategy in reasoning must (i) allocate compute toward promising partial trajectories and (ii) maintain high hardware utilization. We now provide two empirical observations motivating these requirements.

High-quality prefixes manifest enhanced compute efficacy. Recent work shows that successful and failed trajectories often exhibit promising high-quality prefixes well before completion (Uesato et al., 2022; Wang et al., 2024), indicating that useful computation is disproportionately concentrated in early intermediate states. We quantify this in Figure 2: pausing 64 parallel traces midway, ranking them with a lightweight hidden-state scorer adapted from STEP (Liang et al., 2026), and branching exclusively from the highest-scoring

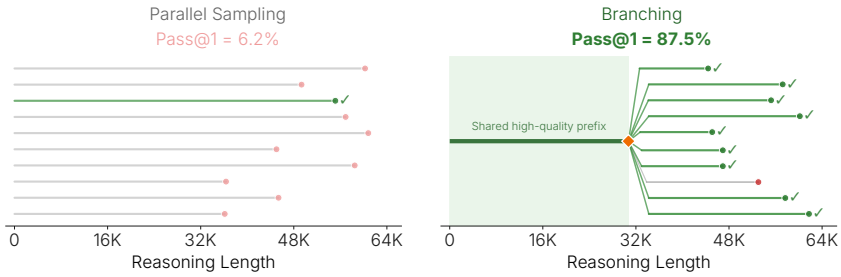

Figure 2: Branching from a high-quality prefix versus independent sampling on a hard AIME 2025 problem (Q27). Branching 64 continuations from the top-ranked prefix achieves 87.5% pass@1 (14× over 6.2% baseline) while sharing the parent's KV-cache (green region) halves memory consumption.

prefix yields dramatic accuracy gains while consuming roughly half the token budget. We observe similarly large lifts on other hard problems (e.g., 1.6% → 39.1% on AIME-25 Q12). While these results highlight the extreme inefficiency of uniform parallel sampling, naively over-committing to a single prefix easily leads to confident but incorrect trajectories (Fu et al., 2025). This necessitates *controlled reallocation*—maintaining a beam of candidates while progressively concentrating compute on the most promising subset.

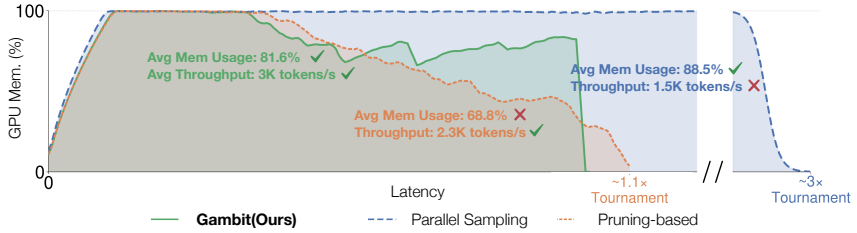

Figure 3: GPU memory utilization and latency profiles during reasoning with batch_size=256 traces on HMMT-25 Q7. Parallel sampling exhausts KV-cache capacity, inflating latency by ∼3×. Pruning-only STEP (Liang et al., 2026) eliminates queueing but steadily loses concurrency. Gambit maintains high utilization throughout at ∼1.1× latency.

Reasoning workloads underutilize hardware. Efficiently serving long reasoning traces in large batches requires balancing concurrency with KV-cache capacity. Standard parallel sampling launches many concurrent traces, but long contexts quickly saturate the KV cache. As illustrated in Figure 3, this forces inference engines such as vLLM (Kwon et al., 2023) or SGLang (Zheng et al., 2024) to severely queue requests, heavily inflating end-to-end latency. Pruning-based approaches alleviate memory pressure (Fu et al., 2025; Liang et al., 2026), but introduce the opposite problem: hardware starvation. Because pruned traces are permanently discarded and not replaced, concurrency steadily declines during generation. As shown in Figure 3, this leaves GPU resources increasingly idle for the remainder of the generation window.

4 Method: Gambit

Motivated by the observations in Section 3, we formalize test-time reasoning as a *constrained compute-allocation problem* over a dynamic population of partial reasoning trajectories. Given a problem prompt $\mathcal{P}$, the objective is to discover a thought-level allocation policy π that maximizes the probability of deriving the ground-truth solution y^*, subject to a strict inference-time hardware budget:

$$\max_{\pi} \mathbb{P}\big(\mathcal{A}_\pi(\mathcal{P}) = y^* \mid \mathcal{P}\big) \qquad \text{s.t.} \qquad \Omega(\pi) \le B, \tag{1}$$

where $\mathcal{A}_\pi(\mathcal{P})$ denotes the final aggregated answer produced by policy π, and $\Omega(\pi)$ represents the peak computational and memory cost incurred during execution. In the context of autoregressive large language models, this hardware budget B manifests primarily as a hard physical constraint on the maximum number of concurrently active traces (preventing compute starvation) and the total KV-cache memory footprint.

A key empirical premise, established in Section 3, is that hidden-state-based scoring functions can identify promising reasoning prefixes midway through generation, and that continuing from such prefixes yields a higher probability of producing a correct answer. This implies that uniform allocation of compute across independent traces is suboptimal. Inspired by beam search (Lowerre, 1976; Sutskever et al., 2014), we therefore instantiate inference as a *thought-level beam search*. Unlike classical token-level beam search—which optimizes sequence likelihood—our objective is to allocate limited compute toward prefixes most likely to yield a correct solution under a strict capacity constraint. Concretely, we operate at the granularity of reasoning steps, use a scoring function as a proxy for continuation value, and maintain a fixed-size active pool throughout execution.

To realize this policy under real hardware constraints, we introduce a fixed-budget algorithm that performs periodic, zero-sum reallocation of compute across traces while maintaining a constant memory footprint (Section 4.1). We further design a decoupled memory management scheme that separates logical search decisions from physical execution state, preventing pathological accuracy collapse under memory pressure (Section 4.2).

4.1 Thought-Level Beam Search

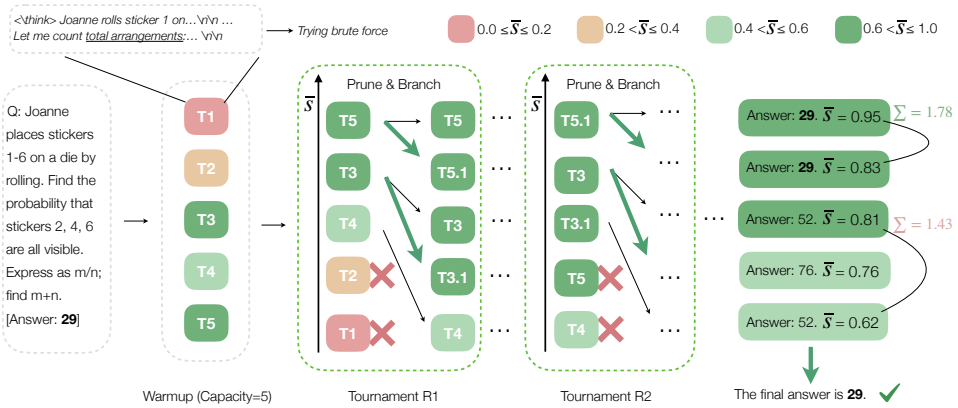

Figure 4: **End-to-end pipeline of Gambit on an AIME problem (capacity C=5, swap size K=2).** During warmup, C=5 parallel traces are evaluated (color intensity denotes running score $\bar{s} \in [0,1]$). Every Δ steps, a tournament ranks the active traces: the K=2 lowest-scoring traces are pruned (×) and replaced by new branches spawned from the highest-scoring prefixes via prefix caching (green arrows). This zero-sum reallocation maintains exactly C active traces throughout generation. Upon completion, answers are aggregated via a score-weighted majority vote, correctly selecting answer 29 ($\sum \bar{s} = 1.78$ vs. 1.43).

