How Far Can Reasoning Alone Take You? Graph-Based Candidate Selection for Test-Time Scaling on Code

Anonymous authors
Paper under double-blind review

Abstract

Large reasoning models spend most of their test-time compute on long chains of thought, and on competitive programming the quality of that reasoning—rather than implementation ability—is the dominant bottleneck. Given N sampled candidates per problem, the question is which one to trust. We ask how much of that selection problem is solvable from the reasoning trace *alone*: no code execution, no answer consensus, no token probabilities. We condense each trace into a summary, lift the summary into a reasoning graph whose topology encodes steps, dependencies and case analysis, and train a graph neural network (GNN) to predict correctness from the graph. On LiveCodeBench-v6, under a leakage-free 10-fold cross-test protocol in which every problem is scored only by models that never saw it during training or checkpoint selection, per-fold ensembles recover 0.29 of the $\mathrm{pass@1} \to \mathrm{pass@64}$ gap ($+9.5$ points absolute over random selection), against 0.12 for a frontier judge reading the same traces and 0.16 for a topology-free MLP over the same summary embeddings; single models recover 0.19, so seed-ensembling—effective here because a small training set and aggressive dropout yield weakly correlated models—accounts for a third of the effect, and the worst of the ten folds still recovers 0.16. A comparison of seven checkpoint-selection rules shows that selecting on a smoothed validation curve dominates raw peak selection in both mean and worst-case usefulness. As a transfer example, on a harder LiveCodeBench-Pro subset the classifier–selector cascade recovers 0.22 ± 0.04. We then characterize the ceiling of reasoning-only selection: 94.5% of incorrect candidates textually claim success, and most residual errors are near-ties where candidates share an algorithm and differ only in implementation details absent from the trace—so much so that the same judge, held fixed, recovers three times as much of the gap from the candidates' code as from their reasoning.

1 Introduction

Recent gains in large language model (LLM) performance are tightly coupled with the compute spent on inference-time reasoning (Snell et al., 2024), and in particular with the long chains of thought (Long CoT) that emerge from reinforcement-learning post-training (Shao et al., 2024; Guo et al., 2025). What matters, however, is the *quality* of that reasoning rather than its raw length: lengthening CoT or adding review steps does not reliably improve accuracy and can hurt it, while structural properties of the trace are far more predictive of correctness (Feng et al., 2025; Chen et al., 2026b). Faithfulness studies make the same point from another angle—models frequently reach correct answers through incorrect reasoning, so answer accuracy and reasoning correctness diverge (Nguyen et al., 2024). Competitive programming sharpens the effect: an olympiad-medalist study of LiveCodeBench Pro finds that frontier models succeed on implementation-heavy problems but fail where intricate algorithmic reasoning and case analysis are required, often producing confidently incorrect justifications (Zheng et al., 2025).

Parallel test-time scaling generates N candidate solutions per problem and must then select one, and in the competitive-code regime the usual selectors are weak or inapplicable. Majority voting and self-consistency (Wang et al., 2023) presuppose a small, countable answer space; confidence-

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.

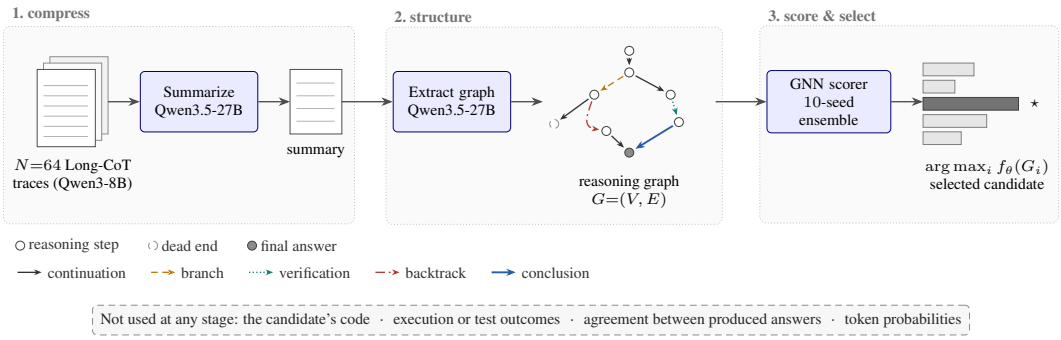

Figure 1: The selection pipeline. Each of the N Long-CoT traces sampled for a problem is condensed by Qwen3.5-27B into a normalized summary and lifted by the same model—a second, shorter pass over the summary—into a typed reasoning graph, which a seed-ensembled two-layer GINE network scores. A recovery edge originates at the last valid step before an abandoned branch, never at the dead end itself.

and logit-based scoring (Kang et al., 2025) is weakly calibrated precisely on the hard instances that matter; execution-based selection (Cho et al., 2026; Li et al., 2025) depends on trusted tests and raises safety concerns when running untrusted code; and generative judges (Toshniwal et al., 2025; Qin et al., 2026) become brittle and expensive as traces grow to tens of thousands of tokens.

We therefore ask a deliberately constrained question: *how much of the selection problem is solvable from the reasoning trace alone?* We exclude, by design, every other signal—no code execution, no consensus over outputs, no token probabilities—so that the measured effect isolates the informativeness of the CoT itself. Concurrent work supports this view, recasting Long CoT as a graph whose structural statistics outpredict length and confidence (Feng et al., 2025) and showing that effective Long CoT has a stable topology emerging from post-training rather than keyword imitation (Chen et al., 2026a). Our pipeline (Figure 1) summarizes each candidate's reasoning, lifts the summary into a reasoning graph whose nodes are reasoning units and whose edges encode dependencies and control flow, and trains a GNN on these graphs to score candidates, selecting the top-scored one per problem.

Why constrain the input at all? The candidate's code is available at selection time, and a judge shown it recovers a larger share of the gap than any reasoning-only method we test. Three considerations motivate the constraint anyway. First, *reasoning correctness is not answer correctness*: a signal read off the artifact reports only whether the output is right, whereas a signal in the trace speaks to the reliability of the process that produced it—which is what one needs when the artifact is hard to verify, and all one has when there is none. Second, *the informativeness of Long CoT is itself the object of study*: what is extractable from a trace lower-bounds what the trace carries, and holding the reader fixed while varying only its input turns the comparison into an experiment about *where the selection signal lives*. Third, *no frontier judge sits in our loop*: side-by-side judging draws on capabilities well beyond the mid-sized models we study, and we run it only to bound what each input affords. We ask how far a model's own reasoning can be trusted without a stronger evaluator—the setting that matters when reasoning is generated at scale and verifying the artifact is costly, unsafe or impossible—so the scorer is a minimal *critic for reasoning models*.

