OPENDEEPTHINK: PARALLEL REASONING VIA BRADLEY–TERRY AGGREGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time compute scaling is a primary axis for improving LLM reasoning. Existing methods primarily scale depth by extending a single reasoning trace. Scaling breadth by sampling multiple candidates in parallel is straightforward, but introduces a selection bottleneck: choosing the best candidate without a ground-truth verifier, since pointwise LLM judging is noisy and biased. To address this, we introduce OpenDeepThink, a population-based test-time compute framework that selects via pairwise Bradley–Terry comparison. Each generation, the LLM judges random pairs of candidates and aggregates votes via Bradley–Terry into a global ranking; top-ranked candidates are preserved and the top three quarters are mutated using the natural-language critiques produced during comparison; the bottom quarter is discarded. OpenDeepThink raises Gemini 3.1 Pro's effective Codeforces Elo by $+405$ points in eight sequential LLM-call rounds ($\sim$27 minutes wall-clock). The pipeline transfers across weaker and stronger models without retuning, and on the multi-domain HLE benchmark, gains appear concentrated in objectively verifiable domains and reverse in subjective ones. We release CF-73, a curated set of 73 expert-rated Codeforces problems with International Grandmaster annotation and 99% local-evaluation agreement against the official verdict.[1]

1 INTRODUCTION

Reasoning models such as o1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025) have established test-time compute scaling as a primary axis for improving LLM reasoning. The gains are sharp on mathematical reasoning, competitive programming, and agentic benchmarks. The dominant paradigm extends the model's chain of thought, either by encouraging longer traces (Jaech et al., 2024; Guo et al., 2025) or by searching over reasoning steps with a learned value function (Snell et al., 2024; Wu et al., 2024). Both approaches are inherently sequential: additional compute buys depth, not breadth, and a single early misstep derails the rest of the trace (Huang et al., 2023). Best-of-N sampling parallelizes naturally but shifts the bottleneck to selection. Picking the best candidate from a pool requires either ground-truth test cases (Cobbe et al., 2021), a trained reward model (Lightman et al., 2023; Wang et al., 2024), or an LLM judge whose pointwise scores are noisy and positively biased (Zheng et al., 2023; Liu et al., 2023). Self-refinement methods (Madaan et al., 2023; Shinn et al., 2023) iterate on a single trajectory and lack population-level selection pressure, tending to patch a fixed approach rather than revise it (Huang et al., 2023). To our knowledge, no existing method simultaneously parallelizes across candidates and combines selection with directed mutation without requiring domain-specific verification infrastructure.

Together, these three properties point to a population-based design rather than a single trajectory: a population enables head-to-head ranking without external verifiers, and the losers can be replaced via directed mutation rather than patched. OpenDeepThink realizes this design (Figure 1): the LLM that generates the candidates also judges them pairwise, Bradley–Terry aggregation (Bradley & Terry, 1952) turns the outcomes into a global ranking, and the natural-language critiques from comparison drive mutations that may abandon the current approach entirely. All calls within a round run in parallel, so the sequential depth is only eight LLM calls ($\sim$285 calls per problem).

[1] `https://anonymous.4open.science/r/CF-73-2CB1`

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

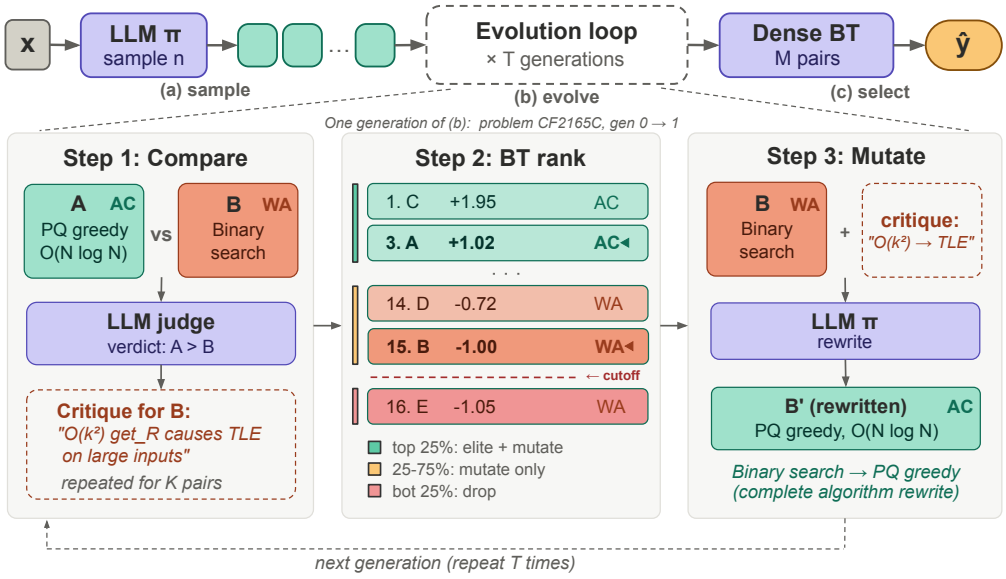


Figure 1: OpenDeepThink pipeline. (a) Parallel sampling of n candidates. (b) T generations of pairwise LLM judging, Bradley–Terry ranking, and feedback-driven mutation. (c) A final, denser Bradley–Terry round selects the submitted solution. AC/WA labels are shown for post-hoc analysis only.

The core mechanism is pairwise Bradley–Terry comparison. Pairwise framing matters because pointwise LLM judging is positively biased (Zheng et al., 2023; Liusie et al., 2024). On a controlled 500-pair diagnostic, pairwise reaches 86% accuracy versus 59% for pointwise.

We evaluate OpenDeepThink on two competition-level programming benchmarks totaling 192 problems and on 82 questions from the multi-domain HLE benchmark (Phan et al., 2025), using Gemini 3.1 Pro as both generator and judge. Our contributions are as follows. (i) We introduce OpenDeepThink, a population-based test-time compute framework that combines pairwise Bradley–Terry selection with feedback-driven mutation, requiring no external verifier or domain-specific infrastructure. (ii) On competitive programming, the framework raises Gemini 3.1 Pro's effective Codeforces Elo by $+405$ points, comparable to the $+411$ that Gemini 3 Deep Think achieves over Gemini 3.1 Pro on LiveCodeBench Pro (Zheng et al., 2025); the same hyperparameters transfer to Gemini 3 Flash and Gemini 2.5 Pro without retuning. (iii) On HLE, gains appear concentrated in objectively verifiable domains and reverse in subjective ones, suggesting that the framework's effectiveness tracks the reliability of pairwise LLM judgment. (iv) We release CF-73, a curated set of 73 expert-annotated Codeforces problems with 99% local-evaluation agreement against the official verdict.

