BEFORE THEY CAN SOLVE: PREDICTING POST-TRAINING CODING-AGENT PERFORMANCE FROM BASE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

How can we predict which base checkpoint is worth an expensive round of agentic post-training? End-to-end $\mathrm{pass@}K$ tests whether successful behavior already appears in a base model's distribution, but it is a poor fit for agentic coding: many base checkpoints cannot reliably produce the well-formed tool invocation required to complete a task end-to-end. Single-shot or short-horizon tasks avoid these tool-calling failures by collapsing a multi-step interaction into a fixed prompt and a single patch, but they sidestep the core capability we care about: maintaining coherent state over many tool-using steps as the repository evolves. To bridge this gap, we treat successful post-trained agent trajectories as a lookahead signal of base-model potential. Replaying each trajectory and rerunning tests after every code-changing step identifies the *decisive step*: the first step whose cumulative patch flips the repository from failing to passing, certifying that the recorded action solves the task given the prior context. Motivated by a coverage principle for agentic traces, we build three screens at this step that do not require a base checkpoint to drive the harness from a cold start: (i) Decisive-Action BPB (bits per byte) measures the probability mass on the certified action, (ii) Patch MCQ tests the checkpoint's choice between that action and alternatives rejected by the same verifier, and (iii) prefix-conditioned $\mathrm{pass@}K$ evaluates support for functionally-correct generations and credits any continuation that the tests accept. Across ten pairs of public base and post-trained models, all three screens rank the cohort in close agreement with post-trained SWE-bench Verified $\mathrm{pass@1}$. As our methods need only a benchmark's successful trajectories and its verifier, they can be applied to turn future agentic coding benchmarks into base-model evaluations.

1 INTRODUCTION

Choosing a base checkpoint to post-train is a high-stakes decision made before its agentic capabilities can be fully realized. Revealing these capabilities requires substantial investment in compute, engineering, and evaluation, all of which comes only after the checkpoint has been chosen.

Existing evaluations fail to address this problem for opposite reasons. End-to-end agentic coding benchmarks such as SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024) are too demanding for base checkpoints: $\mathrm{pass@}K$ remains near zero because untuned base models fail to operate tool-use harnesses (Appendix F). Non-agentic coding benchmarks exercise isolated coding capabilities, but do not test sustained interaction with an evolving repository, and their correlation with post-trained agentic performance is highly inconsistent. Across ten pairs of open-source base models and their eventual post-trained checkpoints, Spearman ρ with post-trained SWE-Bench Verified $\mathrm{pass@1}$ ranges from 0.830 on RepoBench XFirst to -0.394 on HumanEval (Appendix E). In short, current evaluations are either *too agentic to run* or *not agentic enough to predict downstream performance*.

In this paper, we address the question at the heart of this problem: *how can we evaluate agentic coding potential without requiring the base checkpoint to already behave like an agent?* To answer this question, we build on the coverage principle: a promising base checkpoint should retain probability mass on behaviors that post-training may later select, rather than merely maximizing average correctness or conforming to today's agent protocol (Chen et al., 2026; Yue et al., 2026). Agentic

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.

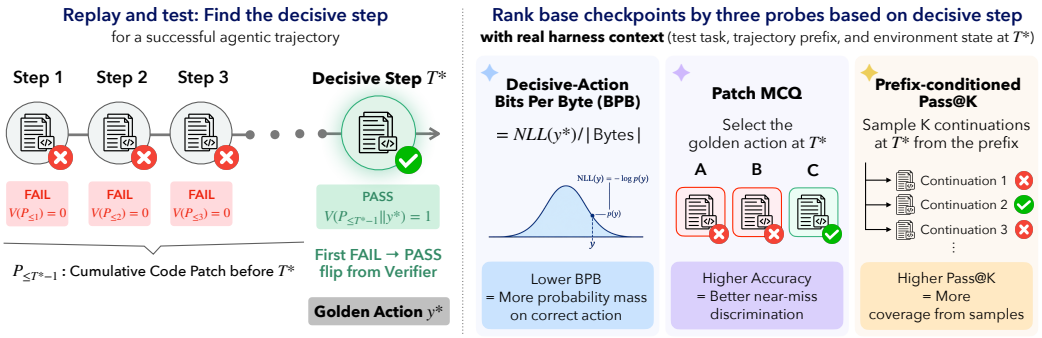


Figure 1: A unified agentic trajectory-derived probing framework. By replaying a coding-agent trajectory on a coding task against the task's own verifier V, we locate the *decisive step* $T^\star$, the earliest step whose cumulative patch resolves the task. At this decisive step, (i) Decisive-Action BPB measures the byte-normalized likelihood which a base model assigns to the golden action; (ii) Patch MCQ evaluates base model's capability of differentiating the golden action from non-resolving alternatives; (iii) Prefix-conditioned pass@K measures a base model's empirical success rate under sampling: it draws K continuations from $T^\star$ and lets the task's own verifier V judge them.

trajectories from frontier models on existing agentic coding benchmarks are a natural source for such desirable behaviors; an important follow-up is therefore *where* in an agentic trajectory to probe that probability mass. We let the verifier of the original agentic coding task decide.

For each step in a frontier post-trained LLM's successful trajectory for an agentic coding task, we reconstruct the cumulative environment state and run the task's tests at that state. This identifies the *decisive step* $T^\star$, i.e., the first step at which the verifier accepts the solution, as visualized in Figure 1. The action taken at that step is our *golden action*: conditioned on the preceding trajectory, executing it resolves the task, while the state immediately before it does not. Unlike the benchmark's *gold patch* (i.e., the human-written reference fix), this is an action produced by a real coding agent and certified by the same verifier that defines task success.

At $T^\star$, we construct three complementary probes as visualized in Figure 1. **Decisive-Action BPB** asks an absolute question: how much probability mass does a checkpoint assign to the golden action? **Patch MCQ** asks a comparative question: can a checkpoint distinguish that action from plausible alternatives rejected by the verifier? **Prefix-conditioned** pass@K asks a generative one: given the same trajectory prefix, can a checkpoint produce a continuation that passes the verifier? The three probes each answer a distinct, interpretable question about model behavior while holding the same agentic context and success criterion fixed. We validate them both by their aggregate correlation with post-trained agentic performance and by whether they correctly order individual checkpoint pairs by that same performance.

Our contributions are three-fold:

- **A framework for converting agentic coding benchmark into base-checkpoint evaluations.** We start from successful trajectories on an agentic benchmark, replay them, and run the task's tests after every code-changing step. The first step where tests pass is the decisive step: the action taken at that step becomes the golden action, and alternative edits attempted at the same point but rejected by the verifier become distractors. Each task yields (i) trajectory context until the decisive step (ii) a golden action (iii) distractors and (iv) an interactive, restorable environment state with an executable task verifier. The construction is a generic recipe rather than a fixed test set: any current or future agentic coding benchmark can be converted into a base-model evaluation.
- **Two static decision-point probes.** Decisive-Action BPB scores the base model by the likelihood it assigns to the golden action, whereas Patch MCQ scores a checkpoint by its choice among the verifier-certified golden action and the rejected distractors.
- **A dynamic continuation probe.** Prefix-conditioned pass@K samples k continuations from the decisive step and verifies each with the task's own tests; any edit that passes is accepted.

Across ten base checkpoints, all three probes produce rankings in close agreement with the ranking by post-trained performance on SWE-bench Verified. Decisive-Action BPB reaches Spearman $\rho = 0.964$ on held-out trajectories from DeepSWE (Huang et al., 2026), a separate agentic coding

benchmark. Patch MCQ reaches $\rho = 0.903$ and prefix-conditioned $\mathrm{pass@}K$ at $K = 16$ reaches $\rho = 0.988$ on trajectories drawn from SWE-bench Verified itself. Together, these results support our method of probing base checkpoints at verifier-certified decision points in real agent trajectories.

2 Related Work

End-to-end agentic evaluation. Long-horizon, end-to-end agentic evaluations place a model inside an interactive environment and score whether the resulting model–harness system completes a task. SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024) provides human-validated repository issues with test-based grading. SWE-bench Multilingual (Yang et al., 2026) extends that task format to nine languages across 41 repositories. SWE-bench Pro (Deng et al., 2025) targets longer-horizon, enterprise-style tasks. Terminal-Bench (Merrill et al., 2026) broadens the scope from repository issues to command-line workflows. DeepSWE (Huang et al., 2026) uses original, multilingual tasks with purpose-built verifiers. FrontierCode (Lu et al., 2026) moves the bar from correctness to *mergeability*, grading maintainer-authored tasks against an ensemble of unit tests. Some base checkpoint evaluation work treats end-to-end $\mathrm{pass@}K$ as a capability ceiling (Yue et al., 2026). However, these benchmarks are poorly suited to meaningfully evaluating base checkpoints that are usually too weak at such agentic capabilities.

Non-agentic code evaluation. Traditional code benchmarks isolate capabilities within bounded input–output tasks. HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) test function synthesis. EvalPlus (Liu et al., 2023) provides more rigorous criteria for assessing the functional correctness of LLM-synthesized code. APPS (Hendrycks et al., 2021) and DS-1000 (Lai et al., 2023) scale to competition problems and realistic libraries. BigCodeBench (Zhuo et al., 2025) adds complex instructions and library use. LiveCodeBench (Jain et al., 2025) emphasizes contamination-resistant competitive programming with one-step repair. CRUXEval (Gu et al., 2024) tests execution reasoning. RepoBench (Liu et al., 2024) and CrossCodeEval (Ding et al., 2023) add repository-level, cross-file context but retain a completion-style target. These benchmarks can be applied to base checkpoints, but do not test agentic behavior.

Base-checkpoint proxies for post-training potential. APTBench (Qin et al., 2025) and RuDE (Li et al., 2026) use answer discrimination to estimate base checkpoints' downstream potential. RuDE contrasts answers labeled by model-judged rubric compliance; our agentic coding setting provides executable verifiers. APTBench's SWE MCQs use reference-derived and LLM-extracted, rewritten, or generated answer choices, including gold patches paired with execution-filtered distractors. Our independent audit of 1,682 APTBench MCQs identifies questions whose designated correct option is incorrect or not uniquely correct (Appendix A); Appendix B reproduces its reported correlations on our cohort. Such defects weaken the interpretation of MCQ accuracy as the ability to distinguish correct from incorrect answers, even when aggregate scores correlate with post-training performance. We therefore construct Patch MCQ at the decisive step: the golden action is verifier-certified, while distractors are alternative actions rejected by the same verifier. This grounds the score in a specific, interpretable criterion: whether a checkpoint can distinguish the golden action from rejected plausible alternatives.

Base-model support and post-training potential. Base checkpoints can contain preference signal that their generations hide: pretrained models rank an answer with its true instruction far above a mismatched one even when their own samples are poor (Hewitt et al., 2024). Chen et al. (2026) define the "coverage" principle, where strong base models already assign high probability mass to high-quality responses. RLVR mainly sharpens solutions base models can already sample rather than creating new ones (Yue et al., 2026). Pushing further, Wu et al. (2025) find RLVR improves $\mathrm{pass@}1$, yet with larger sampling budgets it loses access to more correct solutions than it gains, compared with base checkpoints before RLVR. Together, these results suggest base-model mass bounds post-training gains—exactly the quantity our probes measure.

Likelihood-based evaluation. Likelihood has been studied heavily by the LLM community. Huang et al. (2024) show bits per character on domain corpora is near-linear in downstream benchmark performance . In fact, per-domain loss correlated across public checkpoints has been shown to be strong enough to select pretraining data (Thrush et al., 2025). It is also smoother than task accuracy, giving a more usable development signal than discontinuous task scores (Heineman et al., 2026). Raw likelihood is strongly affected by sequence length, confounding comparisons unless scores are

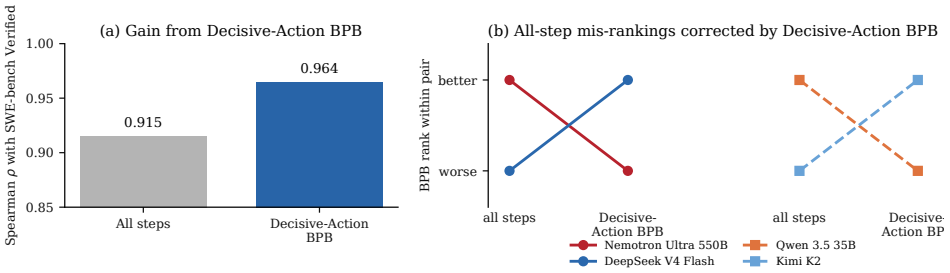


Figure 2: (a) Restricting BPB to the verifier-derived decisive step, instead of averaging over every step, raises its Spearman's rank correlation (ρ) with downstream agent capability, from 0.915 to 0.964 against SWE-bench Verified, across the same ten checkpoints. **(b)** At the checkpoint level, Decisive-Action BPB ranks DeepSeek V4 Flash above Nemotron Ultra and Kimi K2 above Qwen 3.5 35B, in agreement with their downstream performance, thus reversing the incorrect orderings produced by all-step BPB.

length-normalized (Holtzman et al., 2021). Bits per byte (BPB) is one normalization that keeps checkpoints with different tokenizers comparable (Magnusson et al., 2024). We adopt BPB when constructing our evaluation framework.