The scale asymmetry follows. Our pipeline is a 27B model run twice per candidate—once over the trace, once over the much shorter summary—a 22M sentence encoder, and a GNN small enough to train a hundred times over; both LLM passes are amortized, since the resulting graph can be rescored at negligible cost. Because selection quality is a discrete, high-variance statistic, we also adopt an evaluation protocol we found necessary for honest conclusions: a leakage-free 10-fold cross-test scheme in which every problem is scored only by models that never saw it, pooled estimates with per-fold dispersion, and a comparison of checkpoint-selection rules. Data, not capacity, is the binding constraint: the sample size of a group-wise ranker is the number of *problems*, and producing N long traces for each is expensive.

Contributions

- We formulate candidate selection for competitive code as a *reasoning-topology* prediction problem: a reasoning → summary → graph pipeline and a GNN selector that uses only reasoning structure—no execution, no consensus, no token probabilities.
- Under a leakage-free 10-fold cross-test protocol on LiveCodeBench-v6, per-fold ensembles recover $\mathcal{U} = +0.290$ of the pass@1 → pass@64 gap over 392 groups ($+9.5$ points absolute), worst fold $+0.155$, single models $+0.193 \pm 0.040$; a topology-free MLP over the same embeddings recovers $+0.163$, placing half the effect in the graph structure. Frozen and moved to olympiad problems (LiveCodeBench-Pro), the cascade still recovers $+0.22 \pm 0.04$.
- We establish a cost–quality reference point: a pipeline whose largest component is a 27B model, and whose per-candidate work is reused, recovers roughly 2.5 times what a frontier judge extracts from the same traces and four fifths of what it extracts from the candidates' code.
- We compare seven checkpoint-selection rules: smoothed validation selection dominates raw peak selection in ensemble usefulness ($+0.290$ vs. $+0.252$) and worst-fold robustness ($+0.155$ vs. $+0.078$), while loss-based selection can collapse below random. Seed-ensembling is the single largest lever, lifting usefulness from $+0.193$ to $+0.290$.
- We characterize the information ceiling of reasoning-only selection: 94.5% of incorrect candidates textually claim success, 79% of residual errors are near-ties, and the separating information has moved into implementation details that the reasoning graph never contains.

2 Related Work

Reasoning quality and structure predict correctness. Feng et al. (2025) build a graph view of CoT and find that the Failed-Step Fraction—the share of steps in abandoned branches—outpredicts length and review ratio; Chen et al. (2026b) report the same for a reasoning-effort signal, and Chen et al. (2026a) map Long CoT to a "molecular" bond topology emerging from post-training rather than keyword imitation. Faithfulness analyses add that models often reach correct answers through incorrect reasoning (Nguyen et al., 2024), and on competitive code Zheng et al. (2025) attribute failures to reasoning rather than implementation. Unlike these, we learn the structural signal end-to-end and apply it to selection.

Test-time scaling and selection. Test-time scaling improves accuracy by spending more inference compute (Snell et al., 2024). Among N candidates, GenSelect lets an LLM reason its way to the best of N (Toshniwal et al., 2025), V_1 uses pairwise self-verification (Singh et al., 2026), RSA aggregates using the reasoning chains rather than only final answers (Venkatraman et al., 2025), and DAJ trains a reasoning judge for Best-of-N on code (Qin et al., 2026). These consume full traces at frontier-model cost; we compress each trace to a graph and never feed the raw trace to a judge. Execution-grounded selectors remain strong baselines—S* (Li et al., 2025) and Symbolic Equivalence Partitioning (Cho et al., 2026)—while reward-free confidence selection scores candidates from output distributions (Kang et al., 2025); all rely on trusted tests or surface confidence, and our signal requires neither. Finally, Graph of Thoughts models reasoning as a graph of thoughts and dependencies (Besta et al., 2024) and recent work couples GNNs with reasoning graphs to steer generation (Gao et al., 2025); we use graph structure not to steer generation but to *select* among completed candidates.

3 Reasoning Evaluation

Summarization. Long-CoT traces are verbose and exceed practical context limits downstream. We map each trace to a normalized summary that preserves the logical skeleton—claims, sub-goals, case splits and the dependencies among them—while discarding restated context. We use Qwen3.5-27B, chosen for effective context length: our traces span 20–30K tokens, at the edge of the native 32K window of comparable-size predecessors such as Qwen3-32B (128K only via YaRN extrapolation), whereas Qwen3.5-27B supports 262K natively. In a manual audit, mid-size models with

shorter native windows systematically dropped steps from the middle of long traces, corrupting the downstream topology; since the graph is only as faithful as the summary, summarizer reliability at this context length is a prerequisite rather than an implementation detail. Prompts are in Appendix B.

Reasoning graph construction. The same Qwen3.5-27B model then lifts each summary into a reasoning graph $G = (V, E)$: nodes are reasoning steps, edges logical transitions between them. This second pass is cheap—it reads the summary, not the trace. Node features concatenate structural attributes with a text embedding of the step (all-MiniLM-L6-v2 (Reimers & Gurevych, 2019), 384-dimensional, 22M parameters), exposing branching, depth, abandoned branches and the presence and resolution of case analysis—signals shown elsewhere to predict correctness (Feng et al., 2025; Chen et al., 2026a) and invisible to bag-of-tokens or confidence-based selectors. Each node is additionally flagged as a dead end or as the final answer where applicable, and each edge is labelled with one of five relation types, which the scorer consumes through the message-passing operator below. Graphs are then validated against the topological rules of Appendix A, and candidates whose graph fails to parse or to validate are discarded. The extraction procedure and the node and edge typology are given in Appendix A.

Selector architecture. Each node carries 409 features: seven attributes of the step, eighteen structural attributes computed from the graph, and the 384-dimensional MiniLM embedding of its `trigger` and `summary` text. A linear encoder with batch normalization projects them to width 128; edge relations are embedded to the same width. Two GINE layers (Hu et al., 2020) with learnable ϵ and residual connections,

$$m_v^{(l)} = (1+\epsilon)h_v^{(l)} + \textstyle\sum_{u\in\mathcal{N}(v)} \mathrm{ReLU}\big(h_u^{(l)} + w_{uv}\big),$$
$$h_v^{(l+1)} = h_v^{(l)} + \sigma\,\mathrm{MLP}^{(l)}\big(m_v^{(l)}\big),$$

where σ is batch normalization, ReLU and dropout $p = 0.4$, are read out by concatenating sum- and max-pooling over nodes, the same over edge representations formed from $[\,w_{uv} \| h_u \| h_v\,]$, and an encoding of a graph-level vector; a layer-normalized MLP maps the concatenation to two logits, whose softmax gives $f_\theta(G) \in [0, 1]$. The graph-level vector holds the problem's difficulty tag and cross-sample features comparing a candidate with its siblings—computed from their reasoning graphs alone, never their code, output or answer, so the constraint we impose is preserved. The aggregate topological statistics the pipeline computes are not model inputs: whatever the scorer knows about topology it reads off the graph itself (Table 6).