2 Related Work

Test-time compute scaling and verifier-based selection. The dominant approach to improving LLM reasoning at inference scales compute along a single sequential trace, either by training models to produce longer chains of thought (Jaech et al., 2024; Guo et al., 2025) or by searching over reasoning steps with a learned value function (Snell et al., 2024). Self-consistency (Wang et al., 2022) parallelizes naturally by sampling multiple traces and selecting the majority answer, but is limited to tasks with extractable, votable final answers. When the output is a full program or an open-ended argument, selection requires a stronger signal: best-of-N methods rely on trained outcome or process reward models (Cobbe et al., 2021; Lightman et al., 2023; Wang et al., 2024), and tree-search methods require a value function over partial reasoning states (Yao et al., 2023; Hao et al., 2023). Brown et al. (2024) show that coverage scales log-linearly with sample count but that existing selectors plateau without a ground-truth verifier, articulating the exact bottleneck OpenDeepThink targets.

Our framework removes the verifier requirement entirely: it replaces trained reward models and majority voting with a Bradley–Terry ranking over pairwise LLM judgments, an aggregator that handles open-ended outputs and internalizes opponent strength without any labeled data.

Self-refinement. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) iterate on a single trajectory through feedback–rewrite cycles, improving outputs without additional training. However, Huang et al. (2023) demonstrate that intrinsic self-correction, without external feedback or oracle stopping, often degrades performance, because a single trace cannot reliably identify its own errors. OpenDeepThink responds directly to this finding: errors are surfaced between candidates via pairwise comparison, an easier discriminative task than absolute self-evaluation, and only the aggregated signal drives mutation. Dang et al. (2026) independently identify this failure mode as the "Cognitive Well": in solver–grader pipelines, iterative refinement can converge to a confident but incorrect solution that the pipeline's own grader cannot reject, motivating their use of conjecture extraction and independent verification rather than in-place refinement.

Evolutionary methods and LLM-as-judge. OpenDeepThink grafts two lines of work onto a shared scaffold. On the evolutionary side, FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Novikov et al., 2025) use LLMs as mutation operators over populations of programs but depend on programmatic, ground-truth evaluators for fitness; EvoPrompt (Guo et al., 2023) evolves prompts rather than solutions and requires a labeled development set. OpenDeepThink inherits the population–mutation–selection loop but replaces the evaluator with a training-free fitness signal: pairwise LLM judgments aggregated via Bradley–Terry (Bradley & Terry, 1952). On the judge side, Zheng et al. (2023) and Liusie et al. (2024) establish that pairwise LLM judgments align better with human preferences than pointwise scoring, and Chatbot Arena (Chiang et al., 2024) demonstrates that Bradley–Terry maximum-likelihood estimation converts noisy pairwise votes into stable rankings at scale. OpenDeepThink lifts this machinery from offline model ranking to in-the-loop search: the BT score defines elites, routes feedback, and selects the final submission.

Concurrent work. Several concurrent efforts explore parallel test-time reasoning: Population-Evolve (Zhang et al., 2025) maintains a population but aggregates by majority voting; ParaThinker (Wen et al., 2025), PaCoRe (Hu et al., 2026), and Multiverse (Yang et al., 2025) train models for parallel reasoning paths; SSA (Qi et al., 2025) trains a compact aggregator; V_1 (Singh et al., 2026) shows that pairwise self-verification substantially outperforms pointwise scoring and proposes a tournament-based ranking for parallel candidate selection; Squeeze Evolve (Maheswaran et al., 2026) addresses diversity collapse and cost efficiency in verifier-free evolution through multi-model orchestration; PDR (Madaan et al., 2025) generates parallel drafts and distills them into a shared workspace for refinement; Dang et al. (2026) design a competition-math pipeline with conjecture extraction to escape grader failure. Closest to our work, Duel-Evolve (Karlekar et al., 2026) also aggregates LLM self-preferences with a Bradley–Terry model for reward-free test-time evolution, but allocates comparisons via Double Thompson Sampling over 150–200 sequential generations, whereas OpenDeepThink runs in bulk-parallel rounds with a fixed sequential depth of eight LLM calls. Duel-Evolve also conditions its generator on parents annotated with scalar Bradley–Terry posterior means; OpenDeepThink instead recycles the natural-language critiques from comparison as the mutation signal, which carries nearly all of the mutation gain (Table 3). OpenDeepThink is distinguished by being training-free, verifier-free, and applicable to open-ended outputs, with Bradley–Terry aggregation over pairwise critiques providing the selection signal.

3 METHOD

3.1 SETUP

Problem setup. We study the problem of maximizing an LLM's accuracy on hard reasoning tasks under three simultaneous constraints: (i) a fixed per-problem compute budget measured in API calls, (ii) a wall-clock time limit, and (iii) no access to a ground-truth verifier or hidden test cases at inference time. Only the problem statement is available; the solver has no access to web search, external knowledge retrieval, or tool use. The wall-clock constraint is load-bearing: it rules out methods that merely extend a single reasoning chain, since such methods cannot convert additional compute into reduced response time.

Algorithm 1 OpenDeepThink.

Require: Problem x; LLM π (generator and judge); hyperparameters n, K, T, M.
Ensure: Selected solution $\hat{y}$.

$\mathcal{Y}^{(0)} \leftarrow \{y_i^{(0)} \sim \pi(\cdot \mid x)\}_{i=1}^{n}$ ▷ initial parallel sampling
for $t = 1, \ldots, T$ **do**
  $\mathcal{P}^{(t)} \leftarrow$ random pairing assigning each $y_i^{(t-1)}$ to K peers, uniformly
  $\{(c_{ij}^{(t)}, r_{ij}^{(t)}, r_{ji}^{(t)})\}_{(i,j)\in\mathcal{P}^{(t)}} \leftarrow$ parallel judgments by π with randomized presentation order
  $\mathbf{s}^{(t)} \leftarrow \arg\max_{\mathbf{s}} \mathcal{L}_{\mathrm{BT}}(\mathbf{s}; \{c_{ij}^{(t)}\}) - \frac{1}{2}\lambda\|\mathbf{s}\|_2^2$ ▷ L-BFGS
  $\mathcal{E}^{(t)} \leftarrow$ top-$\lceil n/4 \rceil$ of $\mathcal{Y}^{(t-1)}$ by $\mathbf{s}^{(t)}$ ▷ elite preservation
  $\mathcal{D}^{(t)} \leftarrow$ bottom-$\lfloor n/4 \rfloor$ of $\mathcal{Y}^{(t-1)}$ by $\mathbf{s}^{(t)}$ ▷ discard
  for each $y_i^{(t-1)} \notin \mathcal{D}^{(t)}$ in parallel **do**
    $y_i^{(t)} \sim \pi(\cdot \mid x, y_i^{(t-1)}, \mathrm{Aggregate}(\{r_{ji}^{(t)}\}_j))$ ▷ feedback-driven mutation
  end for
  $\mathcal{Y}^{(t)} \leftarrow \mathcal{E}^{(t)} \cup \{y_i^{(t)}\}_{i: y_i^{(t-1)} \notin \mathcal{D}^{(t)}}$
end for
$\mathcal{P}^{\star} \leftarrow$ random pairing assigning each $y \in \mathcal{Y}^{(T)}$ to M peers, uniformly
$\{c_{ij}^{\star}\}_{(i,j)\in\mathcal{P}^{\star}} \leftarrow$ parallel judgments by π
$\mathbf{s}^{\star} \leftarrow \arg\max_{\mathbf{s}} \mathcal{L}_{\mathrm{BT}}(\mathbf{s}; \{c_{ij}^{\star}\}) - \frac{1}{2}\lambda\|\mathbf{s}\|_2^2$
return $\hat{y} \leftarrow \arg\max_{y\in\mathcal{Y}^{(T)}} s_y^{\star}$

