Scaffold-Mediated Post-Training: Co-Evolving Model Parameters and Procedural Scaffold Graphs

Fei Ding[*] **Yongkang Zhang** **Runhao Liu** **Yuhao Liao** **Zijian Zeng** **Huiming Yang**
Alibaba Group Tsinghua University

Abstract

Post-training of large language models optimizes only parameters, while inference-time procedural scaffolds are typically designed independently of parameter training. This disconnect makes it difficult to automatically acquire and internalize complex strategies. We propose **scaffold-mediated post-training**: procedural scaffolds are organized into an evolvable graph structure that co-evolves with model parameters through discovery, distillation, and dynamic recompilation. We instantiate this paradigm as Skill Training. On FeatureBench, automatically discovered skills improve the passed rate by 8.1pp, and after progressive distillation the model still achieves a 27.7% passed rate without any external scaffold (distillation retention rate 85.2%, defined as post-distillation / with-skill passed rate), significantly outperforming standard SFT on the same data.

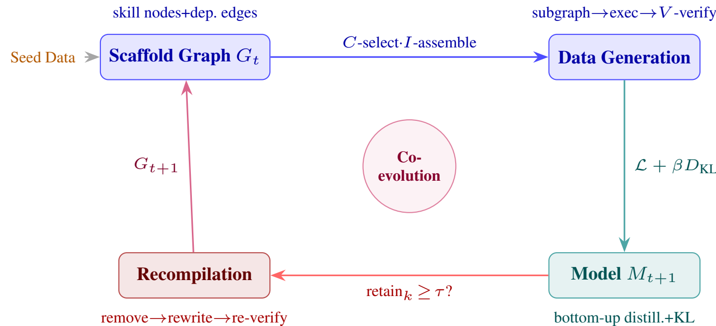

Figure 1: Co-evolution loop in scaffold-mediated post-training. G_t assembles task-specific subgraphs and generates V-verified training data; the model internalizes skill behaviors via progressive distillation; skills meeting the retention threshold are removed and dependent scaffolds are recompiled into G_{t+1}.

1 Introduction

The parameter-learning bottleneck. Post-training of large language models (LLMs) typically treats only model parameters as trainable. However, the procedural strategies required by complex tasks (how to decompose requirements, edit multiple files in topological order, and verify correctness after each step) are difficult to learn from scarce training examples alone (Yang et al., 2025b). Evaluations on FeatureBench (Zhou et al., 2026) show that Claude Opus 4.5 achieves a 76.8% solve rate on SWE-bench (Jimenez et al., 2024) yet only a 45.5% passed rate on FeatureBench, revealing deficiencies in procedural strategies. FeatureBench focuses on end-to-end feature development (involving multi-file

[*]Corresponding author: dignfei@gmail.com

Preprint.

coordination, architectural decisions, and multi-step verification workflows), placing higher demands on procedural strategies.

The limits of scaffold engineering. Inference-time scaffolds such as chain-of-thought (Wei et al., 2022), ReAct (Yao et al., 2023), and agentic workflows can substantially improve performance on complex tasks, but they are typically hand-designed and bolted on at inference time; as scaffolds accumulate, inference becomes increasingly complex and slower.

Scaffold-mediated post-training. This paper proposes a new post-training paradigm that bridges parameter learning and scaffold engineering: **procedural scaffolds themselves become evolvable objects that co-evolve with model parameters**. The model first solves tasks with the aid of an external procedural scaffold graph, which generates high-quality supervised data. The model then absorbs the scaffold's behavior through distillation. Once a scaffold has been internalized into parameters, the external graph undergoes **dynamic recompilation**: the internalized skill is removed from the graph, and higher-level scaffolds that depend on it are automatically rewritten, closing the co-evolution loop between parameter learning and scaffold learning.

We instantiate this paradigm as **Skill Training**. A skill is a node in the procedural scaffold graph, comprising metadata (*meta*), an activation condition (C), an execution procedure (P), a verification checkpoint (V), a compositional interface (I), and anti-patterns (E). The framework operates in three stages (Figure 1):

1. **Skill Discovery**: Structured skills are automatically discovered from seed data to construct an initial procedural scaffold graph.
2. **Scaffold-Graph-Augmented Data Generation**: For each task, a scaffold subgraph is assembled to guide the model through the task and generate verified training data.
3. **Progressive Distillation with Dynamic Recompilation**: Starting from the most frequently invoked bottom-level skills, progressive distillation internalizes skill behavior into model parameters. Internalized skills are removed from the scaffold graph, and higher-level scaffolds that depend on them are automatically rewritten and re-verified.

The main contributions of this paper are:

- **A new paradigm**: We propose scaffold-mediated post-training, in which model parameters and an external procedural scaffold graph co-evolve through discovery, assembly, distillation, and dynamic recompilation.
- **A new representation**: We formalize skills as nodes in a procedural scaffold graph $G = (S, D)$, each containing the six-tuple $(\textit{meta}, C, P, V, I, E)$. Explicit skill calls in P induce dependency edges in the graph, I defines typed labels for inter-node data flow, and V provides quality-filtering signals for distillation.
- **A new mechanism**: We propose dynamic recompilation: once a skill has been distilled and internalized into parameters, the scaffold graph automatically rewrites higher-level scaffolds that depend on it, enabling co-evolution between parameters and scaffolds.
- **Empirical validation**: On FeatureBench, automatically discovered skills raise the passed rate from 24.4% to 32.5%; after progressive distillation the rate reaches 31.5% (22 skills internalized); pure internalization without external scaffolds still achieves 27.7% (retention rate 85.2%), all significantly outperforming standard SFT on the same data (25.5%). Ablations show that V and P contribute 6.9pp and 4.2pp to the retention rate, respectively.

Limitations remain: this work validates the framework only in code generation and uses self-distillation from a single model without exploring teacher diversity. The framework itself is domain-agnostic, and extending it to other domains with executable verifiers, such as mathematical reasoning, is an important direction for future work (Section 7).

2 Related work

2.1 Parameter learning

SFT and RLHF compress capabilities directly into model parameters. STaR (Zelikman et al., 2022) and Quiet-STaR (Zelikman et al., 2024) achieve self-improvement by bootstrapping reasoning chains.

Context Distillation (Snell et al., 2022) distills rich contexts into weights. These methods can update parameters but lack explicit, composable, and verifiable external procedural structures; their distillation targets are flat text or reasoning chains that lack explicit composable interfaces for programmatic cross-task reuse.

2.2 Scaffold learning

APE (Zhou et al., 2023) and OPRO (Yang et al., 2024) automatically optimize prompts; Promptbreeder (Fernando et al., 2024) optimizes prompts via evolutionary algorithms, yet all produce flat text. Voyager (Wang et al., 2024) builds a composable skill library; ExpeL (Zhao et al., 2024) extracts natural-language rules; SkillRL (Xia et al., 2026) jointly trains skills with RL; Skill-It! (Chen et al., 2023) studies data dependencies among skills. DSPy (Khattab et al., 2024) treats prompts as composable modules; SAGE (Wang et al., 2026) accumulates a skill library within an RL framework. These methods construct external scaffolds but all **lack bidirectional co-evolution between parameters and the scaffold graph**.