3 Trajectory-Derived Protocols

In this section, we define how we convert agentic coding benchmarks into base checkpoint evaluations. We define the decisive step in Section 3.1 and then use it to construct three probes: Decisive-Action BPB (Section 3.2), Patch MCQ (Section 3.3), and prefix-conditioned pass@K (Section 3.4).

3.1 Decisive-Step Discovery for Agentic Trajectories

Given a task from an agentic coding benchmark, *e.g*, SWE-Bench, we collect a successful trajectory of L steps from a frontier post-trained LLM using an agentic coding harness, *e.g.,* mini-swe-agent. Let $C \subseteq \{1, \ldots, L\}$ be the indices for the set of steps involving code changes. Replaying through $T \in C$ yields a cumulative code patch $P_{\leq T}$. We define the verifier-derived *decisive step* as

$$T^\star := \min\{T \in C : V(P_{\leq T}) = 1\}, \tag{1}$$

where $V(\cdot) \in \{0, 1\}$ denotes the verifier for the task. As we collected a successful trajectory, it is guaranteed that a step satisfying the above condition exists. The decisive step identifies the evaluation boundary: the evaluation context contains only steps strictly before $T^\star$.

The golden action. Let $y^\star$ denote the LLM response in step $T^\star$ of the successful trajectory. Thus, by the definition of decisive step $T^\star$ in Eq. 1,

$$V(P_{\leq T^\star - 1}) = 0, \qquad V(P_{\leq T^\star - 1} \| y^\star) = 1, \tag{2}$$

where $P_{\leq T^\star - 1} \| y^\star$ denotes the cumulative patch obtained by applying the action $y^\star$ on $P_{\leq T^\star - 1}$. We call $y^\star$ *golden given its prefix*. Before $T^\star$, none of the recorded actions induces a fail-to-pass transition, so rewarding agreement with them mainly measures imitation of the trajectory generator (or incidental edits) rather than problem-solving ability. At $T^\star$, applying $y^\star$ causes the verifier to accept the cumulative patch for the first time. After $T^\star$, later actions may introduce changes unrelated to resolving the task. Therefore, only the decisive step yields a verifier-certified golden action, whereas the steps before and after it are not golden. Figure 2 shows that this choice is not merely principled: restricting BPB to the decisive step increases rank agreement with downstream agent capability, and reverses two checkpoint pairs that all-step BPB orders incorrectly.

3.2 Decisive-Action BPB

Decisive-Action BPB measures how much probability mass a checkpoint assigns to the golden action, given its prefix. Let $x^\star$ be the trajectory prefix before the decisive steps' golden action, $y^\star$. We use one fixed universal chat template and tokenize the concatenation $x^\star \| y^\star$ into t. We then compute $L(\theta)$, the golden action's negative log-likelihood in bits for the tokens fully contained in $y^\star$, and

normalize it by the action's byte length, $|B|$, to obtain BPB. This normalization makes scores comparable across checkpoints using different tokenizers (Gao et al., 2020; Magnusson et al., 2024). Let S denote the set of token indices fully contained in $y^\star$ and B denote the set of bytes belonging to $y^\star$; Decisive-Action BPB is then formulated as

$$L(\theta) := -\sum\nolimits_{j \in S} \log_2 p_\theta(t_j \mid t_{1:j-1}), \qquad \mathrm{BPB}(\theta) := L(\theta)/|B|. \tag{3}$$

A lower BPB is better, meaning the checkpoint places more probability mass on $y^\star$.

Chat Template Masking. Base models may not be familiar with the simple chat template we use for BPB calculations. To reduce the impact of template choice on the results, we mask out formatting text added by the chat template when computing BPB (Appendix H.1).

3.3 PATCH MCQ

Decisive-Action BPB alone cannot establish whether an action resolves a task: two actions can receive similar scores even when a single-character difference causes one to pass verification and the other to fail. Our patch-based multiple-choice probe (Patch MCQ) addresses this limitation through a discriminative k-way choice, presenting the golden action alongside alternatives that fail verification when applied after the same trajectory prefix. The probe asks the checkpoint to select the action that resolves the task, rather than only measuring the likelihood of the golden action.

Conditioned on trajectory prefix $x^\star$, we sample a set of candidate actions from post-trained LLMs, denoted $\mathcal{Y}$. We label each candidate by applying it on the prefix's cumulative patch $P_{\leq T^\star - 1}$ and checking the verifier outcome. The distractor set, $\mathcal{Y}^-$, comprises the candidates that fail verification:

$$\mathcal{Y}^- = \{y : y \in \mathcal{Y} \wedge V(P_{\leq T^\star - 1} \| y) = 0\}. \tag{4}$$

Thus, every distractor is identified by its failure to pass the task's verifier rather than by a human or model-based rubric. While, the golden action $y^\star$ in the original trajectory, is the correct option.

We score all options jointly through answer-letter likelihood. The MCQ context u contains the trajectory prefix and complete option set. Let $\mathcal{L}_k := \{\ell_1, \ldots, \ell_k\}$ denote the answer-letter tokens assigned to the k displayed options (for our four-option panels, $\mathcal{L}_4 = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$). We select the most likely next-token answer letter:

$$\widehat{\ell} := \arg\max_{\ell \in \mathcal{L}_k} p_\theta(\ell \mid u), \tag{5}$$

Thus, answer-letter likelihood compares all displayed actions within one shared context rather than scoring each action independently. Let $\mathcal{Q} = \{q_i\}_{i=1}^N$ denote the set of constructed multiple-choice questions. For each question q_i, let $\hat{\ell}_i$ denote the predicted option and $\bar{\ell}_i$ denote the golden option. The patch MCQ accuracy r is computed by

$$r = \frac{1}{N} \sum_{i=1}^{n} \mathbf{1}(\hat{\ell}_i = \bar{\ell}_i), \tag{6}$$

where $\mathbf{1}(\cdot) \in \{0, 1\}$ is indicator function. To reduce positional bias, we use cyclic option reordering (Zheng et al., 2024). The answer probabilities are aggregated across reorderings before selecting the predicted answer, $\widehat{\ell}$, detailed implementation details are in Appendix I.

3.4 PREFIX-CONDITIONED pass@K

Decisive-Action BPB and Patch MCQ both score the base checkpoint's distribution over actions that were previously generated, but neither tests whether the checkpoint can successfully generate resolving actions. However, on agentic coding tasks, end-to-end pass@K for base models is typically zero even for very large K, since success often requires successive tool calls and agentic interactions with environments, which are not achievable by base models before post-training. *Prefix-conditioned* pass@K measures whether at least one of K sampled continuations from the trajectory prefix $P_{i,\leq T_i^\star - 1}$ successfully resolves the task.

We therefore condition on real trajectories as context: for each successful trajectory i from a frontier post-trained LLM, let $T_i^\star$ be its decisive step and let $P_{i,\le T_i^\star-1}$ denote the cumulative patch as identified in section 3.1. We let $x_i^\star$ be the trajectory prefix before the decisive step's golden action and $\hat{y}_{i,j}$ be a sampled continuation from a base checkpoint. *Prefix-conditioned* $\mathrm{pass@}K$ measures whether at least one of K sampled continuations from the base checkpoint resolves the task:

$$\mathrm{pass@K}(\theta) := \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} \mathbf{1}\left[\exists\, j \le K : V_i\big(P_{i,\le T_i^\star-1} \,\|\, \hat{y}_{i,j}\big) = 1\right]. \tag{7}$$

Appendix J describes the rollout mechanics, including the masking convention at $T_i^\star$, the forward budget, how a base checkpoint is bridged onto the agent interface, and how the k rollouts are verified.

4 Experiments

We evaluate all three trajectory-derived probes on the same ten base checkpoints. We measure how strongly each proposed metric: Decisive-Action BPB, Patch MCQ, and prefix-conditioned $\mathrm{pass@}K$, correlates with models' relative agentic capability after post-training. We use $\mathrm{pass@}1$ performance on SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024), SWE-bench Multilingual (Yang et al., 2026), and Terminal-Bench 2.1 (Merrill et al., 2026) as proxies for post-training agentic capability.

4.1 Methodology

The three probes are built on trajectories collected from GPT-5.6-Sol (OpenAI, 2026b), Opus-5 (Anthropic, 2026b), and Kimi-K3 (Team et al., 2026b) on SWE-Bench Verified, SWE-Pro, and DeepSWE, using mini-swe-agent (Yang et al., 2024) as the harness. The distractors in Patch MCQ are generated from a set of LLMs including Opus-4.8 (Anthropic, 2026a), Sonnet-5 (Anthropic, 2026c), GPT-5.6-Terra (OpenAI, 2026c), GPT-5.6-Luna (OpenAI, 2026a), GLM-5.2 (GLM-5-Team et al., 2026), Gemini-3.7-Flash (Google, 2026b) and Gemini-3.6-Flash (Google, 2026a).

We probe a total of ten base checkpoints: Nemotron-3-Nano (Blakeman et al., 2025), Nemotron-3-Super (Chandiramani et al., 2026), Nemotron-3-Ultra (Blakeman et al., 2026), Qwen-3.5-35B-A3B-Base (Qwen Team, 2026), DeepSeek-V4-Flash-Base (DeepSeek-AI, 2026), Kimi-K2-Base (Team et al., 2025), GLM-4.5-Air-Base (Zeng et al., 2025), Hy3-Preview-Base (Tencent Hy Team, 2026), DeepSeek-V4-Pro-Base (DeepSeek-AI, 2026), and Gemma-4-26B-A4B (Team et al., 2026a).

For each metric, we measure its correlation across the ten base checkpoints with the SWE-bench Verified and Terminal-Bench 2.1 performance of the corresponding post-trained checkpoints. We use publicly reported benchmark scores when available and run the evaluations ourselves when they are not. The full pipeline for computing each metric, from trajectory collection through verification, is described in Appendix D. Further experimental details are provided in Appendix C.

We correlate each probe's per-checkpoint score s (using $s = -\,\mathrm{BPB}$ for Decisive-Action BPB, so higher is better) with d, the downstream $\mathrm{pass@}1$ of the checkpoint's post-trained family. Pearson $r = \mathrm{cov}(s,d)/(\sigma_s\sigma_d)$ measures linear agreement; Spearman ρ, Pearson r on ranks, measures whether a probe orders checkpoints correctly, which is all checkpoint selection needs.

4.2 Correlations with SWE-Verified

Figure 3 shows that each of our probes strongly correlates with downstream SWE-bench Verified performance. While we choose DeepSWE as the source task for probe construction in Figure 3, these strong correlations hold across trajectories collected from different tasks, as shown in Figure 4.

Figure 5 repeats the analysis with probes built from SWE-bench Verified trajectories instead. All three still track post-trained performance closely, at Spearman ρ of 0.915 for Decisive-Action BPB, 0.903 for Patch MCQ, and 0.906 for prefix-conditioned $\mathrm{pass@}K$. Matching the source to the target benchmark does not, however, buy higher rank agreement. For Decisive-Action BPB and prefix-conditioned $\mathrm{pass@}K$, the held-out DeepSWE source agrees more closely ($\rho = 0.964$ versus 0.915, and 0.951 versus 0.906), and only Patch MCQ gains from the match (0.903 versus 0.867).