We now describe our solution to the constrained compute-allocation problem defined in Equation (1). The full algorithm and visualization are in Algorithm 1 and Figure 4. The central challenge is to allocate limited decoding capacity across a population of partial reasoning trajectories so as to maximize the probability of producing a correct final answer.

Gambit performs a zero-sum reallocation of compute over a fixed-capacity pool of reasoning traces. At each round, the algorithm ranks all active trajectories using a scoring function, removes the lowest-scoring traces to free capacity, and immediately reallocates that capacity by branching from the highest-scoring prefixes.

More specifically, each reasoning trajectory is modeled as a sequence of discrete steps ("thoughts") separated by "\n\n" (Pan et al., 2025), denoted by $\tau = (s_1, s_2, \ldots, s_n)$ and C be the operator-specified hardware *capacity* (the maximum number of concurrent traces permitted by the system). Gambit operates in rounds triggered every Δ steps across all running traces (the *check interval*). At each tournament round, let $\mathcal{A}$ denote the set of currently active traces with $|\mathcal{A}| = N$. Let $\mathcal{B} \subseteq \mathcal{A}$ be the *eligible branching candidates*—traces that have generated at least w tokens since their own creation, a constraint that prevents cascading branch explosions from immature children. Traces in $\mathcal{A}$ are ranked by their average score $\bar{s}_i = \frac{1}{n}\sum_{j=1}^{n} f_\theta(\mathbf{h}_{i,j})$,

Algorithm 1 Thought-Level Beam Search in Gambit)

Input: Problem prompt $\mathcal{P}$, capacity C, swap size K, check interval Δ, warmup w, scorer f_θ
Output: Score-weighted majority-vote answer a^*

Main Loop:
$\mathcal{S} \leftarrow \{C \text{ initial traces from } \mathcal{P}\}$ ▷ *Scheduler view*
$\mathcal{T}_{\text{act}} \leftarrow \mathcal{S}$ ▷ *Tree view*
$\mathcal{F} \leftarrow \emptyset$ ▷ *Completed traces*
while $|\mathcal{S}| > 0$ **and** $|\mathcal{F}| < C$ **do**
  GENERATE Advance one thought $\forall \tau \in \mathcal{S}$
  For each τ completing step s_n:
    $\bar{s}_\tau \leftarrow \bar{s}_\tau + \frac{1}{n}(f_\theta(\mathbf{h}_n) - \bar{s}_\tau)$ ▷ *Cumulative score*
  SCHEDULE *On GPU memory pressure*
  if KV-cache is saturated **then**
    $\tau_{\text{vic}} \leftarrow \text{argmin}_{\tau \in \mathcal{S}} \bar{s}_\tau$
    $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\tau_{\text{vic}}\}$ ▷ *Evict as ghost trace*
  end if
  TOURNAMENT *Every Δ thoughts*
  if tournament condition met **then**
    $\mathcal{T}_{\text{act}}, \mathcal{S} \leftarrow \text{TOURNAMENT}(\mathcal{T}_{\text{act}}, \mathcal{S}, C, K)$
  end if
  Move completed traces to $\mathcal{F}$
end while

Function TOURNAMENT($\mathcal{T}_{\text{act}}, \mathcal{S}, C, K$):
$N \leftarrow |\mathcal{T}_{\text{act}}|$
Rank $\tau \in \mathcal{T}_{\text{act}}$ by $\bar{s}_\tau$ descending
$\mathcal{B} \leftarrow \{\tau \in \mathcal{T}_{\text{act}} : \text{GenLen}(\tau) \geq w\}$ ▷ *Eligible parents*
if $N < C$ **then**
  ▷ ***Case 1:*** *Under-capacity*
  $m \leftarrow \min(C - N, |\mathcal{B}|)$
  $\mathcal{C}_{\text{new}} \leftarrow \text{Branch}(\text{Top-}m \text{ from } \mathcal{B})$
else if $N = C$ **then**
  ▷ ***Case 2:*** *At capacity*
  $\mathcal{V} \leftarrow \text{Bottom-}K \text{ from } \mathcal{T}_{\text{act}}$
  $\mathcal{T}_{\text{act}} \leftarrow \mathcal{T}_{\text{act}} \setminus \mathcal{V}$; $\mathcal{S} \leftarrow \mathcal{S} \setminus \mathcal{V}$ ▷ *Prune bottom-k*
  $\mathcal{C}_{\text{new}} \leftarrow \text{Branch}(\text{Top-}K \text{ from } \mathcal{B})$ ▷ *Branch top-k*
end if
$\mathcal{T}_{\text{act}} \leftarrow \mathcal{T}_{\text{act}} \cup \mathcal{C}_{\text{new}}$
$\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{C}_{\text{new}}$
return $\mathcal{T}_{\text{act}}, \mathcal{S}$

Notation: $\mathbf{h}_n$ = last-layer hidden state at step n; Branch(τ) = spawn child from τ's prefix via KV-cache sharing; w = warmup threshold (min steps before eligible for branching).

return $a^* = \text{argmax}_a \sum_{\tau \in \mathcal{F}: \text{ans}(\tau) = a} \bar{s}_\tau$

where f_θ can be any scorer function that takes $\mathbf{h}_{i,j}$, the last-layer hidden state at the boundary of step j in trace τ_i, as input. In the evaluation, Gambit uses the light-weight 2-Layer MLP scorer from STEP (Liang et al., 2026) or custom sequence scorer trained in Section A.1.

It then forms an ordered list $\tau_{(1)}, \ldots, \tau_{(n)}$ with $\bar{s}_{(1)} \geq \cdots \geq \bar{s}_{(n)}$ and two routing cases arise:

Case 1 — Under-capacity ($N < C$): The pool has $C - N$ vacant slots. Gambit selects the top $\min(C - N, |\mathcal{B}|)$ eligible traces and *branches* each one, filling the pool back to capacity:

$$\text{Branch}\left(\tau_{(k)}\right) \text{ for } k = 1, \ldots, \min(C - N, |\mathcal{B}|). \tag{2}$$

Case 2 — At capacity ($N = C$): Gambit executes a rank-based *swap* of size K:

$$\text{Prune}\left(\tau_{(N-k+1)}\right); \text{Branch}\left(\tau_{(k)}\right) \text{ for } k = 1, \ldots, K, \tag{3}$$

Pruning immediately terminates the K lowest-scoring traces and frees their non-shared KV-cache blocks. Branching spawns K new child requests from the K highest-scoring prefixes. The child inherits the parent's KV-cache via prefix caching with minimal overhead. A temperature multiplier can be applied to the child to promote diversity. Because pruning and branching are paired identically, the total active count remains exactly C, enforcing a strict zero-sum memory invariant.

4.2 Systems Integration: Decoupled Memory Management

A critical failure mode arises when logical search decisions are tightly coupled to physical execution state. Under memory pressure, the inference engine often preempts or, in our setting, prunes running traces to satisfy capacity constraints. If the search algorithm directly observes these evictions, the effective number of active traces drops below capacity, triggering repeated under-capacity branching on single Top-1/2 trace. This induces a pathological feedback loop in which compute is repeatedly concentrated on a small set of high-scoring prefixes, collapsing the search distribution and degrading solution quality.

