TASK-EFFICIENT AGENT EVALUATION USING CACHED TRACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Efficient benchmarking estimates a system's full benchmark performance from a small number of task executions. Standard approaches rely solely on the binary correctness of executions from a collection of already-evaluated reference systems. An agentic system, however, produces a rich trace for each execution that can reveal substantially more about its capabilities. Leveraging this information requires useful trace representations. We propose representing a trace by extracting a structured summary of it with an LLM and embedding the summary with a standard text encoder. These representations preserve correctness- and task-related signal while minimizing the presence of system identity-, model family-, and harness-related signals that dominate raw trace embeddings. Given this representation for each task execution from all reference systems, we construct low-dimensional representations of the systems, and predict a new system's score from its position among the reference systems. On SWE-bench Verified and Terminal-Bench 2.0, augmenting an item-response model with this geometry-based prediction reduces mean absolute error from 0.13 to 0.08 (Wilcoxon $p < 10^{-4}$) and from 0.14 to 0.12 (Wilcoxon $p < 10^{-3}$), respectively, when provided a single random executed task. Further, our approach dominates sample average and IRT-only methods for any given number of executed tasks for both datasets.

1 INTRODUCTION

Generative model use is increasingly agentic: systems plan, invoke tools, observe intermediate results, and iterate over long horizons to complete complex tasks (Yang et al., 2024; Wang et al., 2025). As agentic capabilities increase, so does the range of settings in which a given system might be deployed – and, with it, the number of benchmarks on which the system must be evaluated before it can be trusted with consequential work (Kapoor et al., 2026). Agentic evaluation, however, is expensive. Each task execution requires a properly sandboxed environment and a single task execution routinely consumes hundreds of thousands of tokens (Jimenez et al., 2024; Merrill et al., 2026). Full evaluation on a benchmark can regularly exceed $2,000 (Kapoor et al., 2026). Given this cost, efficient evaluation is necessary to improve the evaluation coverage of an agentic system.

Task-efficient evaluation methods address this problem by finding structure in item-level information available from previously evaluated reference systems to mitigate the cost of evaluating a new system. Some task-efficient evaluation methods require only the item-level scores from reference systems (Maia Polo et al., 2024; Vivek et al., 2024; Kipnis et al., 2025; Bowyer et al., 2026). Others leverage information in cached artifacts, such as natural language responses (Acharyya et al., 2024; Helm et al., 2025) or residual streams (Duderstadt et al., 2023), to measure the similarity of systems on a set of tasks. Agentic workflows often produce artifacts when executing a task – file diffs, tool calls, intermediate plans, and error messages – recorded together in an execution trace that can be used to this effect. Existing methods for using cached artifacts rely on off-the-shelf embedding functions that typically capture system or model idiosyncrasies (Sun et al., 2025) better than trace functionality. Along these lines, we make two primary contributions:

1. **`trubric` trace representations.** We introduce `trubric` (*trace rubric*): a trace representation constructed by extracting a task specific structured summary of a trace with an LLM and embedding the summary with a standard text encoder. We show that `trubric` embeddings pre-

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.

Table 1: Mean Absolute Error (MAE) of benchmark score estimators on SWE-bench Verified and Terminal-Bench 2.0 under leave-one-family-out evaluation and different task selection protocols. **Bolded** values are the best method for a given (benchmark, task selection, number of tasks). Underlined values are statistically significant ($\alpha = 0.05$) comparisons to IRT per the two-sided Wilcoxon signed-rank test. The `trubric` and IRT convex combination – which takes advantage of both trace geometry and per-trace score – is the best or near-best in all settings.