2.3 Bridging the two: scaffold-mediated post-training

This paper connects parameter learning and scaffold learning: procedural scaffolds are first discovered and used to guide the model in solving tasks and generating training data (the strength of scaffold learning), then distilled and internalized into parameters (the strength of parameter learning), after which the scaffold graph undergoes dynamic recompilation. This "learn $\rightarrow$ internalize $\rightarrow$ recompile" loop has not been realized in prior work.

3 Method: scaffold-mediated post-training

Engineering motivation. In engineering practice, we built a system of 136+ manually authored skills covering Test-Driven Development (TDD), debugging, and code review workflows. These skills closely match the formal definition in structure (Appendix D), but manual design is prohibitively expensive, directly motivating the automated training research below.

This section first defines the procedural scaffold graph (§3.1), then describes three stages: skill discovery (§3.2) $\rightarrow$ scaffold-graph-augmented data generation (§3.3) $\rightarrow$ progressive distillation and dynamic recompilation (§3.4–3.5). The first two stages constitute **scaffold learning**; the third constitutes **parameter learning and scaffold-graph co-evolution** (Figure 2).

3.1 Procedural scaffold graph

Definition 1 (Skill). A skill s is a node in the procedural scaffold graph, comprising the following core components:

$$s = (meta, C, P, V, I, E)$$

where:

- *meta* (metadata): name, description, category tags, available tool list, and version number.
- C (activation condition): task characteristics and triggering rules that determine when the skill is activated (the LLM automatically evaluates the match against the task description and executes upon match).
- P (execution procedure): structured workflow steps, categorized as either **rigid** (strict step ordering with hard constraints at each step) or **flexible** (advisory patterns that permit discretionary step selection).
- V (verification checkpoint): checkable conditions that must be satisfied after skill execution. Following the **evidence-driven** principle, V requires executable evidence rather than model assertions. Verification may comprise multiple layers (build $\rightarrow$ type-check $\rightarrow$ test $\rightarrow$ security $\rightarrow$ change audit); in rigid skills these are executed sequentially with fail-fast semantics (Appendix I).
- $I = (I.\text{in}, I.\text{out})$ (compositional interface): standardized input/output type label sets, enabling multiple skills to be chained according to dependency relations.
- E (anti-patterns): common failure modes that help the model avoid known pitfalls.

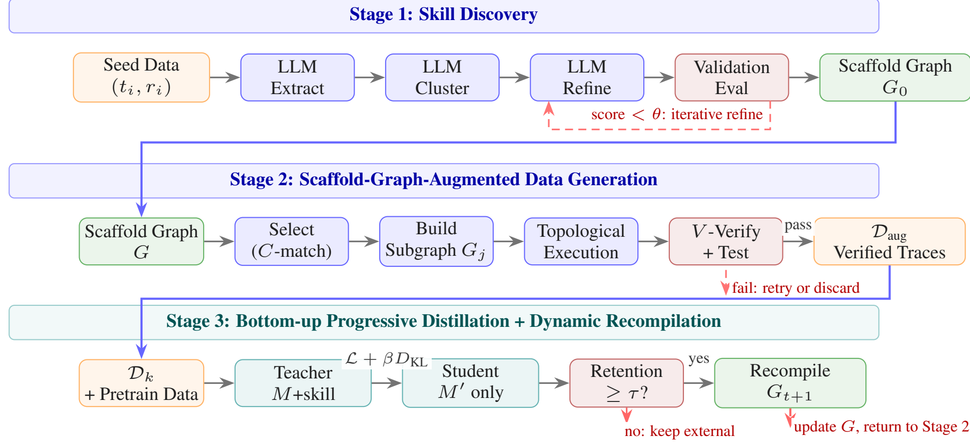

Figure 2: Detailed three-stage data flow. **Stage 1**: iteratively discover skills from seed data and build the scaffold graph G_0. **Stage 2**: for each task, select nodes by C-matching, build subgraph G_j, execute in topological order, and V-verify; only passing traces are retained. **Stage 3**: bottom-up progressive distillation (loss includes KL constraint, mixed with pretrain data); skills whose retention rate meets the threshold are removed from the graph and trigger dependency recompilation. Blue arrows denote cross-stage data flow; red dashed lines denote feedback/fallback paths.

In practice, each skill is implemented as a Markdown file with YAML frontmatter; the design derives from iterative experience with over 136 manually authored skills.

Definition 2 (Procedural Scaffold Graph). A scaffold graph $G = (S, D)$, where S is the set of skill nodes and D is the set of dependency edges induced by explicit skill calls in P: if P_b explicitly invokes s_a, an edge from s_a to s_b is added. I provides typed labels (e.g., `file_list`, `test_result`) for the data flowing along these edges at runtime. Acyclicity is verified via depth-first search (DFS) after graph construction: if a node reappears in the path stack, the cycle is identified and its chain reported, the offending last edge is discarded, and the check is retried. The graph of 30 skills in our experiments is always a DAG. Composite skills correspond to executing a scaffold subgraph: nodes are scheduled in topological order. V provides verification feedback after each node executes.

Multi-round discovery produces hierarchical skill structures: higher-level skills discovered in later rounds explicitly invoke lower-level skills in their P, from which dependency edges naturally arise; Algorithm 1 formalizes these call relationships via interface matching (I_a.out $\cap$ I_b.in $\neq \emptyset$).

Key distinction from prompts. A conventional prompt is flat text; skills possess three structural properties: (1) **Verifiability**—V enables automatic evaluation of execution outcomes, providing a filtering signal for distillation; (2) **Composability**—I defines standardized interfaces that allow multiple skills to be chained; (3) **Conditional activation**—C automates skill selection. Ablation experiments (Section 5.3) quantify the contributions of V and I.

In practice, skills are classified as either **rigid** (enforcing step ordering and verification checkpoints) or **flexible** (advisory patterns permitting flexible selection). Ablation experiments (Section 5.3) progressively remove V and I to verify the contribution of skill's core structural features (E removal causes only 0.8pp drop, not significant). Full ablation configurations and skill examples are provided in Appendix D.

3.2 Stage 1: skill discovery

Input: A high-quality seed dataset $\mathcal{D}_{\text{seed}} = \{(t_i, r_i)\}_{i=1}^{N}$, where t_i is a task description and r_i is a successful solution. N is typically 1000–3000.

Algorithm 1 (Appendix A) provides the complete procedure. Here LLM.Extract, LLM.Cluster, and LLM.Refine are all LLM calls (prompt templates are given in Appendix B). Evaluate compares the passed rate on a validation set before and after skill injection. The threshold θ (default 0.05, i.e., a skill must yield $\geq$5 pp improvement on the tasks that invoke it) controls the minimum effectiveness

requirement; ϵ (default 0.01) controls the iteration convergence criterion. Sensitivity analysis is provided in Appendix C.

Relationship to human-designed skills. The skills produced by automatic discovery are structurally highly similar to manually designed ones (Appendix D provides a component-by-component comparison), yet the process is fully automated, demonstrating that LLMs can induce structured strategies from successful exemplars.

3.3 Stage 2: scaffold-graph-augmented data generation

Objective: Leverage the scaffold graph G_0 to guide the model in solving additional tasks and generate high-quality training data. $\mathcal{T}_{\text{new}}$ comprises 2000 task instances generated by the FeatureBench data pipeline on additional repositories; success is determined by execution-based verification (running the test suite). Algorithm 2 (Appendix A) provides the procedure.