To prevent this, Gambit introduces a **Decoupled View Architecture**, referred as Scheduler/Tree view in Algorithm 1:

- The Scheduler View (Memory Manager): Tracks only traces currently holding physical KV-cache blocks. It opportunistically evicts the lowest-ranked running trace when memory approaches saturation to prevent pushing traces to waiting queue.

- **The Tree View (Beam Search):** Tracks the logical topology of all active traces. Crucially, a trace evicted by the scheduler becomes a "ghost trace"—it generates no new tokens and holds no physical memory, but its `is_active` flag remains `True` in the logical tree.

By decoupling these views, Gambit executes its capacity checks against the *Tree View* rather than the *Scheduler View*. When the scheduler drops a trace, the logical count remains $N = C$. Consequently, the tournament correctly perceives the system as being at capacity and executes a balanced Case 2 swap (pruning the bottom-K and branching the top-K), completely bypassing the greedy Case 1 fallback. Ghost traces simply retain their logical position until they naturally fall into the bottom-K and are permanently pruned. This ensures that system-level scheduling does not distort the intended allocation policy over trajectories.

Termination & Aggregation. Because ghost traces remain logically active but generate no tokens, Gambit monitors the Scheduler View to determine termination. When the physical running count reaches zero, or when C traces have successfully completed, generation stops. Each finished trace τ_i contributes its extracted answer a_i with weight $\bar{s}_i$ to a score-weighted majority vote $a^* = \text{argmax}_a \sum_{i:a_i=a} \bar{s}_i$.

5 Experiments

Our empirical evaluation is designed to answer two primary research questions: (*i*) Does Gambit strictly improve downstream reasoning accuracy over subtractive pruning and unmanaged sampling under the same hardware constraint? (*ii*) How does active compute reallocation impact absolute token consumption and latency compared to those strategies?

5.1 Experimental Setup

Models & Benchmarks. We evaluate across three open-weight large reasoning models of varying architectures and scales: Qwen3-4B-Thinking-2507 (Team, 2025), DeepSeek-R1-0528-Qwen3-8B (Team, 2025), and Phi-4-reasoning-plus-14B (Abdin et al., 2025). This diversity allows us to assess whether score-guided search generalizes across model families. We measure accuracy on four competition-grade mathematical reasoning benchmarks: AIME 2025, AIME 2026 (AoPS, 2025), HMMT 2024, and HMMT 2025 (HMMT, 2024; 2025). We additionally evaluate on GPQA-Diamond (Rein et al., 2023) to test generalization to graduate-level scientific reasoning resistant to pattern matching.

Baselines & Variants. We compare against five inference-scaling baselines: unweighted **Self-Consistency (SC)** (Wang et al., 2023), **Slim-SC** (Hong et al., 2025) (which aggregates via cosine-similarity deduplication at a 0.95 threshold), **DeepConf** (Fu et al., 2025) (which early-stops traces falling below an offline-calibrated top-10% confidence threshold with 16 traces for initialization pool), and **STEP** (Liang et al., 2026) which prunes the lowest-scoring trace triggered by GPU memory constraint. To isolate our algorithmic contributions, Gambit in our main evaluation uses the exact same off-the-shelf 2-layer MLP scorer from STEP. Moreover, we trained our own history-aware scorer in Section A.1 and demonstrate the generalization of Gambit in Section A.2.

Implementation Details. All methods are implemented atop vLLM (Kwon et al., 2023) and evaluated on a single 275 GB NVIDIA B300 GPU. Each problem is allocated a strict budget of $N = 256$ complete traces. For Gambit, tournament hyperparameters are held constant across all benchmarks and models: capacity C=256, swap size K=16, check interval Δ=200 tokens, warmup threshold w=12,000 tokens, and hard-floor δ=0.1. Final answers are aggregated via score-weighted majority vote, utilizing a position-weighted penalty that favors confidence in later generation steps. We have a detailed ablation study in Section A.4 to show the robustness of Gambit on hyperparameters.

5.2 Main Evaluation

Table 1 reports the end-to-end task accuracy and token consumption across all evaluated benchmarks. Because Gambit utilizes the identical baseline MLP scorer as STEP, any variance in accuracy or efficiency is strictly attributable to the underlying search topology.

Beam search strictly dominates baselines with substantial token efficiency. Replacing threshold-based pruning with our active beam search yields universal accuracy gains

Table 1: End-to-end task accuracy and token consumption per question ($\times 10^6$) at $N = 256$. Δ: relative token reduction (%) vs. SC@256. **Bold**: best accuracy / lowest tokens per dataset.

Method	AIME-25		AIME-26		HMMT-24		HMMT-25		GPQA	
	Tok. (Δ)	Acc	Tok. (Δ)	Acc	Tok. (Δ)	Acc	Tok. (Δ)	Acc	Tok. (Δ)	Acc
Qwen3-4B-Thinking										
SC@256	6.02	86.7	5.82	86.7	7.62	50.8	6.86	65.0	2.28	68.2
Slim-SC	3.93 (-34.7)	86.7	3.83 (-34.2)	85.0	4.01 (-47.4)	51.7	3.72 (-45.8)	65.8	1.66 (-27.2)	65.6
DeepConf	3.27 (-45.7)	**90.0**	**3.19 (-45.2)**	86.7	4.28 (-43.8)	58.3	4.15 (-39.5)	66.7	**1.52 (-33.3)**	67.6
STEP	3.96 (-34.2)	86.7	3.51 (-39.7)	87.5	4.02 (-47.2)	61.7	3.88 (-43.4)	66.7	2.09 (-8.3)	66.9
Gambit (Ours)	**3.07 (-49.0)**	**90.0**	3.45 (-40.7)	**88.3**	**3.00 (-60.6)**	**65.0**	**3.49 (-49.1)**	**67.5**	1.83 (-19.7)	**70.2**
DeepSeek-R1-8B										
SC@256	6.76	83.3	6.85	83.3	8.47	55.8	7.65	70.0	2.92	67.1
Slim-SC	5.87 (-13.2)	83.3	6.08 (-11.2)	83.3	7.32 (-13.6)	55.8	6.93 (-9.4)	70.7	2.26 (-22.6)	67.1
DeepConf	3.75 (-44.5)	81.7	3.67 (-46.4)	84.2	4.34 (-48.8)	60.0	3.97 (-48.1)	71.7	**1.68 (-42.5)**	**68.7**
STEP	**3.71 (-45.1)**	83.3	**3.66 (-46.6)**	83.3	4.15 (-51.0)	63.3	4.08 (-46.7)	75.8	2.44 (-16.4)	68.2
Gambit (Ours)	4.21 (-37.7)	**85.8**	4.09 (-40.3)	**85.0**	**3.05 (-64.0)**	**65.6**	**3.81 (-50.2)**	**75.8**	2.21 (-25.3)	68.2
Microsoft-Phi										
SC@256	4.24	86.7	4.20	88.5	5.43	56.7	5.56	73.3	3.05	76.3
Slim-SC	3.43 (-19.1)	85.0	3.50 (-16.7)	86.7	4.72 (-13.1)	55.7	4.48 (-19.4)	74.2	2.24 (-26.6)	72.3
DeepConf	2.56 (-39.6)	85.8	2.75 (-34.5)	86.7	2.86 (-47.3)	57.5	3.02 (-45.7)	74.2	1.61 (-47.2)	74.8
STEP	2.39 (-43.6)	87.5	2.44 (-41.9)	**90.0**	2.75 (-49.4)	56.7	2.78 (-50.0)	75.0	2.10 (-31.1)	76.7
Gambit (Ours)	**1.76 (-58.5)**	**88.3**	**1.86 (-55.7)**	**90.0**	**1.72 (-68.3)**	**58.3**	**1.75 (-68.5)**	**75.8**	**1.59 (-47.9)**	**77.1**

across all benchmarks and model architectures from 4B to 14B. For the 4B model, Gambit matches the highly calibrated DeepConf baseline on AIME-25 (90.0%) while significantly outperforming it on HMMT-24 (+6.7%) and GPQA (+2.6%). When compared directly to the pruning-only STEP baseline, Gambit provides consistent lifts: +3.3% on AIME-25 and HMMT-24 for the 4B model, +2.5% on AIME-25 for the 8B model, and +1.6% on HMMT-24 for the Phi-4 model. These improvements validate our core hypothesis: by continuously replenishing the trace pool via branching rather than allowing concurrency to passively decline, the tournament dynamically shifts the ensemble distribution toward correct regions of the search space, successfully breaking the majority-vote accuracy ceiling.