	SWE-bench Verified ($n = 107$)						**Terminal-Bench 2.0** ($n = 65$)					
	random			adaptive			random			adaptive		
Num. tasks m	1	5	20	1	5	20	1	5	20	1	5	20
population mean	0.139						0.155					
sample score	0.442	0.175	0.082	0.380	0.151	0.072	0.396	0.164	0.073	0.360	0.124	0.132
IRT	0.127	0.088	0.048	0.080	0.062	**0.034**	0.137	0.097	0.052	0.112	0.067	0.039
raw trace	0.099	0.093	0.089	0.096	0.083	0.081	0.135	0.126	0.120	0.128	0.125	0.113
generic	**0.079**	0.071	0.068	0.074	0.070	0.069	0.139	0.127	0.115	0.144	0.130	0.117
trubric	0.081	0.069	0.060	0.068	0.056	0.060	0.127	0.107	0.093	0.110	0.092	0.084
IRT + trubric	0.081	**0.063**	**0.043**	**0.062**	**0.048**	**0.034**	**0.122**	**0.090**	**0.050**	**0.099**	**0.059**	**0.037**

serve information relevant to the task and behavior of the system while minimizing system-, model family-, and harness- specific signal across seven modern text encoders.

2. **Task-efficient evaluation using cached traces.** Given the `trubric` embeddings from a set of tasks for a target system and a collection of reference systems, we evaluate their utility for predicting the target system's full benchmark score on SWE-bench Verified and Terminal-Bench 2.0 under three different agentic system evaluation protocols. We find that `trubric`-based estimators, combined with an item-response model, reduce single task mean absolute error from 0.13 to 0.08 ($p < 10^{-4}$) and from 0.14 to 0.12 ($p < 10^{-3}$) on the two benchmarks, respectively, and are nearly always better than baselines at every task budget (Table 1).

1.1 Problem statement

For our purposes, an agentic system $f \in \mathcal{F}$ is a random mapping from a task space $\mathcal{T}$ to a trace space $\mathcal{X}$. A benchmark is a fixed collection of tasks $T^* = \{t_1, \ldots, t_M\} \subset \mathcal{T}$. Each trace is evaluated by the benchmark's trace-level scoring function $s : \mathcal{X} \to \{0, 1\}$ and the system has a benchmark score $y(f) = \frac{1}{M} \sum_{j=1}^{M} \mathbb{E}\left[s(f(t_j))\right]$, where the expectation is with respect to the randomness of f. We write y for $y(f)$ when the context is clear.

When evaluating a new system, we typically have access to $\mathcal{D} = \left\{\{(x_{ij}, s(x_{ij}))\}_{j=1}^{M}\right\}_{i=1}^{n}$, the traces and trace-level scores of n previously-evaluated reference systems $f_1, \ldots, f_n$, where $x_{ij} \sim f_i(t_j)$. For the target system f_0, we observe traces (and, optionally, trace-level scores) on only a subset $T \subset T^*$ with $|T| = m \ll M$ tasks and our goal is to estimate the system's full benchmark score y_0 using only the target's m execution traces (and, optionally, their scores) and $\mathcal{D}$.

2 Related work

Representing agentic traces. The standard tools for embedding model artifacts are text encoders trained on sentence-level semantic similarity (Reimers & Gurevych, 2019; Su et al., 2023; Nussbaum & Duderstadt, 2025). Agentic traces, however, implicitly or explicitly encode system metadata that may not be relevant for downstream tasks (Sun et al., 2025; Oderinwale, 2026) and are therefore not necessarily encoded well by semantic embedding models. Instead, traces have primarily been studied through taxonomies of failure modes and actions (Cemri et al., 2025; Shu et al., 2026) and LLM judges (Zhuge et al., 2025; Mehtiyev & Assunção, 2026). Parallel work uses LLMs to adapt outputs to be more naturally compared via their semantics (Ravfogel et al., 2024; Peng et al., 2024; Benara et al., 2024). `trubric` applies this to agentic traces by embedding a task-specific structured summary of each trace.

Efficient benchmarking. Task-efficient benchmarking estimates a system's full benchmark score from a small number of executions. The dominant paradigm for efficient benchmarking centers around item response theory (IRT) (Birnbaum, 1968), where a target system's score is estimated

with $m \leq M$ item-level scores by assigning a latent ability based on the relationship between a task's difficulty and discriminability (Maia Polo et al., 2024; Kipnis et al., 2025). Extensions of IRT select informative or adaptive task subsets (Vivek et al., 2024; Truong et al., 2025). A complementary paradigm represents the systems themselves, either from their score profiles (Ruan et al., 2024; Zhuang et al., 2025) or from the artifacts they produce while executing a task (Duderstadt et al., 2023; Acharyya et al., 2024; Helm et al., 2025). For example, the Product Kernel Perspective Space (PKPS) (Anonymous, 2026) represents language models as Euclidean objects obtained via multidimensional scaling of the distances between their embedded responses. Helm et al. (2026a) regress benchmark scores on similar representations and prove that the estimate is task-efficient relative to the sample score. `trubric` enables extending geometry-based methods to agentic systems by providing an effective trace encoding. We show that these two approaches – score-based IRT and trace-based PKPS – can be combined to outperform either for any number of given executed tasks.