Subgraph construction and execution. The LLM selects candidate nodes $\mathcal{S}_j$ by C-matching, extracts subgraph G_j from G_0 in topological order (depth limited to 3 layers). During execution, nodes are dispatched sequentially with data passed via I; each node runs V-verification after execution (one retry permitted; second failure discards the trajectory). A final task-level test suite is run; only fully verified trajectories are retained as training data. Formal definitions are given in Appendix J.

3.4 Stage 3: progressive distillation and dynamic recompilation

Objective: Internalize skill behaviors into model parameters and dynamically recompile the scaffold graph after internalization. Algorithm 3 (Appendix A) provides the complete procedure.

Distillation objective. We adopt a self-distillation approach in which the teacher and student are different configurations of the same model: the teacher receives [skill prompt $\oplus\, t_j$] and produces r_j (the full execution trajectory including intermediate reasoning and action steps); the student receives only $[t_j]$ and is trained to match r_j. The training loss is $\mathcal{L} + \beta \cdot D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})$, where $\mathcal{L}$ is the cross-entropy loss and the KL divergence constraint prevents the parameters from deviating too far from the reference model.

Distillation order. Progressive distillation proceeds bottom-up along the topological order of the scaffold graph G: root nodes with no dependencies (in-degree 0) are distilled first; within the same level, nodes are sorted in descending order of dependency count (out-degree), prioritizing internalization of foundational capabilities with the largest impact. This ordering is uniquely determined by the graph structure and requires no manual weight tuning.

Implementation details. We use full-parameter fine-tuning (Appendix K). The retention rate threshold is $\tau = 0.85$, and the recompilation fallback threshold is $\delta = 0.02$. The reference policy π_{ref} in the KL constraint is statically anchored to the original base model M_0 and is not updated across distillation rounds. Algorithm 3 iterates per skill for clarity; in implementation, skills within the same topological level are merged into a single training round to avoid redundant gradient steps.

Catastrophic forgetting mitigation. Five mechanisms jointly mitigate catastrophic forgetting: (1) **self-distillation**—teacher and student share the same base model, inherently small KL divergence reduces distribution shift; (2) **bottom-up implicit replay**—lower-level skill behaviors are repeatedly replayed in higher-level training data; (3) **pretraining data mixing**—50% pretrain data mixed per round; (4) **KL constraint** ($\beta = 0.01$); (5) **per-round retention check**—skills falling below τ are not internalized and are retained as external scaffolds. Detailed analysis is provided in Appendix E.

3.5 Dynamic recompilation

Dynamic recompilation enables co-evolution of parameters and the scaffold graph. After skill s_k is internalized, its node cannot simply be removed—higher-level skills may depend on it. Recompilation consists of six steps: (1) **Node Removal**: remove s_k from G; (2) **Procedure Rewriting**: an LLM replaces explicit calls to s_k in the higher-level scaffold s_h's procedure P_h with implicit assumptions about the model's internalized capability; (3) **Bridge Regeneration**: regenerate inter-node data-passing instructions; (4) **Interface Update**: update I_h; (5) **Re-verification**: recompiled scaffolds must pass V verification on a validation set (60s timeout per node); (6) **Fallback**: if performance

Table 1: Experimental configurations. Direct Fine-tune trains on the same $\mathcal{D}_{\text{aug}}$ via standard SFT in a single pass, isolating the contribution of structured training pathways.

Configuration	Description
Baseline	Vanilla model executes tasks directly
+ Human Skills	Manually designed skill set
+ Auto Skills	Model auto-discovered skills
+ OPRO Prompt + Distill	OPRO-optimized prompt distilled; no prompt injected at inference
+ Distilled (All-at-once)	All-at-once distillation (τ=0.85, 12 internalized + 18 external)
+ Distilled (Progressive)	Progressive distillation (τ=0.85, 22 internalized + 8 external)
+ Pure Internalization	Progressive distillation with all external skills removed (0 external, parameters only)
+ Direct Fine-tune	Standard SFT on the same $\mathcal{D}_{\text{aug}}$ (with identical pretrain data mixing, no skill decomposition, no progressive curriculum)

drops by $> \delta$, retain s_k as an external scaffold. Recompilation depth is limited to 1 hop; each node is processed independently.

Formally: $M_{t+1} = \text{Distill}(M_t, \mathcal{D}(s_t))$; $G_{t+1} = \text{Recompile}(G_t, s_t, M_{t+1})$.

4 Experimental setup

Our experiments aim to answer five research questions: **RQ1**: Can a scaffold graph be automatically learned from seed trajectories? **RQ2**: Can the scaffold graph improve task solving and data generation? **RQ3**: Can scaffold behaviors be internalized into model parameters? **RQ4**: Is the graph structure (V and I) necessary? **RQ5**: Is dynamic recompilation effective?

4.1 Benchmarks and models

Data and Evaluation. We use the open-source data pipeline from FeatureBench (Zhou et al., 2026) to generate training data. Evaluation strictly follows the FeatureBench paper setup: we use its standard Full set (200 tasks, 24 open-source repositories, execution-based verification), with OpenHands (Wang et al., 2025) as the base agent framework (providing tool interaction and environment management), and skills injected as an additional procedural knowledge layer on top. We report Passed Rate (the average proportion of fail-to-pass tests passed per task). Training data is generated using the data pipeline on additional repositories with no overlap with the evaluation set.

Models. Qwen3-Coder-480B-A35B-Instruct (Yang et al., 2025a) (MoE, 35B activated parameters, full-parameter fine-tuning) and **DeepSeek-V3.2** (DeepSeek-AI, 2025) (MoE, 37B activated parameters, to verify cross-model generality).

4.2 Experimental configurations

4.3 Seed data construction

We use the FeatureBench data pipeline to generate training instances on additional repositories, selecting 2000 seed tasks and 200 validation tasks (for skill quality evaluation, independent of the FeatureBench evaluation set). Selection criteria: involving multi-file modifications, containing strategic steps, and covering diverse development patterns. An additional 2000 tasks are sampled as $\mathcal{T}_{\text{new}}$ for the data generation phase. Seed data details are provided in Appendix G.

4.4 Evaluation metrics

- **Passed Rate**: the average proportion of fail-to-pass tests passed per task. We adopt Passed Rate as the primary metric because skills provide incremental help on procedural strategies (e.g., correctly executing 3 out of 5 steps), and Passed Rate captures such partial improvements. The strict Resolved Rate (all tests must pass) is reported in Appendix L; trends are consistent.
- **Distillation retention rate**: Passed Rate of the distilled model without skill / Passed Rate of the pre-distillation base model with skill (i.e., the Auto Skills passed rate of 32.5%), measuring how

Table 2: Passed rate (%) on FeatureBench Full (200 tasks, OpenHands framework, 5 random seeds).

Configuration	Qwen3-Coder	DeepSeek-V3.2
Baseline	24.4 ± 0.8	26.1 ± 0.7
+ Human Skills	35.0 ± 1.0	36.8 ± 1.1
+ Auto Skills	32.5 ± 1.1	34.3 ± 0.8
+ OPRO Prompt + Distill	26.0 ± 0.8	27.8 ± 0.6
+ Distilled (All-at-once)	29.0 ± 0.9	30.4 ± 1.0
+ Distilled (Progressive)	31.5 ± 1.0	33.3 ± 0.7
+ Pure Internalization	27.7 ± 1.0	29.5 ± 0.8
+ Direct Fine-tune	25.5 ± 0.9	27.2 ± 0.8

much of the skill-augmented capability is retained after distillation (evaluation tasks are highly relevant to the skills).