5.3 Efficiency: Token Consumption, Latency, and Trace Throughput

We analyze system efficiency by quantifying absolute token consumption and hardware utilization over time with implementation of our system on vLLM (Kwon et al., 2023).

Combined with Table 1, Gambit establishes a new tradeoff frontier for efficiency (Figure 5). Across all evaluated architectures, our method achieves strict token reductions relative to traditional parallel sampling. These reductions are particularly pronounced on the Phi-4 architecture, where Gambit decreases the total token budget by 68.5% on HMMT-25 (1.75M tokens versus SC's 5.56M). Similarly, on the Qwen3-4B model, we observe a 60.6% token reduction on HMMT-24. This efficiency is a direct architectural consequence of thought-level branching: by forcing child traces to inherit the physical KV-cache of high-scoring parents, the marginal computational cost of exploring divergent reasoning pathways is vastly diminished. In terms of latency, Gambit achieves more than **2× faster than parallel sampling** and **1.5× speedup other subtractive baselines**, while remaining competitive with STEP in raw latency. Although STEP can appear slightly faster (see Figure 5), it suffers from hardware underutilization: terminated traces are not replaced, causing the

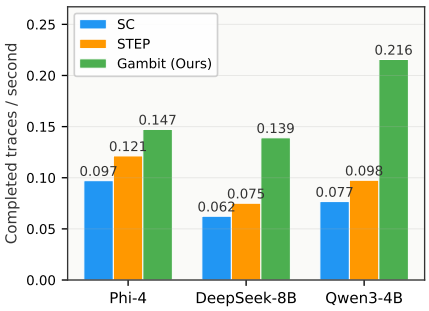

Figure 6: Trace throughput on AIME-26. Gambit effectively doubles the completion rate.

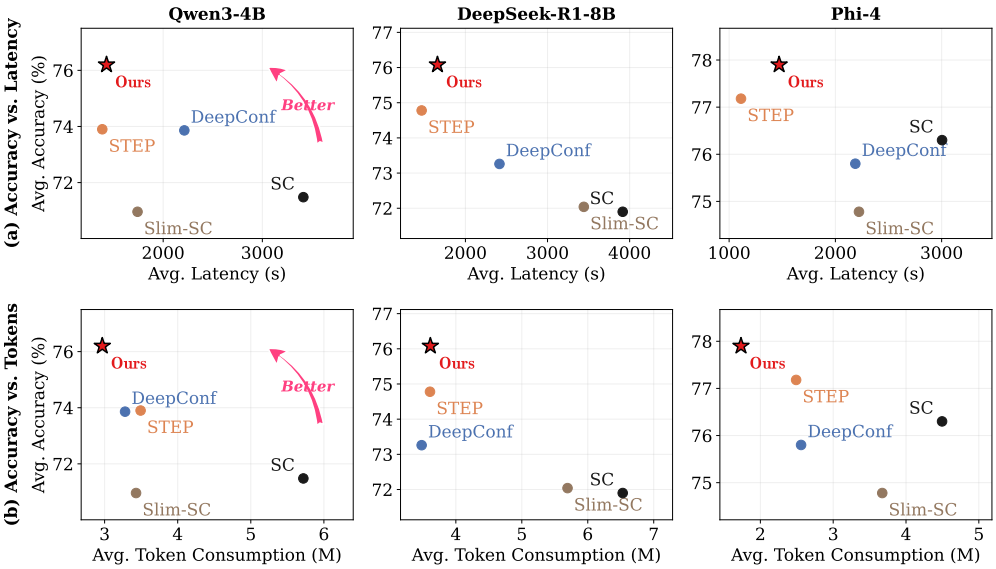

Figure 5: Efficiency vs. Accuracy trade-offs over all benchmarks. (a) Average Latency vs. Accuracy. (b) Average Token Consumption vs. Accuracy. Gambit (Ours) dominates the efficiency frontier, achieving the highest accuracy with the lowest overall token footprint and highly competitive wall-clock latency.

active batch to decay and shrinking the effective vote size. In contrast, Gambit continuously resupplies the trace pool via thought-level branching, maintaining full hardware utilization and yielding over **2**$\times$ **higher trace throughput** (e.g., 0.216 vs. 0.098 on Qwen3-4B; Figure 6).

6 Analysis

6.1 Generation Distribution and Decoding Dynamics

The empirical results in Section 5.3 reveal a non-trivial systems dynamic: Gambit achieves substantial reductions in total token consumption (up to 68.5%), yet its wall-clock latency remains comparable to aggressive pruning methods such as STEP. To explain this apparent discrepancy, we analyze the distribution of generated tokens and trace lengths on AIME-2026.

Prefix Reuse and Unique Token Minimization. The reduction in total token consumption is driven by prefix reuse through branching. Figure 7 shows the distribution of *unique* tokens generated per completed trace, defined as the total generated tokens excluding inherited prefix tokens. Because Gambit branches from shared high-quality prefixes in the KV-cache, it avoids recomputing early reasoning steps. For example, on Phi-4, Gambit generates a median of 5.2K unique tokens per trace, compared to 14.5K for parallel sampling (SC) and 7.7K for STEP. This demonstrates that Gambit explores alternative reasoning paths with substantially lower marginal cost, tightly controlling the total token budget.

Distribution Shift in Total Sequence Length. We next examine the *total* sequence length of completed traces (Figure 8). At first glance, Gambit appears to produce significantly longer traces (e.g., median 35.8K vs. 16.5K for STEP on DeepSeek-R1-8B). However, this does *not* indicate that individual reasoning trajectories are intrinsically longer under Gambit. Instead, this effect arises from a shift in the *distribution* of surviving traces. Subtractive methods such as STEP terminate low-scoring traces without replacement, truncating a large fraction of trajectories early and shifting the length distribution toward shorter sequences. In contrast, Gambit both prunes and *replenishes* the pool by branching from high-quality prefixes.

Crucially, branching does not make individual traces longer; rather, it increases the *frequency* of traces that continue to generate tokens. As a result, the overall distribution shifts to the right, even though the per-trace generation dynamics remain unchanged.