3 METHODS

We first present `trubric`. We then describe constructing representations of the systems $f_0, f_1, \ldots, f_n$ based on `trubric` embeddings that can be used to train a simple regressor in $\mathbb{R}^d$ for task-efficient evaluation. We evaluate the proposed methods in Section 4.

3.1 TASK-SPECIFIC RUBRICS FOR REPRESENTING TRACES

A trace representation for benchmark prediction should place two traces close together when they reflect similar execution behavior on a task. Concretely, we consider five attributes of a trace that a representation might encode: i) task (i.e., which task the trace corresponds to), ii) correctness (i.e., whether the trace produced a correct output), iii) system identity (i.e., which system produced the trace), iv) model family (i.e., the family of the underlying generative model), and v) harness (i.e., the harness used to produce the trace). For efficient benchmarking with traces, an ideal representation encodes (i) and (ii) but not (iii)–(v). `trubric` constructs such a representation in two stages.

First, for each task t_j, an LLM is prompted with the task description to instantiate a task-specific rubric. The rubric is a set of K fields $R_j = (r_j^{(1)}, \ldots, r_j^{(K)})$ where each field describes an aspect of

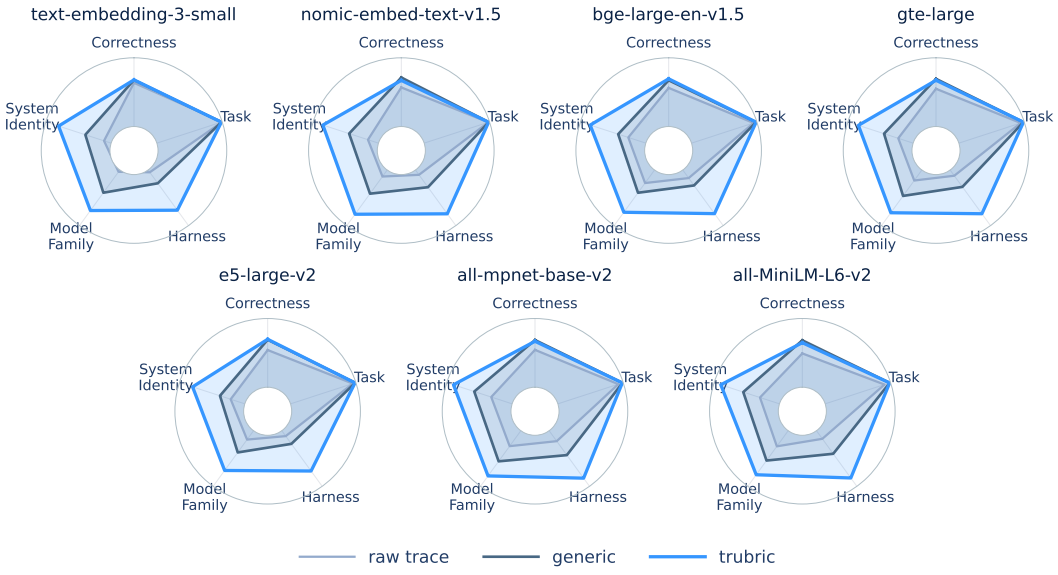


Figure 1: Three different methods for representing traces (embedding the raw trace, embedding a structured summary based on a generic rubric, and embedding a structured summary based on a task-specific rubric) evaluated on preserving / removing attributes (i)–(v). The representation based on `trubric` is better at preserving relevant signal ((i) Task and (ii) Correctness) while mitigating information about system idiosyncrasies ((iii) System Identity, (iv) Model Family, and (v) Harness).

execution behavior (e.g., how the relevant code was found). Rubrics are shared across all traces for a given task and fields are shared across tasks. An example rubric is provided in Appendix A.1.