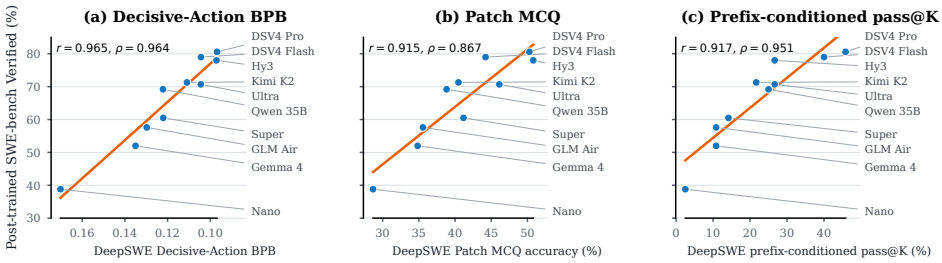


Figure 3: All three DeepSWE-derived probes strongly track post-trained SWE-bench Verified performance. The common DeepSWE source corpus makes the panels directly comparable across Decisive-Action BPB with tool-formatting tokens masked, Patch MCQ, and prefix-conditioned pass@K at $K = 32$. The same ten checkpoints appear in every panel; annotations report Pearson r and Spearman ρ. The BPB axis is reversed so rightward consistently means better.

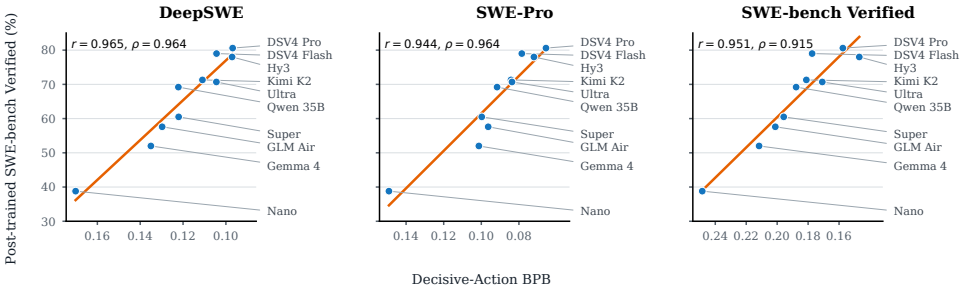


Figure 4: Decisive-Action BPB strongly tracks post-trained SWE-bench Verified performance. Each panel measures the same ten base checkpoints on a different trajectory corpus. Axes run from higher to lower BPB so rightward means better; annotations report Pearson r and Spearman ρ with $-$ BPB.

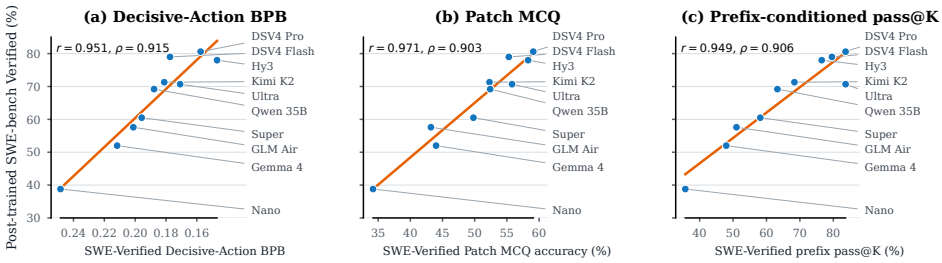


Figure 5: All three probes also track post-trained SWE-bench Verified performance when constructed from SWE-bench Verified trajectories. Panels use the same ten checkpoints and settings as fig. 3, but replace the DeepSWE source trajectories with SWE-bench Verified trajectories. Annotations report Pearson r and Spearman ρ; the BPB axis is reversed so rightward consistently means better. Because the source and downstream target use the same benchmark, this is same-benchmark rather than cross-benchmark evidence.

Bounded code benchmarks do not recover the post-trained ranking. Table 7 in section E correlates the same ten checkpoints' scores on bounded code benchmarks with the same downstream targets. Synthesis benchmarks carry almost no rank signal (MBPP $\rho = 0.091$, HumanEval $\rho = -0.394$), and even the strongest baseline, RepoBench XFirst ($\rho = 0.830$), trails all three DeepSWE-derived probes (ρ of 0.867–0.964).

4.3 Correlations with SWE-Multilingual

Figure 6 tests whether the same DeepSWE-derived probes predict post-trained performance beyond SWE-bench Verified. On SWE-bench Multilingual, rank agreement holds or improves. All three probes keep strong rank agreement with SWE-bench Multilingual, at $\rho = 0.976$ for Decisive-Action

BPB, 0.927 for Patch MCQ, and 0.945 for prefix-conditioned pass@K; the first is the highest of the nine probe–target pairs in figs. 3, 6 and 9, and Patch MCQ agrees more closely here than with SWE-bench Verified (0.867). One plausible contributor is that the DeepSWE source is itself multilingual. Pearson r is noticeably lower, 0.82–0.85, and the gap between the two statistics has a single cause: Gemma 4 26B scores 17.3% on SWE-bench Multilingual against 52.0% on SWE-bench Verified. Correlations with Terminal-Bench 2.1 are reported in section G.

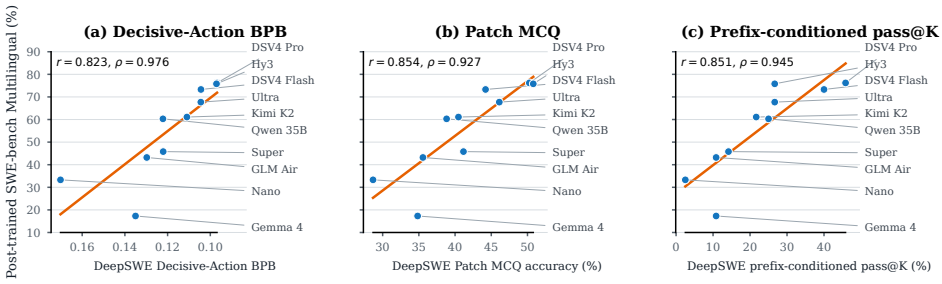


Figure 6: The three DeepSWE-derived probes retain strong rank agreement with post-trained SWE-bench Multilingual performance. Panels use the same ten checkpoints and the same probe settings as fig. 3; annotations report Pearson r and Spearman ρ. The BPB axis is reversed so rightward consistently means better.

4.4 SFT Experiments

Base checkpoint	Post-SFT SWE-Verified	Public SWE-Verified	Decisive-Action BPB	Patch MCQ	Prefix pass@32
Nemotron-3-Nano-Base	28.5	38.8	0.149	33.4	13.1
Nemotron-3-Super-Base	53.1	60.5	0.100	34.4	34.6
Qwen-3.5-35B-A3B-Base	**61.2**	**69.2**	**0.092**	**36.2**	**46.9**

Table 1: Under the same SFT procedure, post-SFT SWE-bench Verified follows the same ordering as the publicly reported scores, and the probes order the checkpoints consistently with it. Probe results are computed with SWE-Pro trajectories.

Under a controlled SFT recipe, the correlation trend continues. The downstream performance results in Sections 4.2 and 4.3 are computed using publicly released post-trained models, each produced by a different lab with their own post-training recipe and evaluation harness. To isolate the contribution of the base checkpoint, we fine-tune three base checkpoints, namely, Nemotron-3-Nano–Base, Nemotron-3-Super-Base, and Qwen-3.5-35B-A3B-Base, with an identical SFT recipe. Each model is trained on the same internal corpus for 1,000 steps, amounting to 16.8B training tokens. Table 1 shows the resulting checkpoints' performance on SWE bench Verified, controlled for the same harness and evaluation hyperparameters. The post-SFT ordering agrees with the publicly reported one, and all three probes computed on SWE-Pro trajectories order the checkpoints the same way. With other trajectory sources the agreement is weaker for some probes (Table 16 in Appendix K), and with only three checkpoints we read this as the correlation trend continuing under a controlled recipe rather than as a precise estimate.

4.5 Ablation Studies

Decisive step ablation. In section 3.1, we hypothesize that decisive steps are the best parts of the trajectory to evaluate. We further examine the importance of decisive steps in (Figure 7b), where we find decisive step BPB correlates much better than all turns.

Format masking ablation. We ablate whether chat-template tokens (specifically the tool-call formatting) contribute to Decisive-Action BPB, holding the trajectory, action, and semantic content fixed. Masking the tool-call formatting improves Spearman agreement on all our corpora (Figure 7a). The difference is especially large for SWE-verified traces. We hypothesize that this is due to the fact that SWE-verified often has early decisive actions. At these early steps the base model has

not seen a lot of prior tool-call formatting examples, causing BPB spikes during evaluation. These findings support our methodology in Section 3.2.

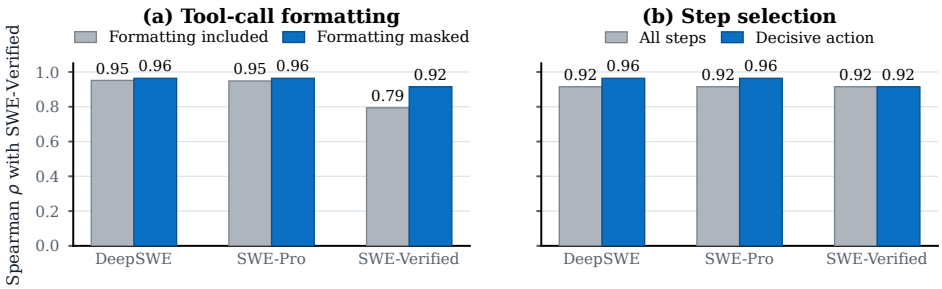


Figure 7: Masking tool-call formatting and scoring only decisive actions improve the BPB screen. Bars report Spearman correlation with post-trained SWE-bench Verified; all correlations use $-\mathrm{BPB}$ so higher is better.

Patch MCQ depends on scoring all options jointly. Table 2 ablates how Patch MCQ selects an answer. Scoring each option independently and selecting the lowest BPB (argmin-BPB) shows only weak agreement with post-trained SWE-bench Verified ($\rho = 0.665$ on DeepSWE questions; $\rho = 0.578$ on SWE-Verified questions). Presenting all four options in a single context and reading the answer-letter likelihood under cyclic rotation increases agreement to $\rho = 0.806$ and $\rho = 0.842$ without reasoning; allowing reasoning before the answer yields smaller additional gains (to 0.867 and 0.903). Thus, joint scoring explains most of the agreement on both question sets, while reasoning provides a modest refinement. Detailed results are in Appendix I.

A modest sampling budget suffices for prefix-conditioned $\mathrm{pass@}K$**.** Table 3 varies the number of sampled continuations K. On held-out DeepSWE prefixes, rank agreement rises from $\rho = 0.915$ at $K = 8$ to 0.952 at $K = 16$ and then plateaus (0.951 at $K = 32$). On same-benchmark SWE-bench Verified prefixes it is already 0.988 at $K = 8$ and $K = 16$ but falls to 0.906 at $K = 32$, where pass rates approach saturation (up to 83.7%) and Nemotron Ultra ties DeepSeek V4 Pro. Larger K therefore does not monotonically improve the screen, and $K = 16$ captures nearly all of the attainable agreement on both corpora. Detailed results are in Appendix J.

Scoring method	DeepSWE	SWE-Verified
Argmin-BPB	0.665	0.578
Rotated, no think	0.806	0.842
Rotated, think	0.867	0.903

Table 2: Patch MCQ scoring-method ablation: Spearman correlation with post-trained SWE-bench Verified on ten base checkpoints.

K	DeepSWE	SWE-Verified
8	0.915	0.988
16	0.952	0.988
32	0.951	0.906

Table 3: Prefix-conditioned $\mathrm{pass@}K$ sampling-budget ablation: Spearman correlation with post-trained SWE-bench Verified on ten base checkpoints.

5 Conclusion

We introduce a framework for evaluating base checkpoints' agentic coding potential before committing to expensive post-training. The framework uses successful agent trajectories to provide the context that base checkpoints cannot yet reliably produce themselves and the task's verifier to identify the decisive step, the first step that resolves the task. Probing checkpoints at the decisive step lets us assess their agentic coding potential without requiring them to operate a tool-use harness across an entire trajectory. Using this framework, we build three complementary screens: Decisive-Action BPB, Patch MCQ, and Prefix-Conditioned $\mathrm{pass@}K$. Across ten base checkpoints, all three produce rankings in close agreement with post-trained SWE-bench Verified performance. The framework thus extends existing and future agentic coding benchmarks into tools for base checkpoint selection before expensive post-training.