Three choices deserve comment. *Pooling sums and the head normalizes with LayerNorm rather than BatchNorm*: sum-pooling makes the representation scale with the number of reasoning steps, and batch normalization would standardize that away across a batch, so the size and detail of a derivation are retained deliberately as a signal—with a flip side we return to below, where a long, dead-end-rich derivation outscores a short correct one. *Depth is two*: reasoning graphs are small (the two examples in our analysis have 6 and 18 nodes), so two rounds of message passing already mix information across most of the graph, and deeper stacks failed to beat the baseline. *Dropout of* 0.4 is aggressive for a network this size and is deliberate: with a small training set it leaves many equally good fits, so independently seeded models err on different problems—the property that makes seed-ensembling our largest lever.

Within a group of N candidates we select $\arg\max_i f_\theta(G_i)$. Each deployed scorer is a *seed-ensemble*: several models trained on the same split from different seeds—varying initialization, data order and dropout mask—whose probabilities are averaged before the argmax. For transfer, our system is a cascade of two components sharing this architecture and differing only in training data and objective: a pointwise correctness *classifier* and a group-wise ranking *selector*.

Checkpoint selection. Group-level pass@1 is discrete and noisy across epochs, so picking the checkpoint at its raw validation peak (BEST-PASS) tends to select lucky outliers. We compare seven rules that either smooth the signal or restrict selection to epochs where the ranking has stabilized: BEST-SMOOTH (peak of the pass@1 curve under a three-checkpoint moving average; our default), BEST-τ and BEST-τ-MRR (best pass@1 resp. MRR inside the zone where the inter-epoch Kendall-τ reaches 0.90), and BEST-MRR, BEST-MARGIN and BEST-LOSS.

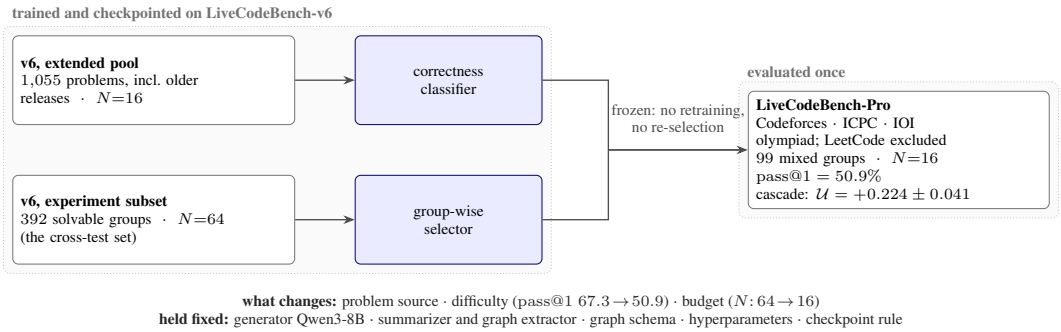

Figure 2: The transfer setting. Both components are trained and checkpointed inside LiveCodeBench-v6—the classifier on an extended pool reaching back to older releases at N=16, the selector on the 392-group cross-test subset at N=64—then frozen and applied to LiveCodeBench-Pro.

4 Dataset

Sources and candidate generation. All candidates are sampled from Qwen3-8B (Yang et al., 2025) and labeled against the benchmark tests; each is paired with its trace, summary and graph. Training data come from LiveCodeBench-v6 (Jain et al., 2024) at two budgets (Figure 2). The correctness classifier is trained on an extended pool of 1,055 problems—reaching back to older releases of the benchmark—at $N = 16$ candidates each. The selector is trained on the same subset on which all our in-domain experiments are run: 454 problems resampled at $N = 64$. Because it ranks within a group it is trained only on *mixed* groups, where at least one candidate passes and at least one fails; evaluation uses all 418 *solvable* groups (at least one correct candidate), of which we drop 26 where extraction failed for some candidate—an empty summary, unparseable JSON, or a graph rejected by validation: the graph is then undefined, and we discard the whole group rather than the candidate, so every retained group keeps its full $N = 64$ and pass@64 is defined identically throughout. These failures are sporadic and leave the difficulty distribution unchanged. The resulting 392 groups are the single evaluation set used for our selector and for every baseline.

Transfer split: a different problem population. Our held-out evaluation changes the *problem population*, not merely the difficulty. LiveCodeBench-v6 mixes LeetCode, AtCoder and Codeforces and is dominated by interview-style and short-contest tasks; 436 of its problems are LeetCode material with a fixed function signature. LiveCodeBench Pro (Zheng et al., 2025) is assembled from olympiad-level contests—Codeforces rounds, ICPC and IOI—and excludes LeetCode by construction, as both easier and more exposed to contamination. The shift therefore has three components: problem source, difficulty (random pass@1 falls from 67.30% to 50.92%) and candidate budget ($N = 64 \rightarrow 16$). Everything else is fixed—generator, summarizer and graph extractor, graph schema, every hyperparameter and the checkpoint rule, all chosen inside v6. We use 99 Pro problems at $N = 16$, restricted to *mixed* groups (1 to 15 correct of 16), so the oracle is 100% and selection is non-trivial everywhere. No Pro data enters training or checkpoint selection.

5 Experiments

Evaluation metric. Let pass@1 be the expected accuracy of a single random candidate ("random") and pass@N the oracle accuracy (Chen et al., 2021). For a selector achieving acc_{sel}, the gap-closing *usefulness* is $\mathcal{U} = (\text{acc}_{\text{sel}} - \text{pass@1})/(\text{pass@}N - \text{pass@1})$, so $\mathcal{U} = 0$ matches random selection and $\mathcal{U} = 1$ the oracle.

Cross-test protocol. All in-domain results follow a leakage-free 10-fold cross-test scheme over the 392 solvable groups. Each fold fixes a disjoint 10% of the groups as its test split and partitions the rest once into train and validation (70/20), so every group is a test group in exactly one fold; folds are formed by grouped stratified K-fold, so groups are never split and the share of correct candidates is balanced across folds. Within a fold we train 10 GNNs (100 in total) on that fixed train

Method	Exec. free	Acc. (%)	$\mathcal{U}$ (%)
Random (pass@1)	✓	67.30	0.0
Confidence (DeepSeek)	✓	69.32	6.1
SEP[†] (execution)	×	80.05	15.8
SxS judge[§] (reasoning)	✓	71.17	11.8
MLP, MiniLM embeds.	✓	72.37	15.5
MLP, Qwen3-Emb-0.6B	✓	71.13	11.7
MLP, Qwen3-Emb-8B	✓	72.63	16.3
Ours, single GNN	✓	73.60	19.3
Ours, per-fold ensemble	✓	**76.79**	**29.0**
Oracle (pass@64)	—	100.0	100.0

Table 1: Selection on LiveCodeBench-v6 ($N = 64$, Qwen3-8B), pooled over the 392 solvable groups under the cross-test protocol; each group is predicted by the ensemble of the fold that held it out. "Single GNN" pools one model per fold; the MLP rows are topology-free ablations under the identical protocol. Both judges are DeepSeek V4 Flash: pointwise, and [§]as a knockout tournament over reasoning traces. [†]SEP needs a shared functional interface and uses its own anchors on the 436 LeetCode problems of v6.

split, varying only the random seed, and select each checkpoint on the shared validation split; the fold's ensemble averages the 10 probabilities before the argmax. We report *per-model* usefulness (over 100 models, each on its own test fold), *per-fold ensemble* usefulness, *pooled* usefulness over all 392 groups—each predicted by the ensemble of the fold that held it out—and the *worst fold*, as a measure of robustness to an unlucky split. The single-model figure $+0.193 \pm 0.040$ is the pooled usefulness of a system that uses one model per fold, mean $\pm$ std over 10 such draws; the per-model $\pm$ in Table 2 is instead the dispersion of the 100 individual models.