Second, for each trace x_{ij} we use an LLM to produce a short summary $z_{ij}^{(k)}$ for each field. The summary of each field is then embedded separately with a semantic embedding model $g : \mathcal{X} \to \mathbb{R}^p$ and concatenated to yield the trace representation $\phi(x_{ij}) = (g(z_{ij}^{(1)}), \ldots, g(z_{ij}^{(K)})) \in \mathbb{R}^{K \cdot p}$. Two examples of an extracted summary for a given task is provided in Appendix A.1.

Figure 1 compares `trubric` to two baselines along axes (i)–(v). Values closer to the edge correspond to a a more ideal representation for efficient benchmarking for a given attribute. Experiment details are provided in Section 4.

3.2 Inference in the Product Kernel Perspective Space

Given trace representations, we represent agentic systems via the Product Kernel Perspective Space (PKPS). Before constructing the geometry of systems, trace representations are centered by subtracting the average across systems for each task and field to emphasize differences in behavior. Let τ denote a text embedding of the task description and $\kappa_\sigma(t, t') = \exp\left(-\|\tau(t) - \tau(t')\|^2/2\sigma^2\right)$. Let $\langle \phi, \phi' \rangle = \phi^T \phi'$ for $\phi, \phi \in \mathbb{R}^p$. For systems f_a and f_b evaluated on task sets T_a and T_b, define the weighted similarity

$$\langle f_a, f_b \rangle = \frac{\sum_{t \in T_a} \sum_{t' \in T_b} \kappa_\sigma(t, t') \, \langle \phi(x_{at}), \phi(x_{bt'}) \rangle}{\sum_{t \in T_a} \sum_{t' \in T_b} \kappa_\sigma(t, t')}, \tag{1}$$

the corresponding squared distance $\delta^2(f_a, f_b) = \langle f_a, f_a \rangle + \langle f_b, f_b \rangle - 2\langle f_a, f_b \rangle$, and $\Delta_{ab} = \delta^2(f_a, f_b)$. With Δ defined on $f_0, f_1, \ldots f_n$, applying classical multidimensional scaling (Torgerson, 1952) to Δ yields d-dimensional Euclidean representations $(\psi_0, \psi_1, \ldots, \psi_n) \in \mathbb{R}^{(n+1) \times d}$ of the systems with respect to the collective task sets. Importantly, Eq. (1) is defined when $|T_a| \neq |T_b|$ if both $|T_a| > 0$ and $|T_b| > 0$, so the target system is naturally represented in the same space as the reference systems. It is typical that $|T_0| = m \ll M$ and $|T_i| = M$ for all $i \in \{1, \ldots, n\}$.

Given $\mathcal{D}$ and $\psi_1, \ldots, \psi_n$, we train a simple regressor on $(\psi_1, y_1), \ldots, (\psi_n, y_n)$ and apply it to ψ_0 to obtain $\hat{y}_{geom}$, an estimate of f_0's full benchmark score y_0. Figure 2 shows the $d = 2$ perspectives when using the raw-trace embeddings (top) and the `trubric` embeddings (bottom) for different numbers of reference models and numbers of executed tasks per system. Each dot is a system colored by its full benchmark score on SWE-bench Verified. The `trubric`-based PKPS contains more information, per a lower mean absolute error on leave-one-system-out evaluation (LOSO; see Section 4). Finally, we note that $\hat{y}_{geom}$ does not use any item-level score information.

4 Experiments

We next introduce the datasets we use to evaluate the trace embeddings and the trace-based geometries for efficient evaluation, describe the trace embedding analysis and three different agent evaluation protocols, and describe baselines. We then evaluate `trubric`-based estimators against baselines across benchmarks, evaluation protocols, task budgets, and design choices.