Notation. We denote a problem instance by x and write $\mathcal{Y}^{(t)} = \{y_1^{(t)}, \ldots, y_n^{(t)}\}$ for the population of n candidate solutions at generation $t \in \{0, 1, \ldots, T\}$. The same LLM π serves as both generator and judge. For a pair (y_i, y_j) at generation t, the judge produces an outcome $c_{ij}^{(t)} \in \{i \succ j,\ j \succ i,\ \text{tie}\}$ together with a pair of natural-language rationales $(r_{ij}^{(t)}, r_{ji}^{(t)})$, where $r_{ab}^{(t)}$ is the rationale supporting candidate a over b. We aggregate the comparisons within generation t into a Bradley–Terry score vector $\mathbf{s}^{(t)} \in \mathbb{R}^n$. We write $\mathcal{E}^{(t)} \subset \mathcal{Y}^{(t-1)}$ for the subset of candidates preserved as elites at generation t, and $\mathcal{D}^{(t)} \subset \mathcal{Y}^{(t-1)}$ for the subset discarded. We write $\hat{y} \in \mathcal{Y}^{(T)}$ for the solution selected after the final round. The pipeline has four hyperparameters: population size n, per-generation comparisons per candidate K, number of evolution generations T, and final-round comparisons per candidate M.

3.2 Selection

Pairwise comparison design. At generation t, each candidate $y_i^{(t-1)}$ is compared against $K = 4$ randomly sampled peers. The comparison prompt instructs the judge to identify the solution more likely to be accepted by a hypothetical online judge, to declare a tie when appropriate, and to supply a brief natural-language rationale $r_{ij}^{(t)}$ for each side; these rationales are reused as feedback during mutation. To mitigate position bias (Zheng et al., 2023), we randomize the presentation order in every comparison. We use the same model for generation and judgment: sharing the model reduces system complexity and demonstrates that the framework does not depend on an auxiliary verifier. Comparisons across different pairs are independent and execute in parallel.

Bradley–Terry aggregation. Given the pairwise outcomes $\{c_{ij}^{(t)}\}$ within a generation, we fit the Bradley–Terry (BT) score vector $\mathbf{s}^{(t)}$ (Bradley & Terry, 1952) under

$$P(i \succ j) = \sigma(s_i^{(t)} - s_j^{(t)}), \tag{1}$$

where σ is the logistic sigmoid and ties contribute half a win to each side. We estimate $\mathbf{s}^{(t)}$ by maximizing the regularized log-likelihood of the observed comparisons with L-BFGS (Liu & Nocedal, 1989), adding a small ℓ_2 penalty for numerical stability and to fix the global shift, since the BT log-likelihood is invariant under additive shifts of $\mathbf{s}^{(t)}$. We prefer BT over raw win rate because each candidate faces only a random subset of opponents; BT internalizes opponent strength, yielding an adjusted ranking that raw win rate cannot. This adjustment is particularly important at $K = 4$, where

sampling noise from the pairing distribution is non-negligible. The same BT formulation underlies large-scale preference evaluation of LLMs (Chiang et al., 2024), and we exploit it here as a soft verifier that distills noisy pairwise signals into a coherent global ranking.

Implementation details. We set $\lambda = 0.01$ with the penalty term $\frac{1}{2}\lambda\|\mathbf{s}\|_2^2$ for all BT fits. Pairwise comparisons sample a random K-regular matching without self-pairs; duplicate pairs within a generation are not permitted. Invalid JSON judge outputs are retried once; remaining failures are treated as ties.

Elite preservation. After fitting the BT scores $\mathbf{s}^{(t)}$, the top 25% of candidates are preserved as elites $\mathcal{E}^{(t)}$ and carried forward unchanged, while the bottom 25% are discarded. The top 75% (elites included) are routed to mutation, so each elite contributes both its original and a mutated variant. The bottom quartile is consistently worse than the middle range, so mutating it offers no benefit. Allowing elites to be mutated alongside the middle ranks maintains diversity without sacrificing selection pressure.

3.3 Mutation

The natural-language rationales $\{r_{ji}^{(t)}\}_j$ produced during pairwise comparison are a byproduct of selection that we deliberately recycle rather than discard. For each non-discarded candidate $y_i^{(t-1)}$ (the top 75%, including elites), we aggregate all feedback directed at it across its K comparisons, yielding a targeted critique grounded in head-to-head failures. The mutator is conditioned on three inputs: (i) the original problem statement, (ii) the current solution, and (iii) the aggregated feedback. The prompt permits the model to refine the existing solution or to abandon it for a fundamentally different approach. Informally, we observed that without this permission the mutator tends to apply local patches to a broken strategy rather than restructure it; we do not test this directly with a controlled ablation. Granting license to restart allows mutation to traverse qualitatively different solution sketches rather than hill-climbing within the neighborhood of a flawed initialization. Our mutation is executed as a synchronous parallel batch: all non-discarded candidates within a generation are mutated simultaneously and independently, each conditioned on its own aggregated feedback. Combined with elite preservation and bottom-quartile discarding, this yields a dynamic in which strong solutions are both retained and mutated, moderate ones are regenerated, and the weakest are eliminated.

4 Experiments

4.1 Setup

Benchmarks. The primary evaluation suite pairs two complementary sources of competition-level programming problems totaling 192 items, each requiring a correct C++ solution with no access to hidden test cases at inference time. CF-73 consists of 73 recent Codeforces problems from Div. 1 and Div. 2 rounds rated roughly 2000 to 3100, annotated by an International Grandmaster and judged against the official Codeforces test suites (Mirzayanov, 2010); our local judge agrees with the official verdict on 99% of submissions. All 73 problems are drawn from rounds held after August 2025, postdating the knowledge cutoff of every model evaluated (Gemini 2.5 Pro, 3 Flash, and 3.1 Pro), making pretraining contamination unlikely. The 1% of disagreements between local and official verdicts are exclusively near-threshold TLE cases caused by differences in machine speed between our evaluation hardware and the Codeforces judging servers. NOI-119 consists of 119 problems from a national informatics olympiad training program whose statements are public but whose tests and evaluation infrastructure remain private, accessed through a hidden online judge that returns binary accept/reject verdicts. Cross-domain experiments use 82 questions sampled uniformly at random from the HLE gold set (Phan et al., 2025), a verified subset of the Humanity's Last Exam benchmark covering mathematics, natural sciences, computer science and AI, humanities, and social sciences.