- **Skill transfer rate**: the proportion of skills that yield a positive improvement on tasks outside the seed set.

Following the FeatureBench evaluation protocol, all 200 evaluation tasks are used for assessment (model training data comes entirely from independent seed and generated sets with no overlap with these 200 evaluation tasks). Each configuration is run with 5 random seeds; we report mean ± standard deviation.

5 Experimental results

5.1 Main results (RQ1–RQ3)

Table 2 presents the passed rate of each configuration on FeatureBench.

Key observations.

(1) **Effectiveness of auto-discovery**: Auto Skills (32.5%) approach Human Skills (35.0%) with a gap of only 2.5pp, indicating that the model can automatically discover skills from 2,000 seed cases at near-human quality (achieving 93% of the human-designed passed rate).

(2) **Value of scaffold-augmented inference**: Auto Skills (32.5%) significantly outperform Baseline (24.4%) by 8.1pp, demonstrating that skill-augmented inference yields a substantial absolute improvement.

(3) **Value of distillation internalization**: Progressive Distilled (31.5%, with 8 external skills) significantly outperforms Direct Fine-tune (25.5%) by 6.0pp. Even after completely removing external skills, the purely internalized model still reaches 27.7% (retention rate 85.2%), outperforming Direct Fine-tune by 2.2pp. Note that both use **the same** $\mathcal{D}_{\text{aug}}$ (all V-verified); the gap arises purely from structured training pathways: per-skill progressive distillation vs. one-pass standard SFT.

(4) Trends are consistent across both models.

5.2 Skill quality analysis

The automatic discovery process extracted approximately 120 candidate strategies from 2,000 seed cases, which were clustered and merged into 45 skill candidates; 30 of these passed validation-set filtering ($\theta = 0.05$), converging in 5 iterations.

Auto Skills cover 78% of test tasks (Human Skills cover 85%), with a false-trigger rate of 12% (Human Skills: 8%). On out-of-seed tasks, 27 out of 30 skills yield a positive improvement (skill transfer rate 90%), indicating good cross-task transferability. A fine-grained analysis stratified by task complexity is provided in Appendix F.

Table 3: Skill component ablation (Qwen3-Coder). Pure internal. rate = infer. passed rate × retention rate. All rows use the progressive distillation pipeline; Flat Prompt's low retention reflects pipeline incompatibility with unstructured prompts. OPRO+Distill in Table 2 uses standard context distillation.

Configuration	Infer. Passed Rate	Pure Internal. Rate	Retention Rate	Composable
Full Skill (complete)	32.5%	27.7%	85.2%	Yes
$-V$ (no verification)	30.0%	23.5%	78.3%	Yes
$-P$ (no explicit calls)	31.0%	25.1%	81.0%	No
$-V-P$ (cond. prompt)	28.5%	20.9%	73.3%	No
Flat Prompt (OPRO)	27.0%	19.0%	70.4%	No

Table 4: Scaffold graph evolution summary. Of 30 skills, 22 were successfully internalized, triggering 50 dependency recompilations (96% successful); the passed rate gradually decreased from 32.5% to 31.5% (-1.0pp).

Metric	Value
Initial scaffold graph nodes	30
Initial dependency edges	52
Successfully internalized skills	22
Retained as external scaffolds	8
Total dependency recompilations triggered	50
Recompilation fallbacks	2
Recompilation success rate	96.0%
Final scaffold graph nodes	8
Final dependency edges	6
Post-recompilation passed rate	31.5%
Passed rate drop	1.0pp

5.3 Graph structure ablation (RQ4)

V's dual role: removing V drops inference-time passed rate by 2.5pp and retention rate by 6.9pp; V provides execution feedback during inference and data filtering during distillation. P's indirect value: removing explicit calls in P drops passed rate by 1.5pp; the dependency graph induced by P enables skill composition and makes distillation targets more modular ($-P$ leaves no dependency edges; the model autonomously decides which skill to invoke, and all skills are merged into a single round of joint distillation). Full degradation to conditional prompt ($-V-P$) differs from Flat Prompt by only 1.5pp in inference-time passed rate (28.5% vs 27.0%), confirming that V and P are the core distinguishing features.

5.4 Dynamic recompilation analysis (RQ5)

Table 4 presents a scaffold graph evolution summary during progressive distillation.

Key observations. (1) **Recompilation success rate**: After internalizing 22 skills, a total of 50 upstream scaffold recompilations were triggered, of which 48 succeeded and 2 fell back (success rate 96%), indicating that the I interface provides sufficient dependency information to support automatic rewriting. (2) **Controllable gradual performance degradation**: As external scaffolds decrease from 30 to 8, the passed rate gradually drops from 32.5% to 31.5% (-1.0pp), far less than the drop from directly removing all scaffolds (-4.8pp). (3) **Graph structure evolution**: The initial graph G_0 has 30 nodes and 52 dependency edges; after distillation, G_T has 8 nodes and 6 edges, representing significant structural simplification. 8 skills are retained as external scaffolds: 6 due to retention rates below τ, and 2 due to recompilation performance drop exceeding δ despite meeting the retention threshold.

5.5 Case studies

Representative cases (detailed in Appendix H): in a success case, a 5-file modification task is completed via skill-guided dependency analysis and stepwise verification, with the model retaining this

behavior after distillation; a failure case reveals that skills primarily improve procedural capabilities but offer limited help for creative decision-making.

6 Discussion

Capability transfer. Scaffold-mediated post-training enables bidirectional capability transfer between two storage media: distillation transfers scaffold capabilities to parameters (G_0's 30 nodes $\rightarrow G_T$'s 8 nodes), while dynamic recompilation feeds parameter state back into the graph structure. Direct Fine-tune (25.5%, same data, one-pass SFT) lacks structured training pathways; hybrid deployment (i.e., Progressive Distilled with 22 internalized + 8 external skills, 31.5%) outperforms by 6.0pp. At inference, hybrid deployment provides $[8 \text{ external skill prompts} \oplus t_j]$ to the model; the instruction-following capability for conditioning on these prompts is already present in the base model.

Self-distillation and confirmation bias. Self-distillation risks reinforcing the model's own biases; three factors mitigate this: (1) V is execution-based verification (build exit codes, test pass rates), not model self-assessment—removing V drops retention by 6.9pp; (2) consistent trends across two different MoE models (Qwen3-Coder/DeepSeek-V3.2); (3) comparison with Direct Fine-tune (27.7% vs 25.5% on Qwen3-Coder, same data) indicates that the structured training pathway provides information beyond the data alone.

7 Limitations

Domain and scale. The current work is validated only in the code generation domain. The framework is domain-agnostic in principle—V and I can be instantiated with domain-specific verifiers and interfaces—but empirical validation in other domains remains future work.

Teacher diversity. Same-model self-distillation is used throughout; stronger or differently architected teachers may further improve quality and mitigate confirmation bias.

Pretraining contamination. The 24 evaluation repositories are likely in the base models' pretraining corpora, but this affects all configurations equally; our conclusions rest on relative differences.