Benchmark data. We consider two agentic benchmarks:

1. **SWE-bench Verified** (Jimenez et al., 2024; OpenAI, 2024) consists of 500 human-validated software-engineering tasks drawn from real GitHub issues. Its public leaderboard includes submissions from 119 unique systems. We consider the 107 systems with complete public trace archives and per-task outcomes. We randomly downsampled to $M = 100$ tasks. We let y_i be the official resolve rate over all 500 tasks. The full list of systems and tasks is included in Appendix C.
2. **Terminal-Bench 2.0** (Merrill et al., 2026) consists of 89 command-line tasks executed under a standardized harness specification. Its leaderboard contains submissions from 75 systems. Each system attempts each task five times. We analyze the 65 unique systems with at least 90% public trace coverage. We use a single trial's trace and binary outcome for a given (system, task) and let y_i be the mean resolve rate across all tasks and all trials. The full list of systems and tasks is included in Appendix C.

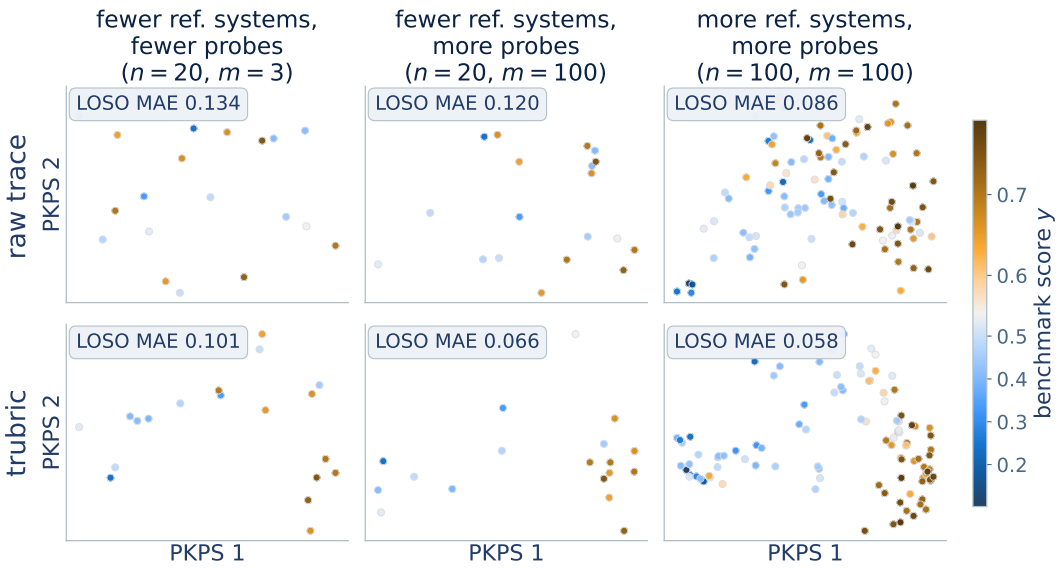


Figure 2: The 2-d Product Kernel Perspective Space (PKPS) induced with raw trace embeddings (top) and the `trubric` embeddings (bottom) under different number of reference systems and number of task executions. Each dot is a system colored by its full benchmark score on SWE-bench Verified. For a given setting, the manifold of systems induced by `trubric` embeddings is more correlated with full benchmark score as measured by MAE under the leave-one-system-out protocol.

Trace embedding evaluation. To quantify what a trace representation encodes, we train balanced linear classifiers to predict each of the five attributes of Section 3.1 for a held-out trace from its embedding. Classifiers for the system-level attributes (iii)–(v) are evaluated on held-out tasks. Classifiers for (i) and (ii) are evaluated on held-out groups of systems. We evaluate trace representations using the cached SWE-bench Verified traces and report balanced accuracy for each attribute. Higher accuracy is preferred for attributes (i) and (ii). Near chance accuracy is preferred for attributes (iii)–(v). Detailed descriptions of the classification tasks are provided in Appendix A.2.

