DISTILL BEFORE YOU MEASURE: DUAL PRUNING FOR EVALUATION-EFFICIENT GPU KERNEL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

GPU kernel optimization requires repeated compilation, correctness checking, and performance measurement. When candidate generation outpaces target-device evaluation, device feedback becomes the search bottleneck. We propose teacher-distilled dual pruning, a two-stage framework for optimization under limited evaluation budgets. Offline, a strong language model analyzes public implementations, patches, and failure cases, distilling optimization actions, applicability conditions, coupled changes, and source-level checks into linked knowledge records. Online, these records guide two pruning layers. Search-space pruning excludes routes incompatible with task and hardware conditions and guides search and generation within the retained space. Candidate-set pruning combines failure knowledge with tool results to inspect generated programs, block confirmed violations, and guide repair and selection before measurement. The framework requires no training or fine-tuning of the online model and supports open- and closed-source models. On 100 KernelBench Level 2 tasks on NVIDIA RTX 4090, it finds faster implementations on 77% of tasks with at most three candidate evaluations per task. The evaluated baselines first reach the same coverage with five to nine evaluations, corresponding to a 40.0%–66.7% reduction in candidate-evaluation budget. Experiments on NVIDIA H100 and with GLM-5.3-Flash further demonstrate applicability across hardware platforms and online models.

1 INTRODUCTION

Efficient GPU kernels are essential to deep learning performance. Their optimization requires iterative design, compilation, correctness checking, and performance measurement. Large language models accelerate design and source analysis through parallel calls, while declining inference costs at comparable capability levels make additional preparation increasingly practical (Stanford HAI, 2025). Verifying correctness and speed, however, still requires target-device execution. When candidate generation outpaces available evaluation capacity, hardware validation becomes the search bottleneck. We therefore study how model computation and prior experience can improve candidate preparation, enabling scarce device evaluations to discover effective optimizations earlier.

Existing approaches improve candidate construction and search: AutoTVM and Ansor combine structured search with cost models (Chen et al., 2018; Zheng et al., 2020), while CUDA Agent strengthens kernel generation through training (Dai et al., 2026). Experience and prediction also guide decisions: KernelSkill reuses knowledge and optimization trajectories (Sun et al., 2026), and GPU Forecasters helps select measurements (Khan et al., 2026). Under small budgets, however, evaluations that expose avoidable design mismatches or implementation defects leave fewer opportunities to test promising alternatives. We therefore link design conditions with implementation checks to guide two decisions: which designs to explore before generation and which programs to measure afterward.

An optimization must first fit the task and hardware conditions; even then, the generated code can omit coupled changes or fail to realize the intended improvement. For example, changing reduction tiling requires coordinated updates to work partitioning, temporary storage, and merge bounds. Specifying the optimization goal alone can leave these dependencies implicit during both design composition and source inspection. Such relationships are scattered across public implementations, patches, and

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.

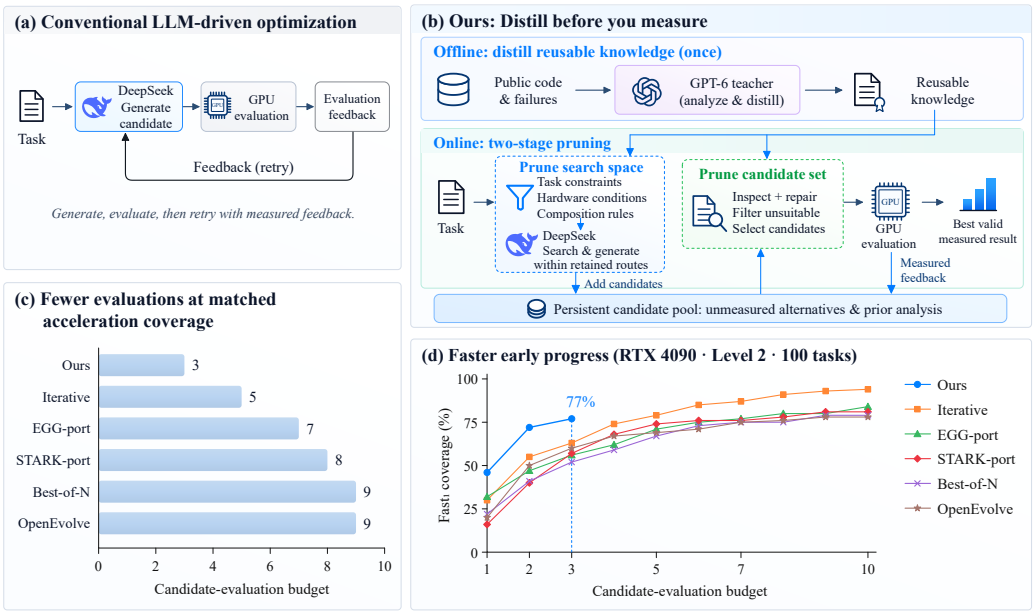

Figure 1: Preparation before scarce device feedback. (a) Generate–evaluate–retry. (b) Offline teacher knowledge guides two-stage pruning. (c,d) On 100 RTX4090 Level 2 tasks, our method reaches 77% Fast_1 coverage at three evaluations; baselines require five to nine.

failure cases. Teacher distillation makes them explicit for use in design exploration and program inspection (Figure 1).

Our teacher-distilled dual-pruning framework comprises offline and online stages. Offline, a strong teacher analyzes public implementations and failures, distilling records that link optimization actions to their applicability conditions, coupled changes, and source checks. Online, task and hardware conditions first exclude incompatible design routes; the model searches and composes designs within the retained routes, then generates their implementations. Options with unknown conditions and freeform routes remain available. The second layer uses failure knowledge and tool results to inspect programs, block candidates with tool-confirmed violations, and guide repair and selection before evaluation. A persistent pool retains unmeasured programs and their assessments for later selection using measured feedback. The framework requires no online parameter updates and supports open- and closed-source models.

On 100 KernelBench Level 2 tasks (Ouyang et al., 2025) on NVIDIA RTX 4090, three evaluations per task yield 77% measured-faster coverage, including 45% with timing-qualified speedups above $1.05\times$. The evaluated baselines first match 77% coverage at five to nine evaluations, corresponding to 40.0–66.7% fewer evaluation slots for our framework. The framework thus trades additional model computation for fewer device evaluations. Experiments on NVIDIA H100 and with GLM-5.3-Flash further examine applicability across hardware and online models.

Our main contributions are as follows:

- **Teacher-distilled optimization knowledge.** We use a strong teacher language model to distill knowledge from public implementations, patches, and failure cases into linked records of optimization actions, applicability conditions, coupled changes, and source checks. These distilled records support design exploration and program inspection across tasks.
- **Dual pruning of designs and programs.** We restrict design exploration using task and hardware conditions before code generation, then inspect, repair, and filter generated programs before measurement. The two layers apply distilled knowledge to both what the model builds and which implementations receive scarce device evaluations.

- **Evaluation efficiency under scarce device feedback.** Fixed-budget and matched-coverage comparisons show higher early acceleration coverage and fewer device evaluations with additional model computation. Knowledge-delivery, capability-removal, candidate-count, cross-hardware, and cross-model studies examine how knowledge use, search configuration, and operating conditions affect the results.

2 Related Work

Kernel generation and reusable experience. AutoTVM and Ansor combine structured search spaces, learned cost models and target measurements (Chen et al., 2018; Zheng et al., 2020). KernelBench evaluates LLM-generated kernels against executable references (Ouyang et al., 2025). STARK combines multi-agent coordination with profiling feedback (Dong et al., 2025); CUDA Agent develops kernel-generation ability through agentic reinforcement learning (Dai et al., 2026). KernelSkill reuses expert skills and task-local memory (Sun et al., 2026), KLineage extracts verified skills from expert lineages (Zhang et al., 2026), and VCE-Skill distills skill-version changes for further evolution (Chen et al., 2026). Our records connect optimization actions to applicability conditions, coupled changes and source checks, guiding both design-space and candidate-set pruning.

Knowledge and evaluation-efficient search. Classical distillation transfers behavior through parameter updates (Hinton et al., 2015); our teacher produces external records for a fixed online model. Surrogate-assisted optimization allocates expensive evaluations with learned models (Ruan et al., 2020). LLAMBO uses LLMs for warmstarting, surrogate modeling and candidate sampling (Liu et al., 2024), while GPU Forecasters predicts ordered speedup categories and trains selective surrogates from measured outcomes (Khan et al., 2026). Test-time scaling studies computation allocation between proposal refinement and verification (Snell et al., 2025); SWE-Search combines tree search, value estimation and iterative feedback for software agents (Antoniades et al., 2025). We allocate model computation to knowledge-guided construction and source assessment before device trials, retaining unmeasured alternatives in a persistent pool for feedback-driven selection.

3 Problem Setting

A task x specifies reference behavior, input/state contracts and permitted implementation changes. A profile h records hardware and toolchain facts, with missing capabilities marked unknown. Given frozen knowledge $\mathcal{K}$, tools $\mathcal{T}$ and a fixed-parameter online model, the optimizer searches retained design routes and selects programs for evaluation using reference source and its own search feedback. The device budget B counts attempted complete-candidate evaluations, including failures and unresolved external effects; unused positions remain distinct. A separate allowance M bounds online model calls and tokens for candidate preparation and search.

Let $\mathcal{E}_{t,B}$ contain the candidates evaluated in the first B positions of task t, $C(c)$ denote correctness under the declared protocol, and $S(c)$ be the central reference-to-candidate latency ratio. Its reporting-eligible subset is $\mathcal{E}^{\star}_{t,B}$ (Section 5). Following KernelBench (Ouyang et al., 2025), define

$$\text{Correct@}B = \frac{1}{n}\sum_{t=1}^{n} \mathbf{1}\{\exists c \in \mathcal{E}_{t,B} : C(c)\}, \tag{1}$$

$$\text{Fast}_p\text{@}B = \frac{1}{n}\sum_{t=1}^{n} \mathbf{1}\{\exists c \in \mathcal{E}^{\star}_{t,B} : C(c) \wedge S(c) > p\}. \tag{2}$$

Fast_1 measures faster-implementation coverage; strict $\text{Fast}_{1.05}$ adds the timing qualification in Appendix B. Rates use the full declared task inventory. Measured framework-equivalent candidates remain eligible and identified by type; deployment-only reference fallback receives no generated-success credit. We seek earlier coverage at fixed B, reporting model usage and native execution cost separately. Our controller has three evaluation positions; prefixes describe intermediate outcomes of the complete policy. Appendix M formalizes this resource objective.

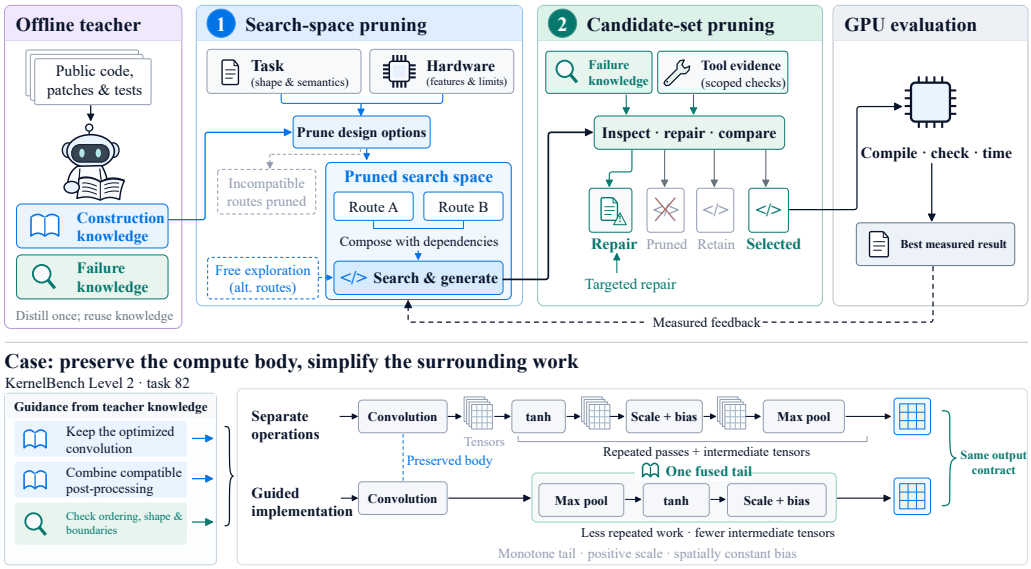

Figure 2: Dual pruning of design options and generated programs. Top: task and hardware conditions prune the design space; composition and generation search within the retained routes. Source assessment then prunes the candidate set before measurement. Bottom: task 82 preserves convolution and combines postprocessing into one tail, reducing repeated work and intermediate tensors while preserving the output contract.

4 METHOD

Offline, the teacher distills reusable knowledge; online, two layers apply it to different pruning objects (Figure 2). The first removes inapplicable design routes before the model searches and generates within the retained space. The second inspects, repairs and filters generated programs before device evaluation. Positive and negative knowledge are content types; either can support either layer. A failure pattern can exclude a design, while a positive optimization record can specify checks on its implementation. Algorithm 1 connects these decisions to measurement.

4.1 DISTILLING CONSTRUCTION AND ASSESSMENT KNOWLEDGE

The offline teacher is a strong language model used for source analysis. Archived public code, patches, tests and hardware documentation ground records that connect an action to applicability conditions, coupled implementation steps and supporting evidence. Positive records add benefit rationales and alternatives; shared mechanism cores and family bindings support composition across operators and backends. Negative records specify failure triggers, required source facts, legal exceptions and quantities that a repair must update together. The records distinguish what the source shows from what the teacher infers. Developer-maintained constructors and checks implement supported operations and are versioned with the records (Appendices C and N.1).

A record summarizes an action or failure pattern a_k, applicability conditions $\mathcal{C}_k$, coupled changes $\mathcal{D}_k$, source checks $\mathcal{Q}_k$ and supporting material e_k:

$$k = (a_k, \mathcal{C}_k, \mathcal{D}_k, \mathcal{Q}_k, e_k). \tag{3}$$

At construction time, it supplies a choice and its dependencies; during assessment, those dependencies become questions about the actual program.

For R rows of length N and partition width b, a partitioned reduction produces $P = \lceil N/b \rceil$ partial results per row. Changing b changes P, so the producer grid, scratch allocation of RP elements and merge bounds must be updated together. A mean still divides by the original row length N.

The launch grid also matters: a program that loops over all partitions of one row still exposes only one independent program for that row. Launching one program per partition instead exposes RP programs, adding scratch traffic and a merge launch (Appendix L.2).

4.2 Search within a knowledge-pruned design space

The first layer turns open-ended code generation into search over explicit optimization routes. Distilled records supply transformations, applicability conditions and composition dependencies. Binding reference semantics, shape, dtype, layout, hardware and toolchain facts removes routes with false prerequisites. The retained routes define the task-specific search space: the model chooses and composes designs within it, then generates their implementations. Unknown conditions remain attached for investigation; explicit freeform routes preserve exploration beyond the catalog. Level 1 uses shared and operator-family records. Level 2 adds graph and region scopes, including producer–consumer relations and supported reference-body-preserving transformations.

For a design option d, let $\mathcal{C}(d)$ collect its required conditions and let $v(q; x, h)$ be a condition's value in $\{\text{true}, \text{false}, \text{unknown}\}$. The retained design set is

$$\mathcal{D}_{x,h} = \{d \in \mathcal{D} : \nexists q \in \mathcal{C}(d) \text{ with } v(q; x, h) = \text{false}\}. \tag{4}$$

Each retained option carries its required changes and source checks. A compact menu exposes the design choices; selecting a route expands its dependencies for implementation. The generator therefore searches a structured space of transformations and compositions before committing to complete code.

Figure 2 illustrates preserving convolution and reordering its postprocessing. For a pooling window, positive scale s and spatially constant channel bias b give $\max_i\{s\tanh(z_i)+b\} = s\tanh(\max_i z_i)+b$. These conditions determine when the reordering is applicable in the design space. The generated fused tail performs the pointwise work after pooling. The second layer checks whether its source implements the ordering and preserves output shape, window boundaries and numerical behavior.

4.3 Evidence-guided candidate-set pruning

The second layer reviews complete programs before GPU evaluation. Tools check specific source properties, while failure knowledge guides inspection and repair. A tool-confirmed violation blocks the candidate; a model-raised concern prompts further inspection, repair or deferral. Candidates that meet the admission rules remain eligible when some concerns are unresolved.

The selector compares each eligible program with the reference and, when available, the best valid measured candidate, focusing on work removed or added. For example, a proposed fusion is inspected for whether the fused helper is called by the forward path, and a partition change is checked against its producer grid, scratch shape and merge bounds. These checks connect a claimed optimization to executed work. A repair creates a new source version whose affected properties are reassessed before selection. When a measurement changes the incumbent, the selector compares the remaining eligible alternatives with the new incumbent source. Table 5 gives the decision rules.

4.4 Persistent-pool search and feedback

Algorithm 1 maintains a persistent pool of deduplicated, unmeasured programs and their assessments. After each evaluation, the controller removes the measured source, updates the incumbent from valid measurements, and retains the other alternatives for selection without regeneration. Permitted supply and repair add sources to this pool; each repair binds an exact parent and shares its lineage allowance. Level 1 uses an initial target $N = 16$ with staged exploitation and exploration. Level 2 starts with two waves of at most four sources, the second conditioned on first-wave source facts. It permits at most two feedback-triggered supply/repair events and one eligible different-body window within the remaining positions (Appendix A). Failed native attempts consume positions.