Difficulty tiers. We stratify the 192 programming problems by the gen-0 pass@1 rate of the main-experiment base model, estimated from the 20 independent samples that seed evolution. The cutoffs are the 33rd and 67th percentiles of that distribution and yield an Easy tier of 53 problems with

pass@1 $> 95\%$, a Medium tier of 75 problems with $35\% <$ pass@1 $\leq 95\%$, and a Hard tier of 64 problems with pass@1 $\leq 35\%$. The source composition varies sharply with difficulty: Easy draws 32 problems from CF and 21 from NOI, Medium splits 31/44, and Hard concentrates in NOI with only 10 of its 64 problems coming from CF. Within each tier, random pass@1 measures how often a single unranked sample succeeds, while BT top-1 measures whether the Bradley–Terry winner is accepted; the contrast between the two isolates aggregation's contribution from raw sampling coverage.

Models. The main experiments use Gemini 3.1 Pro (Comanici et al., 2025) as both the generator and the judge. We additionally report Gemini 3 Flash and Gemini 2.5 Pro under identical hyperparameters and no per-model tuning. Pairwise judgments randomize presentation order throughout.

Compute budget. The per-problem LLM-call budget decomposes into three contributions: the initial sampling of n candidates, T evolution generations each combining pairwise comparisons with mutation of the top 75% (including elites), and a single denser final round of pairwise comparisons for selection. Summing these components gives

$$\text{Total}(n, K, T, M) \;=\; \underbrace{n}_{\text{sampling}} + T \cdot \Big(\underbrace{\tfrac{nK}{2}}_{\text{compare}} + \underbrace{\tfrac{3n}{4}}_{\text{mutate}} \Big) + \underbrace{\tfrac{nM}{2}}_{\text{final select}}, \tag{2}$$

which, at the main-experiment setting $n = 20$, $K = 4$, $T = 3$, $M = 10$, evaluates to approximately 285 calls per problem. All four hyperparameters are held fixed across problems and benchmarks. The pipeline's sequential depth is eight LLM calls: one initial sampling round, two rounds (compare + mutate) per evolution generation for $T = 3$ generations, and one final comparison round. All calls within each round are embarrassingly parallel across candidates. At the observed median per-call latency of approximately 200 seconds for Gemini 3.1 Pro on competition-programming problems, this sequential depth corresponds to roughly 27 minutes under full parallelization. Our implementation waits for all responses within each round; in deployment, over-provisioning parallel requests and discarding late completions could approach this bound without harming selection quality, since we observe that shorter-latency responses tend to have higher accept rates on our benchmarks. The sequential depth remains fixed at eight rounds regardless of population size n or comparison count K.

4.2 Main Result

Table 1 reports per-tier results on the full 192-problem benchmark. The three difficulty tiers cleanly separate the two mechanisms of OpenDeepThink. On Easy and Medium problems, where random pass@1 is already high, BT aggregation alone, applied to the unevolved gen-0 population, reaches saturation on Easy (100%) and near-saturation on Medium (95%); evolution adds almost nothing because there is little to fix. The Hard tier is where evolution becomes load-bearing: random pass@1 climbs from 11% to 36% across three generations, and BT top-1 rises from 23% to 50% (95% CI of gain: $[16, 39]$ pp, paired bootstrap). The $+39$-point gain over random pass@1 decomposes into $+25$ from evolution producing better candidates and $+14$ from selection picking better among them. Aggregation dominates when the base model already solves most problems; evolution dominates at the capability frontier. On CF-73, translating BT top-1 verdicts into effective Codeforces Elo (Section A.2), a single gen-0 random pick yields 2851 (95% CI: $[2716, 2974]$) and the post-evolution BT winner attains 3256 ($[3049, 3655]$), a $+405$-Elo gain comparable to the $+411$ that Gemini 3 Deep Think achieves over Gemini 3.1 Pro on LiveCodeBench Pro (Zheng et al., 2025). Because the two benchmarks differ in problem composition, we compare gains rather than absolute ratings.

Cross-model transfer. Table 2 runs the same pipeline with identical hyperparameters on Gemini 3 Flash and Gemini 2.5 Pro, each evaluated on the difficulty tier matched to its sweet spot. All three models improve on both pass@1 and BT top-1, and the balance between the two mechanisms shifts predictably along the capability axis: Gemini 3 Flash, the weakest model, shows its largest lift in pass@1 ($+30$), meaning evolution is the dominant contributor; Gemini 3.1 Pro, the strongest, shows its largest lift in BT top-1 ($+27$), meaning selection contributes more. No per-model tuning is applied. Two observations discipline the claim: first, evolution and selection contribute unequally but both contribute positively across all three models; second, problems that no model ever solves at gen-0 are rarely rescued. Of the seventeen unsolved problems across the Flash and 2.5 Pro runs, none

Table 1: OpenDeepThink on the CF-73 + NOI-119 benchmark with Gemini 3.1 Pro at n=20, K=4, T=3, M=10. Pass@1 is the empirical accept rate of a single unranked gen-0 sample and lower-bounds what naive sampling achieves. Oracle, shown in gray, is the gen-0 pass@20 score: it counts problems where at least one of the 20 initial candidates is accepted and upper-bounds any gen-0 selector. Ours reports Bradley–Terry top-1 accuracy at each generation. Generations 0–2 use the sparse K=4 comparisons from the evolution loop; Generation 3 uses the final dense M=10 round. All comparisons fall within the per-problem budget of ∼285 calls; no additional comparisons are used for intermediate reporting.

	NOI-119	CF-73	Easy	Medium	Hard	All
# Problems	119	73	53	75	64	192
Pass@1	49%	80%	100%	76%	11%	61%
Oracle	76%	96%	100%	100%	50%	83%
OpenDeepThink (Ours)						
Generation 0	61%	92%	100%	95%	23%	72%
Generation 1	70%	96%	100%	97%	42%	80%
Generation 2	70%	95%	100%	96%	42%	79%
Generation 3	75%	96%	100%	99%	50%	83%

Table 2: Cross-model and cross-domain transfer. **(a)** Cross-model generalization on CF-73 + NOI-119 difficulty tiers. Tier labels are defined relative to 3.1 Pro's gen-0 pass@1 (Section 4.1). Hyperparameters n=20, K=4, T=3, M=10 are held fixed with no per-model tuning. **(b)** HLE BT top-1 accuracy by category, gen-0 versus gen-2. Objective-correctness domains show directional gains; subjective-judgment domains show directional declines. Two singleton categories (engineering, chemistry) are omitted as underpowered. All models are from the Gemini family.