8 Broader impact

The performance bottleneck of current AI coding agents on complex tasks largely stems from the scarcity of procedural strategy data. The scaffold-mediated post-training paradigm proposed here, by treating procedural scaffolds as evolvable objects and distilling them into model parameters, offers a new path to overcome this bottleneck. Developers can train structured skills from high-quality engineering practices (such as TDD and code review workflows), then use distillation to endow the model with these capabilities without external scaffolds. A potential risk is that distillation may internalize unsafe coding patterns from the training data; deployment should be coupled with security audits.

9 Conclusion

We propose scaffold-mediated post-training, where parameters and scaffold graphs co-evolve through multiple rounds of the discover $\rightarrow$ internalize $\rightarrow$ recompile loop—multi-round discovery builds higher-level scaffolds, internalization simplifies the graph structure (Table 4: 30$\rightarrow$8), enabling the model to tackle increasingly complex tasks. Experiments confirm that this mechanism consistently outperforms standard fine-tuning on equivalent data. Code will be released upon acceptance.

References

Mayee Chen, Nicholas Roberts, Kush Bhatia, Jue WANG, Ce Zhang, Frederic Sala, and Christopher Ré. Skill-it! a data-driven skills framework for understanding and training language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 36000–36040. Curran Associates,

Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/70b8505ac79e3e131756f793cd80eb8d-Paper-Conference.pdf.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 13481–13544. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/fernando24a.html.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 54928–54958, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context, 2022. URL https://arxiv.org/abs/2209.15189.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library, 2026. URL https://arxiv.org/abs/2512.17102.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL https://arxiv.org/abs/2602.08234.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang

Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 12028–12068, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/3339f19c5fcee3ad74502947a32be9e6-Paper-Conference.pdf.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025b. URL https://arxiv.org/abs/2504.21798.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 15476–15488. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf.

Eric Zelikman, Georges Raif Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah Goodman. Quiet-STar: Language models can teach themselves to think before speaking. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=oRXPiSOGH9.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38 (17):19632–19642, Mar. 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Qixing Zhou, JiaCheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, Dandan Tu, and Zhaoxiang Zhang. Featurebench: Benchmarking agentic coding for complex feature development. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=41xrZ3uGuI.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

A Algorithm Pseudocode

Algorithm 1 Skill Discovery

Require: Seed dataset $\mathcal{D}_{\text{seed}}$, validation set $\mathcal{D}_{\text{val}}$, maximum iteration rounds T, effectiveness threshold θ, convergence threshold ϵ

Ensure: Base scaffold graph $G_0 = (\mathcal{S}_{\text{base}}, D)$

1: $\mathcal{R} \leftarrow \emptyset$ {Strategy candidate set}
2: **for** each $(t_i, r_i) \in \mathcal{D}_{\text{seed}}$ **do**
3: $\quad \mathcal{R}_i \leftarrow$ LLM.Extract$(t_i, r_i,$ "extract reusable strategies"$)$
4: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_i$
5: **end for**
6: $\mathcal{S}_{\text{cand}} \leftarrow$ LLM.Cluster$(\mathcal{R},$ "merge similar strategies into skill candidates"$)$
7: **for** each $s_k \in \mathcal{S}_{\text{cand}}$ **do**
8: $\quad$ Format s_k into complete skill structure $(\textit{meta}_k, C_k, P_k, V_k, I_k, E_k)$
9: **end for**
10: **for** each $s_k \in \mathcal{S}_{\text{cand}}$ **do**
11: $\quad$ score$_k \leftarrow$ Evaluate$(s_k, \mathcal{D}_{\text{val}})$ {Initial evaluation}
12: **end for**
13: **for** iter $= 1$ **to** T **do**
14: $\quad$ prev_scores$[k] \leftarrow$ score$_k$, $\forall k$ {Cache previous scores}
15: $\quad$ **for** each $s_k \in \mathcal{S}_{\text{cand}}$ **do**
16: $\quad\quad$ **if** score$_k < \theta$ **then**
17: $\quad\quad\quad s_k \leftarrow$ LLM.Refine$(s_k,$ failure case analysis$)$
18: $\quad\quad\quad$ score$_k \leftarrow$ Evaluate$(s_k, \mathcal{D}_{\text{val}})$ {Re-evaluate after refinement}
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: $\quad$ **if** $\max_k |\text{score}_k - \text{prev_scores}[k]| < \epsilon$ **then**
22: $\quad\quad$ **break**
23: $\quad$ **end if**
24: **end for**
25: $\mathcal{S}_{\text{base}} \leftarrow \{s_k \mid \text{score}_k \geq \theta\}$
26: $D \leftarrow \{(s_a, s_b) \mid I_a.\text{out} \cap I_b.\text{in} \neq \emptyset\}$
27: **return** $G_0 = (\mathcal{S}_{\text{base}}, D)$

B Skill Discovery Prompt Templates

Below are the prompt templates used in each stage of Skill Discovery.

B.1 LLM.Extract Prompt

```
You are a strategy extraction expert.  Given a coding task and its
successful solution, extract REUSABLE problem-solving strategies as
structured skills.

TASK: {task_description}

SUCCESSFUL SOLUTION: {solution}

For each strategy, output a structured skill with:            - name:
concise name describing the core idea     - C (condition):  what task
characteristics trigger this skill        - P (procedure):  3-5 step
execution workflow        - V (verification):  checkpoints to confirm
successful execution (must be evidence-based, not self-reported)
- I (interface):  required inputs and produced outputs            - E
(anti-patterns):  common mistakes this skill helps avoid

Rules:  strategies must be GENERAL (not specific to this task),
ACTIONABLE (each step is executable), and VERIFIABLE (V must
reference observable evidence like test output or build status).
```

Algorithm 2 Scaffold-Graph-Augmented Data Generation

Require: Base scaffold graph $G_0 = (\mathcal{S}_{\text{base}}, D)$, new task set $\mathcal{T}_{\text{new}}$
Ensure: Augmented dataset $\mathcal{D}_{\text{aug}}$

1: $\mathcal{D}_{\text{aug}} \leftarrow \emptyset$
2: **for** each $t_j \in \mathcal{T}_{\text{new}}$ **do**
3: $\mathcal{S}_j \leftarrow \text{Select}(G_0, t_j)$ {Match relevant scaffold nodes by C}
4: $G_j \leftarrow \text{BuildSubgraph}(\mathcal{S}_j, D, \text{depth} \leq 3)$ {Expand downstream dependencies, depth ≤ 3}
5: Inject skills from G_j in topological order into LLM context; LLM executes autonomously, passing data via I; 120s timeout per execution step
6: $r_j, \text{trace}_j \leftarrow \text{LLM.Solve}(t_j, G_j)$
7: **if** Any node V verification fails **then**
8: Retry the failed node once; discard this trace on second failure and **continue**
9: **end if**
10: $V_{G_j} \leftarrow \bigcup_{s \in \text{Nodes}(G_j)} V_s$ {Union of subgraph verification checkpoints}
11: **if** $\text{Verify}(r_j, V_{G_j})$ **and** task tests pass **then**
12: $\mathcal{D}_{\text{aug}} \leftarrow \mathcal{D}_{\text{aug}} \cup \{(t_j, \text{trace}_j, r_j)\}$
13: **end if**
14: **end for**
15: **return** $\mathcal{D}_{\text{aug}}$