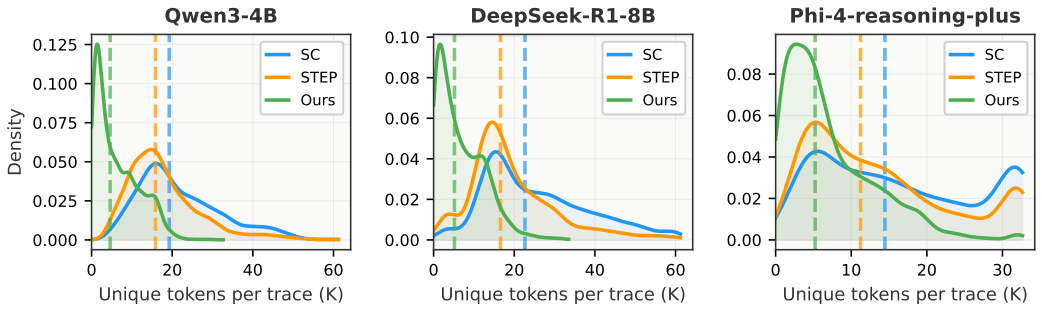

Figure 7: Distribution of *unique* tokens generated per completed trace on AIME-2026. By branching from shared high-quality KV-cache prefixes, Gambit drastically reduces the median number of new tokens required to explore alternative reasoning pathways compared to SC and STEP.

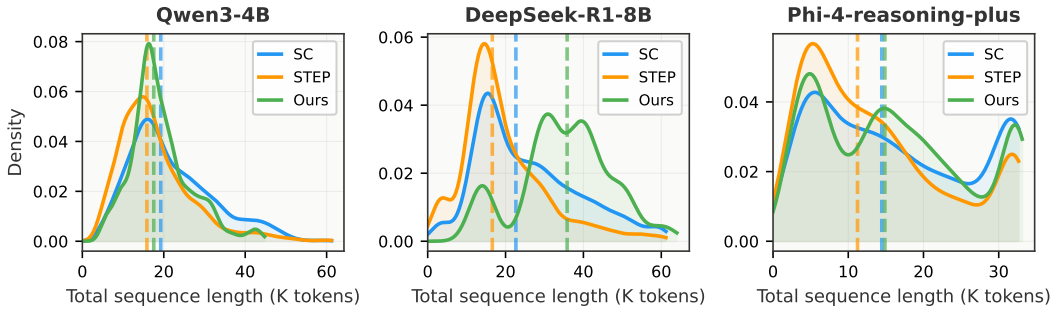

Figure 8: Distribution of *total* sequence length (including inherited prefixes) for completed traces on AIME-2026. Gambit sustains significantly deeper, rigorous reasoning chains than subtractive baselines like STEP, which accounts for the serial decoding latency bottleneck despite massive overall token savings.

Implications for Decoding Latency. This distributional shift explains why token savings do not directly translate into proportional latency reductions. Although Gambit minimizes redundant token generation via prefix sharing, it allocates more compute to continuing high-quality trajectories, resulting in a higher fraction of long-running traces.

Overall, Gambit achieves a favorable trade-off: it dramatically reduces total token consumption while maintaining competitive wall-clock latency by reallocating compute toward promising trajectories rather than uniformly truncating them.

7 Conclusion

Gambit is a thought-level beam search framework that formulates test-time reasoning as a constrained compute-allocation problem over partial trajectories. Rather than scaling inference by increasing the number of independent samples, our approach dynamically reallocates compute toward promising intermediate states while maintaining strict hardware constraints. Our results demonstrate that this shift is both algorithmically and systemically necessary. By actively branching from high-quality prefixes and enforcing a zero-sum allocation policy, it simultaneously improves accuracy and efficiency—achieving consistent gains across benchmarks while reducing token consumption by up to 68.5% and maintaining high hardware utilization. Empirically, this leads to strict dominance over both parallel sampling and pruning-based approaches.

Acknowledgments

We are grateful to the compute support at Dao AI Lab and insightful discussions with Wentao Guo, Rui Pan, Yizheng Zhang, Alan Zhu, and Han Guo.

References

Marah Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Harkirat Behl, Lingjiao Chen, Gustavo de Rosa, Suriya Gunasekar, Mojan Javaheripi, Neel Joshi, Piero Kauffmann, Yash Lara, Caio César Teodoro Mendes, Arindam Mitra, Besmira Nushi, Dimitris Papailiopoulos, Olli Saarikivi, Shital Shah, Vaishnavi Shrivastava, Vibhav Vineet, Yue Wu, Safoora Yousefi, and Guoqing Zheng. Phi-4-reasoning technical report, 2025. URL https://arxiv.org/abs/2504.21318.

Anthropic. Introducing claude 4, 2025. URL https://www.anthropic.com/news/claude-4. Announcement of Claude Opus 4 and Claude Sonnet 4 models.

AoPS. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions, 2025. Accessed: 2025-07-14.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality, 2023. URL https://arxiv.org/abs/2305.18654.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence, 2025. URL https://arxiv.org/abs/2508.15260.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023. URL https://arxiv.org/abs/2305.14992.

HMMT. Archive of february 2024. HMMT Official Archive, 2024. Accessed: 2026.

HMMT. Archive of february 2025. HMMT Official Archive, 2025. Accessed: 2026.

Colin Hong, Xu Guo, Anand Chaanan Singh, Esha Choukse, and Dmitrii Ustiugov. Slim-SC: Thought pruning for efficient scaling with self-consistency. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 34500–34517, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1750. URL https://aclanthology.org/2025.emnlp-main.1750/.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know, 2022. URL https://arxiv.org/abs/2207.05221.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Jinu Lee and Julia Hockenmaier. Evaluating step-by-step reasoning traces: A survey, 2025. URL https://arxiv.org/abs/2502.12289.

Zhixiang Liang, Beichen Huang, Zheng Wang, and Minjia Zhang. Hidden states as early signals: Step-level trace evaluation and pruning for efficient test-time scaling, 2026. URL https://arxiv.org/abs/2601.09093.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words, 2022. URL https://arxiv.org/abs/2205.14334.

Bruce T. Lowerre. *The HARPY Speech Recognition System*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, March 1976. Department of Computer Science. DTIC Accession Number: ADA035146.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

OpenAI. Introducing gpt-5, aug 2025a. URL https://openai.com/index/introducing-gpt-5/. Announcement of GPT-5, OpenAI's smartest AI system with built-in reasoning.

OpenAI. Introducing gpt-oss, 2025b. URL https://openai.com/index/introducing-gpt-oss/.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. Specreason: Fast and accurate inference-time compute via speculative reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=wCbOKbZ7kf.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Peiyang Song, Pengrui Han, and Noah Goodman. Large language model reasoning failures, 2026. URL https://arxiv.org/abs/2602.06176.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014. URL https://arxiv.org/abs/1409.3215.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish

Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024. URL https://arxiv.org/abs/2312.08935.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P. Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning, 2024. URL https://arxiv.org/abs/2405.00451.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

A Appendix

A.1 Evaluating Search Generalization: A History-Aware Sequence Scorer

To evaluate how different scoring signals affect search dynamics, we have trained our history-aware sequence model (Section A.1). The proposed search algorithm is agnostic to the specific scoring function used to evaluate trajectories. However, to study how the quality of the scoring signal affects search behavior (Section A.2), we design an alternative, history-aware *sequence scorer* f_θ to off-the-shelf scorers used in main evaluation. Unlike a Markovian MLP, this sequence scorer maps a prefix of hidden-state vectors to a scalar quality score in $[0,1]$ by attending to the full reasoning trajectory, providing a structurally distinct guidance signal for our ablations.

Data Collection & Step-Level States. We generate n independent traces per problem using standard parallel sampling. Each trace is delimited by double-newline boundaries (`\n\n`) within the model's `<think>` block (Pan et al., 2025), with a binary correctness label $y \in \{0,1\}$ assigned to the final extracted answer. During a single forward pass, we extract the last-layer hidden state $\mathbf{h}_t \in \mathbb{R}^D$ at the token immediately preceding each boundary (Liang et al., 2026). This yields a per-trace step sequence $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T)$.