(a) Cross-model

	3.1 Pro	3 Flash	2.5 Pro
Tier	Hard	Medium	Easy
# Problems	64	75	53
gen-0 pass@1	11%	39%	28%
gen-0 BT top-1	23%	64%	42%
Final pass@1	36%	69%	50%
Final BT top-1	50%	76%	60%
Δ (pass / BT)	+25/27	+30/12	+22/18

(b) Cross-domain (HLE)

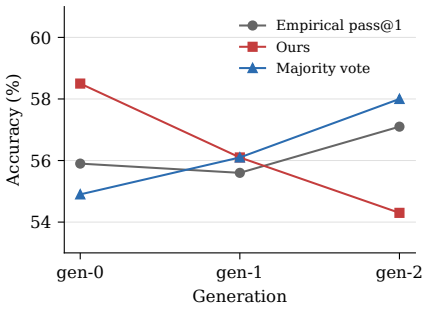
Category	N	gen-0	gen-2	Δ
Mathematics	34	56%	61%	+5
Biology / Medicine	7	43%	57%	+14
Physics	6	33%	50%	+17
Computer Science / AI	15	67%	64%	−2
Humanities / Soc. Sci.	8	75%	50%	−25
Other	10	70%	40%	−30
All	82	58.5%	54.3%	−4

crosses 5% pass@1 in any generation, consistent with the hypothesis that OpenDeepThink amplifies partial competence rather than inducing new capabilities from scratch; we do not test this directly.

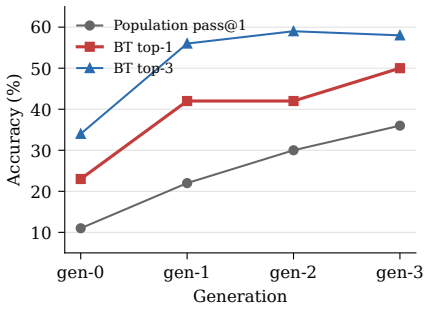
Cross-domain transfer. We evaluate on 82 HLE questions with n=12, K=4, T=2, M=10. The key contrast with competitive programming is immediate: BT top-1 accuracy declines from 58.5% to 54.3% across two generations, while majority vote rises by +3.1 points (Figure 2a). The category breakdown (Table 2b) suggests a directional pattern, though category-level sample sizes (N=6–8 outside mathematics) preclude strong statistical claims. Domains with objectively verifiable answers, namely mathematics, biology, and physics, show directional BT top-1 gains of +5 to +17 points, directionally consistent with competitive programming. Domains with ambiguous correctness criteria, humanities and social sciences, decline by 25 to 30 points. Whether OpenDeepThink helps or hurts tracks the reliability of pairwise LLM judgment: where the judge can discriminate correct from incorrect, BT selection amplifies signal; where it cannot, iterative selection amplifies noise.

5 Discussion

Pairwise selection outperforms pointwise under matched generation quality. The advantage of OpenDeepThink's Bradley–Terry selector over pointwise scoring is not an artifact of better candidates; it persists after generation quality is equalized and traces entirely to the selection mechanism. The



(a) HLE aggregate, N=82.



(b) Iteration ablation on Hard, N=64.

Figure 2: **(a)** HLE aggregate accuracy across evolution generations. Three aggregators are tracked: pass@1, BT top-1 (Ours), and majority vote. BT top-1 is the only aggregator that declines, consistent with the pairwise judge's unreliability on subjective domains (Table 2b). **(b)** Iteration dynamics on the Hard tier (N=64, pairwise K=4). Population pass@1 and BT top-k are shown per generation; gen-3 values use the final dense BT round.

root cause is positive bias in pointwise scoring: pointwise judgment achieves high recall on accepted solutions but poor recall on wrong ones, so it cannot reliably reject incorrect candidates (Zheng et al., 2023; Liusie et al., 2024). Pairwise comparison sidesteps this failure mode by reducing judgment to a relative contrast that requires no calibrated quality threshold. On a 500-pair diagnostic drawn from NOI-119 (Section A.3), pairwise judgment reaches 86% accuracy versus 59% for pointwise. This finding aligns with Singh et al. (2026), who independently demonstrate that pairwise self-verification is a substantially more accurate primitive than pointwise scoring for selecting among parallel candidates. The gap survives two increasingly aggressive controls for generation quality. First, applying 14 pointwise judgments per candidate to the same unevolved gen-0 pool yields only 26% top-1 on the Hard tier at a budget of 300 calls (20 shared initial samples + 280 pointwise judgments). Second, giving the baseline its own refinement budget, six rounds of Self-Refine (Madaan et al., 2023) followed by 8 pointwise votes per candidate (300 calls), closes the generation gap almost entirely (pass@1: 33% vs. OpenDeepThink's 36%), yet top-1 reaches only 41% against 50%: the 9-point residual is pure selection signal. The effect is not merely average-case. Of the 64 Hard problems, OpenDeepThink's BT top-1 is accepted on 32; the strongest pointwise baseline is accepted with certainty—all tied-top candidates correct—on only 6. OpenDeepThink uniquely solves 27 problems that the baseline never reliably solves, while the baseline uniquely solves 1. The baseline's 41% figure reflects the expected accept rate under random tie-breaking among equally-scored candidates, not a deterministic solve count. Additional analyses of aggregator choice, judge-prompt robustness, a non-Gemini model, and the per-stage compute budget are reported in Section A.7.

Negative feedback carries nearly all the mutation signal. Within a single mutation round, the improvement is almost entirely driven by negative feedback; positive feedback is statistically indistinguishable from no feedback at all (Table 3). Telling the mutator what went wrong carries actionable signal; telling it what went right adds nothing beyond what the model already infers from seeing its own solution. Structuring the negative signal as pairwise critique at K=4 nearly doubles the net rescue rate over the no-feedback baseline, because head-to-head contrast surfaces failure modes that a single trajectory cannot self-diagnose, consistent with the discriminative advantage that motivates pairwise selection in the first place. Beyond K=4 the return reverses: K=5 regresses as the mutator receives more contrasts than it can integrate in a single rewrite[2]. Stratifying by difficulty sharpens the picture: on Medium problems, pairwise feedback's rescue advantage over no feedback is $\sim$ 26 points; on Hard problems it narrows to $\sim$ 4 points. Feedback helps the mutator cross a nearby acceptance threshold, not learn a fundamentally new algorithm. Evolution amplifies partial competence.

[2]The primary role of K is to supply enough pairwise observations for reliable Bradley–Terry ranking over n=20 candidates; the feedback signal recycled for mutation is a byproduct of this comparison budget. Because the ranking-precision constraint is statistical rather than model-dependent, we expect the operating point to be relatively stable across judge models, though we have not explicitly verified this.

Table 3: Feedback strategy ablation on 500 solutions from 64 NOI problems. AC and WA denote the online judge's Accepted and Wrong Answer verdicts; 194 solutions are originally AC and 306 originally WA. One round of mutation is applied per strategy. Rescue counts WA → AC transitions; Degradation counts AC → WA; Δ is rescued minus degraded. The chosen setting is **pairwise** K=4.