Main result on LiveCodeBench-v6. Random selection over the 392 solvable groups attains 67.30% against a 100% oracle (Table 1). Our ensembled selector reaches 76.79% pooled, recovering 29.0% of the gap from reasoning structure alone, every prediction coming from an ensemble whose members never saw the group. The confidence-based judge recovers only 6.1%: per-candidate verdict confidence barely separates correct from incorrect solutions. Comparative schemes fare better (Singh et al., 2026); we run the same model as a *side-by-side knockout tournament*, splitting the 64 candidates into eight groups of eight and running a final round over the eight winners, at nine long-context calls per problem. Shown the candidates' *reasoning traces* it recovers 11.8%—less than half of what the reasoning-graph selector extracts from the same information source, at an order of magnitude higher token cost and from a frontier rather than a 27B model. The same tournament applied to the code is treated separately below. The execution-based SEP baseline (Cho et al., 2026) recovers 15.8% against its own anchors: even with access to execution behaviour it closes roughly half of what the reasoning-graph ensemble does.

Topology-free baseline. To separate graph structure from summary semantics, we train an MLP directly on the pooled node embeddings, discarding all edges, under the identical protocol, checkpoint rule and ensembling; the gap to the GNN measures everything the graph adds—connectivity, edge relation types and the size-sensitive read-out—rather than connectivity alone. Pooled over all 392 groups it recovers 15.5% with MiniLM features, 11.7% with Qwen3-Emb-0.6B and 16.3% with Qwen3-Emb-8B, against 29.0% for the reasoning-graph ensemble: semantics account for roughly half of what the full pipeline extracts and structure for the other half. The weakest variant coincides with the reasoning-trace tournament judge (11.8%), which likewise reads content without exploiting structure. Encoder capacity is not the binding constraint, and its effect is not monotonic in size: the 22M MiniLM beats a 0.6B embedder by 3.8 points and is beaten by an 8B one by only 0.8.

Checkpoint-selection rules. Three observations follow from Table 2. First, per-model usefulness barely distinguishes the rules: all except BEST-MARGIN sit in a narrow $+0.18$–0.19 band whose dispersion dwarfs any between-rule difference. Differences emerge only at the ensemble level, where the rules differ in how decorrelated the selected checkpoints' errors are. Second, smoothing dominates raw peaks: BEST-SMOOTH exceeds BEST-PASS by $+0.038$ in ensemble usefulness

Rule	Per-model	Per-fold ens.	Worst
BEST-SMOOTH	$+.193 \pm .171$	$\mathbf{+.290 \pm .105}$	$\mathbf{+.155}$
BEST-τ-MRR	$+.194 \pm .165$	$+.275 \pm .116$	$+.144$
BEST-PASS	$+.184 \pm .150$	$+.252 \pm .114$	$+.078$
BEST-MRR	$+.194 \pm .166$	$+.235 \pm .104$	$+.078$
BEST-LOSS	$+.185 \pm .144$	$+.226 \pm .171$	$-.032$
BEST-MARGIN	$+.129 \pm .177$	$+.212 \pm .128$	$+.001$
BEST-τ	$+.193 \pm .153$	$+.213 \pm .106$	$+.067$

Table 2: Checkpoint-selection rules under the cross-test protocol ($\mathcal{U}$). Per-model: mean $\pm$ std over 100 models, each on its own test fold. Per-fold ens.: the 10-model ensemble, mean $\pm$ std over 10 folds; pooled usefulness over all 392 held-out groups (Table 1) agrees to within 0.002. Worst: worst of the 10 folds.

Method	Runs	Acc. (%)	$\mathcal{U}$
Full cascade	30	61.9 ± 2.0	$+0.224 \pm 0.041$
Selector alone	10	58.99 ± 1.59	$+0.164 \pm 0.033$

Table 3: Transfer to a different problem population: LiveCodeBench-Pro (99 mixed groups, $N = 16$, olympiad problems from Codeforces/ICPC/IOI; random $= 50.92\%$, oracle $= 100\%$). All models are trained and selected on v6 only. Full cascade $=$ correctness classifier $+$ selector; mean $\pm$ std over seeded runs.

and nearly doubles worst-fold robustness, with BEST-τ-MRR close behind—temporal smoothing and rank-stability filtering appear to remove the same failure mode, checkpoints sitting on noisy validation peaks. Third, some natural rules are unsafe: BEST-LOSS produces a fold below random (-0.032), and BEST-MARGIN already loses signal per model.

Seed-ensembling. Averaging the ten models' probabilities within each fold lifts usefulness from $+0.193 \pm 0.040$ to $+0.290$ under BEST-SMOOTH—a larger increase than that of any architectural or loss modification we tried. All ten members are trained on the *same* fixed split, so the gain comes from seed variation, not data coverage. We attribute it to unusually low inter-model correlation: the selector sees very little data, which leaves many equally good fits, and dropout is aggressive ($p = 0.4$), so seeds converge to solutions that err on different problems; averaging before the discrete argmax cancels much of this independent error. Enlarging the training set means generating N Long-CoT candidates for hundreds more problems, which costs more than every other stage combined. Usefulness saturates by roughly ten seeds, so the ensemble is the natural unit of evaluation here.

Cost of selection. Both selectors are dominated by the tokens they read, and both read on the order of the generation budget itself. Producing the $N = 64$ candidates emits roughly $64 \times$ 20–30K tokens of Long CoT per problem. Our pipeline passes each trace once through the 27B model and then re-reads only the summary, about $1/20$ of a trajectory. The tournament judge instead re-reads the candidates it compares: eight group rounds over all 64 traces plus a final round amount to 72 trace-reads over nine long-context passes. The volumes are comparable; the difference is one of *model scale and reusability*. Our reading is done once per candidate by a 27B model on a single accelerator and yields a graph any number of downstream scorers can reuse, whereas the judge's nine passes serve one decision.