B.2 LLM.Cluster Prompt

```
Below is a list of candidate strategies extracted from multiple
tasks. Merge semantically similar or functionally equivalent
strategies into unified skill candidates.
CANDIDATE LIST: {strategy_list}
For each merged skill, preserve the most general C (condition),
union the P (procedure) steps, and ensure V (verification)
covers all merged strategies. Output the merged skill in the
same six-component format. If two strategies have compatible I
(interface), note this as a potential composition edge.
```

B.3 LLM.Refine Prompt

The following template is used for structured skill refinement.

```
The following skill performed poorly on validation tasks. Analyze
the failure cases and refine the skill.
CURRENT SKILL: {skill_definition}
FAILURE CASES: {failure_cases}
For each failure, diagnose the root cause:        (a) C too broad --
triggered on tasks where it should not apply      (b) P incomplete --
missing critical execution steps      (c) V insufficient -- failed to
detect execution errors   (d) I ambiguous -- insufficient data passed
during composition
Available operations on each component:    KEEP: component is correct
as-is EDIT: rewrite component with improvement ADD: add new sub-steps
or checkpoints
Output the refined skill in full (meta, C, P, V, I, E) format.
```

B.4 LLM.Solve Prompt (Stage 2)

The following template is used for scaffold-graph-augmented task solving.

```
You are a coding agent solving a software engineering task with the
help of structured skills.
TASK: {task_description}
ACTIVE SKILLS (in topological execution order):   {skill_1: C, P, V,
I, E} {skill_2: C, P, V, I, E} ...
```

Algorithm 3 Progressive Distillation with Scaffold Recompilation

Require: Augmented dataset $\mathcal{D}_{\text{aug}}$, pretraining data $\mathcal{D}_{\text{pretrain}}$, validation set $\mathcal{D}_{\text{val}}$, scaffold graph G_0, model M, reference policy π_{ref}, retention threshold τ, fallback threshold δ

Ensure: Distilled model M', recompiled scaffold graph G_T

1: $M' \leftarrow \text{copy}(M)$; $G \leftarrow G_0$
2: Sort nodes in G by topological order (ties broken by descending out-degree) {Bottom-up priority}
3: **for** each skill s_k in sorted order **do**
4: $\quad \mathcal{D}_k \leftarrow \{(t_j, \text{trace}_j, r_j) \in \mathcal{D}_{\text{aug}} \mid s_k \in \text{trace}_j\}$
5: $\quad \mathcal{D}_k \leftarrow \mathcal{D}_k \cup \text{Sample}(\mathcal{D}_{\text{pretrain}}, |\mathcal{D}_k|)$ {Mix in 50% pretraining data}
6: $\quad G_{\text{prev}} \leftarrow \text{copy}(G)$ {Save graph snapshot}
7: $\quad$ Fine-tune M' on $\mathcal{D}_k$, minimizing $\mathcal{L} + \beta \cdot D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})$
8: $\quad \text{deps}_k \leftarrow \{s_h \in \text{Nodes}(G) \mid (s_k, s_h) \in D\}$ {Record dependents before removal}
9: $\quad$ Remove s_k from G {Node removal}
10: $\quad$ **for** each $s_h \in \text{deps}_k$ **do**
11: $\quad\quad$ Rewrite P_h, regenerate bridge instructions, update I_h
12: $\quad$ **end for**
13: $\quad$ **if** recompiled G fails V verification on $\mathcal{D}_{\text{val}}$ (60s timeout per node) **then**
14: $\quad\quad G \leftarrow G_{\text{prev}}$; **continue** {Re-verification failed; roll back}
15: $\quad$ **end if**
16: $\quad \text{perf_before} \leftarrow \text{Evaluate}(M', G_{\text{prev}}, \mathcal{D}_{\text{val}})$ {M' with old graph}
17: $\quad \text{perf_after} \leftarrow \text{Evaluate}(M', G, \mathcal{D}_{\text{val}})$ {M' with new graph}
18: $\quad \text{drop} \leftarrow \text{perf_before} - \text{perf_after}$
19: $\quad \text{retain}_k \leftarrow \text{Evaluate}(M', \mathcal{D}_{\text{val}}, \text{without } s_k)/\text{Evaluate}(M, \mathcal{D}_{\text{val}}, \text{with } s_k)$
20: $\quad$ **if** $\text{retain}_k \geq \tau$ **and** $\text{drop} \leq \delta$ **then**
21: $\quad\quad$ Mark s_k as "internalized"
22: $\quad$ **else**
23: $\quad\quad G \leftarrow G_{\text{prev}}$ {Roll back graph state; weight updates retained}
24: $\quad$ **end if**
25: **end for**
26: **return** M', G

```
For each active skill, follow its P (procedure) step by step.  After
completing each skill, run its V (verification) checkpoints and
report evidence (e.g., build exit code, test output).  Pass data
between skills according to their I (interface) specifications.  If
a V checkpoint fails, retry once with a corrected approach before
moving to the next skill.

Output your solution as a code patch.
```

B.5 Procedure Rewriting Prompt (Stage 3)

The following template is used for procedure rewriting during dynamic recompilation.

```
Skill {skill_k} has been internalized into model parameters and
will be removed from the scaffold graph.  The following higher-level
skill depends on it and must be rewritten.

HIGHER-LEVEL SKILL: {skill_h definition}

REMOVED SKILL: {skill_k description}

MODEL PERFORMANCE on skill_k tasks after distillation:
{retention_stats}

Rewrite skill_h's procedure P_h:  replace explicit calls to skill_k
with implicit assumptions that the model can now perform those steps
natively.  Update the data flow instructions (bridge) to remove
skill_k as an intermediary.  Keep all V checkpoints unchanged.

Output the rewritten skill_h in full (meta, C, P, V, I, E) format.
```

C Hyperparameter Sensitivity Analysis

C.1 Skill Effectiveness Threshold θ

Table 5: θ sensitivity analysis. A higher θ retains fewer but stronger skills, yielding higher distillation retention rate but lower coverage.

θ	# Skills Passed	Auto Skills Passed Rate	Distillation Retention Rate
0.02	38	31.8%	82.1%
0.05 (default)	30	32.5%	85.2%
0.08	22	31.4%	86.7%
0.10	15	29.8%	88.3%

$\theta = 0.05$ achieves the best balance between passed rate and distillation retention rate. A threshold that is too low introduces low-quality skills that add noise, while a threshold that is too high leads to insufficient coverage.

C.2 Retention Rate Threshold τ

Table 6: τ sensitivity analysis. A higher τ imposes stricter internalization quality requirements, causing more skills to be retained as external scaffolds.

τ	# Internalized Skills	Post-Distillation Passed Rate	# Skills Requiring External Scaffold
0.70	27	29.0%	3
0.85 (default)	22	31.5%	8
0.95	15	32.0%	15

$\tau = 0.85$ achieves a balance between model autonomy (independence from external skills) and performance. In practice, it can be adjusted based on tolerance for external dependencies.

C.3 Distillation Order Comparison

Table 7: Distillation order sensitivity. "# Forgetting Events" refers to the number of times an already-internalized skill's retention rate falls below τ after a distillation round.

Distillation Order	**Distillation Retention Rate**	**# Forgetting Events**
Topological bottom-up (default)	85.2%	1
Topological bottom-up (no pretrain mixing)	81.8%	2
Topological top-down	79.3%	4
Random order	80.4%	3

Topological bottom-up order performs best (retention rate 85.2%, only 1 forgetting event), because lower-level skills are internalized first and then implicitly replayed during upper-level training. Removing pretrain mixing, bottom-up still achieves 81.8%, higher than random order with pretrain mixing, indicating that topological ordering is the primary contributing factor, with pretrain mixing adding 3.4pp on top. Top-down performs worst (79.3%, 4 forgetting events). Note: this comparison is conducted on a scaffold graph of 30 skills; the differences are expected to be more pronounced in larger graph structures.