Feedback strategy	AC rate	Rescued (WA→AC)	Degraded (AC→WA)	Δ
No feedback	47.6%	48 (15.7%)	4 (2.1%)	+44
Positive only	47.8%	52 (17.0%)	7 (3.6%)	+45
Negative only	49.4%	68 (22.2%)	15 (7.7%)	+53
Pairwise, all pairs	50.4%	70 (22.9%)	12 (6.2%)	+58
Pairwise, K=2	51.6%	73 (23.9%)	9 (4.6%)	+64
Pairwise, K=3	51.2%	69 (22.5%)	7 (3.6%)	+62
Pairwise, K=4	52.4%	74 (24.2%)	6 (3.1%)	**+68**
Pairwise, K=5	50.8%	68 (22.2%)	8 (4.1%)	+60

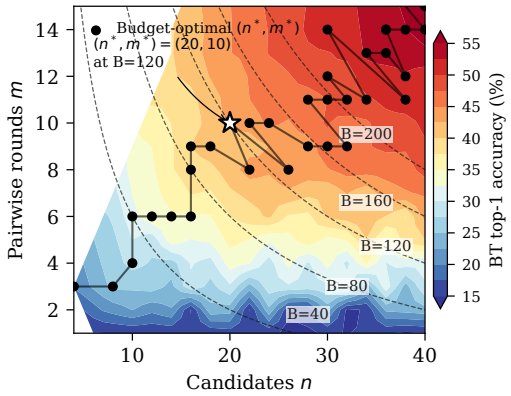


Figure 3: BT-only scaling on 10 Hard problems. Color is BT top-1 accuracy on the (n, m) plane from Monte-Carlo simulation (500 trials per cell, 40 pre-judged candidates per problem). Dashed contours mark equal-budget allocations. Black dots trace the budget-optimal (n^*, m^*); the white star marks $(20, 10)$ at B=120, matching the main pipeline's selection-only budget (excluding evolution calls). Experimental details in Section A.4.

Evolution is front-loaded; dense comparison extracts the residual. Across generations, the largest single improvement occurs at gen-0 → gen-1, where the first mutation round converts the most tractable failures (Figure 2b). Meanwhile, sparse intra-generation comparison (K=4) saturates by gen-2: once candidates become roughly comparable, a low comparison budget can no longer resolve their ranking. The final dense round (M=10) breaks this plateau, extracting an additional +8 points at top-1 beyond sparse gen-2 BT. Evolution and dense selection therefore play distinct and complementary roles: evolution raises the ceiling of the candidate pool; dense Bradley–Terry extracts it. The quadratic cost of comparison ($m^* \approx n/2$ yields $\Theta(n^2)$ total) makes population size the binding constraint; Figure 3 confirms that the budget-optimal allocation at B=120 matches the main pipeline's $(n, M) = (20, 10)$.

6 Conclusion

OpenDeepThink addresses the selection bottleneck that emerges when test-time compute scaling is parallelized rather than deepened. The same LLM acts as both generator and pairwise judge: Bradley–Terry aggregation yields a soft verifier, and the comparison critiques drive feedback-directed mutation. On Codeforces, the framework lifts Gemini 3.1 Pro's Elo by +405 and transfers across models without retuning. On HLE, gains hold where pairwise judgment is reliable and reverse where it is not, so the soft verifier is only as good as the comparisons it aggregates. *Limitations.* The framework has been validated mainly on Gemini-family models; small-scale preliminary results on DeepSeek V4 Flash (Section A.7) suggest that the mechanism transfers. The per-problem cost of ∼285 API calls is substantial and may be prohibitive for latency-sensitive applications. Finally, the 25% elite ratio and the license-to-abandon prompt were chosen by informal tuning rather than controlled ablation.

AI USE STATEMENT

Large language models are a component of the proposed method itself: Gemini 3.1 Pro, Gemini 3 Flash, Gemini 2.5 Pro, and DeepSeek V4 Flash serve as the candidate generators and pairwise judges in all reported experiments (Section 4 and Section A.7). Separately from this role, we used generative AI tools for the following tasks that require disclosure: AI coding assistants helped implement the experiment code and run the experiments. Research ideation, experimental design, and the interpretation of results were carried out by the authors; generative AI tools were used at most for occasional discussion in these tasks, and all final decisions were made by the authors. Additionally, we used generative AI tools for tasks with recommended disclosure: creating and editing software code; editing and rewriting the text for readability; drafting parts of the paper, including the comparison with concurrent work and the text of Section A.7; verifying bibliographic information; LaTeX formatting; and organizing the appendix. We have reviewed all AI-assisted work. All reported numbers are taken from the logs of actual experiment runs and were checked by the authors against these logs. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The full pipeline code is available in an anonymous repository (`https://anonymous.4open.science/r/open-deep-think-E207`), and the CF-73 benchmark is released at `https://anonymous.4open.science/r/CF-73-2CB1`. Section 4.1 specifies the benchmarks, models, and hyperparameters. Section A.2 describes the Elo estimation procedure; Sections A.3 and A.4 describe the setups of the pairwise diagnostic and the BT-only scaling study; Section A.5 documents the pointwise and Self-Refine baselines; and Section A.6 lists all prompt templates verbatim together with the sampling temperatures.

REFERENCES

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Xingyu Dang, Rohit Agarwal, Rodrigo Porto, Anirudh Goyal, Liam H Fowl, and Sanjeev Arora. Escaping the cognitive well: Efficient competition math with off-the-shelf models. *arXiv preprint arXiv:2602.16793*, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

Jingcheng Hu, Yinmin Zhang, Shijie Shang, Xiaobo Yang, Yue Peng, Zhewei Huang, Hebin Zhou, Xin Wu, Jie Cheng, Fanqi Wan, et al. Pacore: Learning to scale test-time compute with parallel coordinated reasoning. *arXiv preprint arXiv:2601.05593*, 2026.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Sweta Karlekar, Carolina Zheng, Magnus Saebo, Nicolas Beltran-Velez, Shuyang Yu, John Bowlan, Michal Kucer, and David Blei. Duel-evolve: Reward-free test-time scaling via LLM self-preferences. *arXiv preprint arXiv:2602.21585*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 2511–2522, 2023.

Adian Liusie, Potsawee Manakul, and Mark Gales. Llm comparative assessment: Zero-shot nlg evaluation through pairwise comparisons using large language models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 139–151, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Lovish Madaan, Aniket Didolkar, Suchin Gururangan, John Quan, Ruan Silva, Ruslan Salakhutdinov, Manzil Zaheer, Sanjeev Arora, and Anirudh Goyal. Rethinking thinking tokens: Llms as improvement operators. *arXiv preprint arXiv:2510.01123*, 2025.

Monishwaran Maheswaran, Leon Lakhani, Zhongzhu Zhou, Shijia Yang, Junxiong Wang, Coleman Hooper, Yuezhou Hu, Rishabh Tiwari, Jue Wang, Harman Singh, et al. Squeeze evolve: Unified multi-model orchestration for verifier-free evolution. *arXiv preprint arXiv:2604.07725*, 2026.