Transfer to olympiad problems. On the Pro subset of Figure 2 the frozen cascade retains $\mathcal{U} = +0.224 \pm 0.041$ over 30 seeded runs, the selector alone accounting for $+0.164 \pm 0.033$ (Table 3): quality degrades gracefully rather than collapsing. Two limits: the shift is in problem source, difficulty and N with the generator held fixed, so this is not evidence about transfer across generator families; and the Pro subset is filtered to mixed groups using ground-truth labels, so it is transfer to a filtered subset rather than to the raw benchmark.

Negative results. On a fixed v6 split, none of six ranking losses (listwise baseline, any-positive softmax CE, hard-listwise, RankNet, max-margin contrastive, RR-LambdaRank) differed from the

baseline statistically (e.g. any-positive CE: $+0.318 \pm 0.029$ over 15 runs versus $+0.323 \pm 0.030$, $t = -0.66$), and neither did eight architectural variants—attention pooling, a virtual node, extra layers, alternative edge encodings, difficulty re-weighting either way. Run-to-run noise dominates modification effects at this scale, which motivated the cross-test protocol and seed-ensembling above.

Node embeddings are necessary too. The mirror ablation—keeping the graph and deleting the node embeddings—was run under an earlier fixed-split protocol and evaluated on the Pro split (Appendix D). Removing text features collapses the hard stratum to the random baseline (-0.004 versus $+0.086$) and roughly halves the aggregate ($+0.081$ versus $+0.155$), with every stratum difference highly significant (t between 6.9 and 8.8, $n = 100$); a larger embedder does not help, beating MiniLM on no stratum. Read together with the topology-free MLP, the two ablations bracket the method: neither structure nor semantics alone reproduces the full pipeline.

6 Analysis: the Information Ceiling

The residual gap is informational, not methodological.

Incorrect candidates claim success. 94.5% of candidates that fail the tests nevertheless textually assert success in their reasoning ("verified", "works", "correct"), making them lexically indistinguishable from correct ones. Whatever separates pass from fail must be read from structure—and structure, too, has limits.

Residual errors are near-ties. Of 85 selection errors on mixed groups, 67 (79%) are near-ties, with a score gap below 0.1 between the chosen incorrect candidate and the best correct one; only 6 (7%) are confident errors (gap > 0.3). The model does not confidently prefer wrong reasoning—it cannot separate candidates whose reasoning carries no separating signal. A canonical hard-stratum example: the selected incorrect candidate has an 18-node graph with detailed dead-ends and rechecks (score 0.911), the correct one a 6-node graph (score 0.138), and both derive the *same* two-pointer $O(N^2)$ algorithm; the failure lies in implementation details—overflow, complexity, time limits—that never appear in the reasoning graph. Consistent with this, ranking stability and quality decouple: inter-epoch Kendall-τ rises to 0.98 while pass@1 stays flat.

The residual signal is in the code. Holding the judge fixed and changing only what it reads localizes the remaining gap. The tournament of Table 1, shown the reasoning traces, recovers 11.8%; shown the *code* those traces produced, on the same 392 groups against the same anchors, it recovers 35.3% (78.83% accuracy). With the reader held constant, the artifact carries roughly three times the separating signal of the reasoning that generated it.

Two things follow. About Long CoT: by the time a trace is complete, most of the information separating a correct candidate from an incorrect one has moved into the artifact, which bounds any reasoning-only selector, learned or generative. About systems: our pipeline recovers 2.5 times what the frontier judge recovers from the traces, while on the code the judge recovers a fifth more than our pipeline does without seeing it (35.3% versus 29.0%)—its advantage comes from the input, not the reader. The near-tie analysis explains why no reader of the reasoning can close the remainder, and yields a testable prediction: because near-ties are precisely the cases a code-level reader resolves, the two signals should err on disjoint problems and composing them should exceed either alone. Reasoning-structure selection is thus a cheap, safe, execution-free complement to—not a substitute for—execution-grounded methods (Cho et al., 2026; Li et al., 2025), and more broadly a minimal reasoning critic for settings where executing or judging the artifact is expensive, unsafe or impossible.

7 Limitations

The most consequential limitation is the scope of the empirical study. All candidates come from a single generator, Qwen3-8B, at a single scale. Our transfer experiment varies the problem population, its difficulty and the candidate budget while holding the generator fixed, so it shows that the signal survives a move from interview-style tasks to olympiad problems, not that it survives a different reasoning style. Whether reasoning-graph topology is a property of Long CoT in general or

of the post-training recipe of one model family is the single most important open question here; the structural signals reported independently across other models (Feng et al., 2025; Chen et al., 2026a) are consistent with the former, but that is corroboration by proxy. Generating $N = 64$ candidates per problem dominates the cost of the pipeline, and our compute budget bounded the dataset well before the method did.

Further constraints qualify the results. The selector depends on summarizer quality and on test-based labels, and the summarizer is the one component we do not ablate for scale. The Pro subset is filtered to mixed groups using ground-truth labels. The feature ablations of Appendix D ran under an earlier fixed-split protocol, so their absolute values are not comparable to the pooled figures. At 392 training groups, run-to-run noise dominates architectural effects, and per-fold ensemble estimates carry substantial dispersion (± 0.105). Finally, the approach cannot resolve candidates whose reasoning is equivalent; we have argued for—but not measured—complementarity with code-level features.

8 Conclusion

We asked how much of the candidate-selection problem in parallel test-time scaling is solvable from the reasoning trace alone, and answered it with a reasoning $\rightarrow$ summary $\rightarrow$ graph pipeline and a GNN selector under a leakage-free 10-fold cross-test protocol. Reasoning structure alone recovers 0.29 of the $\mathrm{pass@1} \rightarrow \mathrm{pass@64}$ gap on LiveCodeBench-v6—roughly 2.5 times what a tournament judge extracts from the same traces (0.12) while reading them nine times over—and degrades gracefully when the frozen cascade is moved to olympiad problems from a different contest population (0.22). A topology-free MLP recovers 0.16 under the identical protocol, placing half the effect in the graph structure.

The most informative comparison is with the same judge reading the candidates' *code*, which recovers 0.35: because it recovers only 0.12 from the traces, its advantage measures the input rather than the reader and locates the residual signal in exactly the artifact our setting excludes—and a pipeline whose largest component is a 27B model reaches four fifths of it without ever seeing the code. Two protocol-level findings, smoothed checkpoint selection and seed-ensembling, proved larger levers than any architectural or loss modification we tried. Reasoning-structure selection is therefore best seen as a cheap, safe, execution-free component to be composed with execution-grounded selection, and as a minimal critic for reasoning models: an estimate of how far a chain of thought can be trusted before any artifact is executed or judged.