D Additional Skill Examples

D.1 Automatically Trained Skill Examples

Skill 1: Requirement Decomposition and Implementation Planning (flexible)

meta: name=requirement-decomposition, tags=[planning, multi-file], tools=[Read, Bash]

C: The task description contains multiple functional requirements or requires modifications to multiple modules.

P (flexible): (1) Decompose task requirements into independent sub-requirements. (2) Identify the files and modules involved for each sub-requirement. (3) Identify dependency relationships among sub-requirements. (4) Order the implementation plan by dependencies. (5) Define acceptance criteria for each sub-requirement.

V (evidence-driven): (a) Cross-check the original requirement list against the subtask list; count of uncovered items must be 0. (b) Run topological sort on subtask dependencies; pass if acyclic. (c) Each subtask includes at least one executable acceptance command (e.g., a test case or build check).

I: input = complete task description, output = ordered list of sub-tasks (each with file list and acceptance criteria).

E: Modifying all files at once without decomposing first—leading to missed dependencies and regression issues.

Skill 2: Test-Driven Development (TDD) (rigid)

meta: name=tdd-workflow, tags=[testing, quality], tools=[Read, Write, Edit, Bash]

C: The task requires adding or modifying functional code, and the project has testing infrastructure.

P (rigid, enforced order): (1) Analyze requirements and determine test strategy. (2) Write failing tests covering normal and edge cases. (3) Write minimal implementation to make tests pass. (4) Refactor code while keeping tests green. (5) Check test coverage $\geq$80%.

V (evidence-driven): (a) Test output from each stage serves as evidence. (b) Existing tests are not broken. (c) Test coverage for new functionality $\geq$80%.

I: input = functional requirements + project test framework info, output = test files + implementation files.

E: Writing implementation code directly without writing tests first—leading to missed edge cases and difficulty in verification.

D.2 Component-by-Component Comparison of Auto Skills and Human Skills

Table 8 provides a component-by-component comparison of the automatically trained TDD skill and the human-designed TDD skill.

Table 8: Component-by-component comparison of automatically trained and human-designed TDD skills. All four components show high correspondence.

Component	Auto Skill (automatically trained)	Human Skill (manually designed)
C	Task requires adding/modifying functionality + testing infrastructure	Triggered when implementing features or fixing bugs
P	5 steps: analyze → write tests → minimal impl. → refactor → coverage	4 stages: RED → GREEN → REFACTOR → COVERAGE, mandatory commit per stage
V	Tests pass + no regressions + coverage $\geq$80%	Git commit evidence per stage + test output + coverage $\geq$80%
I	Input: functional req. + test framework; Output: test + impl. files	planner → tdd → reviewer sequential handoff

Key differences: (1) The human-designed P is more concise (4 stages vs. 5 steps) and includes IDE-specific operations (mandatory git commit); (2) The human-designed V employs a stricter evidence-driven strategy (does not trust the model's claim of completion, requires executable evidence), while the automatically trained V is similar but does not enforce commits; (3) The human-designed I is embedded in a five-tier architecture, while the automatically trained I is a standalone input/output specification. The core structures are highly consistent, demonstrating that automatic training can arrive at skill structures similar to those designed manually.

E Catastrophic Forgetting Analysis

Table 9: Catastrophic forgetting summary during progressive distillation. Of 30 skills, 22 were internalized with an average forgetting of only 1.4pp.

Metric	**Value**
Total skills distilled	30
Successfully internalized	22
Final retention rate of earliest internalized skill	88.1%
Initial retention rate of earliest internalized skill	91.2%
Maximum forgetting magnitude	3.1pp
Average forgetting across 22 internalized skills	1.4pp

The overall forgetting magnitude is small (average 1.4pp, maximum 3.1pp), attributable to five mechanisms: (1) self-distillation introduces less distribution shift than standard SFT; (2) bottom-up distillation causes lower-level skills to be implicitly replayed in subsequent rounds; (3) mixing 50% pretraining data per round prevents distribution shift; (4) the KL divergence constraint ($\beta = 0.01$) prevents parameters from deviating too far; (5) per-round retention rate checks ensure that underperforming skills are retained as external scaffolds rather than forcibly internalized. Lower-level skills (distilled first) are repeatedly implicitly replayed in subsequent rounds, continuously reinforced, achieving the best internalization.

F Fine-Grained Analysis by Task Complexity

Table 10: Passed rate stratified by number of files involved. Skills yield the largest improvement on medium-complexity tasks.

Task Complexity	**# Tasks**	**Baseline**	**+ Auto Skills**	**Improvement**
Easy (1–2 files)	58	40.3%	47.8%	+7.5
Medium (3–5 files)	85	22.5%	32.5%	+10.0
Hard (6+ files)	57	11.0%	17.0%	+6.0

Skills yield the most significant improvement on medium-complexity tasks (+10.0pp). The smaller improvement on easy tasks (+7.5pp) is because the baseline already has a relatively high passed rate; the limited improvement on hard tasks (+6.0pp) is because even with skill guidance, highly complex tasks still exceed the current model's capabilities.

G Seed Data Details

The 2000 seed tasks were batch-generated using the FeatureBench data pipeline on additional repositories, sourced from open-source projects including django, flask, requests, celery, tornado, etc., with **no overlap** with the 24 repositories in the FeatureBench evaluation set, avoiding data leakage. Selection criteria: (1) involving modifications to at least 3 files; (2) containing identifiable strategic steps; (3) covering at least 8 development patterns. 200 validation tasks were used for skill quality evaluation. Evaluation was conducted on the FeatureBench standard Full set (200 tasks, 24 repositories), with no overlap with seed data or training data.

H Case analysis

Case 1: Success—multi-file dependency modification. A task involving 5 file modifications; the Baseline missed 2 dependency files, causing a build failure. After skill injection, "requirements decomposition" planned subtasks and "multi-file consistency check" modified and verified files in

topological order, completing the task successfully. After distillation, the model retained dependency analysis and stepwise verification behavior without skills, indicating the strategy was internalized.

Case 2: Failure—creative decision-making. A task requiring the design of an entirely new data structure. Skills helped with "how to do it" but offered limited assistance for the creative decision of "what to do," revealing that skills primarily improve procedural capabilities rather than creative ones.

Case 3: Configuration file coordination. The task required modifying an application configuration and ensuring all modules depending on that configuration work correctly. The Baseline model modified the main configuration file but missed two test environment configuration files. After injecting the "multi-file consistency check" skill, the model systematically searched for all files referencing the configuration, updating and verifying each one.

Case 4: Skill composition effect. A task requiring a new REST API. The "requirement decomposition" skill split the task into 5 sub-tasks (routing, validation, logic, formatting, testing), and the "TDD" skill handled the testing portion. The two skills were automatically chained via I: the output of requirement decomposition served as input to TDD.

Case 5: Skill false activation. A simple docstring update task was matched by the "requirement decomposition" skill (because it involved docstrings in multiple files). The skill guided the model through unnecessary dependency analysis, increasing execution time without affecting correctness. Such false activations (12% of tasks) suggest room for improvement by raising the θ threshold or refining the C component.