Scorer Architecture. The sequence scorer f_θ is a compact Transformer:

$$\tilde{\mathbf{h}}_t = \mathrm{GELU}(\mathbf{W}_{\mathrm{in}} \cdot \mathrm{LN}(\mathbf{h}_t)), \tag{4}$$

$$\mathbf{z}_1, \ldots, \mathbf{z}_T = \mathrm{SequenceTransformer}(\tilde{\mathbf{h}}_1, \ldots, \tilde{\mathbf{h}}_T), \tag{5}$$

$$\hat{y}_t = \sigma\left(\mathbf{w}_{\mathrm{out}}^{\top} \cdot \mathrm{LN}(\mathbf{z}_t)\right). \tag{6}$$

Each layer utilizes multi-head causal self-attention with Rotary Position Embeddings (RoPE) (Su et al., 2023) and SwiGLU feed-forward networks (Shazeer, 2020). The causal mask is essential: position t attends to all prior steps $1, \ldots, t$, ensuring $\hat{y}_t$ is conditioned on the full reasoning history up to step t. This allows the sequence scorer to detect globally inferior steps that appear locally plausible—providing a high-resolution signal to test Gambit's amplification dynamics.

Training Objective. Because the sequence scorer is queried dynamically during inference via the logit at the final observed step, we train exclusively with a **last-step** binary cross-entropy loss:

$$\mathcal{L}(\theta) = -\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{h}_{1:T}, y) \in \mathcal{D}} [y \log \hat{y}_T + (1-y) \log(1-\hat{y}_T)], \tag{7}$$

where T is the sequence length and $\hat{y}_T$ is the output at the last valid position. Crucially, only the final position contributes a gradient signal, forcing the attention mechanism to actively credit-assign the full trajectory context rather than relying on local features at any single step.

A.2 Generalization Across Scoring Architectures

Table 2: Impact of scorer architecture on end-to-end downstream accuracy. Gambit consistently outperforms the pruning-only baseline (STEP) across different reward models and base LLMs. By actively reallocating compute to promising prefixes, Gambit effectively unlocks the potential of the guidance signal, whereas purely subtractive methods remain fundamentally bottlenecked by their inability to generate new traces.

Base Model	System	AIME-25	AIME-26	HMMT-24	HMMT-25	GPQA
DeepSeek-R1-0528	STEP + SeqScorer	83.3	84.2	61.7	75.8	66.7
	Gambit + SeqScorer	**88.5** (+5.2)	**86.7** (+2.5)	**69.4** (+7.7)	**76.7** (+0.9)	**67.7** (+1.0)
Qwen3-4B	STEP + SeqScorer	86.7	86.7	60.0	65.0	67.1
	Gambit + SeqScorer	**90.0** (+3.3)	**87.5** (+0.8)	**61.7** (+1.7)	**65.8** (+0.8)	**68.2** (+1.1)

A fundamental theoretical difference between subtractive pruning (STEP) and active compute reallocation (Gambit) is how they utilize the underlying guidance signal. In a pruning-only

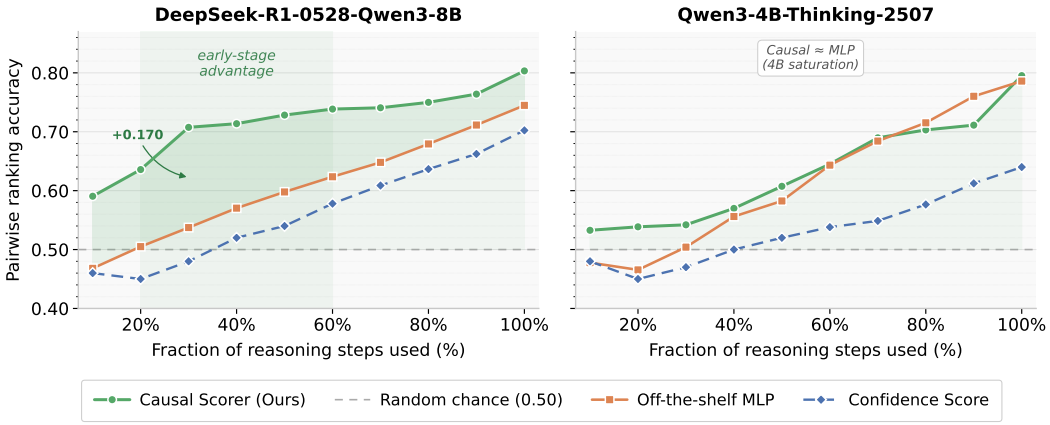

Figure 9: **Pairwise ranking accuracy vs. fraction of reasoning steps used (k%).** Higher accuracy indicates better separation of correct and incorrect traces given only a partial prefix.

system, the scoring mechanism acts as a negative filter: it can terminate unpromising traces, but it cannot actively construct or discover correct ones. In contrast, Gambit treats the scoring mechanism as a generative compass, using it to systematically explore promising regions of the solution space. By actively branching from high-scoring prefixes, Gambit structurally shifts the sampling distribution toward correct answers.

To verify that Gambit's algorithmic superiority generalizes beyond the single, off-the-shelf MLP scorer used in our main evaluation, we evaluate both inference algorithms using our custom, history-aware sequence scorer (Section A.1). This provides a distinct guidance signal derived from the full latent trajectory rather than isolated steps.

As shown in Table 2, replacing the baseline MLP with the trajectory-aware scorer reveals a stark contrast between the two search topologies:

Active search robustly dominates subtractive pruning. Across all base models and benchmarks, Gambit robustly and significantly outperforms the STEP baseline. On the 8B model, Gambit achieves substantial end-to-end accuracy gains over STEP, including +7.7% on HMMT-24 and +5.2% on AIME-25. Similarly, on the 4B model, Gambit maintains consistent superiority, delivering a +3.3% lift on AIME-25 and +1.7% on HMMT-24. This confirms that our tournament algorithm's advantages are fundamentally robust and architecture-agnostic; its compute reallocation mechanism consistently yields higher downstream task accuracy regardless of the underlying reward model.

Subtractive systems bottleneck guidance signals. The magnitude of these performance gaps (e.g., +7.7% on HMMT-24) highlights a critical limitation of pruning-only frameworks. Even when provided with alternative guidance signals, STEP is fundamentally bottlenecked by its inability to actively replenish the trace pool. If the base model fails to generate a sufficient density of correct traces organically, a filter merely results in a smaller, equally biased pool of survivors. Conversely, Gambit fully operationalizes the provided quality signal. By actively branching from prefixes identified as highly promising, the algorithm physically reshapes the output distribution, translating the guidance signal into robust, state-of-the-art benchmark wins without being hypersensitive to the specific choice of scorer.

A.3 Latency Decomposition and System Overhead

A common vulnerability in complex inference-time search algorithms is the introduction of prohibitive CPU-bound bookkeeping or memory-management bottlenecks. To validate that our thought-level beam search imposes minimal structural overhead, we decompose the end-to-end wall-clock latency of Gambit.

Table 3: Latency decomposition for Gambit with Qwen3-4B on AIME-26 with a batch size of $N = 256$. Algorithmic overhead accounts for less than 1% of total execution time, confirming that the beam search mechanism operates efficiently without introducing system bottlenecks.

Execution Component	Time (s)	% Wall Clock
GPU Compute (Forward Pass & Sampling)	37,851.97	99.03%
Beam-search Overhead (Scoring & Tree Mgmt)	370.76	0.97%
Total Wall Clock	**38,222.73**	**100.00%**

As detailed in Table 3, the operational overhead introduced by the tournament mechanism is strictly negligible. The core autoregressive forward passes and GPU sampling routines completely dominate execution, consuming 99.03% of the total wall-clock time.