AI use statement

In this work, we used generative AI tools to assist in implementing the methods. The code was written by the authors with AI assistance for individual functions and code completion, and the final codebase was refactored with a generative AI tool to locate bugs, remove dead and commented-out code and make the code maintainable. The summarization and graph-extraction prompts were written by the authors; a generative AI tool was used to improve their wording and structure, and the final prompts are reproduced verbatim in Appendix B. The refactoring targeted readability; we re-ran the pipeline on the refactored codebase and reproduced the reported selection metrics up to seed-level variation. We have not used generative AI tools to generate synthetic data sets, develop the conceptual framework, formulate or prove mathematical claims, propose or refine hypotheses, design the methodology or experiments, clean or reformat data, or interpret results; translation and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to edit and polish the writing, to search for and identify related work, and to check the references and the formatting of the manuscript. The language models that appear in the paper (the candidate generator, the summarizer and graph extractor, and the judge baselines) are components or objects of the study and are described in the corresponding sections. We have reviewed all AI-assisted work: AI-assisted code and prompts were read, tested and finalized by the authors, and every reported number comes from runs the authors executed and inspected; every AI-suggested reference was checked against the original publication; all AI-edited text was verified by the authors for accuracy. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. URL https://arxiv.org/abs/2308.09687.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Qiguang Chen, Yantao Du, Ziniu Li, Jinhao Liu, Songyao Duan, Jiarui Guo, Minghao Liu, Jiaheng Liu, Tong Yang, Ge Zhang, Libo Qin, Wanxiang Che, and Wenhao Huang. The molecular structure of thought: Mapping the topology of long chain-of-thought reasoning. *arXiv preprint arXiv:2601.06002*, 2026a. URL https://arxiv.org/abs/2601.06002.

Wei-Lin Chen, Liqian Peng, Tian Tan, Chao Zhao, Blake JianHang Chen, Ziqian Lin, Alec Go, and Yu Meng. Think deep, not just long: Measuring LLM reasoning effort via deep-thinking tokens. *arXiv preprint arXiv:2602.13517*, 2026b. URL https://arxiv.org/abs/2602.13517.

David Cho, Yifan Wang, Fanping Sui, and Ananth Grama. Inference-time code selection via symbolic equivalence partitioning. *arXiv preprint arXiv:2604.06485*, 2026. URL https://arxiv.org/abs/2604.06485.

Yunzhen Feng, Julia Kempe, Cheng Zhang, Parag Jain, and Anthony Hartshorn. What characterizes effective reasoning? revisiting length, review, and structure of CoT. *arXiv preprint arXiv:2509.19284*, 2025. URL https://arxiv.org/abs/2509.19284.

Hang Gao, Chenhao Zhang, Tie Wang, Junsuo Zhao, Fengge Wu, Changwen Zheng, and Huaping Liu. Learn to think: Bootstrapping LLM logic through graph representation learning. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 8058–8067, 2025. URL https://arxiv.org/abs/2505.06321.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/1905.12265.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. URL https://arxiv.org/abs/2403.07974.

Zhewei Kang, Xuandong Zhao, and Dawn Song. Scalable best-of-N selection for large language models via self-certainty. *arXiv preprint arXiv:2502.18581*, 2025. URL https://arxiv.org/abs/2502.18581.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15964–15978, 2025. URL https://arxiv.org/abs/2502.14382.

Minh-Vuong Nguyen, Linhao Luo, Fatemeh Shiri, Dinh Phung, Yuan-Fang Li, Thuy-Trang Vu, and Gholamreza Haffari. Direct evaluation of chain-of-thought in multi-hop reasoning with knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2862–2883, 2024. URL https://arxiv.org/abs/2402.11199.

Peijia Qin, Ruiyi Zhang, Qi Cao, and Pengtao Xie. DAJ: Data-reweighted LLM judge for test-time scaling in code generation. *arXiv preprint arXiv:2601.22230*, 2026. URL https://arxiv.org/abs/2601.22230.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019. URL https://arxiv.org/abs/1908.10084.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Harman Singh, Xiuyu Li, Kusha Sareen, Monishwaran Maheswaran, Sijun Tan, Xiaoxia Wu, Junxiong Wang, Alpay Ariyak, Qingyang Wu, Samir Khaki, Rishabh Tiwari, Long Lian, Yucheng Lu, Boyi Li, Alane Suhr, Ben Athiwaratkun, and Kurt Keutzer. V_1: Unifying generation and self-verification for parallel reasoners. *arXiv preprint arXiv:2603.04304*, 2026. URL https://arxiv.org/abs/2603.04304.

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. URL https://arxiv.org/abs/2408.03314.

Shubham Toshniwal, Ivan Sorokin, Aleksander Ficek, Ivan Moshkov, and Igor Gitman. GenSelect: A generative approach to best-of-n. In *2nd AI for Math Workshop @ ICML*, 2025. URL https://arxiv.org/abs/2507.17797.

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan Obando-Ceron, Yoshua Bengio, Brian R. Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, Nikolay Malkin, and Moksh Jain. Recursive self-aggregation unlocks deep thinking in large language models. *arXiv preprint arXiv:2509.26626*, 2025. URL https://arxiv.org/abs/2509.26626.

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. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2203.11171.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Zihan Zheng et al. LiveCodeBench Pro: How do olympiad medalists judge LLMs in competitive programming? *arXiv preprint arXiv:2506.11928*, 2025. URL https://arxiv.org/abs/2506.11928.

A REASONING-GRAPH CONSTRUCTION

Extraction procedure. Extraction runs in two LLM phases, both served from the same Qwen3.5-27B endpoint behind an OpenAI-compatible vLLM server, with thinking enabled, temperature 1.0, top-p 0.95, a generation budget of $65{,}536$ tokens and 80 concurrent requests; work is sharded across nodes and merged at file barriers. *Phase 1* condenses the raw trace into a numbered, chronological "cognitive step-by-step breakdown" (Figure 3). The prompt compresses explanations but forbids dropping topology: pivot and trigger phrases such as "Wait, let me rethink" or "Alternatively" must be carried over verbatim, abandoned paths must be summarized rather than deleted and explicitly marked, and verification steps must be retained. *Phase 2* reconstructs the logical topology of that list (Figure 4), under rules that push against chronological chaining—the prompt states that a straight line $1 \to 2 \to 3$ is to be resisted unless a strict derivation forces it, and penalizes edges whose justification is mere adjacency.