5 Evaluation Design

Tasks and comparisons. Experiments use public development tasks. Our primary comparison uses all 100 KernelBench Level 2 tasks, one independent search per task, an initial pool of at most

eight candidates and $B = 3$. Best-of-N, Iterative, OpenEvolve, EGG-port and STARK-port share the evaluation protocol. Their $B = 1, \ldots, 10$ results are consecutive prefixes of one recorded trajectory per method and task. All first-five-position prefixes are complete; later completion is detailed in Appendix E. Runs use RTX4090 hosts and nonthinking DeepSeek-V4.1-Flash. We repeat the Level 2 comparison on H100 with four baselines and on 24 shared tasks with GLM-5.3-Flash. Level 1 provides a complementary $N = 4, 8, 16$ pool-size study on 24 common tasks, using the corresponding recorded searches; its full $N = 16$ results are in Appendix E.3.

Metrics and resources. For all methods, speedups of $S \geq 10$ are excluded from performance aggregation (KernelBench Contributors, 2026). Their evaluation slots, correctness verdicts and tasks remain in the reported records. Appendix E reports the source audit and sensitivity analysis. Strict $\text{Fast}_{1.05}$ requires stability and a minimum ratio above 1.05 across seeds and reference brackets. AM/GM are secondary success-only summaries. Appendix B defines the metrics and timing protocol.

For baseline m and coverage target q, the first observed crossing is $b_m(q) = \min\{b : \text{Fast}_{1,m}@b \geq q\}$; slot saving is $1 - b_{\text{ours}}/b_m(q)$. Tokens measure model usage; native-worker durations cover the complete evaluation pipeline, including compilation, correctness checks and timing. H100 trajectory selection and follow-up costs are in Appendices H and J.3.

Knowledge and implementation studies. On 24 development-seen Level 2 tasks stratified by source and family, capability removals disable positive guidance and dedicated construction support (`NO_POS`), negative guidance and specialized checks/patches (`NO_NEG`), or both (`NEITHER`). They retain generic model search, reference inputs, artifact interfaces and native evaluation. Each runs once at $N = 8, B = 3$, in balanced orders across two hosts, using FULL from the preceding independent four-arm run on the same tasks. Coarse route replaces both channels with a body-retention/tail-fusion hint and generic search. A same-source excerpt study varies positive content within shared support; raw-source search removes teacher-specific support. A separate three-arm batch varies knowledge delivery to construction, assessment, or neither while retaining common tools and search support. Fixed-source confirmation repeats selected sources with fresh seeds. Appendix D and Table 31 map interventions, service settings and controls.

6 Results and Analysis

6.1 Early acceleration under a small evaluation budget

On 100 RTX4090 tasks, ours reaches 46%, 72% and 77% Fast_1 coverage at $B = 1, 2, 3$, leading the strongest baseline by 14, 17 and 14 percentage points (Table 1). Strict coverage also leads at each prefix.

Algorithm 1 Measurement-budgeted search with dual pruning

Require: Reference x, hardware h, knowledge $\mathcal{K}$, tools $\mathcal{T}$, evaluation budget B, model allowance M
Ensure: Best valid measured source $c^\star$, or $\varnothing$

$D \leftarrow \text{PruneDesignSpace}(x, h, \mathcal{K})$ ▷ Prune false prerequisites; retain unknowns
$F \leftarrow \text{SearchAndGenerate}(D; M)$; $c^\star \leftarrow \varnothing$; $b \leftarrow 0$
while $b < B$ **do**
  Assess sources in F using scoped checks and relevant knowledge
  Investigate or repair relevant concerns within M; reassess changed sources
  $A \leftarrow \text{Admissible}(F)$ ▷ Block confirmed contradictions
  if $A = \varnothing$ **then break**
  $c \leftarrow \text{Select}(A, c^\star, x, h; M)$
  $r \leftarrow \text{NativeEvaluate}(c)$; $b \leftarrow b + 1$ ▷ An attempted evaluation consumes a slot
  Bind r to c; remove c from F; update $c^\star$ from valid measurements
  if execution requires an operational stop **then break**
  if $b < B$: retain F; add permitted feedback-triggered supply or a child
end while
return $c^\star$

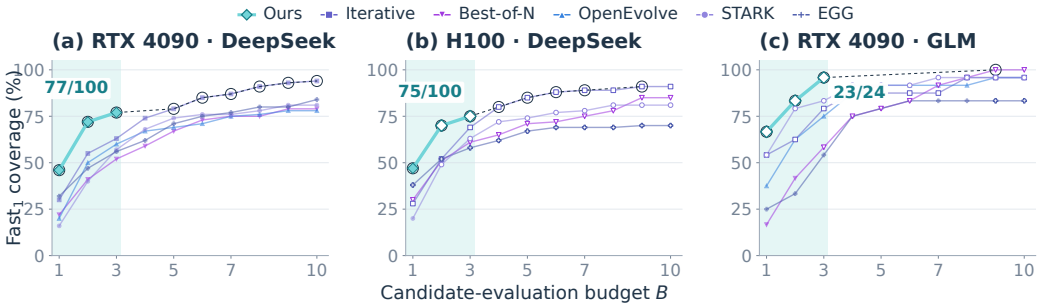

Figure 3: Evaluation budget and Fast_1 coverage on Level 2: (a) RTX4090 with DeepSeek-V4.1-Flash, (b) H100 with DeepSeek-V4.1-Flash, and (c) RTX4090 with GLM. Denominators are 100, 100 and 24 tasks. Shading marks $B \leq 3$; direct labels identify our measured $B = 3$ points. Outlined points and dashed links mark the observed Pareto set; hollow markers identify incomplete prefixes. Our curves end at measured $B = 3$.

Table 1: Coverage (%) and first evaluation budgets matching our $B = 3$ targets: 100 RTX4090 Level 2 tasks. Strict criterion: Appendix B; dashes: unreached through $B = 10$.

	Fast_1			Strict $\text{Fast}_{1.05}$			First target budget	
Method	$B = 1$	$B = 2$	$B = 3$	$B = 1$	$B = 2$	$B = 3$	77 Fast_1	45 strict
Best-of-N	22	41	52	5	10	13	9	—
Iterative	30	55	63	14	23	29	5	9
OpenEvolve	20	50	60	9	20	28	9	10
STARK-port	16	40	57	11	26	34	8	5
EGG-port	32	47	56	21	30	39	7	5
Ours	**46**	**72**	**77**	**32**	**41**	**45**	**3**	**3**

The second slot adds 26 accelerated tasks and the third adds five, concentrating the largest improvement in the first two evaluations. At $B = 3$, ours obtains 90 correct and 77 faster tasks versus Iterative's 95 and 63; both lie on the correctness–Fast_1 frontier (Figure 5, Appendix F.1).

Matching our 77-task Fast_1 coverage takes five to nine evaluations for the baselines: **40.0–66.7% fewer evaluations** for ours. At the 45-task strict target, savings are **40–70%** against the four reaching baselines; Best-of-N remains below this target through $B = 10$ (Table 1). Across all three Fast_1 anchors, savings range from 40.0% to 71.4% (Table 12); Tables 15 and 16 give sensitivity bounds.

Trading model computation for GPU evaluations. At matched 77% Fast_1 coverage, ours trades more model computation (94.763M reported online tokens versus 1.565–7.225M) for a smaller GPU-evaluation budget (three slots per task versus five to nine) and less native-worker time (12.64 versus 28.51–58.02 hours; Table 2). The additional model computation supports construction, inspection and selection before device trials.

Of the 77 faster tasks, 47 exceed a central $1.05\times$ ratio and 45 satisfy strict timing. Three fresh-seed repeats of 48 selected sources and 12 reference-equivalent controls pass all 180 correctness and stability checks. Our six selected strict sources remain strict in every repeat; reference controls span $0.9931\times$–$1.0136\times$, with none strict (Appendix K.6).

6.2 Transfer across hardware and online models

On H100, ours leads Fast_1 through the first three budgets, reaching 47%, 70% and 75% (Figure 3b). The largest margin is **18 percentage points at** $B = 2$, against 52% for the strongest baseline. At $B = 3$, strict coverage reaches 34 tasks, compared with EGG-port's 36. Iterative, STARK-port and Best-of-N first match our 75-task Fast_1 target at four, six and seven positions; EGG-port has not

Table 2: Resources to first reach 77 Fast_1 tasks: native-worker hours and reported tokens (five of our attempts have unknown usage). Offline work, startup controls and superseded searches are separate.

Method	First B	GPU receipts	GPU-worker h	Tokens (M)	HTTP attempts
Ours	3	300	12.64	94.763	2,031
Best-of-N	9	887	32.51	1.565	900
Iterative	5	498	29.83	2.387	500
OpenEvolve	9	893	58.02	6.630	900
STARK-port	8	793	33.32	7.225	1,398
EGG-port	7	694	28.51	5.096	1,087

reached it in the observed prefixes. These crossings correspond to 25.0–57.1% fewer slots (Tables 21 and 22).

The second H100 slot adds 23 faster tasks, of which 18 improve an already correct but slower first result. Nineteen of the same 23 tasks use candidates generated in the initial pool. Linear tasks contribute broad coverage (27/37 at $B = 2$); Conv2d contributes stronger gains (14/21 strict tasks versus at most 8/21 for a baseline). Appendix J examines the corresponding source changes.

With GLM-5.3-Flash on 24 tasks, ours obtains **23/24** correct-and-faster results at $B = 3$, versus at most 20/24 for a baseline using the same model. Best-of-N and Iterative match 23 at $B = 8$, OpenEvolve at $B = 9$, and STARK-port at $B = 7$ in its partially observed prefixes (Figure 3c). Strict counts are 14 for ours, 15 for Iterative and 14 for STARK-port. At matched 23/24 Fast_1 coverage, ours uses 2.74 GPU-worker hours versus 5.35–9.26 for the four reaching baselines, with 18.57M versus 2.57–3.74M reported tokens (Appendix I).

6.3 Knowledge delivery and realized optimization

Knowledge delivery. With tools and construction support held fixed, `GEN_ONLY` supplies knowledge to generation and repair, `ASSESS_ONLY` to assessment, and `GENERIC_PREP` uses generic preparation. Their strict-success counts are 5/8/9, 5/8/9 and 4/5/8 at $B = 1, 2, 3$ (Appendix K.2). At $B = 2$, both knowledge arms reach eight tasks versus five for generic preparation, a 12.5-percentage-point difference. The two knowledge arms share seven final strict successes, with two unique to each. ASSESS_ONLY uses 8.512M reported tokens versus 19.169M for GEN_ONLY, a 55.6% reduction (Figure 4b).

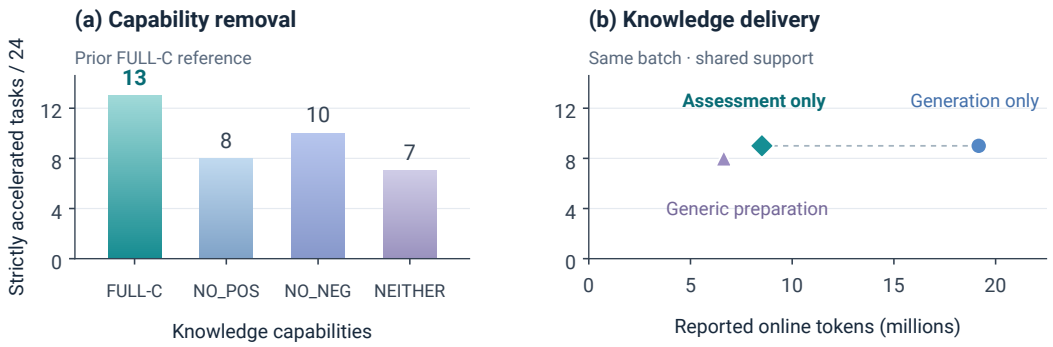

Figure 4: Two knowledge studies on 24 tasks at $N = 8, B = 3$. (a) Capability removal, with prior FULL-C as reference. (b) Same-batch knowledge delivery under shared tools and construction support: reported tokens versus strict coverage. Table 3 lists configurations.

Under shared support, organized positive guidance (FULL-E, prior) adds six Fast_1 tasks over same-source excerpts with 16.8% fewer reported tokens; strict coverage differs by one task (Table 3). In a separate study without teacher-specific support, raw-source search yields 12 correct, nine faster and

Table 3: Knowledge studies ($N = 8, B = 3$, 24 tasks). Delivery: same batch; capability/coarse route: prior FULL-C; same-source: prior FULL-E, shared support. Settings/prefixes: Appendices D and K.

Arm	Correct	Fast_1	Strict $\text{Fast}_{1.05}$	Reported tokens (M)
Knowledge delivery: same-batch, shared support				
GEN_ONLY	19	16	9	19.169
ASSESS_ONLY	21	17	9	8.512
GENERIC_PREP	20	17	8	6.638
Capability and coarse-route comparisons				
FULL-C (prior)	23	19	13	21.909
NO_POS	20	11	8	13.770
NO_NEG	22	18	10	12.708
NEITHER	21	16	7	7.555
Coarse route	24	19	6	5.978
Same-source positive guidance: shared support				
FULL-E (prior)	20	19	11	22.763
Source excerpts	19	13	10	27.368

one strict result on 16 completed tasks, versus 13, 13 and seven for the same 16-task subset of the main experiment (FULL-M) (Appendix K.5).

Implementation differences. On task 54, FULL-C and `NO_NEG` preserve convolution and fuse its tail ($1.367\times$, $1.356\times$), whereas `NEITHER`'s explicit convolution loops reach $0.012\times$. In the main-experiment task 82 source, valid reordering reduces tanh evaluations from 16 to one per window (Figure 2). Coarse route's task 48 defines but never calls its fused helper, reaching $1.0020\times$ versus FULL-C's $1.6844\times$ (Appendix K.3).

After task 35's integer-unpacking failure, ASSESS_ONLY reuses an initial-pool alternative supporting scalar/pair kernel sizes and a fused tail: $1.600\times$ at position two. GEN_ONLY instead generates a feedback-driven child reaching $1.184\times$ at position three.

6.4 Candidate-pool size and model computation

On 24 Level 1 tasks spanning nine operator families, $N = 4, 8, 16$ yield 9/15/19 faster results and seven strict accelerations each at $B = 3$ (Table 4). $N = 8$ matches $N = 16$ correctness with 28.6% fewer reported tokens, while $N = 16$ gives higher Fast_1 coverage. This coverage–computation trade-off motivates $N = 8$ for Level 2; Level 1 retains $N = 16$ (Appendix G).

Table 4: Initial-pool results at $B = 3$ on 24 tasks. AM/GM use each task's best correct central ratio ($n = 20, 24, 24$); tokens cover the selected searches.

Pool	Correct	Stable	Fast_1	Strict	AM	GM	Tokens (M)	HTTP
$N = 4$	20	20	9	7	1.270	1.038	19.943	435
$N = 8$	24	24	15	7	1.273	1.101	26.581	608
$N = 16$	24	23	19	7	1.328	1.189	37.251	788

7 Conclusion

Teacher-distilled dual pruning uses offline knowledge to prune design routes before generation and candidate programs before evaluation. Applicability conditions guide construction; coupled changes and source checks guide inspection, repair and selection. A persistent pool reuses unmeasured alternatives as feedback updates selection. On 100 KernelBench Level 2 tasks, three evaluations yield 77% measured-faster coverage, matched by the evaluated baselines at five to nine. H100 and GLM experiments extend the evaluation across hardware and online models. The framework turns model computation and distilled knowledge into better-prepared candidates for scarce target-device trials.

AI USE STATEMENT

OpenAI Codex assisted with source analysis, offline knowledge extraction, software development, artifact analysis, and manuscript drafting. AI tools also assisted in refining the methodological design, planning experiments, and interpreting results. Online models provide candidate construction and evidence-conditioned investigation, repair, and comparison. GPT Image supplied the cartoon assets. The workflow, generated-code comparison, and motivation figure were assembled as editable PowerPoint objects; the numerical panels use recorded experimental results. Authors remain responsible for source attribution, method accuracy, and empirical claims.

REPRODUCIBILITY STATEMENT

The anonymous source package contains the manuscript, figures, experimental data and build instructions. Numerical plots are generated from the included per-position metrics. Appendices C–K describe the knowledge artifacts, prompts, tools, evaluation settings, and run-specific outcome and cost records.

REFERENCES

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. SWE-Search: Enhancing software agents with Monte Carlo tree search and iterative refinement. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a1e6783e4d739196cad3336f12d402bf-Abstract-Conference.html.

Jianming Chen, Xuanbin Ye, Yawen Wang, Junjie Wang, Yuanzhe Hu, Qing Wang, and Fanjiang Xu. VCE-Skill: Enhancing skill self-evolution with version-change experience, 2026. URL https://arxiv.org/abs/2608.16544v1.

Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Learning to optimize tensor programs. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/8b5700012be65c9da25f49408d959ca0-Abstract.html.

Weinan Dai, Hanlin Wu, Qiying Yu, Huan-ang Gao, Jiahao Li, Chengquan Jiang, Weiqiang Lou, Yufan Song, Hongli Yu, Jiaze Chen, Wei-Ying Ma, Ya-Qin Zhang, Jingjing Liu, Mingxuan Wang, Xin Liu, and Hao Zhou. CUDA Agent: Large-scale agentic RL for high-performance CUDA kernel generation, 2026. URL https://arxiv.org/abs/2602.24286.