LLM USAGE DISCLOSURE

In this work, we used generative AI tools for design or provide feedback on research methodology or experiments, implement methods, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis, and interpret results. We have not used generative AI tools for generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We aim to release, alongside the paper, the corpus construction and turn-extraction rules for all corpora (SWE-Pro, SWE-Verified and DeepSWE), the universal and seven alternative chat-template renderers, the payload/identifier/combined-span region-mask definitions, the decisive-step discovery code (candidate detection, replay driver, bisect/scan search), the Patch MCQ set-building and cyclic-rotation scoring code, the prefix-conditioned $\mathrm{pass@}K$ codes, and the bounded-code baseline measurements. Appendix L gives the full checklist.

REFERENCES

Anthropic. Claude opus 4.8. https://www.anthropic.com/news/claude-opus-4-8, 2026a.

Anthropic. Claude opus 5. https://www.anthropic.com/news/claude-opus-5, 2026b.

Anthropic. Claude sonnet 5. https://www.anthropic.com/news/claude-sonnet-5, 2026c.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, et al. Nemotron 3 nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2512.20848*, 2025.

Aaron Blakeman, Aaron Thomas, Aastha Jhunjhunwala, Abhibha Gupta, Abhinav Khattar, Adam Rajfer, Adi Renduchintala, Adil Asif, Aditya Vavre, Adriana Flores Miranda, et al. Nemotron 3 ultra: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2606.15007*, 2026.

Aakshita Chandiramani, Aaron Blakeman, Abdullahi Olaoye, Abhibha Gupta, Abhilash Somasamudramath, Abhinav Khattar, Adeola Adesoba, Adi Renduchintala, Adil Asif, Aditya Agrawal, et al. Nemotron 3 super: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2604.12374*, 2026.

Fan Chen, Audrey Huang, Noah Golowich, Sadhika Malladi, Adam Block, Jordan Ash, Akshay Krishnamurthy, and Dylan Foster. The coverage principle: How pre-training enables post-training. In *International Conference on Learning Representations*, volume 2026, pp. 112225–112293, 2026.

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

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Yangruibo Ding, Zijian Wang, Wasi Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, et al. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. *Advances in Neural Information Processing Systems*, 36:46701–46723, 2023.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Google. Gemini 3.6 flash. https://ai.google.dev/gemini-api/docs/models/gemini-3.6-flash, 2026a.

Google. Gemini 3.7 flash. https://ai.google.dev/gemini-api/docs/models/gemini-3.7-flash, 2026b.

Alex Gu, Baptiste Roziere, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida Wang. CRUXEval: A benchmark for code reasoning, understanding and execution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 16568–16621, 2024.

David Heineman, Valentin Hofmann, Ian Magnusson, Yuling Gu, Noah Smith, Hanna Hajishirzi, Kyle Lo, and Jesse Dodge. Signal and noise: A framework for reducing uncertainty in language model evaluation. *Advances in Neural Information Processing Systems*, 38:17073–17114, 2026.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

John Hewitt, Nelson F Liu, Percy Liang, and Christopher D Manning. Instruction following without instruction tuning. *arXiv preprint arXiv:2409.14254*, 2024.

Ari Holtzman, Peter West, Vered Shwartz, Yejin Choi, and Luke Zettlemoyer. Surface form competition: Why the highest probability answer isn't always right, 2021. URL https://arxiv.org/abs/2104.08315.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. Deepswe: Measuring frontier coding agents on original, long-horizon engineering tasks. *arXiv preprint arXiv:2607.07946*, 2026.

Yuzhen Huang, Jinghan Zhang, Zifei Shan, and Junxian He. Compression represents intelligence linearly. *arXiv preprint arXiv:2404.09937*, 2024.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345. PMLR, 2023.

Xiaoyuan Li, Yubo Ma, Kexin Yang, Moxin Li, Keqin Bao, Wenie Wang, Fuli Feng, and Dayiheng Liu. On predicting the post-training potential of pre-trained llms. *arXiv preprint arXiv:2605.11978*, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Tianyang Liu, Canwen Xu, and Julian McAuley. RepoBench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations*, 2024.

Eric Lu, Ben Pan, Deniz Birlikci, Sam Lee, Ray Wang, Rohan Choudhury, Fermi Ma, TC Qin, Carlo Baronio, and Silas Alberti. Introducing FrontierCode. https://cognition.com/blog/frontier-code, 2026.

Ian Magnusson, Akshita Bhagia, Valentin Hofmann, Luca Soldaini, Ananya H Jha, Oyvind Tafjord, Dustin Schwenk, Evan Walsh, Yanai Elazar, Kyle Lo, et al. Paloma: A benchmark for evaluating language model fit. *Advances in Neural Information Processing Systems*, 37:64338–64376, 2024.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026.

OpenAI. Introducing SWE-bench Verified. https://openai.com/index/introducing-swe-bench-verified/, August 2024.

OpenAI. Gpt-5.6-luna. https://openai.com/index/gpt-5-6, 2026a.

OpenAI. Gpt-5.6-sol. https://openai.com/index/gpt-5-6, 2026b.

OpenAI. Gpt-5.6-terra. https://openai.com/index/gpt-5-6, 2026c.

Jiarui Qin, Yunjia Xi, Junjie Huang, Renting Rui, Di Yin, Weiwen Liu, Yong Yu, Weinan Zhang, and Xing Sun. Aptbench: Benchmarking agentic potential of base llms during pre-training. *arXiv preprint arXiv:2510.24397*, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026a.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026b.

Tencent Hy Team. Hy3. https://hy.tencent.com/research/hy3, 2026.

Tristan Thrush, Christopher Potts, and Tatsunori Hashimoto. Improving pretraining data using perplexity correlations. In *International Conference on Learning Representations*, volume 2025, pp. 17475–17509, 2025.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why rlvr may or may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

John Yang, Kilian Lieret, Carlos Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. *Advances in Neural Information Processing Systems*, 38, 2026.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Advances in Neural Information Processing Systems*, 38:57654–57689, 2026.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. Large language models are not robust multiple choice selectors. In *International Conference on Learning Representations*, volume 2024, pp. 19426–19454, 2024.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025.

A Independent Audit of SWE -MCQ Construction

To evaluate whether APTBench's (Qin et al., 2025) SWE MCQs have a uniquely correct designated answer and incorrect distractors, we audited all 1,682 questions in four components: Fix Patch, Plan, Test Patch, and Handle Error. We developed a rubric with agent assistance and applied it through automated review, assessing whether the designated answer correctly addresses the stated question and whether another answer choice is also correct. The audit included code execution and inspection of the underlying repository where needed. table 4 reports the audit results: 727 questions (43.2%) were classified as Invalid, meaning that the rendered question does not support the designated answer choice as uniquely correct.

Table 4: Rubric-based assessment of 1,682 APTBench SWE MCQs. An MCQ is classified as Invalid when the question does not have a uniquely correct designated answer and incorrect distractors. Percentages are relative to all questions in each row.

Component	All questions	Invalid questions	Invalid (%)
Fix Patch	232	63	27.2%
Plan	243	81	33.3%
Test Patch	1,060	450	42.5%
Handle Error	147	133	90.5%
Total	1,682	727	43.2%

The audit identifies failures of both answer correctness and uniqueness. For example, Handle Error MCQ #67 asks how SceneNN data was processed, but its designated correct option recommends reinstalling `torch_scatter`, which does not correctly answer the stated question. In Plan MCQ #66, answer choices A and C contain identical text, "I need to run the script again," but only A is designated correct. This MCQ therefore accepts answer choice A as correct while rejecting identical answer choice C.

These examples show that selecting the designated answer choice need not mean identifying a uniquely correct answer: scoring can reward an incorrect answer or reject an identical answer that was not designated correct. Such defects weaken the interpretation of MCQ accuracy as the ability to distinguish correct from incorrect answers, even when aggregate scores correlate with post-training performance, as our APTBench evaluation in section B illustrates. These findings motivate Patch MCQ's stricter construction (section 3.3): explicit validation of each answer choice. Patch MCQ requires the golden action to resolve the task and each distractor to fail the same verifier from the same environment state (section 3.3), grounding its answer choices in observed task outcomes.

B APTBench Reproduction

We evaluate APTBench on the same ten base checkpoints used to evaluate the three probes, as described in section 4.1. Table 5 reports Pearson and Spearman correlations between these checkpoints' APTBench scores and the SWE-bench Verified performance of their corresponding post-trained releases.

C Experimental Setup Details

Decisive-Action BPB corpora and checkpoints. We evaluate agent trajectories from DeepSWE, SWE-Pro, and SWE-bench Verified on the same ten checkpoints listed in table 6. The primary Decisive-Action BPB results and the all-step versus decisive-step ablation use this matched ten-checkpoint cohort throughout. On DeepSWE, the canonical comparison reported in figs. 2 and 7 increases Spearman correlation with post-trained SWE-bench Verified from 0.915 to 0.964.

Patch MCQ cohorts. Table 10 uses two four-option question sets, each with chance accuracy 0.250: DeepSWE contributes 756 questions over 392 instances, and SWE-bench Verified 576 questions over 249 instances. Both cover the same ten checkpoints under matched `none` preambles. We report one accuracy per checkpoint and setting, as recorded in our source measurements; the run-to-run spread behind those figures is not stored alongside them, so we attach no repeat counts or rerun intervals to individual cells and do not read small per-checkpoint differences as measurements of a difference. Detailed construction, coverage, and provenance auditing are reported with the results and in section I.

Prefix-conditioned $\mathrm{pass@}K$ cohort. The sampled probe of section 3.4 replays decisive-step prefixes on 120 DeepSWE instances and on 98 SWE-bench Verified prefixes, over the same ten-checkpoint cohort, at $k \in \{8, 16, 32\}$ rollouts per instance. Rollouts fork at the decisive step: earlier steps are served verbatim from the captured trajectory, the decisive step is masked, and the candidate checkpoint keeps control for the rest of the episode, with each rollout scored by the task's own benchmark verifier. DeepSWE is cross-domain relative to downstream SWE-bench Verified;

Table 5: APTBench reproduction on the same ten base checkpoints evaluated by the three probes (section 4.1). Pearson and Spearman correlations between APTBench scores and SWE-bench Verified performance of the corresponding post-trained release for each checkpoint.

	SWE-bench Verified	
APTBench metric	Pearson	Spearman
APTBench-SWE average	0.942	0.879
APTBench-DR average	0.962	0.903
APTBench-SWE: EnvSetup Plan ACC	0.910	0.818
APTBench-SWE: EnvSetup Action EM	0.851	0.842
APTBench-SWE: EnvSetup Handle Error ACC	0.871	0.770
APTBench-SWE: IssueFix Locate ACC	0.457	0.377
APTBench-SWE: IssueFix Fix Patch ACC	0.792	0.830
APTBench-SWE: IssueFix Plan ACC	0.873	0.806
APTBench-SWE: IssueFix Action EM	0.394	0.232
APTBench-SWE: IssueFix Test Patch ACC	0.434	0.316
APTBench-DR: Closed Plan EN ACC	0.923	0.842
APTBench-DR: Closed Plan ZH ACC	0.903	0.855
APTBench-DR: Closed Action EN EM	0.834	0.646
APTBench-DR: Closed Action EN ROUGE-1	0.929	0.903
APTBench-DR: Closed Action ZH EM	0.881	0.821
APTBench-DR: Closed Action ZH ROUGE-1	0.964	0.988
APTBench-DR: Open Plan EN ACC	0.823	0.491
APTBench-DR: Open Citation EN ACC	0.695	0.721
APTBench-DR: Open Citation ZH ACC	0.859	0.927
APTBench-DR: Open Action EN ACC	0.762	0.650
APTBench-DR: Open Action ZH ACC	0.750	0.515

Table 6: Evaluation cohort and downstream targets: SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024) and Terminal-Bench 2.1 (Merrill et al., 2026). Each row pairs a base checkpoint with the post-trained release whose reported performance we use as the downstream target; naming it matters because several families shipped more than one. All targets are pass@1 percentages.