Mike Mirzayanov. Codeforces. `https://codeforces.com`, 2010.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Jianing Qi, Xi Ye, Hao Tang, Zhigang Zhu, and Eunsol Choi. Learning to reason across parallel samples for llm reasoning. *arXiv preprint arXiv:2506.09014*, 2025.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Harman Singh, Xiuyu Li, Kusha Sareen, Monishwaran Maheswaran, Sijun Tan, Xiaoxia Wu, Junxiong Wang, Alpay Ariyak, Qingyang Wu, Samir Khaki, et al. v_1: Unifying generation and self-verification for parallel reasoners. *arXiv preprint arXiv:2603.04304*, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

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. *arXiv preprint arXiv:2203.11171*, 2022.

Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute. *arXiv preprint arXiv:2509.04475*, 2025.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language models secretly decide how to parallelize and merge generation. *arXiv preprint arXiv:2506.09991*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Yanzhi Zhang, Yitong Duan, Zhaoxi Zhang, Jiyan He, and Shuxin Zheng. Population-evolve: a parallel sampling and evolutionary method for llm math reasoning. *arXiv preprint arXiv:2512.19081*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Zihan Zheng, Zerui Cheng, Zeyu Shen, Shang Zhou, Kaiyuan Liu, Hansen He, Dongruixuan Li, Stanley Wei, Hangyi Hao, Jianzhu Yao, et al. Livecodebench pro: How do olympiad medalists judge llms in competitive programming? *arXiv preprint arXiv:2506.11928*, 2025.

A TECHNICAL APPENDICES AND SUPPLEMENTARY MATERIAL

A.1 HLE CATEGORY BREAKDOWN

The HLE category breakdown is reported in Table 2(b) in the main text. Two singleton categories, engineering and chemistry, are omitted as underpowered.

A.2 ELO RATING ESTIMATION

The effective Elo ratings of Section 4.2 are computed on the CF-73 subset, since NOI problems do not have published Codeforces ratings. We adopt the standard Elo logistic model in which the probability that a player at rating R_{model} solves a problem at rating R_{problem} is

$$P(\text{solve}) \;=\; \frac{1}{1 + 10^{(R_{\text{problem}} - R_{\text{model}})/400}}.$$

Each problem's R_{problem} is taken from the Codeforces website. We estimate R_{model} by maximum a posteriori (MAP) under a Gaussian prior $\mathcal{N}(3100,\, 500^2)$ centered loosely on the published rating of Gemini 3.1 Pro, optimizing the posterior with `scipy.optimize.minimize_scalar` over the bounded interval $[1000,\, 5000]$.

We report two scenarios. For gen-0 pass@1, the per-problem likelihood is Binomial with $n = 20$ independent gen-0 samples and k accepted; this measures the rating implied by naive sampling. For the final BT top-1, the per-problem likelihood is Bernoulli, treating the BT-ranked top candidate as a single submission that is either accepted or rejected; this measures the rating of the post-evolution selector. The same prior, optimizer, and per-problem ratings are used in both scenarios.

Confidence intervals are obtained by bootstrap resampling. We draw 1000 resamples by resampling problems with replacement (a problem contributes its full likelihood factor each time it is drawn), refit R_{model} on each resample, and report the 2.5 and 97.5 percentiles as the 95% CI. Resampling at the problem level captures the dominant source of variance, since per-problem outcomes are the noisy units; within-problem sample variance is already absorbed into the Binomial likelihood for the gen-0 scenario (this absorption does not apply to the BT top-1 scenario, which is per-problem Bernoulli).

A.3 PAIRWISE DIAGNOSTIC SETUP

The 500 (AC, WA) solution pairs used in the pairwise-vs-pointwise diagnostic of Section 5 are drawn from $\sim$60 NOI-119 problems. Each problem contributes 20 candidate solutions whose AC/WA labels are determined by the online judge against its private test suite, independently of any Bradley–Terry output. We form within-problem pairs from the $\binom{20}{2}$ combinations, retain only those containing one AC and one WA solution, and sample 500 pairs uniformly across the resulting pool. The pointwise variant scores each solution in isolation; the pairwise variant compares the two. Same judge model, randomized presentation order (Zheng et al., 2023). Pointwise attains 96.4% accuracy on AC but only 62.2% on WA, yielding a joint correctness of 59.2%. Pairwise reaches 86.2%. Since AC/WA labels are determined by the online judge independently of any BT output, the diagnostic does not feed circularly into the main result.

A.4 BT-ONLY SCALING STUDY SETUP

The scaling study of Figure 3 isolates Bradley–Terry aggregation from evolution on 10 Hard problems with gen-0 AC rate $\in [1/20,\, 4/20]$. For each problem we draw 40 candidates from π, label every candidate against the public test suite, and pre-compute all 780 pairwise judgments via full round-robin: 39 rounds of 20 pairs each. For each budget $B = n + m \cdot n/2$ and population size n, we Monte-Carlo sample n candidates without replacement, simulate $m = \lfloor 2(B - n)/n \rfloor$ rounds of random-pairing comparison with 500 trials per cell, aggregate with Bradley–Terry, and record whether the top-1 pick is accepted. Optimal n grows sublinearly with B at roughly $n^* \propto B^{0.6}$, so additional budget is better spent on broader sampling than on deeper comparison at the margin.

A.5 Baseline Details

Section 5 compares OpenDeepThink against two pointwise baselines. We document their implementation here.

Pointwise judge. Each candidate solution is scored in isolation using the same base model (Gemini 3.1 Pro) at temperature 1.0. The prompt asks whether the solution is correct for all valid inputs and requires a final line of exactly `VERDICT: YES` or `VERDICT: NO`. Responses that do not match this format are discarded. For each candidate, the judge is called N times independently (N=14 for the standalone pointwise baseline, N=8 for the Self-Refine variant), and the candidate's score is the count of `YES` verdicts. The top-1 pick is drawn uniformly at random from the candidates with the highest `YES` count; the reported accuracy is the fraction of AC solutions in this tied-top set, equivalent to the expected accept rate under random tie-breaking.

Self-Refine baseline. Starting from the same 20 gen-0 candidates as OpenDeepThink, each candidate is independently refined for six rounds. Each round uses a single-step prompt that asks the model to review its current solution and either output it unchanged or produce an improved version; unlike the two-step feedback-then-refine protocol of Madaan et al. (2023), our implementation merges feedback and rewriting into a single call. No cross-solution information is provided: each trajectory sees only the problem statement and its own most recent code. After six rounds, the 20 refined candidates are scored with 8 pointwise votes each and selected by the same `YES`-count procedure described above. The total budget is 300 calls per problem (20 shared initial samples + 120 refinement + 160 pointwise), comparable to OpenDeepThink's 285.

A.6 Prompt Templates

The full pipeline code is available at `https://anonymous.4open.science/r/open-deep-think-E207`.