Juncheng Dong, Yang Yang, Tao Liu, Yang Wang, Feng Qi, Vahid Tarokh, Kaushik Rangadurai, and Shuang Yang. STARK: Strategic team of agents for refining kernels, 2025. URL https://arxiv.org/abs/2510.16996v1.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

KernelBench Contributors. KernelBench: Evaluation code and excessive-speedup detection. https://github.com/ScalingIntelligence/KernelBench/blob/7a0fae93624b60b8e9b67a862acabb5dbf8f0951/src/kernelbench/eval.py, 2026. Version pinned in the URL; lines 412 and 691–695. Accessed September 24, 2026.

Zaid Khan, Justin Chih-Yao Chen, Jaemin Cho, Elias Stengel-Eskin, and Mohit Bansal. GPU forecasters: Language models as selective surrogates for kernel runtime optimization, 2026. URL https://arxiv.org/abs/2605.31464v1.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2402.03921.

Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. KernelBench: Can LLMs write efficient GPU kernels? In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47356–47415. PMLR, 2025. URL https://proceedings.mlr.press/v267/ouyang25a.html.

Xiaoran Ruan, Ke Li, Bilel Derbel, and Arnaud Liefooghe. Surrogate assisted evolutionary algorithm for medium scale expensive multi-objective optimisation problems, 2020. URL https://arxiv.org/abs/2002.03150.

Sakana AI. AI CUDA Engineer: Evaluation post-mortem. Author's public account, March 3, 2025. URL https://sakana.ai/ai-cuda-engineer-post-mortem/. English description of the Japanese-language article. Accessed September 24, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

Stanford HAI. The 2025 AI Index report. Stanford Institute for Human-Centered Artificial Intelligence, 2025. URL https://hai.stanford.edu/ai-index/2025-ai-index-report. Top Takeaway 7: capability-normalized inference cost, November 2022–October 2024. Accessed September 24, 2026.

Qitong Sun, Jun Han, Tianlin Li, Zhe Tang, Sheng Chen, Fei Yang, Aishan Liu, Xianglong Liu, and Yang Liu. KernelSkill: A multi-agent framework for GPU kernel optimization, 2026. URL https://arxiv.org/abs/2603.10085.

Shuoming Zhang, Qiuchu Yu, Yangyu Zhang, Ruiyuan Xu, Xiyu Shi, Guangli Li, Xiaobing Feng, Huimin Cui, and Jiacheng Zhao. Learning when to optimize: Verified optimization skills from expert GPU-kernel lineages, 2026. URL https://arxiv.org/abs/2605.28213v1.

Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. Ansor: Generating high-performance tensor programs for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 863–879. USENIX Association, 2020. URL https://www.usenix.org/conference/osdi20/presentation/zheng.

A Independent Controllers and Information Access

Algorithm 1 gives the common search procedure. The level-specific implementations share source identity, conditional knowledge, scoped inspection, candidate-pool retention, and native accounting. Their candidate schedules reflect the different structure of single-operator and composite tasks.

A.1 Level 1

The single-operator controller constructs candidates through conditional compositional plans and materializes complete source. Bounded generation can replenish supply; later positions use an exact measured parent for exploitation and an alternative route for exploration. On-demand investigation addresses relevant unresolved source or repair issues. Source tools distinguish confirmed findings, advice, and unknown scopes.

Comparison may split sources into batches followed by a finalist comparison. It consumes actual code, reference work, matching measured parents and current evidence. Before dispatch the selected source is checked again. Model concerns can trigger bounded repair. A child is rechecked and investigated before receiving preferential selection. Candidates with unresolved source concerns remain marked unknown and undergo the shared source checks and native gates.

The single-operator configuration uses an initial candidate target of $N = 16$ and $B = 3$. Runtime scheduling, numerical-comparison handling and source-accounting corrections are integrated into this implementation. Candidate count is a level-specific search hyperparameter; the two pruning layers and their information interfaces remain shared.

A.2 Level 2

The composite-operator controller starts with at most eight implementations in two waves of four. The second wave receives first-wave source and static facts, without additional target measurements. Each code request produces one complete source, and replenishment retains the existing candidate pool.

Task, family, graph and region contracts condition generation. Supported body-preserving scaffolds can reserve a construction slot. Their sources then follow the common assessment, selection and native evaluation path. Source analysis precedes semantic inspection, and source-identical evidence is reused under matching dependencies. Comparison normally orders the eligible candidate pool in one request; capacity-driven brackets record actual coverage. Missing comparison output yields a recorded fallback order.

Later positions retain the candidate pool and allow at most two supply/repair events, using native candidate failures or a bounded incumbent-performance continuation. A slow-route rule can allocate at most one remaining position to an eligible different-body route within the remaining budget. The controller retains bounded source checks and generic source-linked feedback repairs.

A.3 Online decision contracts

Table 5 specifies the implemented Level 2 decisions. Comparison consumes full candidate source, the reference, findings tied to the current source version, applicable contracts and permitted native history. Its prompt asks for separate comparisons with the reference and the current measured incumbent: cite expressions for work removed *and added*, including body delegation, serial loops, memory traffic and materialization. Work already absent from the incumbent is not counted again. Model reasons guide ordering; source checks govern admission and native receipts update the measured incumbent.

Comparison output requires each supplied ID exactly once and a corresponding reason. A bounded format-recovery request handles malformed replies; absence of usable ranking records the fallback. Capacity batches retain the top two sources from groups larger than two and one from smaller groups, followed by a finalist comparison. Changed children carry their own source identities, while the exact-parent lineage shares its repair allowance.

Table 5: Concrete inputs, triggers and outputs of the released composite-task controller. Prompt and consumer excerpts are retained with source hashes.

Decision	Input and trigger	Output and next action
Inspect	Complete source, applicable failure duties, legal exceptions and current tool reports.	Source-quoted claims with candidate/duty IDs and a repair suggestion; optional check requests. Confirmed scoped tool contradictions block admission.
Compare	Eligible sources, reference, actual incumbent and current evidence. Assess realized work and unresolved premises.	JSON order of supplied IDs, one reason per ID and the required baseline identity. Rank once if capacity permits; otherwise compare batches and their finalists.
Fallback	Singleton or unavailable bounded comparison.	Preserve the input candidate order, initially creation order; record fallback coverage. Recheck eligibility before dispatch.
Repair / improve	Candidate-native failure, or a stable valid incumbent not yet used for performance continuation.	Exact-parent child changing one optimization intent plus necessary dependencies; fresh checks and pool competition. At most two supply/repair events.
Different body	Remaining slot, stable valid measurements all below $0.75\times$, and eligible contrasting body routes.	At most one slot goes to the highest model-ranked different-body finalist. Preserve model order and spending order separately.
Measure / retain	Selected source, remaining position and passed operational controls.	Source-bound native receipt; consume the position and retain the best measured valid implementation.

A.4 Evidence and recovery

Roles receive relevant contracts, required candidate source and permitted history. Candidate/source, reference, device, library, tool and observation identities control reuse. Each changed source receives fresh judgments and its own measurement. Repair allowances accumulate across the child lineage.

Infrastructure faults and uncertain effects have separate execution records. Every formal attempt consumes its position, including failure. Reference health, candidate correctness, memory-safety checks, timing and operational controls remain distinct. Unknown external effects stop redispatch; failed controls or unconfirmed worker cleanup stop the affected execution path. Framework fallback and custom kernel success are reported separately.

B Metrics, Measurement, and Statistical Analysis

B.1 Correctness and evidence eligibility

Let $C_{\mathrm{num}}(c)$ indicate that the required numerical and behavioral comparisons pass. The behavioral contract includes output metadata, permitted alias relationships, input side effects, persistent state and elementwise numerical agreement. Let $C(c)$ require both those checks and explicit success of the validation tools required by the reported protocol. Reported correctness requires complete numerical, behavioral and required sanitizer evidence from the frozen native protocol. Results separately identify numerical, validation-complete, and timing-eligible coverage. Missing or interrupted evidence has an unverified status.

The reporting chain binds source identity, evaluation positions, component receipts, and measurements to the corresponding frozen run. A later replay or continuation is identified separately and its additional cost is retained.

B.2 Timing and threshold success

The configured development evaluator separates reference-before, candidate, and reference-after endpoints into processes. It uses two input/state seeds, with four warmups and twenty measured forward invocations per endpoint and seed. Each measured invocation uses freshly constructed

input/model state. The timed region includes the complete forward call and device synchronization, yielding task-level latency under the declared execution setting.

For seed block s, let $m_{0,s}^{-}$ and $m_{0,s}^{+}$ be the reference bracket medians and $m_{c,s}$ the candidate median. Define the conservative comparison

$$R_{\min}(c) = \min_{s,\, b\in\{-,+\}} \frac{m_{0,s}^{b}}{m_{c,s}}. \tag{5}$$

The separate strict $\text{Fast}_{1.05}$ threshold requires $R_{\min}(c) > 1.05$, reporting eligibility (Section 5), complete timing evidence, and the stability checks below. The recorded central ratio defines $S(c)$ in Fast_p; the strict metric additionally requires all timing blocks and stability checks to pass.

The configured checks require a median absolute deviation divided by median of at most 0.10 for each pooled endpoint, cross-seed central-ratio spread of at most 1.10, and within-seed reference-bracket spread of at most 1.10. The implementation and sample inventories record these thresholds. Higher secondary speed thresholds use the same strict comparison operator and evidence gates.

For task t with evaluated candidates $\mathcal{E}_{t,B}$, timing-eligible coverage is

$$\text{Timed}_t@B = \mathbf{1}\{\exists c \in \mathcal{E}_{t,B} : C(c) \wedge T(c)\}. \tag{6}$$

Equation 2 uses the central ratio. Stable-only Fast_1 adds $T(c)$; strict $\text{Fast}_{1.05}$ adds $T(c)$ and $R_{\min}(c) > 1.05$, with the same candidate screen. Search-time coverage uses the original candidate positions. Independent confirmation retains the selected sources and records its execution cost separately (Appendix K.6).

B.3 Aggregation, failures, and deployment

All task-level rates use the declared complete inventory. API failure, invalid source, construction failure, correctness failure, unstable timing, resource failure, unresolved effect, and unused budget are separate terminal categories. All categories remain in the fixed cohort denominator; per-family results supplement the overall summary.

For a reference-relative deployment summary, define

$$R_t^{\text{deploy}} = \max\left(1, \max_{c\in\mathcal{E}_B : C(c)\wedge T(c)} R_{\text{center}}(c)\right), \tag{7}$$

where the inner maximum is absent if no generated candidate is eligible and R_{center} is the declared central paired ratio. The leading 1 represents reference fallback, counted separately from generated success. Any geometric mean of R_t^{deploy} is accompanied by generated correctness, threshold success, and fallback rates. This deployment summary is computed from qualified measurement records. Speed among only eligible generated candidates is labeled conditional and shown with its count.

B.4 Costs and time to quality

Three ledgers are maintained: library construction; online model work; and target evaluation. Construction records source reading, judgment outputs, available model usage, and auditable time/cost evidence. Online work records logical calls, physical HTTP attempts, invalid responses, input/output/total tokens where available, correction and repair calls, latency, and route changes. Admission reserves and observed token usage are separate fields. Timeouts with uncertain completion retain an unknown remote-cost status.

Device costs include reserved and dispatched candidate attempts, worker phases, correctness invocations, warmups, timings, failed runs, diagnostics, profiling, and any extra confirmation. Actual device busy time, reservation occupancy, worker duration, and end-to-end elapsed time are distinct. In the documented development records, the exact GPU-busy field is unavailable; worker duration and elapsed campaign time retain their respective definitions. Operational keepalive is disclosed separately from algorithmic evaluation.

For a target quality threshold, tasks still unsuccessful at a declared budget/time cap are cap-censored; terminal failures and unresolved effects are reported in their respective categories. Plots restricted to tasks reaching the threshold are labeled conditional. Baseline curves use observed prefixes of longer trajectories for first-attainment comparisons.

B.5 Task-level analysis and diagnostic scope

The task is the comparison unit. The capability-removal study uses one search per task and removal arm and reuses FULL from the preceding independent run on the same 24 tasks. Same-task comparisons retain the common inventory and each run's execution conditions. Repeated timing samples characterize the measured candidate within that search.

The reported budget curves use observed prefixes of the selected trajectories. Post-hoc coverage anchors define the matched-budget comparisons. Missing prefixes retain their unknown positions, with completion bounds reported where relevant. Equation 8 distinguishes candidate availability from selection; reported coverage follows the recorded search trajectories.

C Knowledge and Execution Artifacts

Experiment map. Appendices C–K organize experimental settings, detailed results and source-level analyses as follows.

Appendix	Contents
C	Knowledge construction, retained artifacts and executed implementation routes.
D	Search configurations, study interventions and service settings.
E	RTX4090 budget prefixes, matched coverage, reporting sensitivity and source audits; full Level 1 results.
F	Source-hash sensitivity and joint correctness–acceleration coverage.
G	Initial-pool size: coverage, token cost and task-level outcomes.
H	H100 coverage, matched budgets and trajectory completion.
I	GLM transfer: coverage, matched resources and implementation differences.
J	Implementation traces, knowledge links and H100 execution provenance.
K	Capability removal, knowledge delivery, content comparisons and fixed-source confirmation.

Construction examples and hardware-conditioning rules follow in Appendix L.

The reproducibility unit includes archived sources, positive and negative records, conditional consumers, prompts, tools, configuration, native protocol and execution records. The active catalogs support the two level-specific consumers. Level 1 has 272 active positive cards/466 options; Level 2 has 405/587 and graph/region delivery. Each negative procedure specifies its trigger, legal exceptions and repair dependencies.

Source records preserve available repository revisions, paths and content hashes. Knowledge construction produces linked artifacts: source evidence, a reusable relationship, conditional bindings, and its generation or assessment view. Positive entries specify actions, conditions, dependent parameters and alternatives. Negative entries specify a trigger, source facts needed to check it, legal exceptions and coupled repair steps. The row-reduction dossier in Appendix L provides a complete example of these fields.

Knowledge construction. *Extraction:* source analysis was conducted interactively in Codex. The retained protocol specifies reading a pinned source span, identifying a local decision, writing its steps, requirements, parameter relationships and alternatives, and separating source observations from teacher proposals. *Independent review:* the documented library reconstruction used a separate GPT-6 Astra agent with maximum reasoning to check specified cards against their evidence. *Revision:* the authors reconciled findings with the pinned sources and updated the records; developers maintained schemas, consumers, constructors and checks. Table 6 summarizes the workflow. The online model chooses and composes routes, writes variants and uses evaluation feedback during search.

For example, the composite-library build retains reduction-axis and output-dtype branches as semantic contracts. It also relabels proposed empty-output skipping, external postactivation and downstream-consumer constructions as teacher proposals where the cited code supplies only a local primitive or scheduler behavior.

Knowledge use has four evidential levels: exposure in a request; declared choice; a supported property of actual source; and native validation. Source inspection assesses conformance beyond request exposure and declared choices. Source hashes join materialization, reports, repair parents, selection,

Table 6: Teacher analysis, author review and conditional knowledge delivery.

Stage	Authoring decision	Retained artifact
Read and scope	Verify the source span; isolate a local choice and its actual call, indexing or state path.	Source revision, file/span hashes, parent record and decision slot.
Form the relationship	Specify requirements, steps, parameter relations, alternatives and choices left to generation.	Source facts and teacher reasoning in separate fields, with evidence for each binding.
Retain or merge	Keep independent choices; attach repeated mechanisms as additional source bindings.	Record-to-card migration and explicit composition/dependency relations.
Review and revise	Check the cited implementation and legal neighboring cases; resolve identified discrepancies.	Scoped review records and revised cards linked to their evidence.
Condition delivery	Bind semantic branches as contracts and apply explicit task/backend/resource conditions to choices.	Shared mechanism catalog, family/region bindings and conditional priorities.

dispatch and receipts. Reuse requires matching context, and children receive new checks. Cross-arm source or result imports are excluded.

Executed implementation routes. We classify the selected Fast_1 winner's executed forward path by whether it uses a library convolution or matrix-operation body followed by a custom kernel, contains a complete custom body, or is framework-equivalent. The classification follows reachable helpers and checks the selected source hash against its native receipt.

Table 7: Static forward-path classification of selected, screened $B = 3$ Fast_1 winners, classified from their executed forward paths.

Panel	Winners	Complete custom	Library body + custom tail	Framework-equivalent
RTX4090, 100 tasks	77	0	77	0
H100, 100 tasks	75	1	71	3
GLM, 24 tasks	23	0	21	2

D Implementation Details and Experimental Settings

D.1 Level-specific search configurations

The shared dual-pruning method uses level-specific search schedules. The release configurations are $N = 16$ for Level 1 and $N = 8$ for Level 2, with three formal candidate positions for both. The Level 1 pool-size comparison in Appendix G characterizes the coverage–compute tradeoff and supports $N = 8$ as a performance–compute compromise. Level 2 uses this value as its fixed initial pool size, evaluated on the full benchmark.

D.2 Knowledge and settings

The inspected Level 1 active positive view contains 272 cards and 466 options; Level 2 contains 405 cards and 587 options. Counts refer to active catalog entries and selectable options. Both consume base negative obligations/procedures; Level 2 adds graph and hierarchical views. Retrieval, request and decision records trace their use.