I Verification Mechanism Details

The design practice of 136+ manually authored skills led to a six-stage verification workflow:

1. **BUILD**: Build passes (exit code 0)
2. **TYPES**: Type checking passes with no errors
3. **LINT**: Code style checks pass
4. **TEST**: Tests pass + coverage $\geq$80%
5. **SECURITY**: Security scan (no sensitive information leakage)
6. **DIFF**: Change audit (no unexpected file modifications)

Stages are executed in strict order, halting on first failure. This layered verification provides high-quality filtering signals for distillation: only traces that pass all verifications are retained as training data.

Verification follows the evidence-driven principle: the model's claims of completion are not accepted; executable evidence is required (build exit codes, test execution output, coverage numbers, git commit records).

J Formal Composition Mechanism

Given two skills s_a and s_b, when $I_a.\text{out} \cap I_b.\text{in} \neq \emptyset$, the composed skill s_{ab} is defined as:

$$s_{ab} = (\textit{meta}_{ab},\ C_a \wedge C_b,\ P_a \oplus \text{bridge} \oplus P_b,\ V_a \cup V_b,\ I_{ab},\ E_a \cup E_b)$$

where $I_{ab}.\text{in} = I_a.\text{in} \cup (I_b.\text{in} \setminus I_a.\text{out})$, $I_{ab}.\text{out} = I_b.\text{out} \cup (I_a.\text{out} \setminus I_b.\text{in})$, $\oplus$ denotes sequential concatenation, and bridge is an automatically generated data-passing instruction. This definition preserves external inputs and unconsumed outputs. If conflicting checkpoints exist in $V_a \cup V_b$, verification is performed only on the final output, with intermediate V serving as soft constraints.

K Computational Cost Details

Table 11: Computational cost by stage. All experiments were conducted on 512×H100-80GB. Skill Discovery and Data Generation involve extensive LLM inference calls and execution-based verification, constituting the primary computational bottleneck. Distillation is standard fine-tuning with cost comparable to conventional SFT. Note that the total cost of this framework should not be directly compared with standard SFT—SFT uses pre-existing data, whereas this framework includes the data production process (Skill Discovery + Data Generation), which is an additional but necessary investment.

Stage	Primary Cost	Relative Cost
Skill Discovery	2000 seeds $\times$ LLM extract (once) + $\sim$45 candidates $\times$ T eval/refine rounds	High
Data Generation	2000 tasks $\times$ scaffold graph assembly $\times$ LLM inference $\times$ V verification	Highest
Distillation	K rounds of full-parameter fine-tuning	Medium
Dynamic Recompilation	Dependent nodes $\times$ LLM rewriting $\times$ re-verification	Low

Fine-tuning hyperparameters. Full-parameter fine-tuning uses the AdamW optimizer with learning rate 1×10^{-5}, cosine decay schedule, warmup ratio 5%, effective batch size 256, weight decay 0.01, 1 epoch per distillation round, and maximum sequence length 160K. Both models use identical fine-tuning hyperparameters.

Inference hyperparameters. Qwen3-Coder-480B-A35B-Instruct: temperature=0.7, top_p=0.8, top_k=20, repetition_penalty=1.05. DeepSeek-V3.2: temperature=1.0, top_p=0.95. Both models are configured with a maximum context length of 160K.

L Resolved Rate Supplementary Results

Table 12 reports the strict Resolved Rate (all fail-to-pass tests must pass for a task to count as resolved). Resolved Rates on FeatureBench Full are generally low, because end-to-end feature development tasks typically involve multiple tests and the bar for passing all of them is high. The relative trends across configurations are consistent with Passed Rate: Auto Skills outperform Baseline, Progressive Distilled outperforms Direct Fine-tune, confirming the robustness of main-text conclusions.

Table 12: Strict Resolved Rate (%) on FeatureBench Full. Trends are consistent with Passed Rate (Table 2).

Configuration	Qwen3-Coder	DeepSeek-V3.2
Baseline	3.4	5.4
+ Auto Skills	6.0	8.0
+ Distilled (Progressive)	5.5	7.5
+ Direct Fine-tune	4.0	5.5

M General capability preservation

To verify that progressive distillation does not degrade the base model's general capabilities, we evaluate DeepSeek-V3.2 before and after distillation on multiple standard benchmarks. Table 13 shows that post-distillation performance on general (MMLU-Pro), scientific reasoning (GPQA Diamond), and mathematics (AIME 2025) remains essentially unchanged.

Table 13: General capability comparison before and after distillation (DeepSeek-V3.2). All metrics fluctuate by $\leq$0.4pp, confirming no degradation.

Benchmark	**Before**	**After**	**Change**
LiveCodeBench (Pass@1-COT)	83.1%	83.5%	+0.4
MMLU-Pro (EM)	84.7%	84.7%	+0.0
GPQA Diamond (Pass@1)	82.4%	82.3%	−0.1
AIME 2025 (Pass@1)	92.8%	92.9%	+0.1

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state four contributions (new paradigm, new representation, new mechanism, empirical validation) and these are supported by experiments in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 7 discusses limitations including domain scope (code generation only), teacher diversity (same-model self-distillation), and provides concrete transfer directions.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper is primarily empirical. Formal definitions (Definition 1, 2) are provided for skill and scaffold graph, but no theorems or proofs are claimed.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 4 provides evaluation benchmark (FeatureBench), model specifications, experimental configurations (Table 2), seed data construction, evaluation metrics, and statistical methodology (5 random seeds). Algorithms 1–3 in the appendix provide complete pseudocode. Prompt templates are in Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The complete framework will be open-sourced upon acceptance, as stated in the conclusion. FeatureBench is an existing open-source benchmark.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Sections 3 and 4 specify all hyperparameters: $\theta = 0.05$ (Section 3.2), $\tau = 0.85$, $\delta = 0.02$, $\beta = 0.01$ (Section 3.4), 5 random seeds, 200 evaluation tasks (FeatureBench Full), 2000 seed tasks, full parameter fine-tuning (Section 4). Sensitivity analyses are in Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Table 2 reports mean $\pm$ standard deviation across 5 random seeds. The factor of variability (model inference randomness) is stated in Section 4.4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4.1 specifies model size (35B and 37B activated parameters, full parameter fine-tuning). The cost appendix provides a breakdown by stage (Discovery and Data Generation are the main bottlenecks; Distillation is standard fine-tuning), with hardware specified as 512×H100-80GB.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses publicly available benchmarks and open-source models. No human subjects or private data are involved.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: Section 8 (Social Impact) discusses the positive impact: the framework addresses the data scarcity bottleneck in complex coding tasks, provides a new path to break through performance plateaus, and lowers the technical barrier for complex programming by distilling engineering best practices into model parameters.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper proposes a training methodology, not a pre-trained model release.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: FeatureBench (MIT License), Qwen3-Coder (Apache 2.0), and DeepSeek-V3.2 (MIT License) are properly cited. All are publicly available open-source/open-weight assets with permissive licenses.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: No new datasets or models are released at submission time. The code will be open-sourced upon acceptance.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human subjects research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: The paper does not involve human subjects research.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research?

Answer: [Yes]

Justification: This paper studies LLM post-training to automatically acquire procedural strategies. LLM calls are core to skill discovery, data generation, and dynamic recompilation (Section 3).

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.