The remaining fraction of execution time (0.97%) encompasses all algorithmic interventions required to dynamically manage the search space. The majority of this overhead ($\sim$0.60% of total time) is attributed to *guidance synchronization*—the inter-process communication (RPC) required to transmit latent hidden states to the reward model worker and retrieve trajectory scores. This functions primarily as a brief synchronization stall rather than a computational bottleneck. *Tree topology management* accounts for approximately 0.25%, encapsulating the CPU-bound bookkeeping necessary to sort active traces and dispatch prune-and-branch commands. Finally, *continuous scheduling and garbage collection* ($\sim$0.12%) manages the per-step priority queues and hard-floor threshold scans, ensuring that degenerated traces are opportunistically evicted without interrupting the continuous batching cycle.

Ultimately, this profiling confirms that highly dynamic, thought-level beam search topologies can be executed with virtually no systems penalty. By effectively mitigating CPU and memory-management overheads, Gambit ensures that the efficiency gains achieved through token reductions translate directly into real-world latency improvements.

A.4 Hyperparameter Ablations

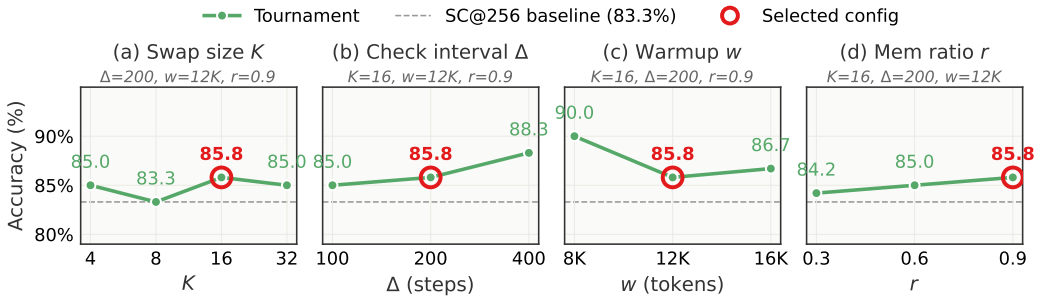

Figure 10: Hyperparameter sensitivity of Gambit on AIME-25 with DeepSeek-R1-0528-8B (batch size 256, C=256). Each panel varies one parameter while fixing the others to the selected configuration (K=16,Δ=200,w=12K,r=0.9). The dashed line denotes the SC@256 baseline (83.3%). The selected configuration (red circle) lies within a broad high-performing regime, consistently outperforming the baseline.

We analyze the sensitivity of Gambit to key design parameters, including swap size $K \in \{4,8,16,32\}$, beam search interval $\Delta \in \{100,200,400\}$, warmup threshold $w \in \{8K,12K,16K\}$, and GPU memory usage ratio δ (Figure 10). Across all dimensions, performance varies smoothly without sharp degradation, indicating that Gambit is robust to hyperparameter choices. The selected configuration (K=16, Δ=200, w=12K, r=0.9) lies in a broad high-performing region and consistently outperforms the SC@256 baseline.

Each parameter reflects a balance between accuracy and efficiency. Small swap sizes limit the ability to refresh low-quality traces, while overly large values can disrupt stable decoding; K=16 provides a consistent improvement. Frequent updates (Δ=100) act on insufficiently

developed prefixes, while infrequent updates (Δ=400) delay corrective selection; Δ=200 yields a stable cadence. Similarly, branching too early can harm reasoning quality, whereas excessive delay wastes compute, and a moderate warmup (w=12K) allows traces to establish a reliable foundation before selection. Finally, increasing the GPU memory usage ratio r improves performance, highlighting the importance of effective KV-cache utilization for parallel decoding. Overall, Gambit operates in a stable regime where moderate hyperparameters achieve strong performance while maintaining efficient and well-conditioned decoding dynamics.

A.5 Runtime Example of Gambit

To make the runtime behavior of Gambit concrete, we open up a single tournament tree and walk through four representative traces it produced on AIME 2025 Q12. The configuration is the standard one used throughout Section 5: capacity C=256, swap size K=16, check interval Δ=200, warmup w=12K, hard floor δ=0.1, off-the-shelf MLP scorer f_θ from STEP, and model DeepSeek-R1-0528-Qwen3-8B. The problem asks for the expected number of regions when two perpendicular diameters and 25 random chords (with endpoints in different quadrants) are drawn through a disk; the ground-truth answer is 204.

Tree at a glance. The run produced a tournament tree with 1,595 nodes:

- 256 root traces seeded at the start of decoding;
- 1,339 branches spawned over the run by zero-sum prune-and-branch operations (Algorithm 1);
- 1,083 traces eventually *pruned*; their partial outputs are not preserved in the tree export;
- 256 traces eventually *completed*, contributing answers to the score-weighted majority vote;
- 42 of the 256 completed traces returned the correct answer 204; the score-weighted vote selects 204 with $\sum \bar{s}_\tau = 24.02$ versus 21.81 for the runner-up.

Baseline contrast: parallel sampling at twice the trace budget cannot solve this question. For comparison, on the same problem, with the same model, plain self-consistency at n=512 (no scoring, no pruning, no branching; δ = none, every trace runs to completion in parallel) consumes 21 million tokens over 15K seconds of wall-clock time — roughly 13× the new tokens and 3.4× the wall clock of the Gambit run above — and nonetheless fails to recover the correct answer. Of its 512 completed traces, only 34 (6.6%) return 204, while the incorrect answer $\frac{487}{3}$ wins the unweighted plurality with 90 votes (17.6%); $\frac{637}{3}$ comes second with 59 (11.5%) and 204 comes only third. With the same setup, allocated by Gambit via thought-level beam search, Gambit both delivers the correct answer and uses an order of magnitude fewer generated tokens.

The subtree we follow. We pick one root prefix τ_0 that survived past the warmup window. Its surviving depth-1 descendant, denoted τ_p, became the *highest-scoring* depth-1 trace in the entire run with running score $\bar{s}_{\tau_p}$=0.610. Each tournament round in which τ_p ranks in the top-K executes a single *branching event*: τ_p's prefix is shared into two trajectories, one of which is τ_p continuing decoding, and the other is a fresh depth-2 child that begins its own continuation from the same prefix via prefix-cache reuse. We follow three such branching events for τ_p, each producing one depth-2 child of a different eventual fate. Schematically:

```
τ0 (root, depth 0)
   |
   + τp (depth 1; completed at decoding step ~34,000;
         s̄=0.610, ans = 203)
      |
      |--- branching event at decoding step ~26,500
      |     + τC (depth 2; completed; s̄=0.566,
      |           ans = 559/3)  --- alternate-explore child
      |
      |--- branching event at decoding step ~26,000
```

```
|     + τ_D (depth 2; pruned in a later round;
|           no preserved text)  --- pruned child
|
|--- branching event at decoding step ∼30,000
     + τ_B (depth 2; completed; s̄ = 0.632,
           ans = 204)  --- correct child
```

Each branching event is a single 1-to-1 split of τ_p's prefix: τ_p continues, one new depth-2 child begins. The three events above are the ones whose products we reproduce below; τ_p takes part in additional branching events at later tournament rounds, all following the same 1-to-1 pattern. We now reproduce the four marked traces τ_p, τ_B, τ_C, τ_D verbatim, each with metadata followed by the model's relevant output. "`[. . . ]`" marks elisions for length.