D.3 Ablation configuration

The capability experiment uses the original 24 source-stratified Level 2 tasks, one independent search for each removal, $N = 8, B = 3$, and two RTX4090 hosts. Its 72 searches close all 216 candidate

Table 8: Level-specific implementation and evaluation settings.

Configuration	Construction and search	Reporting
Level 1, $N = 16$	Family/cross-family routing; retained pool, staged exploit/explore and on-demand investigation; $B = 3$.	Single-operator configuration; outcome records in Appendix E.3.
Level 2, $N = 8$	Hierarchical routing; initial 8 in two waves; scoped inspection, pool comparison, bounded continuation; $B = 3$.	Independent complete Level 2 experiment.
H100 panel	Independent Level 2; at most eight initial candidates, $B = 3$, hardware-conditioned guidance.	100-task panel; execution and completion details in Appendix H.

Table 9: Selected settings; configuration and provenance are recorded for each experiment.

Item	Configuration
Formal budget	At most three candidate positions; controls separately charged.
Initial supply	Level 1: target16; Level 2: at most8 in two initial waves (4+4).
Review	Scoped tools, targeted investigation and source-bound evidence.
Level 2 model	DeepSeek-V4.1-Flash, official service, nonthinking.
Level 2 protections	600k-token warning,4M emergency boundary,48 logical/64 physical requests; emergency ceilings; actual usage is recorded separately.
Level 2 continuation	At most two supply/repair events and one different-body window; no extra formal positions.
Native evidence	Exact-source correctness, sanitizer and timing under frozen hardware, software and numerical permissions.

positions. Six fixed arm orders balance the three removals across tasks. FULL is reused from the preceding four-arm experiment on the identical tasks. All request nonthinking DeepSeek-V4.1-Flash; FULL uses the official service and the new removals use the designated compatible service. Torch 2.6.0+cu124, Triton 3.2.0, the private sanitizer, numerical permissions and native evaluation rules are retained. Table 10 defines the intervention.

Table 10: Capability-removal definitions. Every arm retains public artifact interfaces, the common CUDA bridge, source/receipt identity, native correctness, sanitizer and timing checks, budgets, and process recovery.

Arm	Removed channel capabilities	Retained components
NO_POS	Positive cards, priorities, composition and dependency closure; forced protected route, prefix/tail constructors and predefined slow-body rotation/selection hints.	Negative procedures; generic planning, coding, comparison and repair.
NO_NEG	Failure cards/procedures, error-specific repair tutorials, specialized source/shape/address checks, and automatic Triton, extension and reference-binding patches.	Positive guidance and its necessary implementation conditions; generic investigation and repair.
NEITHER	Both sets of channel-specific capabilities.	Generic model search, full reference inputs and raw same-run feedback.

D.4 KNOWLEDGE DELIVERY BY STAGE

The stage-delivery experiment uses the same fixed 24 tasks, $N = 8, B = 3$, with one search per task and arm across three four-GPU RTX4090 hosts. Each host processes eight tasks and all three arms for each task. The new arms request nonthinking DeepSeek-V4.1-Flash through the compatible

service; FULL-C is the preceding official-service run already defined above. Native evaluation, the private CUDA 12.4 environment and the task cohort are unchanged.

All three stage-delivery arms retain protected-body/tail support, deterministic checks, generic model review and repair, the candidate pool, and native evaluation. Table 11 specifies direct teacher delivery by request. Same-search findings and native feedback can enter subsequent repair in every arm; assessment-only delivery therefore retains the feedback loop while using generic construction prompts.

Table 11: Direct knowledge delivery on the shared execution path. Teacher denotes the corresponding construction or assessment knowledge view; Generic uses ordinary task/source instructions. Capability removals are separately defined in Table 10.

Request	GEN_ONLY	ASSESS_ONLY	GENERIC_PREP
Planning and generation	Teacher	Generic	Generic
Repair	Teacher + feedback	Generic + feedback	Generic + feedback
Source inspection	Generic	Teacher	Generic
Candidate comparison	Generic	Teacher	Generic

D.5 Comparability

Within each campaign, the arms share task IDs, budget prefixes, numerical permissions and native profiles. Their records include provider and model IDs, dates, tokens, cache usage and retries. Worker time, device time and model waiting are separate resource measures. No historical candidate, winner or measured feedback enters a new ablation search. Slot budgets, total cost and elapsed time are reported on separate axes.

Compared search policies. All comparison arms are local shared-protocol implementations of the named strategies, evaluated with the common task interface and native evaluator. Best-of-N samples independent candidates in a fixed order; Iterative revises after each execution result. OpenEvolve retains a bounded fitness/novelty population for parent selection and mutation. STARK-port maintains a search tree with planning, coding and debugging roles. EGG-port schedules two seeds, algorithmic and mapping branches, three tiling choices, memory refinement and final tuning. Target-device queries for the latter two consume their allotted evaluator positions. The released ports specify these prompts, branch schedules and evaluator permissions. The baseline campaign runs a ten-position policy; reported $B = 1$ through $B = 10$ curves are prefixes of that policy. Our controller is configured for three positions. Budget matching uses the first observed crossing in those recorded trajectories.

E Full-Benchmark Results, Budget Matching and Sensitivity

The mainline rows come from the independent full-benchmark experiment, covering 100 tasks and 300 closed positions, with one selected trajectory per task. The baseline snapshot selects exactly one trajectory for each of five methods and 100 tasks, with 4,994 closed positions. Two remaining outcomes are unknown: Best-of-N task 72 has nine positions, and STARK task 78 has five. All three mainline prefixes and all baseline prefixes through five are complete. The baseline implementations are shared-protocol ports; runtime dates and some driver versions differ.

Mainline model and execution cost. The 100-task mainline records 94,762,793 reported online tokens and 2,031 HTTP attempts; five attempts have unknown remote usage and are excluded from that token sum. The native receipt wall-time field sums to 45,499.63 seconds (12.64 hours), the common resource field used in Table 2. A separate worker-process elapsed-time field totals 46,148.13 seconds (12.82 hours). Both fields are retained in `RTX_MATCHED_PREFIX_COSTS.csv`; receipt time is used consistently for cross-method comparison. Teacher construction and operational controls have separate accounts.

Reporting rule and strict acceleration. The reporting rule in Section 5 excludes 57 stable positions in 15 method–task trajectories. The source audit below records the affected implementations and

Table 12: First observed baseline budget / percentage of slots saved by ours at matched Fast_1. For example, $B = 2$ versus OpenEvolve $B = 7$ gives $1 - 2/7 = 71.4\%$. Targets are selected after observing our run.

Target coverage Our budget	46% $B = 1$	72% $B = 2$	77% $B = 3$
Best-of-N	3 / 66.7%	6 / 66.7%	9 / 66.7%
Iterative	2 / 50.0%	4 / 50.0%	5 / 40.0%
OpenEvolve	2 / 50.0%	7 / 71.4%	9 / 66.7%
STARK-port	3 / 66.7%	5 / 60.0%	8 / 62.5%
EGG-port	2 / 50.0%	6 / 66.7%	7 / 57.1%

Table 13: Full Fast_1 prefix curves (%). All tasks remain in the denominator.

Method	$B = 1$	$B = 2$	$B = 3$	$B = 4$	$B = 5$	$B = 6$	$B = 7$	$B = 8$	$B = 9$	$B = 10$
Best-of-N	22	41	52	59	67	73	75	75	79	79
Iterative	30	55	63	74	79	85	87	91	93	94
OpenEvolve	20	50	60	67	69	71	75	76	78	78
STARK-port	16	40	57	68	74	76	76	78	81	81
EGG-port	32	47	56	62	71	75	77	80	80	84
Ours	46	72	77	—	—	—	—	—	—	—

their original numerical verdicts. For strict $\text{Fast}_{1.05}$, ours obtains 32/41/45 tasks at $B = 1/2/3$. The 45-task target is first reached by EGG/STARK/Iterative/OpenEvolve at 5/5/9/10, corresponding to 40–70% fewer candidate-evaluation slots. Best-of-N does not reach that strict target within ten slots.

Separating supply from selection. For a fixed diagnostic pool, let A denote existence of a target-quality candidate and S its selection within budget. Since S implies A,

$$\Pr(S) = \Pr(A)\Pr(S \mid A). \tag{8}$$

E.1 Coverage sensitivity to the reporting screen

Across 300 mainline and 4,994 baseline positions, the eligible, numerically passing candidates with $S \geq 10$ occupy 71 baseline positions: 57 timing-stable and 14 without timing eligibility. Both groups lie within the same eight tasks: 23, 42, 44, 50, 72, 77, 80 and 96. This inventory includes all numerical successes relevant to Fast_1, before imposing timing stability.

Let $f_m(B)$ be a baseline's screened task count and E the union above. Counting every affected task as an additional success for every baseline, at every prefix, gives the upper bound $\min\{100, f_m(B) + |E|\}$. It includes already-successful tasks and positions later than B, deliberately allowing more gains than reinstatement can add. Table 15 applies this bound to the strongest baseline at each prefix.

Table 14: Stability sensitivity: require timing eligibility in addition to Fast_1. Our targets are unchanged, but some baseline crossings move.

Target coverage Our budget	46% $B = 1$	72% $B = 2$	77% $B = 3$
Best-of-N	3 / 66.7%	7 / 71.4%	9 / 66.7%
Iterative	2 / 50.0%	4 / 50.0%	5 / 40.0%
OpenEvolve	2 / 50.0%	7 / 71.4%	9 / 66.7%
STARK-port	3 / 66.7%	5 / 60.0%	8 / 62.5%
EGG-port	3 / 66.7%	6 / 66.7%	8 / 62.5%

Table 15: Conservative Fast_1 task-count bounds on the same 100 tasks. Every baseline receives eight additional successes at every prefix. The added successes are an analytical allowance, not measurements.

Budget	Ours	Best screened baseline	Baseline upper bound	Margin
$B = 1$	46	32	40	6
$B = 2$	72	55	63	9
$B = 3$	77	63	71	6

Thus the RTX4090 early Fast_1 ordering is preserved under this allowance. It also bounds the budget needed to match our 77-task target. Write $\bar{f}_m(b) = \min\{100, f_m(b) + 8\}$ and let $b_m^+(77)$ be the first budget attaining 77 tasks when screened successes are reinstated within the recorded trajectory. Inverting the count upper bound gives

$$\underline{b}_m = \min\{b : \bar{f}_m(b) \geq 77\}, \qquad b_m^+(77) \geq \underline{b}_m, \qquad 1 - \frac{3}{b_m^+(77)} \geq 1 - \frac{3}{\underline{b}_m}. \tag{9}$$

Table 16 evaluates these lower bounds from Table 13. For example, Iterative has at most 71 tasks at three positions under the allowance; Best-of-N has at most 75 at five. Their first possible target budgets are therefore at least four and six.

Table 16: Analytical lower bounds on baseline budget and our slot savings for 77 Fast_1 tasks, allowing every baseline eight extra successes at every prefix. The second column is the last insufficient count upper bound; the final two columns give the first possible budget and the corresponding minimum saving from our three evaluations.

Method	$\bar{f}_m(\underline{b}_m - 1)$	$\underline{b}_m$	$1 - 3/\underline{b}_m$
Iterative	71	4	25.0%
OpenEvolve	75	5	40.0%
STARK-port	76	5	40.0%
EGG-port	70	5	40.0%
Best-of-N	75	6	50.0%

The method-specific saving lower bounds are 25%, 40% or 50%, with at least 25% against every baseline on these recorded Fast_1 trajectories. The measured crossings in Table 12 give the primary 40.0–66.7% savings.

E.2 Source audit of high-magnitude candidates

KernelBench flags excessive speedups with a default warning threshold of $10\times$ (KernelBench Contributors, 2026); documented benchmark-bypass failures also motivate source inspection (Sakana AI, 2025). We inspect the excluded implementations below. We examined the active source paths and saved correctness records for all 57 timing-stable excluded positions (54 distinct source hashes) across eight tasks. Task 42 provides a concrete failure cluster: nine positions report 16.7–$17.5\times$ and pass the recorded numerical tolerance, yet their source formulas differ from the reference. With stride one and zero padding, the required spatial mean of the transposed convolution is

$$\bar{y}_{n,o} = b_o + \frac{\sum_i \left(\sum_{h,w} x_{n,i,h,w}\right)\left(\sum_{k,l} W_{i,o,k,l}\right)}{H_{\text{out}} W_{\text{out}}}. \tag{10}$$

Table 17 identifies the two incorrect shortcuts and the corresponding independent algebraic counterexamples.

All nine pass the recorded numerical tolerance. The independent witnesses demonstrate non-equivalence to the reference formula and show how apparent high-magnitude speedups can survive sampled correctness checks.

The H100 baseline return contains the same error pattern on task 42: Best-of-N position six reports $33.452\times$ with numerical, sanitizer, and timing checks marked as passed in the returned summary, but

Table 17: Level 2 task 42: high-magnitude, protocol-passing implementations with source-level formula errors. Witness errors come from independent float64 CPU algebraic probes.

Source group	Positions	Recorded S	Deviation from Eq. 10	CPU witness max. error
Best-of-N and STARK-port	6	16.73–17.52×	Uses the input spatial mean without the input/output-area factor.	0.4899
OpenEvolve	3	17.02–17.17×	Divides the convolution bias by output area instead of adding it directly.	1.4466

its source uses the input spatial mean in place of the output-area-scaled sum in Eq. 10. The same formula error therefore appears on both devices.

We also screened the active forward paths of all 47 mainline winners above $1.05\times$, inspected eight tails in depth, and inspected the three fastest below-cutoff winners per baseline. This inspection identifies a task 72 implementation that folds BatchNorm weights but uses the original weights in its effective kernel. The source-specific quarantine below covers identified formula deviations at any speedup magnitude.

E.3 Level 1 results with an initial pool of 16

Table 18 reports the Level 1 configuration with an initial candidate target $N = 16$ and three positions per selected search. Candidate-count comparisons are reported in Appendix G.

Table 18: Level 1, $N = 16, B = 3$. Means use 90 and 85 tasks, respectively; these are alternative eligibility summaries of the same selected searches.

Outcome	Numerical	Stable	Strict Fast	AM	GM
Selected $B = 3$, all numerical successes	90	—	28	1.358	1.094
Selected $B = 3$, timing-eligible only	90	85	28	1.375	1.095

Five numerical successes (7/9/42/61/63) lack timing eligibility. Median-only speedups exceed 1.05 on 31 tasks, whereas strict acceleration is 28. The selected searches report 160,632,037 tokens; the retained original/replay/diagnostic/new search chain reports 184,812,835 tokens and 353 candidate/diagnostic attempts, apart from operational controls. Task-level run origins, follow-up context and cumulative costs are retained in the reproduction records.

F Source Sensitivity and Joint Coverage

Full-cohort source-hash sensitivity. The source audit identifies eight distinct source hashes spanning nine positions on task 42 with a formula difference, and one hash at STARK-port task 72 position three that uses the original rather than the folded BatchNorm weight in its effective kernel. We quarantine these identified hashes symmetrically, retain every task and position, and select from the remaining measured candidates. This source-path challenge is separate from the uniform magnitude screen and quantifies the effect of the identified algebraic differences.

The performance columns remain unchanged from the primary $B = 3$ comparison; the source-specific exclusions reduce Best-of-N and STARK-port protocol-correct coverage by one task each.

F.1 Joint coverage across hardware and model settings

Figures 5 and 6 compare two coverage objectives at the same three-position budget. Upper-right points provide greater coverage on both axes. A method is nondominated when no other observed method is

Table 19: Full-100 source-hash sensitivity, with the primary magnitude screen and ten identified positions quarantined. All task denominators and original candidate-position indices are retained. Correct/100 denotes recorded protocol success after the stated post-hoc source exclusions.

Method at $B = 3$	Correct/100	Fast_1/100	Strict $\text{Fast}_{1.05}$/100
Best-of-N	72	52	13
Iterative	95	63	29
OpenEvolve	79	60	28
STARK-port	77	57	34
EGG-port	72	56	39
Ours	90	77	45

at least as good on both axes and strictly better on one. The fronts are computed independently for each setting.

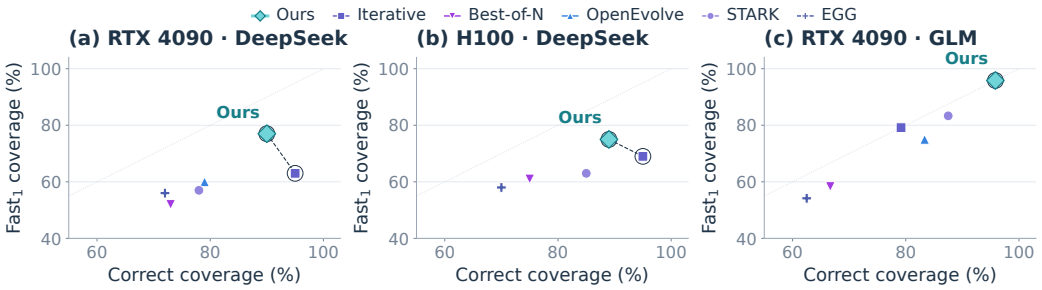

Figure 5: Correctness and Fast_1 coverage at $B = 3$. Ours and Iterative form the observed Pareto set on the two DeepSeek-V4.1-Flash settings; ours leads both axes under GLM. The denominators are 100, 100, and 24 tasks. Outlined points and dashed links identify the observed non-dominated set.