Base checkpoint	Post-trained model	SWE-bench Verified	SWE-bench Multilingual	Terminal-Bench 2.1
Nemotron-Nano-Base	Nemotron-Nano	38.8	33.3	6.7
Nemotron-Super-Base	Nemotron-Super	60.5	45.8	38.6
Nemotron-Ultra-Base	Nemotron-Ultra	70.7	67.7	53.9
Qwen-3.5-35B-A3B-Base	Qwen-3.5-35B-A3B	69.2	60.3	48.3
DeepSeek-V4-Flash-Base	DeepSeek-V4-Flash	79.0	73.3	78.7
Kimi-K2-Base	Kimi-K2-Thinking	71.3	61.1	34.8
GLM 4.5-Air-Base	GLM-4.5-Air	57.6	43.2	23.6
Hy3-Preview-Base	Hy3	78.0	75.8	63.3
DeepSeek-V4-Pro-Base	DeepSeek-V4-Pro	80.6	76.2	78.7
Gemma-4-26B	Gemma-4-26B-IT	52.0	17.3	39.0

the SWE-bench Verified arm uses prefixes and downstream validation on the same benchmark. Because the candidate drives the harness after the fork, these scores are properties of a model–harness system and are not comparable across agent formats.

Statistical protocol. All reported comparisons use the explicitly stated completed coverage for that experiment: ten models for the primary Decisive-Action BPB, Patch MCQ, and prefix-conditioned pass@K downstream correlations, 99 trajectories for the template intersection, and the per-column available cells for table 10. We report Pearson r and Spearman ρ (average ranks for ties) as descriptive point estimates, with no bootstrap intervals available, treating all coefficients as indicative rather than calibrated predictive guarantees.

D EVALUATION INFRASTRUCTURE

All three probes run on one pipeline and diverge only at the final scoring stage. Figure 8 shows the stages and what dominates the cost of each. The design constraint throughout is that verifier executions, not forward passes, are the expensive resource: a single candidate step must be replayed into a repository and have the task's own test suite run against it, so the pipeline is organized to bound how many times that happens per trajectory.

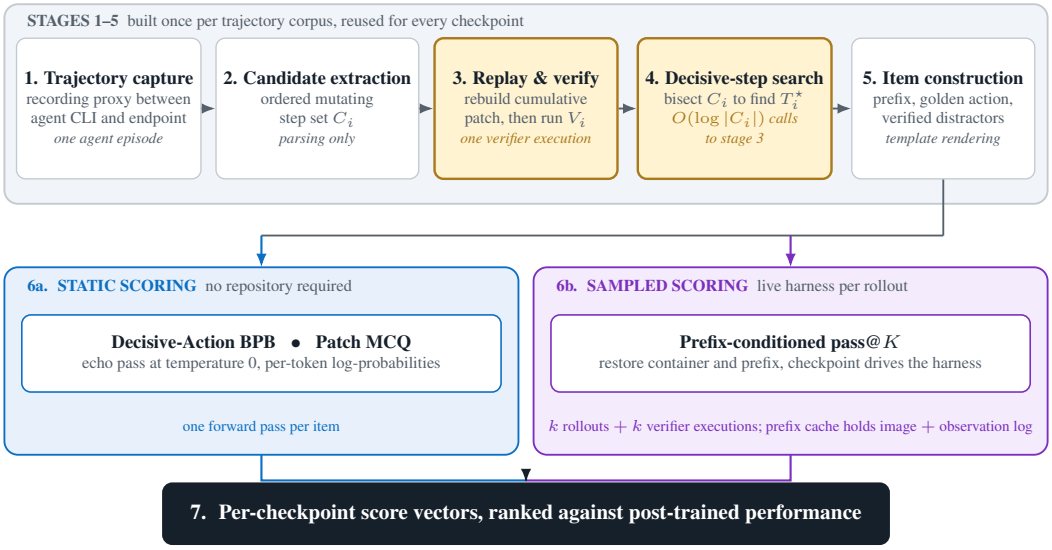


Figure 8: The pipeline shared by all three probes. Stages 1–5 run once per trajectory corpus and produce artifacts reused for every checkpoint; stage 6 is the only per-checkpoint work, and it is where the static and sampled probes diverge by orders of magnitude. Verifier executions, not forward passes, are the scarce resource, which is why stage 4 bisects rather than scans and why the sampled path caches prefixes.

Trajectory capture. A recording proxy sits between the agent CLI and the model endpoint, tees every wire request and response into a local store tagged by (batch, instance), and a post-hoc pass groups those rows into per-step records. Capture is therefore agent-agnostic, which is what lets one pipeline ingest four agent families – a Responses-API agent, a Messages-API agent, a chat-completions agent, and a host-side agent that executes each shell command through the container – and recover the family automatically from the captured wire format during replay. It also means the recorded step boundaries are the agent's own, not a reconstruction of them.

Per-instance execution environment. Every instance runs in its own container built from the benchmark's published task image, with the repository checked out and the interpreter environment the task ships. Before an episode the working tree is reset to the task's base commit with `git reset -hard` followed by `git clean -fd`; the `-x` flag is deliberately omitted, because these images keep compiled extensions and packaging metadata as git-ignored build artifacts, and deleting them leaves a tree that cannot import the package under test. The patch is extracted with `git add -A` and a binary diff against the base commit, so the same ignore rules keep those artifacts out of the extracted patch. Prompts carry the problem statement only; the benchmark's hint fields are deliberately excluded as a leakage control.

Replay and verification. Stage 3 is the only component that executes untrusted code. For a candidate step it replays the recorded steps into a fresh container, extracts the cumulative patch, and runs the benchmark's own verifier, recording the replay driver, the patch content or hash, the verifier command and environment, the pass/fail outcome, and an explicit infrastructure-error status (section L). That last field is load-bearing: a verifier that rejects a patch is a normal search observation, whereas a verifier that could not run at all must not be read as rejection, so the two are recorded separately and only the latter invalidates a trial. Replay is strict in one specific way – a replayed command that does not match the recorded one raises rather than falling through to live execution. Falling through would leave the conversation the candidate reads describing a filesystem that a different command

sequence produced, which yields a wrong-but-plausible score rather than an obvious crash. When the candidate takes over, its k rollouts are scored in k separate harness runs, because the harness keys predictions by instance identifier and k patches for one instance would otherwise collide.

Search strategy and its cost. Stage 4 bisects C_i rather than scanning it, which reduces the verifier executions per trajectory from $|C_i|$ to $O(\log |C_i|)$ and is the single largest infrastructure saving in the pipeline. Bisection assumes monotonicity – once a cumulative patch passes, later prefixes also pass – and a linear `-scan` mode exists for corpora that violate it. Both modes first verify $\max C_i$ as a gate, so a trajectory whose final step does not resolve is discarded after one verifier execution instead of an entire search. All decisive-step metadata reported in this paper was produced by bisect.

Throughput limits and the failure mode they create. Both scoring paths are endpoint-bound rather than client-bound: raising client concurrency on a saturated endpoint changed static-scoring throughput not at all, while two hosts differed by more than an order of magnitude on identical work. Shared endpoints also impose a concurrency ceiling, and exceeding it fails in a way that is not self-announcing. Past the ceiling the retry budget exhausts, a rate-limited forward returns nothing, and the rollout falls back to a prefix-only patch that is indistinguishable from a legitimate failed attempt – a degraded window reads as a low score rather than as an error. We observed one candidate score zero under saturation and 0.347 on a clean re-run, with the size of the correction tracking each arm's measured rate-limit fraction. Two safeguards follow: runs are capped below the measured ceiling, and empty predictions written during a degraded window are purged rather than resumed over, since resumption logic keyed on file existence would otherwise treat them as complete forever. Long prefixes are dropped when they exceed an endpoint's context limit, and because the longest prefixes are also the most-explored instances, those drops are a selection effect and comparisons are made on the completed intersection (section C).

Sampling configuration. For base checkpoints driving the harness, sampling temperature is 0.6.

E Bounded Code Benchmarks as Baselines

Section 1 argues that bounded code benchmarks – those a base checkpoint can complete without sustained interaction with an evolving repository – do not measure whether it can become a capable coding agent. Table 7 quantifies that claim on the ten-checkpoint cohort of table 6 by correlating each benchmark's base-model score with the same two post-trained downstream targets used throughout. BigCodeBench (Zhuo et al., 2025) is named in section 1 as a member of this family but is not among our measurements, so it does not appear here; the two LiveCodeBench variants and the two CRUXEval directions are reported separately because they score different abilities.

Table 7: Bounded code benchmarks as baselines for post-trained agentic performance, with our three probes repeated for reference. Correlations are against each post-trained downstream target on the cohort of table 6; the probe rows use the DeepSWE-derived probes of figs. 3 and 9, with $-\,\mathrm{BPB}$ for Decisive-Action BPB. Two baselines cover only eight checkpoints, so their coefficients are not matched pairs with the rest. At $n \leq 10$ with no interval estimates these are descriptive associations, and single-row differences should not be over-read.

		vs. SWE-bench Verified		vs. Terminal-Bench 2.1	
Evaluation	n	Pearson	Spearman	Pearson	Spearman
MBPP	10	0.151	0.091	0.037	0.036
HumanEval	10	−0.369	−0.394	−0.650	−0.571
LiveCodeBench	10	0.523	0.455	0.429	0.383
CRUXEval-I	10	0.473	0.394	0.287	0.201
CRUXEval-O	10	0.814	0.758	0.618	0.523
RepoBench XFirst	10	0.690	**0.830**	0.557	0.505
Decisive-Action BPB, decisive step	10	0.965	0.964	0.863	0.833
Patch MCQ, think 1000	10	0.915	0.867	0.871	0.796
Prefix-conditioned pass@K, $K = 32$	10	0.917	0.951	0.947	0.930

The family spans from negatively ranked to moderate, and none matches our method. RepoBench XFirst is the strongest bounded baseline by rank agreement ($\rho = 0.830$), which is the

figure section 1 and section 5 quote as the comparison point for Decisive-Action BPB. CRUXEval-O follows and is genuinely informative ($r = 0.814$, $\rho = 0.758$): predicting the output of a program exercises reasoning about execution, which is closer to what resolving a repository issue demands than synthesizing a standalone function is. The synthesis benchmarks supply almost no signal on this cohort – MBPP is flat ($\rho = 0.091$) and HumanEval is *negatively* ranked ($\rho = -0.394$) – and the pattern is consistent with range compression rather than noise alone: across these ten checkpoints MBPP spans only 67.2–86.0 and RepoBench XFirst 73.2–82.8, while the downstream target spans 38.8–80.6. A benchmark whose scores are bunched near the top cannot order checkpoints that differ by more than forty points downstream.

What this does and does not establish. Two readings are unsupported. First, these coefficients do not show that bounded benchmarks measure nothing: CRUXEval-O and RepoBench XFirst both carry real signal, and a practitioner with no trajectory corpus would be better off consulting them than nothing. Second, the comparison is not controlled. The baselines are single published-style scores, our probes are computed on trajectories we selected, and every downstream score comes from a different lab's post-training recipe and harness. What the table supports is narrower and sufficient for section 1: within one cohort, agreement with post-trained agentic performance varies enormously across bounded benchmarks, the best of them still trails all three trajectory-derived probes, and nothing in the family distinguishes itself in a way that would let a practitioner pick one in advance.

F End-to-End Evaluation of Base Checkpoints

If base checkpoints could simply be run end-to-end on an agentic coding benchmark, the probes in section 3 would be unnecessary. To assess the need for the proposed probes, we evaluate base models directly in an end-to-end setting. We run six base checkpoints on SWE-bench Verified, driving the harness from the task statement (with no trajectory context), and results are list in table 8.

Table 8: End-to-end evaluation of base checkpoints on SWE-bench Verified, alongside each family's post-trained score on the same benchmark. All values are $\mathrm{pass@1}$ percentages.