Trace τ_p — the high-scoring depth-1 founder (completed, answer 203, $\bar{s} = 0.610$)

Metadata. Branching depth 1; created at decoding step 13,934 as a branch from root τ_0; generated 20,669 new tokens after inheriting 14,004 tokens from τ_0 via prefix-cache reuse; participated in several subsequent branching events as the high-scoring parent τ_p (three of which we follow below); final completion at decoding step $\sim$34,000 with extracted answer 203.

Excerpt — opening.

```
<think>
Alex divides a disk into four quadrants with two perpendicular
diameters.  So, that's like the x and y axes, dividing the disk
into four equal parts.  Then he draws 25 more lines by connecting
two random points on the perimeter in different quadrants.  ...
And we need to find the expected number of regions that these 27
lines divide the disk into.
```

`[. . .  the trace sets up linearity of expectation, derives the chord-length distribution and the chord-vs-quadrant configuration, integrates to obtain the chord-chord intersection probability, and arrives at` $E[N_i] = 8$ `new regions per chord. . . . ]`

Excerpt — final calculation (the slip).

```
For each of the 25 random chords, 8.

So total E[R] = 1 + 2 + 25 * 8 = 3 + 200 = 203.

[...]

Now, expected regions 203.

Now, I think that's the answer.
</think>
The total expected number of regions is the sum of the new regions
created by all lines: 3 from the diameters and 25 x 8 from the
random chords, giving 3 + 200 = 203.

\boxed{203}
```

Note the slip in the very first line of the excerpt: τ_p writes "$1+2+25\cdot 8$" but evaluates to $3+200=203$, treating the leading 1 and 2 as the diameter contributions and forgetting that the disk itself contributes one initial region. The final boxed answer is therefore off by one.

Trace τ_B — depth-2 child of τ_p (completed, answer 204, $\bar{s} = 0.632$)

Metadata. Branching depth 2; spawned from τ_p at decoding step $\sim$30,000 during a tournament round in which τ_p ranked first; inherited 31,404 tokens from τ_p via prefix-cache reuse; generated 3,649 new tokens of its own; running score $\bar{s}_{\tau_B} = 0.632$, the highest among all 256

completed traces; extracted answer 204. Because the score is the largest, τ_B contributes the single largest mass to the score-weighted vote.

Last ≈ 100 *characters of inherited prefix (shared with* τ_p*).*

```
... in the diameters, we computed their contribution separately.
```

Excerpt — start of τ_B*'s newly generated continuation.*

```
.

Now, another way to think: total expected regions is
E[R] = E[1 + sum_{all lines} new regions created]

But since the lines include the diameters and the random chords,
and we have computed for each.

[...]

Now, so total E[R] = E[new regions by diameters]
               + sum_{i=1}^{25} E[new regions by C_i]
               = 3 + 25 * 8 = 203.

Now, but is there any overlap or something? No, because each
new region is created by exactly one line.

Now, so it should be correct.

Now, but let me confirm with a smaller case.
```

Excerpt — τ_B *catches the parent's slip.*

```
Suppose we have only the two diameters and no random chords.
Then E[R] = 4, but according to formula, diameters contribute 3,
no random chords, so 3, but should be 4, inconsistency.

Mistake here.

When we add the diameters, they create 3 new regions, but
initially there was 1 region, so total regions after diameters
is 1 + 3 = 4.

[...]

Ah, I see, the issue is that the initial region is not accounted
for in the sum.

In the standard formula, the total number of regions is equal to
1 (the initial whole space) plus the sum over all lines of the
number of new regions created by that line.

[...]

In our calculation for the random chords, we are only summing the
new regions created by each line, but not including the initial
region.

In the expression for E[R], it should be
     E[R] = E[1 + sum_{all lines} new regions created]
```

Excerpt — final answer.

```
</think>
The disk is initially divided into four quadrants by two
```

```
perpendicular diameters, which create 4 regions.  ...

The two diameters create a total of 3 new regions ...

Each random chord, when added, creates a random number of new
regions, but the expected number of new regions created by each
random chord is 8.  ...

With 25 random chords, the sum of the expected new regions created
by all random chords is 25 x 8 = 200.

The sum of the expected new regions created by all lines is
3 + 200 = 203.

The total expected number of regions is the initial region plus
this sum: 1 + 203 = 204.

\boxed{204}
```

τ_B's contribution is small in token count (only 3,649 new tokens, almost free thanks to prefix-cache reuse) but decisive in content: the child explicitly performs a smaller-case sanity check, identifies the missing $+1$ from the disk, and writes the corrected equation $1 + 203 = 204$. This is the prototypical case of *branching repairs the parent*. No subtractive method (STEP, DeepConf, Slim-SC) can produce this kind of repair — they can only kill the parent for being below threshold or produce confidently wrong answers.

Conclusion

The four traces above span the full state space a trace can occupy in a Gambit run. τ_p is a long-lived high-scoring trace that itself completes *and* serves as a branching parent. τ_B alone contributes the largest single mass to that vote.

Table 4: Systems-level efficiency at n=256 on a single B300-275 GB. Token consumption is reported in thousands (K); latency in wall-clock seconds per question. **Bold** indicates the best result per column within each model–metric block.

		AIME-25	AIME-26	HMMT-24	HMMT-25	GPQA
Qwen3-4B-Thinking-2507						
Tokens (K)	SC	6,023	5,817	7,622	6,857	2,276
	Slim-SC	3,930	3,829	4,010	3,723	1,659
	DeepConf	3,266	**3,192**	4,278	4,148	**1,516**
	STEP	3,960	3,506	4,023	3,884	2,089
	Gambit (Ours)	**3,071**	3,449	**3,002**	**3,493**	1,826
Latency (s)	SC	3,465	3,338	5,358	4,313	593
	Slim-SC	1,905	1,841	2,412	1,995	555
	DeepConf	2,095	2,188	3,231	2,948	605
	STEP	1,390	1,232	2,016	**1,507**	786
	Gambit (Ours)	**1,350**	**1,177**	**1,912**	2,179	**527**
DeepSeek-R1-0528-Qwen3-8B						
Tokens (K)	SC	6,764	6,849	8,465	7,652	2,919
	Slim-SC	5,874	6,080	7,323	6,933	2,256
	DeepConf	3,753	3,666	4,337	3,972	**1,679**
	STEP	**3,711**	**3,661**	4,147	4,077	2,444
	Gambit (Ours)	4,211	4,089	**3,053**	**3,806**	2,211
Latency (s)	SC	4,078	4,107	5,640	4,873	880
	Slim-SC	3,462	3,564	5,071	4,342	771
	DeepConf	2,692	2,582	3,256	2,800	744
	STEP	**1,519**	**1,469**	**1,902**	**1,715**	730
	Gambit (Ours)	1,571	1,842	2,014	2,169	**702**
Phi-4						
Tokens (K)	SC	4,243	4,198	5,426	5,559	3,050
	Slim-SC	3,426	3,503	4,716	4,482	2,242
	DeepConf	2,557	2,749	2,860	3,022	1,608
	STEP	2,390	2,435	2,754	2,780	2,097
	Gambit (Ours)	**1,756**	**1,855**	**1,717**	**1,752**	**1,590**
Latency (s)	SC	2,760	2,636	3,848	4,046	1,730
	Slim-SC	2,123	2,166	3,069	2,704	1,048
	DeepConf	1,864	1,749	2,437	2,832	2,056
	STEP	**1,155**	**1,139**	**1,197**	**1,222**	**846**
	Gambit (Ours)	1,480	1,538	1,623	1,589	1,123

A.6 End-to-end Latency and Token Consumption

The following Table 4 records the data of wall-clock time and total token consumption for each model and benchmark.