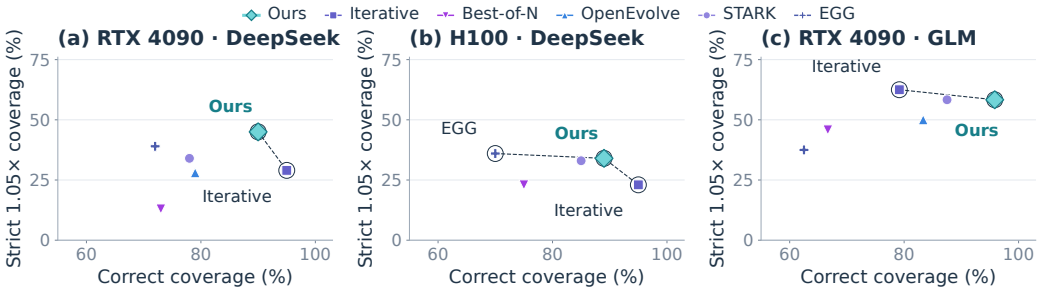

Figure 6: Correctness and strict acceleration at $B = 3$, using the same task denominators as Figure 5. Strict acceleration additionally applies the recorded timing qualification. Ours and Iterative form the observed frontier on RTX4090 with either online model; EGG, ours, and Iterative form the H100 frontier.

G Candidate-Pool Size and Model Computation

G.1 Coverage at a Fixed Device Budget

We compare initial targets $N = 4, 8, 16$ on the same 24 Level 1 tasks, with three formal candidate-evaluation positions per search. The cohort covers matrix multiplication, pointwise operations,

normalization, pooling, reductions, convolutions, scans, losses and attention. Its task IDs are 1, 3, 6, 12, 19, 23, 26, 33, 35, 40, 42, 45, 47, 49, 50, 51, 54, 57, 67, 89, 90, 94, 95 and 97. All 72 positions close in each selected group.

Table 4 in the main text summarizes the final coverage and reported costs.

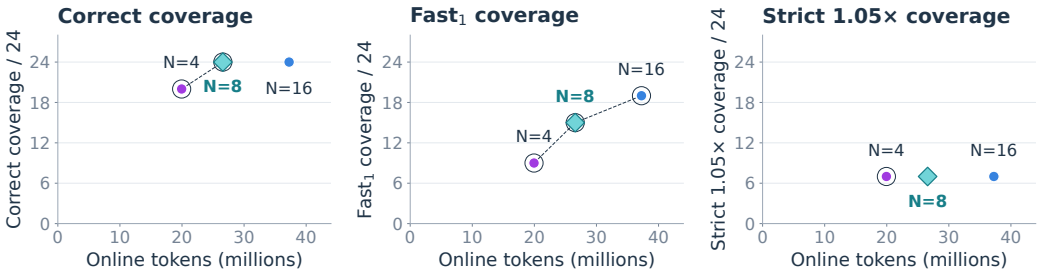

Figure 7: Model computation and task coverage for $N = 4, 8, 16$ at $B = 3$. Each panel uses the same 24 Level 1 tasks and reported online tokens. Outlined points and dashed links mark the observed Pareto set, favoring lower cost and higher coverage. $N = 8$ matches $N = 16$ correctness with 28.6% fewer tokens; $N = 16$ has higher Fast$_1$, while all three settings have seven strict accelerations. Execution conditions are given below.

The $N = 4$ setting uses 19.943 million tokens, 25.0% fewer than $N = 8$, but returns four fewer correct tasks and six fewer Fast$_1$ successes. $N = 8$ and $N = 16$ both achieve correctness on all 24 tasks; $N = 8$ uses 28.6% fewer reported tokens, with 15 and 19 Fast$_1$ successes, respectively. Both achieve seven strict accelerations. On the 20 tasks correct in all three groups, GMs are $1.038\times$, $1.153\times$ and $1.216\times$. We use $N = 8$ as the fixed setting for Level 2.

Table 20: Correct/Fast$_1$/strict counts along each search prefix. All values count tasks out of 24.

Initial pool	$B = 1$	$B = 2$	$B = 3$
$N = 4$	14/7/6	19/8/7	20/9/7
$N = 8$	23/11/3	24/14/5	24/15/7
$N = 16$	21/14/3	24/19/6	24/19/7

The additional supply is most visible before the final slot: $N = 16$ already reaches 19 Fast$_1$ tasks at $B = 2$, compared with 14 for $N = 8$ and eight for $N = 4$. The smaller pool also produces early strict gains: $N = 4$ has six strict successes at $B = 1$, versus three for each larger pool. The equal final strict counts arise from different search paths and successful task sets.

G.2 Which implementations change?

Figure 8 shows the complete task inventory. Four $N = 4$ searches (57, 67, 90 and 97) produce no correct implementation within the budget. Large-K matrix multiplication (task 6) records $0.131\times$, compared with $1.008\times$ for $N = 8$.

Relative to $N = 8$, $N = 16$ gains Fast$_1$ on tasks 45, 50, 51, 54 and 67, and loses it on task 47. Four additions (50, 51, 54 and 67) lie at 1.001–$1.006\times$: they mainly recover from slower implementations to approximately reference performance. For example, task 54 moves from $0.233\times$ to $1.005\times$, while task 67 moves from $0.503\times$ to $1.006\times$. Task 45 changes from $0.9995\times$ to $1.077\times$.

Larger gains remain task-dependent. LayerNorm (task 40) improves from $3.547\times$ with $N = 8$ to $5.101\times$ with $N = 16$, whereas MaxPool2d (42) and cross entropy (95) favor $N = 8$: $2.102\times$ versus $1.012\times$ and $2.745\times$ versus $1.924\times$, respectively. The strict-success sets are 23/40/42/45/89/94/95 for $N = 4$, 23/40/42/89/90/94/95 for $N = 8$, and 23/40/45/89/90/94/95 for $N = 16$.

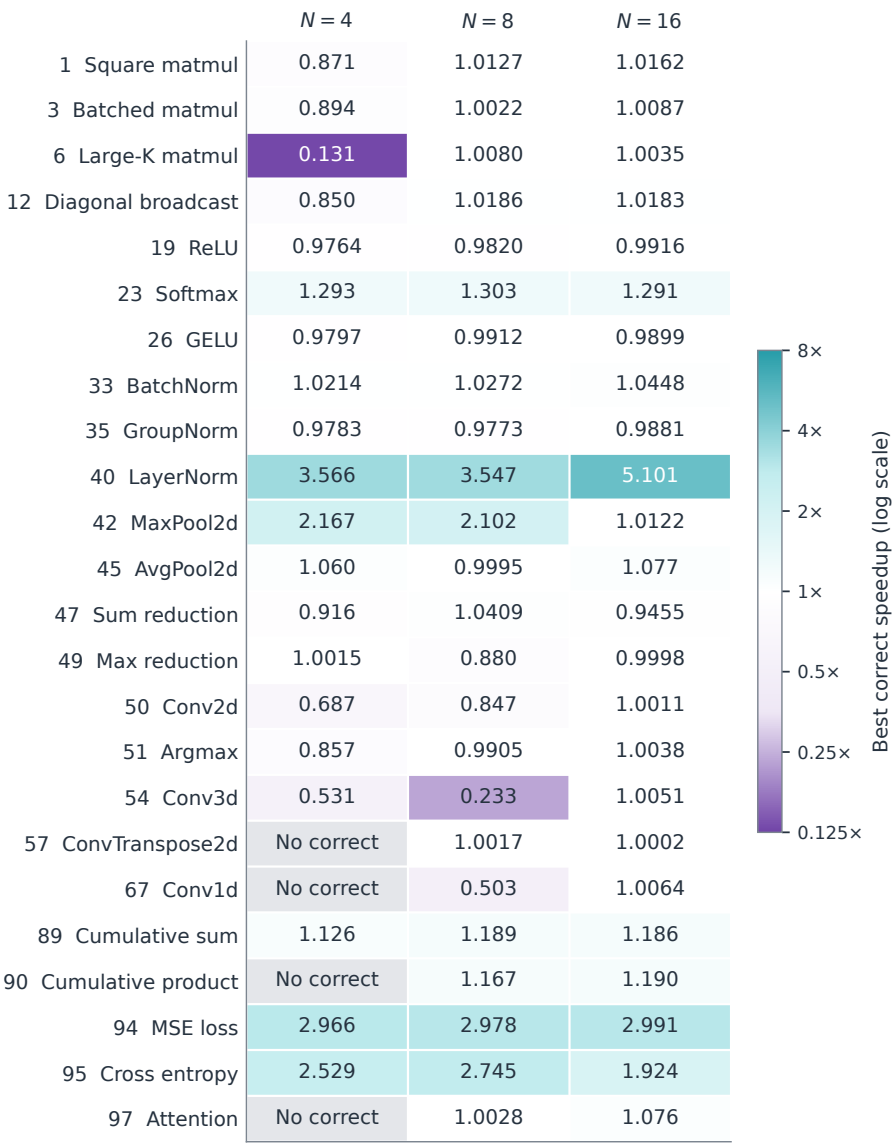

Figure 8: Best correct central speedup for all 24 tasks within $B = 3$. Color encodes the logarithm of speedup around $1\times$; grey denotes no correct implementation. Numerical labels preserve near-reference differences. The strict timing criterion is reported separately in Table 4.

Table 21: Low-budget coverage on H100, Level 2 (%; all 100 tasks). Fast_1 follows Equation 2; Strict uses the timing-qualified $\text{Fast}_{1.05}$ criterion. Budget prefixes index selected trajectories; correctness and incomplete-search records are retained. Bold marks the highest coverage in each column; pp denotes percentage points.

Method	Fast_1@1	Fast_1@2	Fast_1@3	Strict@3	Correct@3
Best-of-N	30	51	61	23	75
Iterative	28	52	69	23	**95**
EGG-port	38	52	58	**36**	70
STARK-port	20	49	63	33	85
Ours	**47**	**70**	**75**	34	89
Fast_1 lead (pp)	**+9**	**+18**	**+6**	—	—

G.3 Execution conditions and accounting

N denotes the initial candidate target; adaptive generation and repair add later candidates. The $N = 4$ and $N = 8$ groups materialize 239 and 361 candidates over their searches. The corresponding reduction in reported tokens is 25.0%. The $N = 4$ and $N = 8$ configurations share the same search controller and knowledge; the supplied runs also include the common compiler-abort handling needed to retain completed outcomes.

$N = 4$ is the new 24-task campaign; $N = 8$ and $N = 16$ use the same-task subsets of their existing outcome records. The $N = 16$ subset comprises 22 initial-run records and follow-up records for tasks 19 and 89. All 24 reference-source hashes match across groups. Runs occur in different batches on RTX4090 hosts; the four $N = 4$ hosts use driver versions 595.71.05, 570.124.06, 550.163.01 and 595.71.05, with matching Torch 2.6.0+cu124, Triton 3.2.0 and sanitizer binaries.

Token totals cover reported usage in the selected searches; operational controls and unknown remote usage are separate. For $N = 16$, earlier unsuccessful work preceding the selected follow-ups is retained in the campaign ledger rather than included in this selected-run token panel.

The completed $N = 8$ Level 1 full-benchmark run supplies additional context: all 100 tasks and 300 positions close, yielding 90 correct tasks, 87 timing-stable tasks, 47 Fast_1 successes and 28 strict accelerations. This full run reports 110.489 million tokens and 2,468 HTTP requests. The 24-task analysis uses its corresponding subset without selecting a different candidate outcome. Level 1 retains $N = 16$ for the broader observed coverage. The Level 1 comparison supports the $N = 8$ performance–compute compromise used for Level 2; the composite-task evaluation tests the method with that fixed setting.

H H100 Coverage and Evaluation-Budget Frontier

Cross-device execution and reporting. The H100 panel compares all 100 Level 2 tasks, using at most eight initial candidates and three candidate-evaluation slots for ours, and up to ten slots for Best-of-N, Iterative, EGG-port and STARK-port. The returned baseline configuration identifies H100 80GB, a local DeepSeek-V4.1-Flash deployment, temperature 0.7, and an 8,192-token output limit. Model serving and runtime conditions differ from the RTX4090 panel; these are separately executed system comparisons under the documented device and service settings. The H100 mainline aggregate preserves its original and follow-up execution provenance in the supplied result data.

All H100 performance tables use the screened Fast_1 of Equation 2. The screen removes 43 baseline positions: 6 Best-of-N, 23 Iterative, 1 EGG-port and 13 STARK-port, and none from ours. Tasks, original position indices, other candidates and correctness verdicts are retained. Ours remains at 47/70/75 Fast_1 tasks. Strict $\text{Fast}_{1.05}$ additionally requires timing eligibility and the frozen minimum seed/reference-bracket ratio to exceed 1.05. All metrics keep the full 100-task denominator.

Figure 3b places all observed H100 budgets on the evaluation–coverage plane. Ours occupies the frontier at $B = 1, 2, 3$; Iterative extends it at $B = 4, 5, 6, 7, 9$. The horizontal axis counts selected-

Table 22: Matched-coverage selected-trajectory budgets on H100. Savings equal $1 - 3/b_m(75\%)$. Crossings use observed prefix values. EGG-port does not reach the target in its observed positions.

Method	First observed budget for 75% Fast_1	Slots saved by ours
Ours	**3**	—
Iterative	4	**25.0%**
STARK-port	6	**50.0%**
Best-of-N	7	**57.1%**
EGG-port	Not observed through $B = 10$	—

trajectory candidate positions; model computation and offline construction are treated as separate resource categories.

Early coverage and stricter performance. At $B = 2$, ours adds 23 accelerated tasks beyond its first evaluation, reaching 70% versus 52% for the strongest baseline. A third evaluation raises coverage to 75%. Table 23 separates this broad acceleration coverage from correctness and larger, timing-qualified gains. At $B = 3$, Iterative has the highest correctness coverage, while EGG-port and STARK-port obtain 36 and 33 strict accelerations, respectively, compared with 34 for ours. The most pronounced joint correctness and strict-acceleration advantage for ours occurs at the first evaluation: 77 versus 54 correct tasks and 28 versus 18 strict accelerations, comparing against the respective strongest baseline.

Table 23: H100 correctness and acceleration criteria (tasks out of 100). Strict acceleration uses the same frozen timing qualification across methods.

Method, $B = 3$	Correct	Stable correct	Fast_1	Strict $\text{Fast}_{1.05}$
Best-of-N	75	72	61	23
Iterative	95	93	69	23
EGG-port	70	68	58	36
STARK-port	85	84	63	33
Ours	89	88	**75**	34

Table 24: Full H100 Fast_1 prefix curves (%); column numbers denote budget B. Dashes indicate unrun budgets.

Method	1	2	3	4	5	6	7	8	9	10
Best-of-N	30	51	61	65	71	72	75	78	85	85
Iterative	28	52	69	80	85	88	89	89	91	91
EGG-port	38	52	58	62	67	69	69	69	70	70
STARK-port	20	49	63	72	74	77	78	81	81	81
Ours	47	70	75	—	—	—	—	—	—	—

Completion and evidence records. Ours has 292 recorded positions of 300, with five searches incomplete. The four baselines have 3,546 recorded positions of 4,000 and 318 complete ten-position trajectories of 400. Table 25 retains each method's completion status.

Sensitivity to remaining measurements. Holding recorded results fixed, suppose every baseline task with an incomplete prefix and no observed acceleration becomes accelerated when its missing positions are filled. The maximum coverage over the four baselines would then be 42%, 57%, and 71% at $B = 1, 2, 3$, respectively, versus our observed 47%, 70%, and 75%. These gap-completion bounds retain the low-budget Fast_1 lead under all assignments of the missing outcomes.

Table 25: H100 completion status. Coverage uses the complete 100-task inventory.

Method	Final budget	Complete searches / 100	Recorded positions
Ours	3	95	292 / 300
Best-of-N	10	89	935 / 1,000
Iterative	10	90	950 / 1,000
EGG-port	10	66	783 / 1,000
STARK-port	10	73	878 / 1,000

I ONLINE-MODEL TRANSFER ON GLM

Protocol. This panel compares the complete framework with five baseline algorithms, using GLM-5.3-Flash for every method on the same 24 Level 2 tasks: 1, 3, 6, 8, 9, 10, 11, 14, 17, 18, 20, 28, 35, 36, 45, 52, 55, 72, 79, 83, 89, 91, 92 and 98. Ours uses an initial pool target of eight and three candidate-evaluation positions; baselines use up to ten. The service configuration enables thinking with high reasoning effort and an output ceiling of 65,536 tokens. Method-specific prompts and inference schedules are retained. This panel compares algorithms under a shared model configuration. The main metric is Fast_1 on all 24 tasks. Correctness retains the native protocol verdict.

I.1 SMALL-BUDGET COVERAGE AND MATCHING BUDGETS

Ours obtains 23 correct and 23 accelerated tasks within three evaluations, compared with at most 21 correct and 20 accelerated tasks for a baseline (Table 26). The Fast_1 margin is 12.5 percentage points; strict acceleration distinguishes smaller measured gains from more substantial improvements (Figure 6c). Ours reaches 14 strict accelerations, Iterative reaches 15, and STARK-port reaches 14.

Table 26: GLM comparison at $B = 3$. Strict denotes the conservative $\text{Fast}_{1.05}$ timing criterion. Closed-prefix counts describe measurement completeness; correctness is reported in its own column. All metrics keep the 24-task denominator.