Base checkpoint	End-to-end $\mathrm{pass@1}$	Post-trained SWE-Verified
DeepSeek-V4-Pro-Base	0.0	**80.6**
DeepSeek-V4-Flash-Base	0.0	79.0
Nemotron-Ultra-Base	0.0	70.7
Qwen-3.5-35B-A3B-Base	**20.2**	69.2
Nemotron-Super-Base	0.0	60.5
Nemotron-Nano-Base	0.0	38.8

End-to-end evaluation cannot rank base checkpoints, and the one signal it gives points the wrong way. Five of the six checkpoints resolve no task at all, so the benchmark ties them at the floor and cannot order them. Only Qwen-3.5-35B-A3B-Base resolves any, at 20.2%, which ranks it first end to end even though its post-trained family is fourth of the six, behind both DeepSeek V4 models and Nemotron Ultra. End-to-end $\mathrm{pass@1}$ therefore ties five of the six checkpoints at the floor, and its one non-zero score places a mid-ranked family first: it is not merely uninformative but, where it discriminates at all, misleading. The exception of Qwen-3.5-35B-A3B-Base more plausibly reflects its familiarity with agent-formatted interaction than coding ability.

G Correlations with Terminal-Bench 2.1

Figure 9 tests whether the DeepSWE-derived probes of section 4.3 also predict post-trained performance on Terminal-Bench 2.1, which scores command-line workflows rather than repository issues. On Terminal-Bench 2.1, the prefix-conditioned $\mathrm{pass@}K$ transfers best. Agreement is lower for the two static probes, at $\rho = 0.833$ for Decisive-Action BPB and 0.796 for Patch MCQ, while prefix-conditioned $\mathrm{pass@}K$ holds at 0.930.

Where the Terminal-Bench gap comes from. Two checkpoints account for most of the disagreement between the benchmarks. Kimi K2 ranks fourth on SWE-bench Verified but eighth on

Terminal-Bench, and Gemma 4 26B ranks ninth and sixth. The probes side with the SWE view of both, placing Kimi K2 fifth or sixth and Gemma 4 26B eighth or ninth, which is what costs them agreement with Terminal-Bench. The shortfall is thus a property of how the two benchmarks rank these models rather than a failure of the probes on the SWE ranking they were built from. Capturing Terminal-Bench-specific standing would likely require trajectories from command-line tasks, and aggregating probes across trajectory sources from several task families is a natural way to do so.

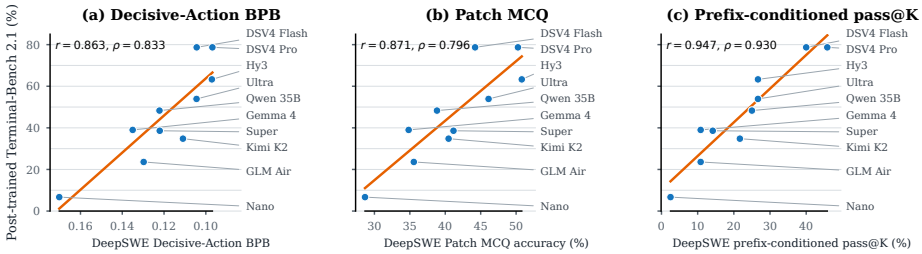


Figure 9: DeepSWE prefix-conditioned pass@K **at** $K = 32$ **has the strongest descriptive agreement with post-trained Terminal-Bench 2.1 performance.** Panels use the same ten checkpoints and the same probe settings as fig. 3; annotations report Pearson r and Spearman ρ. The BPB axis is reversed so rightward consistently means better.

H Decisive-Action BPB Method Detail

H.1 Universal Chat Template and Formatting Masks

Universal rendering. We serialize every trajectory with the same model-independent plain-text template rather than using each checkpoint's native chat template. Messages remain in their original chronological order, and the renderer uses the following visible labels:

Table 9: Universal serialization of trajectory components. Angle-bracketed text denotes the original message content.

Source component	Universal representation
Tool definitions	`Tool definitions: <JSON>`
System message	`System: <content>`
User message	`User: <content>`
Assistant text	`Assistant: <content>`
Assistant tool call	`Assistant (tool call): <JSON>`
Tool response	`Tool Result: <content>`

An assistant message may contain both explanatory text and one or more tool calls; in that case, both the `Assistant:` and `Assistant (tool call):` records are emitted. The template contains no model-specific control tokens such as ChatML markers or model-specific step delimiters: all labels above are ordinary text.

Canonical tool-call serialization. Tool definitions and tool calls are serialized with `json.dumps(value, ensure_ascii=False)`. This preserves source field insertion order, uses the standard spacing after commas and colons, and emits Unicode directly. If `function.arguments` is already a JSON string, it remains a string inside the outer tool-call object rather than being parsed into a nested object; reparsing would change the prediction target and make it artificially easier. Random tool-call IDs are removed before rendering because they carry no semantic information and are not predictably determined by the trajectory. All other tool-call fields are preserved.

Prefix and target construction. For an assistant step at position i, the prefix is the universal rendering of the tool definitions and messages strictly before i, while the target is the same rendering applied to message i alone. Tool definitions therefore appear in the prefix but are not repeated in

the target. Using the same serializer on both sides prevents a representation mismatch at the prefix–target boundary.

Formatting mask. The complete rendered prefix and decisive response are visible to the checkpoint as conditioning context, but the primary Decisive-Action BPB result accumulates loss only over assistant-generated semantic content: natural-language assistant text, tool function names, and tool arguments. It excludes fixed role labels; tool-call wrapper keys such as `type`, `function`, `name`, and `arguments`; JSON braces, brackets, quotes, commas, and colons belonging to that wrapper; and trailing record separators. Our tool-formatting-included ablation adds the serialized tool-call wrapper to the scored region, but still excludes the fixed role labels. The rendered input is otherwise identical, so the two settings differ only in which target bytes contribute to the BPB numerator and denominator.

I PATCH MCQ DETAIL

Table 10: Source-current four-option Patch MCQ accuracy with cyclic rotation and arithmetic aggregation; chance is 0.250. Both question sets use matched `none` preambles. Each cell is the one accuracy our source measurements record for that checkpoint and setting; run-to-run spread is not stored with these values (section C), so column-to-column gaps of a few thousandths should not be read as differences. Bold marks the largest value per column.

	DeepSWE		SWE-Verified	
Base checkpoint	No think	Think	No think	Think
Hy3-Preview-Base	**0.4456**	**0.5080**	0.552	0.583
DeepSeek-V4-Pro-Base	0.4206	0.5026	0.550	**0.591**
Nemotron-Ultra-Base	0.4235	0.4611	0.538	0.558
DeepSeek-V4-Flash-Base	0.4009	0.4421	0.536	0.553
Kimi-K2-Base	0.4322	0.4047	**0.597**	0.523
Qwen-3.5-35B-A3B-Base	0.3953	0.3883	0.521	0.524
Nemotron-Super-Base	0.3472	0.4116	0.500	0.498
GLM 4.5-Air-Base	0.3513	0.3556	0.425	0.432
Gemma-4-26B	0.3265	0.3483	0.443	0.440
Nemotron-Nano-Base	0.3172	0.2867	0.306	0.342
Mean	0.386	0.411	0.497	0.504

Table 11: Patch MCQ read ablation on both question sets; chance is 0.250. Argmin-BPB scores each option independently and picks the lowest byte-normalized negative log-likelihood; rotated log-probability is the answer-letter read, presenting all four options jointly under four cyclic rotations, and is reported here at think-1000 to match table 10. Gap is argmin-BPB minus rotated log-probability, so negative values favor the joint read. Rows are ordered by DeepSWE rotated log-probability.

	DeepSWE			SWE-Verified		
Base checkpoint	Argmin	Rotated	Gap	Argmin	Rotated	Gap
Hy3-Preview-Base	0.266	0.508	−0.242	0.359	0.583	−0.224
DeepSeek-V4-Pro-Base	0.283	0.503	−0.220	0.405	0.591	−0.186
Nemotron-Ultra-Base	0.267	0.461	−0.194	0.366	0.558	−0.192
DeepSeek-V4-Flash-Base	0.266	0.442	−0.176	0.380	0.553	−0.173
Kimi-K2-Base	0.279	0.405	−0.126	0.387	0.523	−0.136
Qwen-3.5-35B-A3B-Base	0.271	0.388	−0.117	0.325	0.524	−0.199
Nemotron-Super-Base	0.269	0.412	−0.143	0.366	0.498	−0.132
GLM 4.5-Air-Base	0.265	0.356	−0.091	0.384	0.432	−0.048
Gemma-4-26B	0.262	0.348	−0.086	0.354	0.440	−0.086
Nemotron-Nano-Base	0.262	0.287	−0.025	0.347	0.342	+0.005
Mean	0.269	0.411	−0.142	0.367	0.504	−0.137

Question-set construction detail. Table 10's DeepSWE question set is built by a max-flow selection over three source sets capped at two questions per instance. It contains 756 questions over 392

Table 12: Single-rotation versus full cyclic rotation on the DeepSWE question set, think-1000 arm, over the same ten checkpoints as table 10; chance is 0.250. k=1 takes the answer from the first rotation alone and k=4 is the shipped aggregate, both recomputed offline from the same finished runs, so no checkpoint was re-scored; each value is the mean of three rounds. The k=4 column is the DeepSWE think-1000 column of table 10. Rotation helps on 9 of 10 checkpoints, but not uniformly: the gain spans -0.005 to $+0.081$, wider than the gap between adjacent checkpoints in the k=4 ranking, and it reorders the top two. Bold marks the best value per column.

Base checkpoint	k=1	k=4	Gain
DeepSeek-V4-Pro-Base	**0.4555**	0.5026	+0.047
Hy3-Preview-Base	0.4271	**0.5080**	+0.081
Nemotron-Ultra-Base	0.4195	0.4611	+0.042
DeepSeek-V4-Flash-Base	0.3881	0.4421	+0.054
Kimi-K2-Base	0.3741	0.4047	+0.031
Nemotron-Super-Base	0.3727	0.4116	+0.039
Qwen-3.5-35B-A3B-Base	0.3454	0.3883	+0.043
GLM 4.5-Air-Base	0.3333	0.3556	+0.022
Gemma-4-26B	0.3126	0.3483	+0.036
Nemotron-Nano-Base	0.2913	0.2865	−0.005
Mean	0.3720	0.4109	+0.039

Table 13: Downstream agreement of every Patch MCQ scoring method on the same ten checkpoints, with both Pearson and Spearman correlations. Rows pair a question set with a scoring method; the SWE-Verified question set draws its options from the same benchmark as the first target, so those rows are same-benchmark evidence against it while all rows are cross-domain with respect to Terminal-Bench 2.1. At $n = 10$ with no interval estimates we do not read small differences as separating rows.

		vs. SWE-bench Verified		vs. Terminal-Bench 2.1	
Question set	Scoring method	Pearson	Spearman	Pearson	Spearman
DeepSWE	Argmin-BPB	0.601	0.665	0.420	0.382
DeepSWE	Rotated, no think	0.900	0.806	0.686	0.547
DeepSWE	Rotated, think 1000	0.915	0.867	0.871	0.796
SWE-Verified	Argmin-BPB	0.429	0.578	0.359	0.213
SWE-Verified	Rotated, no think	0.909	0.842	0.725	0.578
SWE-Verified	Rotated, think 1000	0.971	0.903	0.894	0.900

instances; the four gold-option length ranks and four answer slots each occur exactly 189 times, so the pick-longest baseline is exactly 0.250. The SWE-Verified question set merges the earlier pair pool with two additional single-action rollout waves, deduplicates by source, instance, target step, and trial, and retains 576 length-rebalanced questions over 249 instances.