Phase 2 emits a single JSON object with a `root_node_id`, a list of `nodes` and a list of `edges`. A node records a short `trigger` label—often quoting the phrase in the trace that opens the step, such as a pivot or a doubt—a one-sentence `summary` of the step, and the two boolean flags `is_dead_end` and `is_final_answer`. An edge records a `source` list, a `target`, a `relation` drawn from the closed vocabulary of Table 5, and a free-text `justification`. Two properties of the schema matter downstream. `source` is a list, so a step may depend on several predecessors and a single record can expand into several directed edges. And abandoned reasoning is represented explicitly rather than deleted: a dead-end node hangs off the chain with its flag set, and the recovery edge originates at the node *preceding* it, so the surviving path and the abandoned one are both visible in the topology. Outputs are then parsed—a leading Markdown fence is stripped if present—and validated against the topological rules the prompt states: exactly one root, whose in-degree is zero; exactly one node flagged `is_final_answer`; every non-root node with at least one incoming edge; dead ends terminal, with out-degree zero; no `backtrack_recovery` edge originating at a dead end; every node either a dead end, the final answer, or on a path to one; every relation drawn from Table 5; every node reachable from the root; and the whole graph acyclic. Candidates whose output fails to parse or fails validation are dropped before training.

Node typology and structural attributes. Nodes are not drawn from a closed vocabulary: the `trigger` label is free text, and the categorical information the scorer sees is carried by the two boolean flags of Table 4 together with the degree and position statistics computed from the graph itself.

Field	Role
`trigger`	Short label for the step, often quoting the phrase that opens it ("Pivot", "Dead End Discovery")
`summary`	One-sentence restatement of the step; the text that is embedded
`is_dead_end`	The step is abandoned rather than built upon
`is_final_answer`	The step closes the derivation
`root_node_id`	Graph-level; marks the entry step

Table 4: The extracted node schema. Structural node attributes s_v are formed from the two flags together with graph statistics.

The text that is embedded is `trigger` and `summary` joined into a single string, encoded with all-MiniLM-L6-v2 and L_2-normalized; identical strings are deduplicated and cached across the corpus, so the encoder runs once per distinct step text rather than once per node. Each node additionally carries eighteen structural attributes, all normalized to $[0, 1]$: depth from the root, in- and out-degree, indicators for branching and merging, indicators for an outgoing `verification` and an incoming `backtrack_recovery`, closeness to the final node, whether every path from the node leads to a dead end, position in the step list, distance to the nearest dead end, the longest path through the node, whether the node lies on a critical root-to-final path, the share of dead ends in its subtree, and a four-step random-walk positional encoding. Seven further base attributes complete the 25 non-embedding dimensions.

Edge typology. Every edge carries one of the five relations of Table 5, embedded and consumed by the GINE message-passing operator. The accompanying `justification` string is retained in the released graphs but is not encoded: `use_edge_text_embeddings` is false, so no edge text reaches the network.

Relation	Meaning
`logical_continuation`	The target follows directly from the source
`branch_alternative`	An alternative line is taken because the source is rejected or too costly
`verification`	The target checks the source against examples or rederives it
`backtrack_recovery`	The chain resumes from an ancestor after a dead end
`conclusion`	The target closes the derivation

Table 5: The five edge relation types.

B PROMPTS

All prompts are reproduced verbatim; placeholders in angle brackets are substituted at run time.

```
You are an Expert Cognitive Analyst. Your task is
to deeply analyze a raw Chain-of-Thought (CoT)
text and summarize it into a "Cognitive
Step-by-Step Breakdown" without losing ANY of its
logical topology (branches, dead ends,
verifications, or backtracks).

YOUR OBJECTIVE
Compress verbose explanations and calculations
into concise summaries, BUT strictly preserve the
exact chronological sequence of every cognitive
shift.

WHAT YOU MUST PRESERVE:
1. All Pivot/Trigger Phrases: If the original text
says "Wait, let me rethink", "Alternatively",
"Let's check this", or "This is wrong", you MUST
include these exact phrases in your summary for
that step.
2. Dead Ends: Do not delete abandoned paths. They
are crucial. Summarize what the path was and
explicitly note that it hit a dead end.
3. Verifications: Summarize any backward-checking
or validation steps.

OUTPUT FORMAT
Return a numbered, chronological list of cognitive
steps
```

Figure 3: Phase 1: the prompt that condenses a Long-CoT trace into a normalized summary. The trace is prepended to it.

```
You are an Expert Cognitive Graph Architect. Your task is to take a "Cognitive Step-by-Step
Breakdown" (a summarized chronological list of thoughts) and reconstruct their TRUE logical
topology into a Directed Graph.

YOUR OBJECTIVE
Connect these sequential steps based on STRICT LOGICAL DEPENDENCE, not naive chronological
order. Human and AI reasoning branches, hits dead ends, and backtracks. Your graph must
reflect this true cognitive geometry.

TOPOLOGICAL GRAPH RULES (CRITICAL):
1. DEFAULT TO NON-LINEARITY: Actively resist the urge to create a straight line
(1->2->3->4). Assume the reasoning is non-linear unless a strict logical derivation
absolutely forces a linear step. Always ask: "Does this step truly build on the immediate
predecessor, or does it refer back to an earlier step?"
2. SINGLE ENTRY POINT: The graph MUST have exactly one root node (usually Step 1).
3. DEAD ENDS ARE TERMINAL: If a step is marked as a "Dead End", it MUST have an out-degree
of 0. NO subsequent steps can logically originate from it.
4. LOGICAL BACKTRACKING: If Step N is a Dead End, and Step N+1 starts a new approach, Step
N+1 MUST NOT reference Step N. Link Step N+1 to the most recent valid node where the branch
actually started.
5. CONVERGENCE: Multiple paths can converge if a verification step checks multiple previous
branches, or if the final conclusion aggregates previous findings.
6. NO HANGING NODES: Every node MUST satisfy one of these conditions:
It is marked as "is_dead_end: true" (terminal node on abandoned path), OR
It is marked as "is_final_answer: true" (there must be EXACTLY ONE such node), OR
It has at least one outgoing edge that eventually leads to the final answer node.
Verification steps are NOT side steps - they are linear progression steps that contribute
to the critical path and must be connected accordingly.

EDGE RELATION TYPES TO USE:
- "logical_continuation": Target is a direct mathematical/logical next step of Source.
- "branch_alternative": Target proposes a new method, branching off from Source.
- "backtrack_recovery": Target recovers from a previous dead end, resuming from Source.
- "verification": Target verifies or tests the claim made in Source.
- "conclusion": Target is the final answer derived from Source.

OUTPUT FORMAT
Provide ONLY valid JSON. Do not include markdown formatting. Use this exact schema:
{
    "root_node_id": <integer>,
    "nodes": [
      {
        "id": <integer, matching the Step N number>,
        "trigger": "<the exact trigger phrase provided in the input summary>",
        "summary": "<brief 1-sentence summary of the Action/Thought>",
        "is_dead_end": <true if state is Dead End, false otherwise>,
        "is_final_answer": <true if this is the final conclusion, false otherwise>
      }
    ],
    "edges": [
      {
        "source": <array, id of the nodes this logically stems from>,
        "target": <integer, id of the current node>,
        "relation": "<logical_continuation | branch_alternative | backtrack_recovery |
                     verification | conclusion>",
        "justification": "<One short sentence explaining why Target logically depends on
                          Source. You will be heavily penalized for lazy chronological
                          chaining (e.g., Target N referencing Source N-1 without a
                          genuine dependency).>"
      }
    ]
 }
```

Figure 4: Phase 2: the prompt that lifts a normalized summary into a reasoning graph. The summary is prepended to it. Line breaks inside the schema are ours, for width.