Method	Correct/24	Fast_1/24	Strict/24	Closed B3 prefixes/24
Ours	23	23	14	23
Best-of-N	16	14	11	23
Iterative	19	19	15	23
OpenEvolve	20	18	12	24
EGG-port	15	13	9	24
STARK-port	21	20	14	23

Figures 5c and 6c show the joint coverage tradeoffs under GLM. Ours leads correctness and Fast_1; under the strict threshold, ours and Iterative form the observed frontier.

Ours reaches 16, 20 and 23 Fast_1 tasks at $B = 1, 2, 3$. At the 23-task target, Best-of-N and Iterative first match at $B = 8$, and OpenEvolve at $B = 9$; these correspond to 62.5%, 62.5% and 66.7% fewer evaluation slots (Table 27). These three crossings remain unchanged under the recorded completion bounds. STARK-port first reaches 23 at its observed $B = 7$; its earlier prefixes include missing positions. EGG-port has 20 accelerated tasks in its observed $B = 10$ prefix. Matching uses aggregate accelerated-task counts at observed budgets; online costs are reported separately.

Figure 3c shows the observed GLM budget–coverage frontier. Ours reaches its 23-task coverage at $B = 3$; later measured baseline points complete the comparison.

I.2 COMPLETION AND RESOURCE ACCOUNTING

The selected records contain 135 completed trajectories out of 144 and 1,223 closed positions out of a planned maximum of 1,272. The final continuation adds five positions to OpenEvolve task 6, preserving its earlier source and measurement records. Table 29 reports completion and recorded

Table 27: First observed budgets matching our 23-task Fast$_1$ coverage at $B = 3$. Missing positions retain their unmeasured status.

Method	First observed B	Slot saving
Best-of-N	8	62.5%
Iterative	8	62.5%
OpenEvolve	9	66.7%
EGG-port	—	—
STARK-port	7	57.1%

Table 28: Recorded resources across the same 24 tasks through each method's first observed prefix reaching 23 Fast$_1$ tasks. Worker hours sum native GPU-worker receipt durations across workers. Tokens count reported usage only: 12 of our intents lack a response receipt, and some baseline responses omit usage. Startup probes and superseded searches are accounted for separately.

Method	First B	GPU receipts	Worker hours	Tokens (M)	HTTP intents
Ours	3	69	2.74	18.569	437
Best-of-N	8	177	5.35	2.572	223
Iterative	8	175	7.91	3.049	230
OpenEvolve	9	205	9.26	3.736	256
STARK-port	7	151	6.16	3.063	313

online usage. Earlier superseded searches add 302,370 reported tokens and 14 HTTP intents; the deduplicated observed task-API total is 39,414,449 tokens and 2,189 intents. Startup probes and remote calls without usage receipts are outside that reported-token sum.

Table 29: Observed outcomes at each method's maximum budget and reported online tokens for the selected trajectories.

Method	B	Complete/24	Positions	Fast$_1$/24	Strict/24	Tokens (M)
Ours	3	23	69	23	14	18.569
Best-of-N	10	23	232	24	19	3.205
Iterative	10	21	225	23	18	3.799
OpenEvolve	10	24	240	23	18	4.268
EGG-port	10	23	238	20	17	4.313
STARK-port	10	21	219	23	18	4.958

Unclosed records are Best-of-N task 89 (2 positions), Iterative tasks 11 (7), 3 (8) and 72 (0), STARK-port tasks 72 (0), 36 (4) and 79 (5), EGG-port task 55 (8), and ours task 89 (0). Ours task 89 ends after two unusable planning responses. Iterative task 72 ends with an output contract failure; other incomplete records include transport interruptions, cancelled execution, a lane failure and an unstarted task. These records retain an unverified status at unobserved positions. On the common 22 tasks with closed B3 prefixes for all methods, ours obtains 22 Fast$_1$ tasks, compared with at most 19 for a baseline. On the full 24-task cohort, even assigning success to every missing baseline B3 position yields at most 21 Fast$_1$ tasks, below our observed 23.

Fourteen correct candidate positions reach the magnitude cutoff: one from ours, two Best-of-N, five OpenEvolve, four EGG-port and two STARK-port. Removing them changes none of the six methods' B3 coverage counts. On our task 72, the excluded high-magnitude candidate does not affect coverage: the other two candidates obtain approximately $1.225\times$ and $1.227\times$, preserving its coverage contribution.

I.3 Online realization and model transfer

The online model uses knowledge and feedback to construct, repair and select implementations. Of our 69 closed positions, 55 pass the correctness protocol. Correct task coverage rises from 17 at the first position to 23 at the third. Task 35 fails twice on a Triton global-constant access before a third implementation passes at approximately $1.59\times$. Task 9 improves from $0.41\times$ to $1.02\times$, illustrating that restoring a useful implementation and obtaining substantial acceleration are separate outcomes. The 23 Fast_1 tasks comprise 14 strict accelerations and nine additional central-ratio successes. Transport errors are recorded separately from these code-generation and performance outcomes.

For a secondary descriptive comparison, we extract the identical 24 task identifiers from the historical DeepSeek-V4.1-Flash benchmark records (Figure 9). Ours increases from 16 to 23 Fast_1 tasks; the baselines are all unchanged or higher, and STARK-port increases from 12 to 20. Strict coverage rises from 8 to 14 for ours. The GLM run uses thinking/high while the historical DeepSeek-V4.1-Flash run uses nonthinking; service and run conditions also differ.

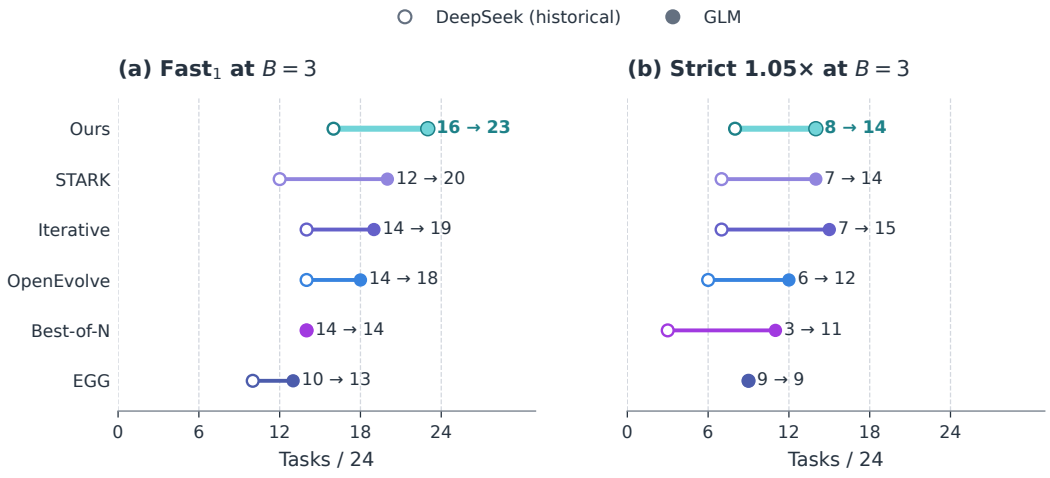

Figure 9: Secondary comparison on the same 24 task identifiers, using the uniform candidate cutoff. Open circles denote the historical DeepSeek-V4.1-Flash configuration; filled circles denote GLM. The primary algorithm comparison is within each online-model configuration.

J Tracing Early Coverage to Implementation Choices

J.1 Where the second evaluation helps

The H100 transition from 47 to 70 accelerated tasks contains 18 tasks whose first candidate was correct but not faster, and five without a correct first candidate. Seventeen additions are Linear composites; the other six comprise three ConvTranspose3d and one each of ConvTranspose2d, Conv3d and Conv2d. Generation keys identify 19 second-slot winners as members of the initial pool and four as targeted repairs. On RTX4090, the corresponding 26 additions comprise 15 previously correct but unaccelerated tasks and 11 without a correct first result.

At H100 $B = 2$, paired Fast_1 outcomes are 32 ours-only versus 13 Best-of-N-only tasks; 28 versus 10 for Iterative; 30 versus 12 for EGG-port; and 33 versus 12 for STARK-port. Of the ours-only tasks, 30, 27, 25 and 33 respectively already have complete baseline prefixes.

Coverage and gain magnitude have different sources. Linear achieves 27/37 Fast_1 coverage but one strict acceleration. Conv2d reaches 15/21 and 14/21. At $B = 3$, the 75 H100 Fast_1 tasks split into 36 with best central ratio in $(1, 1.05]$, five above 1.05 without strict qualification, and 34 strict successes. The RTX4090 split is 30, two and 45.

Table 30: H100 coverage at $B = 2$. Strict denotes timing-qualified $\text{Fast}_{1.05}$. Each baseline column is the maximum among four methods in that family and metric.

Family	Tasks	Our Fast_1	Best baseline	Our strict	Best baseline
Linear	37	27	17	1	4
Conv2d	21	15	13	14	8
Conv3d	12	9	8	5	6
ConvTranspose2d	11	7	8	4	6
ConvTranspose3d	19	12	14	8	7

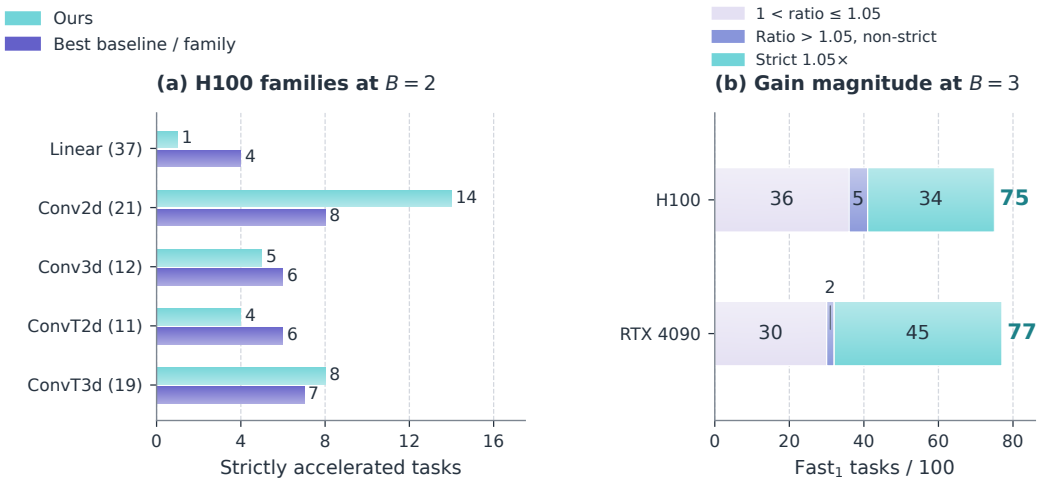

Figure 10: Family-level strict gains at $B = 2$ and the decomposition of our Fast_1 coverage. All 100 tasks are retained; the strongest baseline is selected separately within each family.

J.2 SOURCE-LEVEL REALIZATION

Access estimates count logical elements in the source; speedups measure end-to-end execution.

Fused pooling tail with retained convolution (task 82). Our H100 implementation retains library Conv2d, then fuses tanh, scaling, channel bias and 4×4 max pooling. The first result is $1.626\times$. The second changes window traversal from 16 scalar-offset load expressions and sequential maxima to four row-vector loads with within-row reductions, reaching $1.728\times$. Both read the same 16 values per output; this changes access organization and reduction structure while preserving the element count. Iterative's best first-two-slot result is $0.996\times$: its effective source performs separate eager tanh, multiplication, bias and pooling. Best-of-N, Iterative and EGG-port first record acceleration on this task at their third positions.

One-pass channel reduction (task 92). After library convolution and a GroupNorm statistics kernel, our tail fuses normalization, tanh, HardSwish, residual addition and channel logsumexp. The first candidate traverses channels twice, for the maximum and then the exponential sum. The next uses a running maximum and rebased sum in one traversal, improving from $1.882\times$ to $2.192\times$. The source removes one full reread of the $128 \times 64 \times 126 \times 126$ FP32 activation (520.22 MB of logical read demand) and a repeated nonlinear chain, while adding rebase exponentials. The statistics kernel and output domain stay fixed.

Shared route, different realization (task 57). Our successful second candidate retains library convolution and fuses ReLU/HardSwish in one tail, reaching $1.855\times$ with strict qualification. EGG-port produces a similar in-place tail at its first position, with central ratio $1.927\times$ and unstable timing. The two sources share the fused route and differ in their recorded timing stability.

Recovering a useful pool member (task 39). The three ratios are $0.294\times$, $1.011\times$ and $0.296\times$. The second candidate belongs to the initial pool: it retains library GEMM and uses a tool-materialized scale/evaluation-BatchNorm tail. Keeping the measured best preserves it after the third attempt. Iterative reaches $1.016\times$ at $B = 2$ through affine folding.

J.3 KNOWLEDGE LINKS AND EXECUTION PROVENANCE

Candidate records associate knowledge-view hashes, mechanism identifiers, generation actions, source hashes and native measurements. Task 82 names backend-plus-custom construction, positional scalar capture and shared pointwise domains. Task 92 additionally names normalization and stable reduction relations, including rebasing when the running maximum changes. These relationships appear in the measured code. Task 39's materialized tail records the shared tool as its construction source, with no mechanism IDs.

A complete RTX4090 knowledge-to-receipt trace (task 82). The conditional generator view contains backend-plus-custom construction (CV01), a shared output domain (PW01), and path closure across the fused region (X01). The recorded second-wave generation request includes these expanded positive records in its execution contract; its returned candidate is identical to the stored generation artifact. The selected source preserves `self.conv(x)` and places pooling, tanh, scale and channel bias in one Triton tail, mapping the output domain in 8×64 tiles with boundary masks. The third native position binds the same source hash to a passing correctness, sanitizer and stable-timing receipt: 12.6541 ms versus 26.9024 ms for the reference, or $2.126\times$. The first two positions remain recorded failures; the selected source belongs to the second initial generation wave, rather than a repair child. The online model chooses the tile size and concrete operation ordering during generation.

Selected trajectories and campaign cost. The H100 aggregate selects 97 tasks from the primary batch (283 receipts) and tasks 8, 66 and 78 from follow-ups (nine receipts). The reconstruction script preserves a primary trajectory whenever it has a readable receipt, consulting follow-ups otherwise. The selected panel has 292 receipts and at most three positions per trajectory. Separately, the operator's campaign summary records 100 primary plus eight follow-up task-runs, 2,136 logical calls, about 138.49 million tokens and 308 reported positions (284 primary and 24 follow-up), with 10.4 summed GPU-process hours. These totals include work outside the selected panel. For the three tasks

supplied by a follow-up, the plotted budget axis starts at the selected follow-up's first position. The selected 292 receipts account for 130.000 million reported tokens and 34,325 seconds of summed GPU-worker receipt time; the campaign totals also retain earlier and other follow-up work.

The retained 97 primary trajectories alone yield 46, 69 and 74 Fast_1 tasks at $B = 1, 2, 3$; the three selected follow-up trajectories add one task at each prefix, giving the plotted 47, 70 and 75. The operator-reported primary total of 284 positions leaves one position among the three replaced primary task-runs beyond the 283 retained primary receipts. Their original candidate outcome is unavailable in the delivered archive. Under that recorded position count, primary-only coverage lies in 46–47, 69–70 and 74–75 at the respective prefixes. The campaign's 308 positions remain the resource account for all recorded task-runs.

K KNOWLEDGE AND IMPLEMENTATION STUDIES

The studies examine integrated channel capabilities, the stage of knowledge delivery, positive-content organization within shared support, and generic search guided by raw sources or a coarse route. Table 31 names each FULL reference by its experiment.

Table 31: FULL reference map. C/F/S denotes correctness, Fast_1 and strict $\text{Fast}_{1.05}$. Each label identifies one recorded run; the following sections specify task cohorts and service settings.

Reference	Use	$B = 3$ C/F/S
FULL-M	Main 100-task experiment; same-task subsets for raw-source comparison.	90/77/45
FULL-C	Prior 24-task reference for channel removals, Coarse route and knowledge-delivery studies.	23/19/13
FULL-E	Prior 24-task excerpt-study reference for shared-support positive-content comparison.	20/19/11

K.1 CAPABILITY REMOVAL: COMPLETE FIXED-COHORT RESULTS

The fixed, source-stratified development cohort contains tasks 3, 5, 6, 14, 16, 25, 33, 35, 36, 44, 47, 48, 49, 54, 55, 62, 72, 73, 74, 79, 81, 82, 83 and 91. It comprises 5 Conv2d, 5 Conv3d, 5 ConvTranspose2d, 4 ConvTranspose3d, 4 Linear and 1 other composite task. The capability study disables the dedicated support of each removed channel (Table 10). The three removals form one batch of 72 searches and 216 closed positions, compared with FULL-C. Position records contain 214 completed native results and two cleanup-confirmed timeouts; timeouts consume their positions and retain earlier verified implementations. Run settings and exact-source identities accompany the records.

Table 32: Capability-removal prefixes; C/F/S are correct, Fast_1 and strict $\text{Fast}_{1.05}$ counts out of 24. FULL-C is defined in Table 31. All correct candidates in the removal batch satisfy timing stability.

Arm	$B = 1$ C/F/S	$B = 2$ C/F/S	$B = 3$ C/F/S	Tokens (M)
FULL-C	10/8/5	20/17/12	23/19/13	21.909
NO_POS	14/6/2	19/9/5	20/11/8	13.770
NO_NEG	16/13/7	19/17/9	22/18/10	12.708
NEITHER	16/8/3	19/12/5	21/16/7	7.555

At $B = 3$, FULL has the highest count in all three coverage measures in this comparison. Relative to `NEITHER`, retaining positive capabilities in `NO_NEG` adds one correct, two Fast_1 and three strict tasks. Retaining only negative capabilities gives eight strict tasks versus seven for `NEITHER`, with 20 versus 21 correct and 11 versus 16 Fast_1 tasks. The prefix curves locate FULL's cumulative advantage after the second and third measurements; `NO_NEG` leads all three counts at $B = 1$.