Presentation and answer extraction. Each question q_i is presented under four cyclic option reorderings (rotations), indexed $r \in \{0, 1, 2, 3\}$ with $r = 0$ the original ordering. Each reordering is an independent request whose MCQ context $u_i^{(r)}$ contains the trajectory prefix and all four options, so the options are always compared within one shared context as in eq. (5). Let $\ell_r(o) \in \mathcal{L}_4$ denote the answer letter under which option o is displayed in reordering r; because the reorderings are cyclic, every option is displayed under every letter exactly once across the four requests. From each request we read the next-token probability of each answer letter, restricted to the top-20 candidate tokens the endpoint returns, and renormalize over the answer letters,

$$\tilde{p}_\theta\big(\ell \mid u_i^{(r)}\big) := \frac{p_\theta\big(\ell \mid u_i^{(r)}\big)}{\sum_{\ell' \in \mathcal{L}_4} p_\theta\big(\ell' \mid u_i^{(r)}\big)}, \qquad \ell \in \mathcal{L}_4, \tag{8}$$

assigning zero probability to any letter absent from the returned candidates. Because a given letter names a different option in each reordering, these probabilities are aggregated per option rather than per letter: each option's aggregate is the mean probability of the letter it was displayed under, taken over the four reorderings,

$$\bar{p}_i(o) := \frac{1}{4} \sum_{r=0}^{3} \tilde{p}_\theta\big(\ell_r(o) \mid u_i^{(r)}\big), \tag{9}$$

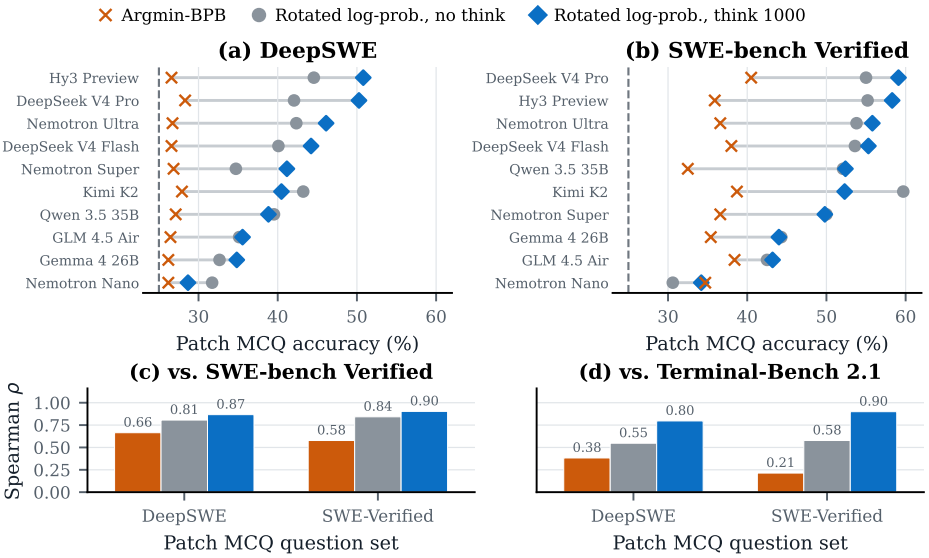


Figure 10: Patch MCQ accuracy and downstream agreement, by scoring rule. Panels (a) and (b) rank the same ten checkpoints on each question set; panels (c) and (d) plot the rank agreement tabulated in table 13 against both downstream targets. The joint answer-letter read separates the cohort more sharply than scoring each option independently, and the gap between the two reads is larger than the gap between thinking protocols.

and the prediction is the option with the largest aggregate, reported by its letter in the original ordering so that it is directly comparable with the golden option $\bar{\ell}_i$ in the accuracy of section 3.3:

$$\widehat{\ell}_i := \ell_0\Big(\arg\max_o \bar{p}_i(o)\Big), \tag{10}$$

with exact ties broken by position in the original ordering. With a single reordering this reduces to eq. (5). Averaging probabilities, rather than counting per-reordering votes, keeps a near-tie between two options from being rounded away before the four reads are combined, and it cancels the position prior because that prior falls equally on every option over a full cycle. With reasoning enabled, $u_i^{(r)}$ additionally contains the checkpoint's own reasoning of up to 1,000 tokens, and the letter probabilities are read at the answer position that follows it; reconstructing the answer from the distribution before the reasoning describes a different quantity and yields near-chance accuracy.

Only complete reorderings are scored. A reordering fails to read if none of the answer letters appears among the returned candidates, and a question counts toward accuracy only if all four of its reorderings read successfully; a question with any failed reordering is left unscored, since eq. (9) is defined only over a complete cycle. This is a correctness requirement rather than a cleanliness preference. Rotation removes the position prior by making it common to every option so that it cancels in the comparison, and that cancellation is exactly what an incomplete set of rotations destroys: a partially rotated item still carries the prior and therefore measures a different quantity than a fully rotated one. Admitting partial rows changes per-checkpoint results by amounts comparable to the effects we report, so every arm in this section is filtered to complete rotations.

Rotation count. Table 12 isolates how much the joint read owes to rotation itself. Taking the answer from the first rotation alone, rather than aggregating all four, costs 0.039 accuracy averaged over the cohort, and the cost is uneven: it ranges from -0.005 on Nemotron Nano, the one checkpoint rotation does not help, to $+0.081$ on Hy3 Preview. That spread is wider than the gap between adjacent checkpoints in the four-rotation ranking, and it is enough to reorder the top two: DeepSeek V4 Pro leads on a single read (0.4555 versus 0.4271) but trails on the aggregate (0.5026 versus 0.5080). Part of a checkpoint's standing under Patch MCQ therefore reflects how much the debiasing protocol recovers for it, not its per-read judgement alone, so the read protocol belongs with the score whenever Patch MCQ numbers are quoted.

Argmin-BPB scoring ablation. Table 11 compares the two scoring methods on both question sets, with the rotated-log-probability columns taken at think-1000 so they match table 10 exactly.

The joint read wins on all ten checkpoints on DeepSWE and on nine of ten on SWE-Verified, where Nano is the lone exception by 0.005. The two reads also rank the cohort differently, at Spearman $\rho = 0.543$ on DeepSWE and 0.322 on SWE-Verified, and argmin-BPB's agreement with post-trained SWE-bench Verified $\mathrm{pass@1}$ is correspondingly weaker ($\rho = 0.665$ and 0.578, versus 0.867 and 0.903 for the joint read; table 13 reports every scoring method against both downstream targets, and fig. 10 plots the same comparison), so the choice of scoring method changes probing conclusions and not just absolute levels.

Reasoning effects require matched protocols. Both question sets now provide format-matched `none`-preamble comparisons. On DeepSWE, the ten-checkpoint mean rises from 0.386 to 0.411 ($+0.025$): seven checkpoints improve, with changes ranging from $+0.082$ for DeepSeek V4 Pro to -0.030 for Nano. On SWE-Verified the mean rises only from 0.497 to 0.504 ($+0.008$), also with 7/10 improving, and Kimi drops by 0.074 – a single checkpoint accounting for most of the difference between the two means. Without run-to-run spread for these values we do not treat small per-checkpoint differences as measured effects, and no-think and think-1000 remain distinct protocols rather than one uniformly dominating the other.

J Prefix-Conditioned $\mathrm{pass@}K$ Detail

Fork point and masking convention. The decisive-step index is inclusive: masking at $T_i^\star$ means steps $1, \ldots, T_i^\star - 1$ are replayed verbatim from the recorded trajectory and the candidate must itself produce the action at $T_i^\star$. Replaying one step further and generating nothing reproduces the recorded resolving patch exactly, which we use as a control that the replay path is faithful before any candidate is scored.

Forward budget. Unlike a single-action evaluation, the candidate has to finish the task after the fork, so its phase is multi-turn and needs a stopping rule; every reported run caps it at three generated steps. When that budget is exhausted the proxy returns a synthetic response carrying no action, so the agent exits cleanly and the patch is still extracted, rather than the episode running to timeout and yielding nothing.

Driving the harness from a base checkpoint. A post-trained candidate is relayed: the agent's own request is forwarded to it unchanged. A base checkpoint cannot be driven that way, because it has no chat template and cannot reliably emit a structured action, so we bridge it onto the interface instead. The bridge intercepts each request the agent CLI makes, renders it into a role-labeled plain-text transcript using the same renderer as Decisive-Action BPB (section H.1), samples a continuation from the checkpoint's raw completion endpoint, and repackages the result as the response the agent expects, so the agent itself runs unmodified. Figure 11 shows this round trip and the prompt it produces. The prompt is assembled in a fixed order,

$$\text{transcript} \parallel \text{steer} \parallel \text{primer} \parallel \text{prefill},$$

and the order matters as much as the contents. The *steer* is a final `System:` line stating that the investigation is complete and asking for exactly one action that edits the existing source; given only a neutral primer, a base checkpoint tends to keep exploring or to write a reproduction script instead of a fix. The *primer* is the turn header `Assistant (tool call):`, which places the next token at the start of an action. Placing the steer after the primer instead of before it cut the share of forwards emitting a parseable action to 12%.

Prefilling the action envelope. The primer alone is not enough, and prefilling is what makes a base checkpoint usable in the harness at all. With only the primer, 47.9% of generations on one checkpoint continued it as prose rather than acting, and 56% of those fabricated a `Tool Result:` or `User:` turn: the checkpoint kept writing the transcript instead of acting in it. Adding the steer did not fix this, since 79% of the generations that still failed to parse simply echoed the steer back as another `System:` block. The fix is to append the opening of the action envelope after the primer, so that the checkpoint's first sampled token already lands inside an open action and a prose continuation becomes ungrammatical; the fragment is prepended back onto the sample before parsing, so the parser sees a complete action. On eight DeepSWE prefixes with 48 generations per arm, prefilling raised the share of forwards emitting a parseable action from 27% to 50%, raised the share that edit

the repository from 10% to 27%, and cut steer echoes from 19% to 8%. The envelope is wire-specific and must open exactly what the agent parses: for the function-calling agent on DeepSWE it is `[{"`, the start of the tool-call JSON array, and for the text-mode agent on SWE-bench Verified it is the opening of the fenced command block that agent reads. Stop sequences follow the wire in the same way: they are kept in text mode, where they are load-bearing, and disabled for the function-calling agent.

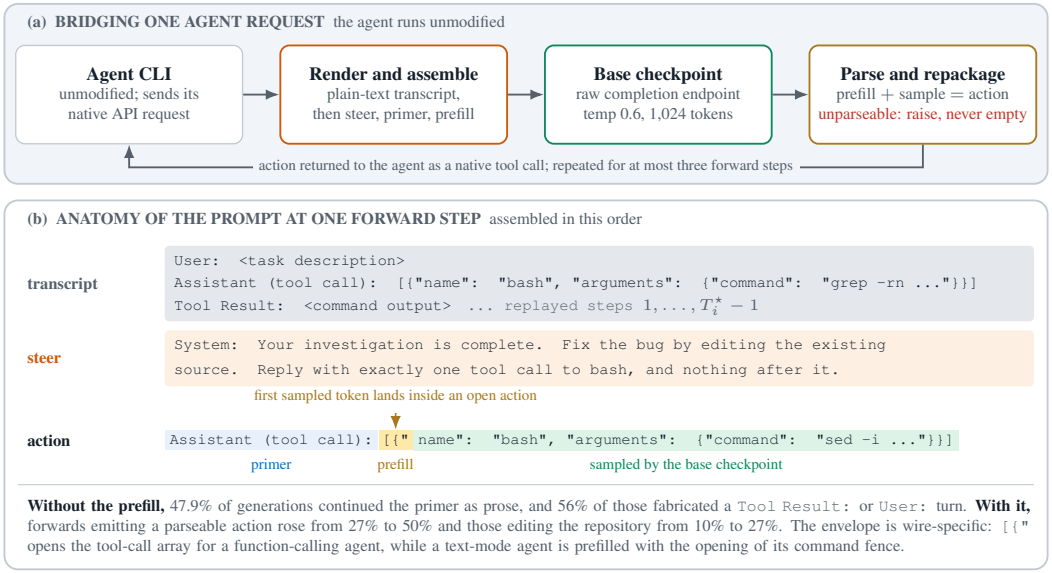


Figure 11: Driving an agent harness from a base checkpoint. (a) The bridge sits between an unmodified agent CLI and the base checkpoint: it renders each agent request into a plain-text transcript, samples a continuation from the checkpoint's raw completion endpoint, and returns the parsed action to the agent as a native tool call. (b) The prompt at one forward step, assembled as transcript, steer, primer, and prefill. The prefill pre-opens the action envelope so that the checkpoint's first sampled token already lands inside an open action; only the green span is written by the base checkpoint.

Reading the action back. The completion is parsed for the checkpoint's own action with no instruct model in the loop: nothing repairs malformed output, so what is scored is exactly what the base checkpoint wrote. A turn that cannot be parsed raises an error rather than returning an empty action, because the agent would read an empty action as the episode being finished, and the rollout would then end looking like a legitimate failed attempt rather than a formatting failure. Each forward is sampled at temperature 0.6 (section D) with a 1,024-token cap: at 512 tokens, 95% of generations hit the cap and were truncated mid-command, and raising it to 1,024 lifted the parse rate from 68% to 80%. Because prefilling forces the first token to open an action, it leaves the checkpoint no room to deliberate, so we also tested a two-step variant that samples a short plan before the action. It lowered the share of forwards emitting a parseable action from 56% to 38%, since the spliced plan gave the checkpoint more transcript to continue instead of acting; every reported run therefore samples each action directly, with no plan step, within the three-step forward budget.