C Implementation Details

Listing 1: The deployed selector configuration, verbatim. One run corresponds to one (fold, seed) pair; the fold assignment is supplied externally so that the cross-test splits are identical across seeds and across ablations. The correctness classifier of the transfer cascade shares this configuration and differs in its training pool and in being trained pointwise.

```
arch: gine              hidden_dim: 128
num_layers: 2           dropout: 0.4
pool_mode: add          use_edge_pool: true
use_meta: true          use_cls_prob_features: false
use_text_embeddings: true
text_embedding_model: all-MiniLM-L6-v2
text_embedding_dim: 384
use_edge_text_embeddings: false
use_cross_sample_features: true
use_semantic_cross_sample: true

lr: 3e-4                weight_decay: 1e-4
batch_size: 64          num_epochs: 80
grad_clip: 1.0          min_delta: 1e-4
early_stop_metric: pass_at_1
early_stop_patience: 40
tau_converged_threshold: 0.9

use_listwise_loss: true     use_pairwise_loss: false
listwise_mode: listwise     listwise_alpha: 0.85
listwise_temperature: 1.0   listwise_margin: 0.5
listwise_warmup_epochs: 3
listwise_pos_neg_balance: true
listwise_entropy_weight: true
listwise_difficulty_weight: false
filter_mixed_groups_train: true
use_class_weights: false    use_grouped_sampler: false
use_swa: false
```

Objective. Write s_i for the logit of the correct class assigned to candidate i and P_g for the positives in group g. The listwise term is a group-level softmax cross-entropy with a uniform target over the positives,

$$L_{\text{list}}(g) = -\frac{1}{|P_g|} \sum_{i \in P_g} \log \frac{\exp(s_i/T)}{\sum_{j \in g} \exp(s_j/T)},$$

so all correct candidates are pulled above all incorrect ones in the same group; the deployed run uses $T = 1$. Two per-group weights modulate it. Positive/negative balancing multiplies the group loss by $n_g/(2|P_g|)$, which removes the bias towards groups with many positives. Entropy weighting multiplies it by $H(p_g)/\ln 2$ with $p_g = |P_g|/n_g$, so a balanced $8/8$ group carries weight 1 while a $1/15$ group carries ≈ 0.34; nearly trivial groups are thereby discounted. The final objective is $\alpha L_{\text{list}} + (1-\alpha) L_{\text{CE}}$ with $\alpha = 0.85$, the pointwise cross-entropy retaining a calibration signal that the group-wise term alone does not provide. Five further group modes (hard-listwise, soft top-1, RankNet, margin contrastive, RR-LambdaRank) are implemented and are the losses compared in the negative results.

Compute. Candidate generation and both LLM extraction phases run on a single node of eight Ascend 910B accelerators with 64 GB of device memory each, under CANN 9.0, with the model served through vLLM-ascend 0.21. Training the GNN itself is inexpensive by comparison: the network is small enough to train on CPU, so the full 10×10 cross-test grid is a negligible fraction of one summarization pass over the corpus. Both judge baselines query DeepSeek V4 Flash at the provider's default decoding settings.

D Node-Embedding Ablation

Each node carries a text embedding, raising a confound: the GNN could be predicting correctness from step semantics rather than topology. The topology-free MLP removes structure; here we remove semantics instead. These runs use an earlier fixed-split protocol (100 seeded runs trained on

Component	Value
Node features	7 base + 18 structural + 384 MiniLM = 409
Graph-level vector	one-hot difficulty tag + cross-sample features; aggregate graph statistics unused
Edge features	learned relation-type embedding (5 types)
Node encoder	linear → BatchNorm, width 128
Message passing	2× GINE, width 128, learnable ϵ
	inner MLP linear-BN-ReLU-linear, 128→256→128
	BatchNorm, ReLU, dropout, residual per layer
Neighbour aggregation	sum
Dropout	0.4
Read-out	node sum ‖ node max ‖ edge sum ‖ edge max ‖ meta
Head	LayerNorm → MLP → 2 logits
Score	softmax prob. of the correct class
Trainable parameters	all; text embeddings precomputed
Loss	$\alpha L_{\text{list}} + (1-\alpha)L_{\text{CE}}$, α=0.85, T=1
	pos/neg balancing, entropy weighting
	3 warm-up epochs, no difficulty re-weighting
Training groups	mixed groups only
Optimizer / LR	AdamW / 3×10^{-4}
Scheduler	cosine annealing, warm restarts (T_0=10, T_{mult}=2)
Weight decay	10^{-4}
Batch size / epochs	64 / 80
Gradient clipping	1.0
Early stopping	pass@1, patience 40, δ=10^{-4}
Seeds per fold	10
Checkpoint rule	BEST-SMOOTH
Hardware	8× Ascend 910B, single node

Table 6: Selector configuration. The same architecture is used for the pointwise correctness classifier of the transfer cascade.

Stratum	MiniLM	Qwen3-Emb-0.6B	No-emb.
Easy	$+0.128 \pm .101$	$+0.111 \pm .117$	$+0.094 \pm .083$
Medium	$+0.263 \pm .123$	$+0.224 \pm .093$	$+0.120 \pm .107$
Hard	$+0.086 \pm .086$	$+0.077 \pm .097$	$-0.004 \pm .098$
All	$+0.155$	$+0.134$	$+0.081$

Table 7: Node-embedding ablation on the transfer split (LiveCodeBench-Pro, N=16): usefulness $\mathcal{U}$ under three node featurizations—the default MiniLM (384-dim), the larger Qwen3-Emb-0.6B (1024-dim), and structural attributes only. Mean ± std over 100 seeded runs.

v6), so absolute values are not comparable to the pooled cross-test figures, and we report them on the transfer split because in-domain metrics under that protocol are distorted by checkpoint selection.

Two conclusions follow. First, text node embeddings are *necessary*: removing them collapses the hard stratum to the random baseline (-0.004 versus $+0.086$) and roughly halves the aggregate ($+0.081$ versus $+0.155$), with every stratum difference highly significant (t between 6.9 and 8.8, $n = 100$). Reasoning topology alone is not sufficient out of distribution: semantic node features become load-bearing precisely where the graph statistics shift. Read with the topology-free MLP, the two ablations bracket the method—neither structure nor semantics alone reproduces the full pipeline. Second, a larger embedder does not help: raising dimensionality from 384 to 1024 beats MiniLM on no stratum and is significantly worse on medium (-0.039, $t = -2.54$). We keep the 384-dim MiniLM as the default.