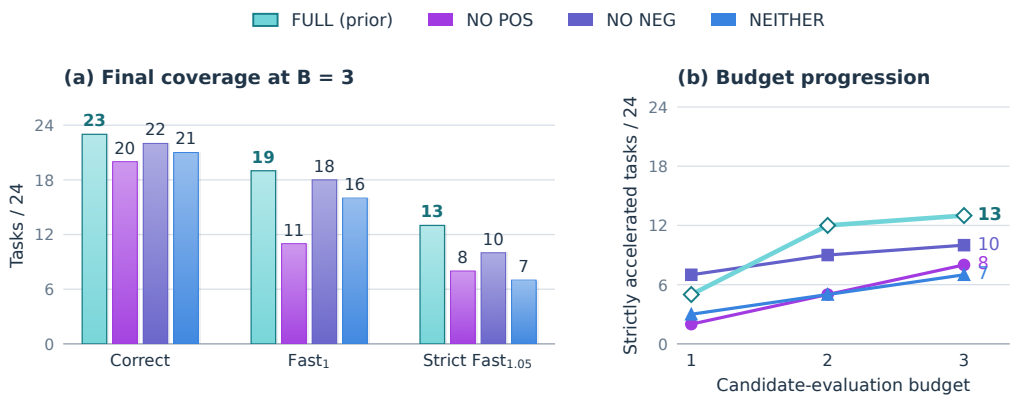

Figure 11: Capability ablation on the fixed 24 tasks. Left: $B = 3$ coverage counts; right: strict acceleration across the original budget prefixes. FULL uses the FULL-C reference; removal curves use the capability batch. All curves retain the original positions and 24-task denominator.

Paired strict outcomes. Comparing task identities gives FULL-only / removal-only strict successes of 8/3 versus `NO_POS`, 6/3 versus `NO_NEG`, and 7/1 versus `NEITHER`. The corresponding net margins are 5, 3 and 6 tasks.

Implementation routes and repair. On task 16, `NO_POS` replaces transposed convolution and its tail with a kernel that accumulates over input channels and kernel positions ($0.042\times$). FULL and `NO_NEG` preserve the library convolution and fuse Mish, addition, clamping and scaling ($1.395\times$ and $1.396\times$). On task 54, FULL and `NO_NEG` retain convolution and fuse multiplication, LeakyReLU and GELU ($1.367\times$ and $1.356\times$); `NEITHER`'s explicit convolution loops reach $0.012\times$.

On task 35, all three `NO_NEG` candidates repeat the unavailable `triton.language.math.tanh` call and fail execution. `NO_POS` reaches a strict $1.104\times$ result; FULL reaches $1.609\times$. Task 91 shows independent discovery of a similar efficient route: FULL and `NEITHER` both retain transposed convolution and fuse channel softmax and postprocessing, reaching $1.669\times$ and $1.700\times$. The exact source files and measurement identities accompany the aggregate records in `data/capability_ablation24/`.

Candidate outcomes and cost. Of the 72 positions per removal, 41, 50 and 48 are correct for `NO_POS`, `NO_NEG` and `NEITHER`. Their remaining positions comprise 11/9/11 numerical or output mismatches, 19/13/12 compilation, execution or resource failures, and 1/0/1 timeouts, respectively. The two timeouts retain earlier correct candidates and close within $B = 3$. The new arms report 34.033 million tokens across 1,460 usage-bearing receipts, with 93.4% input tokens. Twelve additional transport-unknown attempts remain separately recorded with unknown remote usage.

Table 33: Reported model usage and conditional $B = 3$ speedups. AM/GM use each arm's best stable, correct candidate on its n successful tasks; coverage in Table 32 retains all 24 tasks. FULL-C's total usage is 21.909 million tokens; dashes denote fields not decomposed here.

Arm	Input (M)	Output (M)	Receipts	AM (n)	GM
FULL-C	–	–	–	1.119 (23)	0.954
NO_POS	12.881	0.889	496	0.918 (20)	0.739
NO_NEG	12.032	0.676	476	1.111 (22)	0.958
NEITHER	6.883	0.672	488	1.091 (21)	0.901

K.2 KNOWLEDGE DELIVERY BEFORE AND AFTER GENERATION

This study varies where teacher knowledge enters the online workflow while retaining the common tools and construction support (Table 11). The three new arms use the fixed 24-task cohort. `GEN_ONLY` supplies knowledge to construction and repair; `ASSESS_ONLY` supplies it to source assessment; `GENERIC_PREP` uses generic preparation at both stages. Table 34 also includes the previously defined FULL-C reference.

Table 34: Stage-delivery outcomes on all 24 tasks. C/F/S denotes correctness, Fast_1 and strict $\text{Fast}_{1.05}$. Each prefix retains the original evaluation positions; model usage covers each search through its final recorded outcome.

Configuration	$B = 1$ C/F/S	$B = 2$ C/F/S	$B = 3$ C/F/S	Tokens (M)
FULL-C	10/8/5	20/17/12	23/19/13	21.909
GEN_ONLY	16/10/5	18/14/8	19/16/9	19.169
ASSESS_ONLY	14/9/5	18/12/8	21/17/9	8.512
GENERIC_PREP	19/13/4	20/14/5	20/17/8	6.638

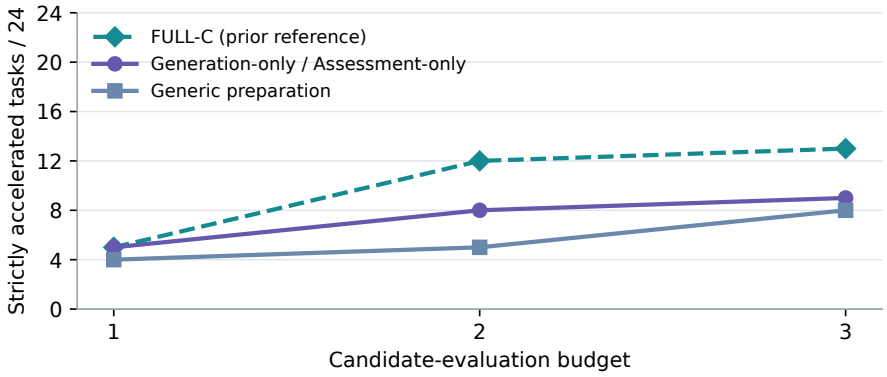

Figure 12: Strict acceleration across the three recorded budget prefixes. The two single-stage arms have identical aggregate counts (5/8/9), shown as one curve; their successful task sets differ. FULL-C is the prior reference, and the three new arms share one batch. All counts use the full 24-task cohort.

Earlier strict acceleration. Both single-stage knowledge arms reach eight strict tasks at $B = 2$; generic preparation reaches eight at $B = 3$. At this observed coverage target, the required per-task evaluation cap falls from three to two, a 33.3% reduction. At $B = 3$, the counts are nine, nine and eight. The prefix comparison thus locates the clearest separation at the second measurement. FULL-C reaches 12 strict tasks at $B = 2$ and 13 at $B = 3$, giving final margins of four, four and five tasks (16.7, 16.7 and 20.8 percentage points). Generic preparation has the highest first-position correctness, at 19/24; the final strict counts describe which searches turn executable candidates into timing-qualified acceleration.

Paired tasks and model computation. At $B = 3$, FULL-C-only / other-only strict successes are 5/1 against `GEN_ONLY`, 5/1 against `ASSESS_ONLY`, and 5/0 against `GENERIC_PREP`. Within the new batch, `GEN_ONLY` versus `GENERIC_PREP` has five exclusive strict successes (16, 35, 47, 49, 62) and four in the reverse direction (5, 72, 79, 83). The assessment-only comparison has four (16, 35, 44, 47) and three (5, 72, 79), respectively. The two knowledge arms share seven strict successes (16, 35, 47, 54, 73, 74, 91). Generation-only delivery uniquely succeeds on 49 and 62, and assessment-only on 44 and 83. Their identical aggregate curves therefore contain different task outcomes. Assessment-only delivery uses 55.6% fewer reported tokens than generation-only delivery, with two more correct tasks and one more Fast_1 task. The difference is concentrated in input tokens: 7.936M for assessment-only versus 18.437M for generation-only delivery. Generic

preparation reports 6.638M total tokens. Table 37 separates input, output and cache usage for the three arms.

Table 35: Measured same-batch implementation cases: each entry is the best eligible source within its original $B = 3$ search. Exact sources, source hashes and native position records accompany `data/stage_delivery24/`.

Task	GEN_ONLY	ASSESS_ONLY	GENERIC_PREP	Source-level difference
16	1.392×	1.387×	0.999×	Fused pointwise tail
35	1.184×	1.600×	1.005×	Selection of a fused tail
47	1.080×	1.085×	1.001×	Realized tail optimization
73	1.364×	1.830×	1.294×	Batch-normalization folding

Candidate reuse and repair. On task 16, both knowledge arms preserve the `ConvTranspose2d` body and combine Mish, addition, hardtanh and scaling into one tail. Generic preparation's best source keeps the separate framework operations. On task 35, the first ASSESS_ONLY candidate fails when unpacking an integer kernel size. The search then selects an initial-pool candidate that handles both integer and pair-valued sizes. It fuses subtraction, HardSwish, 2×2 max pooling and Mish, reaching $1.600\times$ at the second evaluation, compared with generic preparation's $1.005\times$. GEN_ONLY reaches $1.184\times$ through a repaired candidate at the third evaluation. On task 73, the assessment-only source folds evaluation-mode batch normalization and scaling into convolution weights and bias inside `forward`, then calls `F.conv2d`. All three arms produce accelerated implementations on this task, with different realized transformations.

Position and resource accounting. The batch contains 72 searches: 69 use all three positions and three end when no additional admissible candidate is available. The original total is 210 positions. Generation-only task 44 has no evaluated candidate; assessment-only task 72 and generic-preparation task 44 retain their earlier correct results after one and two positions. All 24 tasks remain in each denominator.

Table 36: Stage-delivery resource ledger. Correct counts evaluated positions with recorded correctness. Worker hours sum original position durations. Tokens sum received usage; Unknown counts transport attempts without received usage. Recovery and diagnostics have separate time and budget records.

Arm	Positions	Correct	Tokens (M)	HTTP	Unknown	Worker h
GEN_ONLY	69	46	19.169	531	77	1.857
ASSESS_ONLY	70	41	8.512	572	94	1.749
GENERIC_PREP	71	53	6.638	487	49	1.934

The fraction of evaluated positions with recorded correctness is 66.7%, 58.6% and 74.6%, respectively. Position-level validity and budgeted strict task coverage describe different search outcomes: the former counts passing evaluations, while the latter counts tasks attaining a timing-qualified gain.

Table 37: Reported usage from 1,370 API receipts. Cached and uncached tokens partition input; total reported tokens equal input plus output. Transport attempts without received usage appear separately in Table 36.

Arm	Input (M)	Cached (M)	Uncached (M)	Output (M)	Receipts
GEN_ONLY	18.437	1.546	16.891	0.732	454
ASSESS_ONLY	7.936	1.136	6.800	0.576	478
GENERIC_PREP	6.201	0.690	5.511	0.438	438

Three saved assessment-only candidates on tasks 48, 55 and 82 remain unverified and receive no correctness or performance credit. Follow-up reference controls pass on 48 and 82; the reference

diagnostic on 55 times out, ending the saved-source checks. The zero-API diagnostic adds 3,603.74 worker-process seconds in a separate account.

K.3 Coarse strategy and executable optimization

Configuration. On the same fixed 24 tasks, Coarse route receives one instruction: *prefer retaining optimized library calls for the main expensive operation and implementing compatible postprocessing as a fused custom kernel; preserve the reference semantics*. Positive and negative records, retrieval, hardware priorities, composition closures, protected-body and tail constructors, forced routes, specialized source checks and automatic implementation repairs are removed. The controller retains generic model planning, coding, review, comparison and feedback-driven continuation. The shared artifact interface, native correctness, sanitizer, timing and source-identity checks remain active.

Each task targets two waves of four initial implementations and at most three native evaluations. The run uses two RTX4090 hosts and official nonthinking DeepSeek-V4.1-Flash. Before launch, the comparison fixes the preceding same-task FULL run used in Table 32. Table 38 reports all 24 tasks with their original positions.

Table 38: Coarse-route control on the fixed 24-task cohort. C/F/S denotes stable correctness, Fast_1 and strict $\text{Fast}_{1.05}$. The FULL-C reference is fixed before the coarse-route run.

Configuration	$B = 1$ C/F/S	$B = 2$ C/F/S	$B = 3$ C/F/S	Tokens (M)
FULL-C	10/8/5	20/17/12	23/19/13	21.909
Coarse route	20/11/3	24/15/5	24/19/6	5.978

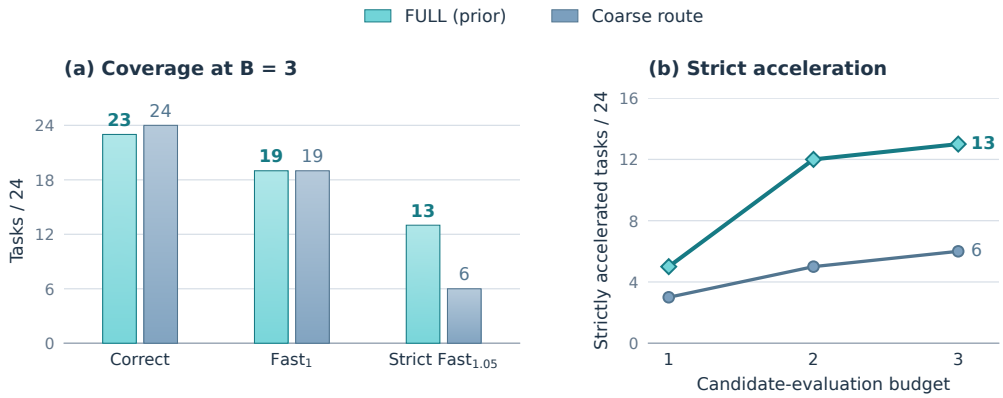

Figure 13: Route-level guidance and realized acceleration. Left: $B = 3$ coverage on all 24 tasks. Right: strict acceleration through the first three evaluation positions. FULL attains 13 strict successes and the coarse-route control six; their Fast_1 coverage is equal.

Coverage and gain strength. At $B = 3$, FULL has nine exclusive strict successes (tasks 5, 47, 48, 54, 62, 72, 74, 79 and 83); Coarse route has two (44 and 49). Four tasks (16, 35, 73 and 91) are strict successes in both. The net gain is seven tasks, or 29.17 percentage points. Of Coarse route's 19 Fast_1 winners, 13 lie between $1.00007\times$ and $1.01138\times$; six meet the strict criterion. The strict prefixes are 5/12/13 for FULL and 3/5/6 for Coarse route.

Realization in the measured source. Twelve coarse-route winners preserve the reference forward's abstract syntax tree after removal of its docstring. On task 48, the source defines a fused postprocessing kernel and a launcher, but forward never calls them. Instead, scaling, tanh, multiplication and sigmoid run as separate framework operations, giving $1.0020\times$; FULL reaches $1.6844\times$. On task 54, the

coarse route keeps multiplication, LeakyReLU and GELU as separate calls ($1.0025\times$), while FULL's measured source executes all three inside one tail kernel ($1.3671\times$).

Coarse route also produces efficient transformations. Task 16 fuses Mish, addition, clamping and scaling, reaching $1.3923\times$ versus FULL's $1.3955\times$. Task 73 folds inference-time batch normalization and scaling into convolution parameters, reaching $1.8413\times$ versus $1.3072\times$.

Candidate supply and continuation. The 192 scheduled initial implementation requests yield 173 valid source emissions and 116 distinct source hashes; 57 emissions duplicate an existing source. Including continuation gives 120 distinct candidates. Tasks 3, 5 and 82 each produce two distinct initial sources. On task 5, the model requests no further modification after both have been tested; on tasks 3 and 82, repair outputs exhaust bounded format recovery. These three searches end with no new untested candidate and retain their earlier correct implementations. The remaining 21 searches use all three positions, for 69 consumed positions in total.

There are 52 correct candidate evaluations, 13 compilation or execution failures, and four numerical mismatches. Seven failed evaluations invoke an unavailable Triton tanh interface. The action ledger records 144 invalid outputs: 102 violate the artifact envelope, including 75 with an extra schema field. Sixty-two actions end after bounded recovery.

Table 39: Model cost and conditional $B = 3$ speedups for the coarse-route control. AM/GM use each configuration's best stable correct candidate on its n successful tasks. All 24 tasks remain in the coverage denominator.

Configuration	Tokens (M)	Positions	AM	GM	n
FULL-C	21.909	72	1.119	0.954	23
Coarse route	5.978	69	1.084	1.071	24

Resource profile. Coarse route uses 72.71% fewer tokens than the fixed FULL reference. Its 522 HTTP requests all return responses; reported usage totals 5,978,265 tokens, including 5,726,009 input tokens. Adaptation, child generation and comparison account for 85.2% of usage. Native worker-process time sums to 10,607.07 seconds across GPUs, with task 55 accounting for 5,181 seconds. Startup controls and GPU-allocation keepalive are separate operational costs. Exact position records, source pairs and the frozen comparison specification are included in `data/coarse_route24/`.