Prefix caching. The k rollouts at one decisive step replay identical prefix steps, which on a 352-rollout measurement accounted for 48% of rollout wall time. Caching that work requires two artifacts, because a prefix is two kinds of state: a committed container image carries the filesystem the candidate edits, and an observation log carries the conversation it reads. Restoring only the image saves nothing, since a host-side agent rebuilds its message history by re-executing commands; restoring only the log is worse, because the candidate then edits a pristine tree while reading a history that says otherwise. On ten prefixes the cache reproduced byte-identical patches and cut prefix wall time by 96%, roughly 40% per arm over the full pool, at about 64 GB of images per 120 prefixes. Committing a container preserves the filesystem but drops running processes, so prefixes whose recorded commands leave a background service behind are detected by probing both containers and dropped from the cache, with the uncached path used as the fallback.

Scoring the rollouts. The k rollouts for one instance are scored in k separate harness runs, because the benchmark harness keys predictions by instance identifier and k patches for one instance would otherwise collide in a single predictions file. Generation is endpoint-bound while the harness is CPU- and disk-bound, so the two are pipelined: one rollout index is generated while the previous index is scored in the background. We validated the pipelined path as byte-identical to the serial one before using it for the reported runs.

Table 14: Prefix-conditioned pass@K on DeepSWE and SWE-bench Verified trajectory prefixes, in percent. DeepSWE uses 120 held-out instances; SWE-bench Verified uses a separately eligibility-filtered cohort of 98 prefixes. Both cover the same ten checkpoints, but the two halves aggregate over different cohort sizes, so their means are not matched pairs. Rollouts fork at the decisive step $T_i^\star$: steps before are served verbatim, $T_i^\star$ is masked, and the checkpoint keeps control for the rest of the episode. An instance counts as resolved if at least one of k continuations passes the verifier. Row order follows table 10; bold marks the largest value per column.

	DeepSWE			SWE-bench Verified		
Base checkpoint	$k=8$	$k=16$	$k=32$	$k=8$	$k=16$	$k=32$
`hy3-base`	16.67	20.83	26.67	61.22	70.41	76.53
`deepseek-v4-pro-base`	**37.50**	**41.67**	**45.83**	**68.37**	**77.55**	**83.67**
`ultra-base`	12.50	17.50	26.67	58.16	68.37	**83.67**
`dsflash-base`	28.33	33.33	40.00	63.27	72.45	79.59
`kimi-k2-base`	14.17	16.67	21.67	55.10	62.24	68.37
`qwen-base`	17.50	20.00	25.00	52.04	57.17	63.27
`super-base`	5.83	10.83	14.17	41.84	50.00	58.16
`glm-air-base`	8.33	9.17	10.83	34.69	42.86	51.02
`gemma-base`	4.17	6.67	10.83	31.63	39.80	47.96
`nano-base`	1.67	2.50	2.50	18.37	26.53	35.71
Mean	14.67	17.92	22.42	48.47	56.74	64.80

Table 15: Downstream validation of prefix-conditioned pass@K against both targets of table 6, on the same ten-checkpoint cohort used throughout the primary comparisons; higher is better agreement. SWE-bench Verified prefix rows are same-benchmark evidence against the SWE-bench Verified target, not cross-domain validation; both prefix corpora are cross-domain with respect to Terminal-Bench 2.1. The last two rows repeat the canonical all-step/decisive-step comparison from figs. 2 and 7, correlated against $-$ BPB on the same ten checkpoints. No value is marked best: at $n = 10$ with no bootstrap intervals, we do not read these differences as separating the rows.

		vs. SWE-bench Verified		vs. Terminal-Bench 2.1	
Probe	n	Pearson	Spearman	Pearson	Spearman
DeepSWE prefix pass@K, $k=8$	10	0.840	0.915	0.870	0.802
DeepSWE prefix pass@K, $k=16$	10	0.872	0.952	0.912	0.900
DeepSWE prefix pass@K, $k=32$	10	0.917	0.951	0.947	0.930
SWE-bench Verified prefix pass@K, $k=8$	10	0.990	0.988	0.902	0.875
SWE-bench Verified prefix pass@K, $k=16$	10	0.984	0.988	0.911	0.875
SWE-bench Verified prefix pass@K, $k=32$	10	0.949	0.906	0.886	0.820
Decisive-Action BPB, all steps (reference)	10	0.960	0.915	0.861	0.809
Decisive-Action BPB, decisive step only (reference)	10	0.965	0.964	0.863	0.833

Consistency on one previously mis-ranked pair, but not independent evidence. On held-out DeepSWE prefixes, prefix-conditioned pass@K ranks DeepSeek V4 Flash above Nemotron Ultra 550B at every k (40.00% versus 26.67% at $k = 32$), matching the correction made by Decisive-Action BPB. This is a consistency check rather than an independent correction, because the protocol already forks at the decisive step. The second BPB case, Qwen 3.5 35B versus Kimi K2, remains a disagreement: at $k = 32$ the sampled probe places Qwen above Kimi (25.00% versus 21.67%) even though Kimi scores 2.1 points higher downstream. The SWE-bench Verified arm uses prefixes from the same benchmark as the downstream target and is not held out in that sense.

Where the probes disagree. Figure 12 shows both arms against both targets. At $k = 32$, the held-out DeepSWE arm disagrees with downstream SWE-bench Verified pass@1 on two of 45

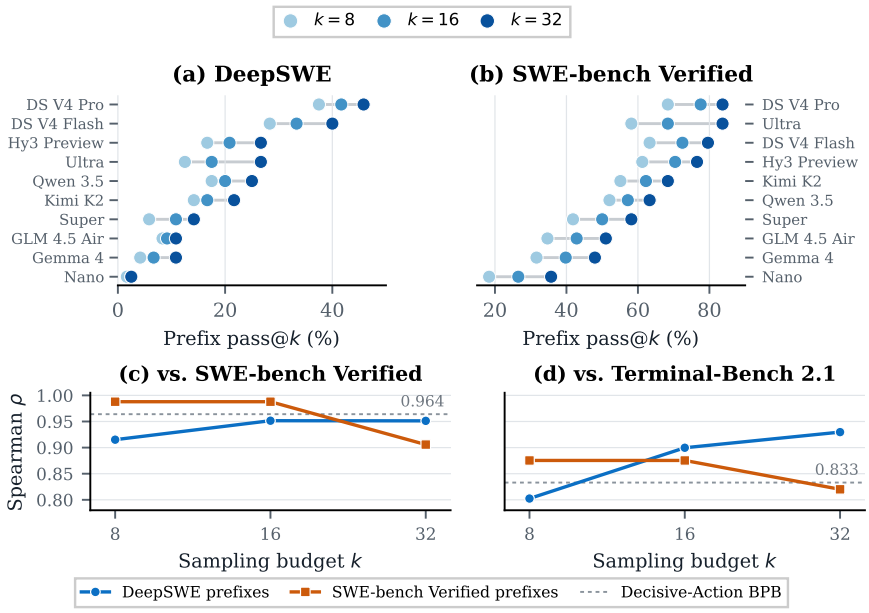


Figure 12: Prefix-conditioned pass@K rankings and downstream agreement. Panels (a) and (b) rank the same ten checkpoints under DeepSWE and SWE-bench Verified trajectory prefixes at $k \in \{8, 16, 32\}$, tabulated in table 14. Panels (c) and (d) plot the rank agreement of those scores with each downstream target from table 15; the dashed line marks Decisive-Action BPB against the same target for reference. The two panels are sorted independently and use different x ranges, so marker positions are not comparable across them.

checkpoint pairs, both involving Kimi K2, which the sampled probe places below Nemotron Ultra 550B (21.67% versus 26.67%) and below Qwen 3.5 35B (21.67% versus 25.00%) despite Kimi scoring higher downstream, by 0.6 and 2.1 SWE-bench Verified points respectively. The SWE-bench Verified prefix arm disagrees on three pairs, all involving Nemotron Ultra, which it ranks above DeepSeek V4 Flash (83.67% versus 79.59%), Kimi K2 (83.67% versus 68.37%), and Hy3 Preview (83.67% versus 76.53%) despite Nemotron scoring lower downstream in each case. Kimi also leads the SWE-bench Verified no-think Patch MCQ setting in table 10, so the cheap and sampled probes disagree about this checkpoint in opposite directions across narrow downstream gaps. This is precisely the case we reserve for direct investigation rather than for arbitration by either proxy.

K Additional SFT Correlation Results

Table 16 shows correlations between our probes and the post-SFT SWE-bench Verified scores of Nemotron 3 Nano, Nemotron 3 Super and Qwen 3.5 35B A3B, using different SWE trajectories as sources. The BPB rows use the same canonical per-checkpoint measurements as the main ten-checkpoint analysis. At $n = 3$, one pairwise inversion changes Spearman correlation by 0.5, so these coefficients are descriptive; extending the controlled SFT experiment to more base models is left to future work.

L Reproducibility Checklist

- **Corpus construction:** source records, success criteria, and step-extraction rules for SWE-Pro and DeepSWE.
- **Prefix/target rendering:** the universal template and all seven alternative renderers, including message normalization and reasoning removal.
- **Scoring regions:** payload, identifier, and combined-span mask definitions, including exact assistant-content, tool-content, tool-formatting, and reasoning boundaries.

Evaluation	vs. Post-SFT SWE-Verified ($n = 3$)		vs. Public SWE-Verified ($n = 3$)		vs. Public SWE-Verified ($n = 10$)	
	Pearson	Spearman	Pearson	Spearman	Pearson	Spearman
DeepSWE						
Decisive-Action BPB	0.97	0.50	0.96	0.50	0.97	0.96
Patch MCQ	0.91	0.50	0.90	0.50	0.87	0.80
Prefix pass@32	0.97	1.00	0.98	1.00	0.95	0.93
SWE-Pro						
Decisive-Action BPB	0.99	1.00	0.99	1.00	0.94	0.96
Patch MCQ	1.00	1.00	1.00	1.00	0.66	0.61
Prefix pass@32	1.00	1.00	1.00	1.00	0.85	0.83
SWE-bench Verified						
Decisive-Action BPB	0.99	1.00	0.99	1.00	0.95	0.92
Patch MCQ	0.99	1.00	0.99	1.00	0.89	0.90
Prefix pass@32	1.00	1.00	0.99	1.00	0.89	0.82

Table 16: Comparison of probe correlations across source tasks and post-SFT SWE-bench Verified results for Nemotron 3 Nano, Nemotron 3 Super, and Qwen 3.5 35B A3B. We show correlations against public SWE-bench Verified scores for the same 3 models, as well as all 10 models in our panel.

- **Step-candidate detection:** per-step parsed action type, raw trajectory step index, mutating-vs-read-only classification, the ordered candidate list C_i, individual skip reasons, and the parser/tool-rule version.
- **Replay and verifier decisions:** for each tested candidate step, the replay driver, cumulative patch content or hash, verifier command and environment, pass/fail outcome, and infrastructure-error status.
- **Step-discovery search:** search mode (bisect or `-scan`), the ordered sequence of candidates tested, the resulting $T_i^\star$ or explicit no-pair outcome.
- **Target construction provenance:** pair-builder identity and version, and fields distinguishing original trajectory steps from generated candidate continuations.
- **Scoring eligibility and drops:** dropped-row reasons and the common completed intersection used for each reported comparison.
- **Endpoint versions:** exact weights and serving builds for every named checkpoint in each cohort.
- **Bounded-code baselines:** exact per-model MBPP, HumanEval, LiveCodeBench, CRUXEval-I/O, and RepoBench XFirst measurements used in the bounded-code comparison, including missing-value and cohort handling.
- **Downstream vectors:** exact per-model SWE-bench Verified and post-trained $\mathrm{pass@1}$ mappings used in every correlation, with cohort joins and tie handling.
- **Patch MCQ set generation and scoring:** benchmark-instance and decisive-step eligibility, positive/negative construction rules, prompt template, rotation scoring code, and all model-free leakage baselines.