Three prompt templates govern the pipeline. Generation and mutation share the same system prompt. The judge uses no system prompt; its full instruction is in the user message. Temperature is 1.0 for generation and mutation, 0.0 for judging. All placeholders are literal substitutions with no additional formatting.

Generation. Used in gen-0 sampling to produce n initial candidates per problem.

System

````
You are an expert competitive programmer.
Output your solution as a single ```cpp ... ``` block, preceded by
brief reasoning.
````

User

```
{problem}
```

Pairwise comparison. Used in per-generation comparison and the final dense BT round. Returns a JSON object with a winner verdict and per-side feedback. Presentation order of Solutions A and B is randomized per comparison to mitigate position bias.

User (no system prompt)

````
You are a competitive programming expert.

## Problem Statement
{problem}

## Solution A
```cpp
````
````

````
{code_a}
```

## Solution B
```cpp
{code_b}
```

Which solution is more likely to receive an Accepted verdict from
an online judge --- meaning it produces correct output within the
time and memory limits for all valid inputs?

If both solutions appear incorrect (wrong answer, TLE, or other
issues), choose the one that requires fewer modifications to become
Accepted.

If they are fundamentally identical or equally likely to be
Accepted, output TIE.

Respond with a JSON object and nothing else, in exactly this
format:
{
"feedback_a": "one sentence on Solution A's key strength or
critical flaw",
"feedback_b": "one sentence on Solution B's key strength or
critical flaw",
"winner": "A or B or TIE"
}
````

Mutation. Used to mutate non-discarded candidates (top 75%, including elites) each generation. The variant with feedback is the default; the variant without feedback is a fallback for candidates with no comparisons in the current generation.

System

```
Same as generation.
```

User (with feedback)

````
## Problem
{problem}

## Solution
```cpp
{code}
```

## Pairwise Feedback
This solution was compared against other solutions multiple times:

{feedback_sections}

## Task
Write a solution that maximizes the probability of Accepted.  You
may refine the existing solution or take a different approach if
the current one is fundamentally flawed.

Think briefly, then output your final solution as a single ```cpp
...  ``` block.
````

The {feedback_sections} placeholder is filled with the candidate's pairwise critiques from the current generation, partitioned by outcome:
````

Feedback section format

```
### Wins (this solution was judged better):
- {feedback}
### Ties (judged equally likely to be Accepted):
- {feedback}
### Losses (this solution was judged worse):
- {feedback}
```

Empty sections are omitted. Within each feedback string, the judge's references to "Solution A" / "Solution B" are rewritten to "this solution" / "the other solution" so the mutator receives self-relative critiques. The without-feedback variant omits the Pairwise Feedback section entirely; all other text is identical.

A.7 Additional analyses

This section reports four supplementary analyses: the choice of rank aggregator, robustness of the judge prompt, a preliminary run on a non-Gemini model, and a per-stage breakdown of the compute budget.

Aggregation ablation. We compare Bradley–Terry against Borda count (win $+1$, tie $+0.5$, loss $+0$) on all 192 CF/NOI problems and 82 HLE questions, reusing the stored pairwise comparison records at zero API cost (Table A.1). Of the 192 problems, 186 produce identical top-1 selections under both methods; the two aggregators are statistically indistinguishable.

Table A.1: Bradley–Terry vs. Borda count aggregation, computed on the same pairwise comparison records.

Setting	N	BT top-1	Borda top-1	Δ	95% CI
Final selection (M=10)	192	82.8%	82.8%	+0.0 pp	$[-1.8, +1.8]$
Sparse gen-0 (K=4)	192	72.3%	72.3%	+0.0 pp	$[-3.2, +3.0]$
HLE	82	54.9%	55.7%	−0.8 pp	$[-4.3, +2.4]$

Judge prompt robustness. We test the judge prompt on 400 discordant pairs (one AC, one WA) stratified by difficulty (Hard 200 / Medium 150 / Easy 50). To control for temporal model drift, the original prompt is re-run as a fresh baseline (V0-fresh) alongside two perturbations: a paraphrased prompt (V1) and the original prompt with Solutions A and B swapped. All runs use Gemini 3.1 Pro at temperature 0 (Table A.2). V0-fresh agrees with its own historical record on 74.2% of individual judgments, which sets the noise floor for temperature-0 reproducibility; the 72.5% agreement of V1 with V0-fresh is indistinguishable from this floor. In the swap run, the judge prefers the first-presented solution 47.9% of the time (ideal: 50%). The pipeline additionally randomizes presentation order by design.

Table A.2: Judge prompt robustness on 400 discordant pairs. CIs are from a cluster bootstrap.

Variant	Accuracy	TIE rate	Agreement w/ V0-fresh	Δ vs. V0-fresh	95% CI
V0-fresh	86.3%	27.0%	—	—	—
V1 Paraphrase	85.1%	26.2%	72.5%	−1.2 pp	$[-5.4, +2.9]$
Swap (A/B reversed)	87.7%	27.0%	71.0%	+1.4 pp	$[-2.5, +5.3]$

Preliminary cross-model run. We run the full pipeline on DeepSeek V4 Flash (Codeforces Elo 3052, a non-Gemini architecture) on 5 CF-73 problems with identical hyperparameters (n=20, K=4, T=3, M=10) and no code changes across all 14 LLM call sites (Table A.3). On 2170f, the AC rate rises from 0% to 45% over three generations; on 2145g, the gen-3 pool contains only $2/20$ correct solutions (10% random hit rate), yet BT selects a correct one. This is a small-scale check and is not intended as a full evaluation.

Table A.3: DeepSeek V4 Flash on 5 CF-73 problems with unchanged hyperparameters. "Random hit rate" is the fraction of correct solutions in the gen-3 pool.

Problem	Tier	gen-0 AC	gen-3 AC	BT #1	Random hit rate
2170f	Easy	$0/20$	$9/20$	AC	45%
2162g	Easy	$3/20$	$6/20$	WA	30%
2145g	Medium	$1/20$	$2/20$	AC	10%
2161e	Medium	$0/20$	$0/20$	WA	0%
2138e2	Hard	$0/20$	$0/20$	WA	0%

Compute breakdown. Table A.4 decomposes the 285-call budget by stage. On Hard problems, accuracy rises from 10.6% (pass@1) to 50.0% (BT top-1) across the 285 calls. Under a matched budget (285 vs. 300 calls), OpenDeepThink reaches 50.0% versus 40.6% for Self-Refine with pointwise selection (Section 5).

Table A.4: Per-stage breakdown of the 285-call budget. The first row is the expected single-sample pass rate without any comparison, corresponding to Pass@1 in Table 1.

Stage	Cumulative calls	BT top-1 (All)	BT top-1 (Hard)
gen-0 sampling (pass@1)	20	60.9%	10.6%
gen-0 BT selection	60	72.3%	23.4%
gen-1 evolve + BT	115	79.6%	42.2%
gen-2 evolve + BT	170	79.2%	42.2%
gen-3 evolve + BT	225	81.7%	46.9%
Final dense BT	285	82.8%	50.0%