K.4 Same-source positive excerpts

This arm keeps the same 24 tasks, $N = 8$, $B = 3$, negative library, menus, conditional routing, tools and templates. It replaces expanded positive guidance with cited passages from the same local public sources. It tests organized positive guidance against direct excerpts, retaining teacher-built infrastructure. All 24 searches and 72 positions complete.

Table 40: Same-source contrast on 24 tasks. FULL-E is the historical same-task reference; both arms retain the same support infrastructure.

Positive delivery	Correct	Fast_1	Strict	Tokens (M)	HTTP
FULL-E	20	19	11	22.763	502
Source excerpts	19	13	10	27.368	484

FULL-E has six more Fast_1 tasks (25 percentage points) and uses 16.8% fewer reported tokens. Correctness and strict acceleration differ by one task each. On the 17 jointly correct tasks, conditional GMs are $1.230\times$ and $1.049\times$.

The wire audit checks all 484 task HTTP receipts: 235 expanded requests (225 generation-related and ten format-recovery) consume excerpts spanning 105 distinct cards. These views match the frozen passages. Inputs constitute 97.1% of the arm's 27.368 million tokens, locating its main resource use in source-bearing inputs; availability probes are separate.

Source inspection locates concrete realization failures. Task 49 treats consecutive NCDHW addresses as channels during softmax, rather than following the spatial channel stride, in all three attempts. Task 83 repeatedly refers to unbound globals from a separate reference namespace. Task 14 retains a custom GEMM/reduction route at approximately $0.447/0.444/0.445\times$.

A same-task baseline comparison. The computation paths in Figure 2 are illustrated by the recorded RTX4090 Level 2 task 82 sources. We compare the best stable correct source within $B = 3$ for our method with the strongest stable correct baseline on that task. For task 82, both selected sources occupy the third candidate position. OpenEvolve measures $1.001509\times$, compared with $0.998609\times$ for Best-of-N and $0.998773\times$ for Iterative; EGG-port and STARK-port have no eligible correct source within this prefix. Our source measures $2.125992\times$. The case was selected after inspecting the generated implementations.

The reference applies convolution, tanh, a scale of 2, channel-wise bias, and 4×4 max pooling. OpenEvolve uses `torch.tanh`, in-place scale and bias, then `max_pool2d`; these remain separate eager operations. Our source retains convolution and reduces each pool window inside a single Triton kernel before tanh, scaling, and bias. Its output tile is 8×64. The positive scale and spatially constant bias preserve the maximum ordering. The source therefore evaluates tanh once per complete pooled window instead of 16 times, and avoids separate full-resolution pointwise passes.

K.5 Public passages with generic model search

This configuration retrieves original passages from the same local public corpus and supplies the full reference program to generic planning, coding, review, comparison and repair. It uses $N = 8, B = 3$. Teacher cards, conditional menus and composition rules, forced protected routes, prefix/tail constructors, failure tutorials, specialized source checks and automatic implementation patches are removed. Artifact interfaces, source identity, native correctness, Sanitizer, timing and process management remain shared evaluation services. The experiment compares this raw-source search path with the integrated teacher-guided construction and inspection path.

Completed-task snapshot. The two completed shards contain tasks 5, 6, 14, 16, 33, 35, 44, 48, 49, 54, 55, 72, 79, 81, 82 and 91: 16 searches and 48 closed evaluation positions. Table 41 uses these same 16 tasks for both rows; the planned cohort contains 24 tasks. The FULL row uses the original full-benchmark trajectories. The raw-source run requests nonthinking DeepSeek-V4.1-Flash; eight tasks use the compatible service and eight use the recorded official-service continuation. Existing candidates and evaluation positions are retained during continuation.

Table 41: Same-task $B = 3$ comparison on the 16 completed raw-source tasks. The FULL row reuses the corresponding full-benchmark sources; the raw-source row is the new capability configuration. Counts retain all 16 tasks and their three original positions.

Configuration	Correct	Fast_1	Strict $\text{Fast}_{1.05}$
FULL (full-benchmark reference)	13	13	7
Raw-source generic search	12	9	1

Realized computation. Task 82 exposes the difference between receiving material and realizing its conditional transformation. The raw-source winner calls library convolution, tanh, scaling, bias addition and max pooling separately, reaching $1.003762\times$. FULL retains convolution and executes window reduction before the monotone pointwise tail inside one Triton kernel, reaching $2.125992\times$. Its three independent confirmation measurements are $2.1298\times$, $2.1291\times$ and $2.1299\times$. The source-level change reduces tanh evaluations from 16 to one per complete pool window and eliminates separate full-resolution pointwise passes. Task 54 gives a complementary source pair: raw-source search independently preserves convolution and fuses multiplication, LeakyReLU and GELU in Triton, reaching $1.353794\times$ versus FULL's $1.350947\times$.

Candidate and request accounting. The run records 105 candidate artifacts including repair children; 23 of 48 evaluated positions pass correctness. The other positions comprise ten numerical

or contract mismatches, seven unavailable Triton API/namespace errors, five compilation errors, two CUDA execution errors and one type error. The request ledger contains 75 rejected outputs, including 43 artifact-envelope or field mismatches and three truncated completions; generic format recovery and deterministic fallback ordering retain their recorded outcomes. Reported usage is 6.809 million tokens across 352 HTTP attempts, including 16 transport-unknown attempts whose remote usage is recorded as unknown.

K.6 Independent fixed-source confirmation

After search, we freeze the best stable, correct, reporting-eligible $B \leq 3$ source for each available method–task pair on 12 tasks: 5, 6, 14, 25, 33, 49, 55, 73, 74, 82, 83 and 91. This produces 48 candidate sources across six methods, plus one reference-equivalent control per task. Each source is executed three times in fresh worker processes under the common correctness, Sanitizer and timing protocol. Correctness uses seeds 26092401, 26092402 and 26092403 with the existing input challenges, shapes and tolerances. Timing retains the reference-bracketing protocol. Source hashes are fixed before confirmation. The 180 executions are a separate zero-API diagnostic account, with no feedback into search or additional credit toward $B = 3$.

Table 42: Independent confirmation of selected existing sources. Each count requires the stated condition in all three repeats. Source counts follow availability of an eligible historical winner; the table reports source reproducibility within those selected sets. All 180 executions satisfy the timing-stability check.

Method	Frozen sources	Correct 3/3	Fast_1 3/3	Strict 3/3
Ours	10	10	8	6
Best-of-N	5	5	3	0
Iterative	9	9	7	4
OpenEvolve	8	8	5	2
EGG-port	8	8	7	6
STARK-port	8	8	4	3
Reference-equivalent	12	12	1	0

Reproducible acceleration and reference calibration. All 48 candidate sources pass correctness in all three repeats. Our six sources meeting the original strict threshold retain strict acceleration in every repeat: tasks 5, 73, 74, 82, 83 and 91. Task 82 repeats at approximately $2.13\times$, task 91 at $1.72\times$ and task 83 at $1.44\times$. Their repeat ranges appear in Table 43.

The 36 reference-equivalent measurements span 0.993068–$1.013605\times$; six of the 12 controls have a median ratio above one, and none exceeds $1.05\times$. Our task 33 source measures $1.014122/1.002557/0.997442\times$, while task 14 measures $0.9956/1.0011/1.0044\times$.

Table 43: Our ten eligible fixed sources. Ranges include all three independent confirmation measurements. Tasks 6 and 49 have no eligible source under the selection rule and contribute no frozen implementation.

Task	Original speedup	Independent repeat range
5	$1.0993\times$	1.0987–$1.0992\times$
14	$1.0074\times$	0.9956–$1.0044\times$
25	$1.0089\times$	1.0093–$1.0128\times$
33	$1.0032\times$	0.9974–$1.0141\times$
55	$1.0086\times$	1.0013–$1.0086\times$
73	$1.3213\times$	1.3022–$1.3051\times$
74	$1.3311\times$	1.3449–$1.3482\times$
82	$2.1260\times$	2.1291–$2.1299\times$
83	$1.4254\times$	1.4356–$1.4417\times$
91	$1.7156\times$	1.7159–$1.7176\times$

L Retained Construction Examples and Hardware Conditioning

L.1 A complete reduction relationship

The implementation dossiers use pinned Triton tutorial and primitive source, including the fused-softmax tutorial and standard reduction primitives. Those sources establish the observed local reduction and masking patterns. Teacher analysis extends these local patterns into a multi-stage row reduction with explicit resource tradeoffs.

For R contiguous rows of length $N > 0$ and partition width b, the mean uses $P = \lceil N/b \rceil$ partitions. The producer writes one fp32 partial for each (r, p) at $rP + p$. The consumer reads the P partials for row r, combines them, and divides once by the original N. The merge domain contains P partial values. The scratch extent is RP fp32 values, and its producer/consumer traffic alone is $8RP$ bytes; input/output traffic and launch overhead are additional. Partitioning increases available parallelism at the cost of an extra launch and scratch traffic.

For an illustrative $N = 1003$, changing b from 256 to 512 changes P from 4 to 2. Scratch shape, producer grid, and the merge domain change together; the denominator remains 1003. The constructor updates the producer grid, scratch extent and merge bounds from the new partition count.

What each component contributes. The dossier retains direct, streamed, and partitioned routes. The first covers a row in one block; the second scans fixed-width blocks within one program. Both avoid scratch and a second launch, but streaming retains only one program per row. Partitioning exposes more parallelism with added traffic. Row count can therefore change the rationale at fixed row length. Native evaluation resolves the performance crossover between these alternatives.

Masked producer lanes contribute zero. Numerical validation checks floating-point reassociation, intermediate range, special values and the declared tolerance. The available backend and declared memory limits gate preparation. Established shape and stride facts activate a supported route; missing facts retain an unresolved status. Model generation and other supported recipes remain available according to their own contracts.

L.2 Following the same relationship through search

Table 44 describes how the row-mean relationship above can be used at different information states. The example follows route selection, construction and inspection.

Table 44: A shared relationship supports different decisions as information becomes available.

Decision	Available relationship and facts	Consequence / boundary
Choose a design	R, N, task/profile facts, and the direct/streamed/partitioned alternatives. Partitioning adds programs, launches, and scratch traffic.	Condition the route on its implementation rationale; native evaluation measures the crossover between alternatives.
Construct	$P = \lceil N/b \rceil$ binds the producer grid, scratch extent, and merge domain; normalization retains N.	The owned route recomputes these fields. A model-generated variant needs its own source evidence for the same coupled relationships.
Assess benefit	Inspect the actual launch grid and work assignment. One program per row may loop over all partitions while preserving the partial array.	Its launch exposes R independent programs. Record that realized parallelism and assess the remaining benefit rationales.
Assess source	For $N = 1003, b = 512$, inspect two produced partials and any four-position merge, including allocation, initialization, and active masks.	Out-of-bounds or uninitialized active reads support an objection; allocated zero padding or masking may resolve it; absent facts remain unknown.
Repair / select	A source-bound objection names the inconsistent dependency and the quantities that must remain valid after editing.	A changed child receives fresh review, and the selector consumes the updated source-bound findings.

A launch with one program for each row-partition pair exposes RP independent programs; a loop over partitions inside each row's program exposes only R. The source review checks this launch structure, scratch allocation and merge bounds together (Appendix N.5).

L.3 Coverage and local checks

The active positive catalogs contain 272 cards and 466 options for Level 1, and 405 cards and 587 options for Level 2. These counts describe searchable knowledge entries and their selectable options. In the reported Level 2 route, scoped protected-prefix construction preserves reference operations and materializes supported tails; model-written variants undergo their own checks.

Review context retains independently known shapes, scalars, broadcasting and reduction geometry. Each tool finding states its support scope; pattern matches identify relations for further source inspection.

L.4 Conditional views and projection

The conditional-delivery stage evaluates task and hardware-profile conditions and retains missing facts as unknown. Discovery uses family and graph/region scope and exposes a compact option menu. Selected bindings carry card and option identities, node/region scope and parameter values; consumers expand the associated implementation conditions and dependencies. Required task guidance is delivered separately, including when the option list is empty.

Level 2 role views factor repeated material and deliver candidate-relevant contracts, with separate limits for offline expansion and actual request size. Bindings identify selected knowledge; downstream source checks assess implementation conformance. Missing conditions and unresolved interactions remain explicit in the consumer view.

L.5 Which hardware facts change a decision

Fact	Consumer / effect	Follow-up validation
Backend and tool availability	Gate supported backend routes and textual requirements.	Compile the generated candidate against the available backend.
Primitive API surface	Restrict preparation to observed operations/signatures.	Validate numerical behavior and measure latency natively.
Compute capability	Supplies the configured dtype-availability rule, including bf16.	Check route-specific instruction requirements and measure performance.
Observed total memory	Informs explicit memory conditions and runtime checks.	Check current free memory and actual allocation at runtime.
Other observed device properties	Available in profile/model context; only explicit consumers use them.	Trace which properties the binding and selection consumers use.
Device name	Display and model context.	Resolve required capabilities from explicit observations.

The H100 results and implementation traces appear in Appendices H and J.

M The Resource Objective

We optimize task coverage under a fixed target-evaluation budget and account separately for model computation and reusable offline analysis.

Quality at a fixed budget. For a declared cohort of N tasks and online policy π with model allowance M, define

$$Q_\pi(B; M) = \frac{1}{N} \sum_{t=1}^{N} \mathbb{E}\Big[\mathbf{1}\{\exists c \in \mathcal{E}_{t,B}^{\pi,\star} : C(c) \wedge S(c) > 1\}\Big]. \tag{11}$$

Here $\mathcal{E}_{t,B}^{\pi,\star}$ applies the reporting eligibility defined in Section 5. The expectation is over online randomization. The reported curves give the corresponding observed average $\widehat{Q}_\pi(B)$ from one

selected search per task and retain the full task denominator. Each policy's model usage is recorded separately from B. Pool size and rejection counts are recorded as process diagnostics.

Resources at a common quality target. For a cohort coverage target α, compare the smallest tested native budget at which $\widehat{Q}_\pi(B)$ reaches α. The matched-budget panels use our observed $B = 1, 2, 3$ coverages as coverage targets. If no measured point reaches a target, it remains unreached within the tested range. Crossings use observed budget points; model tokens and target-device time are reported on separate axes. First-success counts and times retain unsuccessful tasks and distinguish cap censoring, terminal failures and unknown effects.

Reuse and deployment. For deployment planning, let K be the offline knowledge-construction cost, R the number of searches reusing the frozen artifacts, L the online-model cost per search, and G the native-evaluation cost at matched coverage, all expressed in the same resource valuation. The amortized cost is $K/R + L + G$. Relative to an alternative with costs L_0, G_0, reuse is attractive when $G_0 - G > (L - L_0) + K/R$. Tokens retain input, output and cache categories; native receipts supply accumulated worker duration. The measured budget curves characterize G through evaluation positions and receipt time, while offline construction remains a separate ledger.

N KNOWLEDGE-GUIDED ASSESSMENT INTERFACE

The assessment interface is

$$\mathcal{A}(c, x, h; \mathcal{K}, \mathcal{T}, H) = (\mathcal{F}_c, \mathcal{O}_c, \mathcal{U}_c, \mathcal{W}_c), \tag{12}$$

where $\mathcal{F}_c$ denotes scoped tool findings, $\mathcal{O}_c$ source-linked model objections, $\mathcal{U}_c$ unknowns, and $\mathcal{W}_c$ qualitative work and performance arguments.

N.1 PROCEDURE FORMATION

A negative record identifies a relation, activation conditions, required facts, failure case, legal exception and repair dependencies. A stale merge domain, for example, motivates checking allocation, producer coverage, initialization and masks together. Allocated zero padding and masked inactive lanes supply the corresponding legal exceptions. These fields define the inspection procedure and its support conditions. Source observations and teacher interpretations retain provenance; performance arguments account for work both removed and added.

N.2 DELIVERY

Source, task, chosen mechanisms and hardware route procedures. Positive contracts can inform inspection, while negative findings can motivate a changed construction. Requests and assigned duties record exposure. Unsupported or undisclosed duties remain unassessed. Finite checks run only within their support scopes; model investigation handles selected remaining questions. Library procedures use executable checkers where available and otherwise guide source investigation.

N.3 DECISION USE

Confirmed current-source tool contradictions can block admission. Model claims remain hypotheses that can request evidence or bounded repair. Empty output, parse failure, omitted duties and exhausted allowances retain an unassessed status. Source changes invalidate prior conclusions. Comparison uses current code, findings, unknowns and matching measurements to order eligible alternatives. Unknowns may be measured under the native protocol; confirmed hard contradictions retain their admission block independently of rank.

N.4 COMPARISON COVERAGE

Comparisons record candidates and source coverage. Capacity-driven brackets rank the candidates presented in each request, with inspection scope recorded.

N.5 INSPECTION AND REPAIR TRACE

A trace joins a procedure, activation facts, exact source, observations, questions, disposition and any child. Arithmetic witnesses based on model-supplied mappings retain those assumptions. Partition repairs must update affected producer, scratch and consumer quantities together. Scoped source checks reexamine the affected relations, and native evaluation measures the child's correctness and speed. The assessment path materializes source, obtains scoped facts, delivers guidance, consumes findings, compares or repairs, rechecks and dispatches within budget, invoking model investigation on demand.