Trace embedding baselines. We compare `trubric` embeddings to two baseline representations: *raw trace*, which uses g to embed the start and end of each trace, and *generic*, which follows the two-stage construction of Section 3.1 with fixed field descriptions in place of a task-specific rubric. Unless otherwise stated, both the generic and `trubric` representations use $K = 6$ fields, with DeepSeek-V4-Flash (DeepSeek-AI, 2026) as both the rubric generator and summarizer. For the embedding function g, we consider seven text encoders: `text-embedding-3-small` (Neelakantan et al., 2022), `nomic-embed-text-v1.5` (Nussbaum et al., 2025), `bge-large-en-v1.5` (Xiao et al., 2024), `gte-large` (Li et al., 2023), `e5-large-v2` (Wang et al., 2022), `all-mpnet-base-v2`, and `all-MiniLM-L6-v2` (Reimers & Gurevych, 2019). These encoders span open- and closed-source models, multiple architectures and training procedures, and embedding dimension sizes. The prompts and fixed field descriptions are provided in Appendix A.3.

Efficient benchmarking evaluation protocols. Modern leaderboards contain many entries that share an underlying model or harness. To emulate the process of evaluating a new model or harness, as well as a new combination of the two, we consider three different evaluation protocols: *leave-one-system-out* where only the target itself is excluded from the references, *leave-one-family-out* where references whose underlying model family matches the target's are additionally excluded, and *leave-one-harness-out* where references sharing the target's harness are excluded. Detailed descriptions of a system's assignment to a model family or harness are provided in Appendix C.

Efficient benchmarking baselines. We compare `trubric`-based predictions against five baselines: the *population mean* (the mean benchmark score of the reference systems), the *sample score* (the mean of the target's m observed trace-level scores), *raw trace* (the PKPS pipeline applied to the raw-trace embeddings), *generic* (the PKPS pipeline applied to the generic rubric embeddings), and

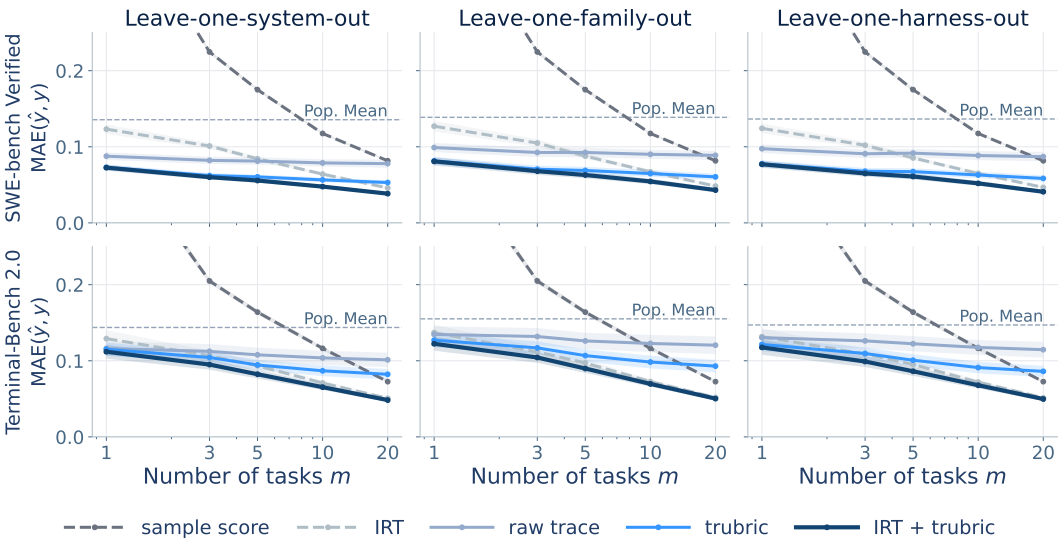


Figure 3: MAE of benchmark score estimators as a function of the number of executed tasks for SWE-bench Verified (top) and Terminal-Bench 2.0 (bottom) under three different realistic evaluation protocols: leave-one-system-out (left), leave-one-family-out (center), and leave-one-harness-out (right). Shaded areas represent ± 1 S.E. The `trubric` and IRT combination dominates (i.e., is never worse and is sometimes better) across all protocols, datasets, and number of executed tasks.

two-parameter *IRT*. We also consider the convex combination $\hat{y}_\alpha = \alpha\, \hat{y}_{\text{IRT}} + (1-\alpha)\, \hat{y}_{geom}$, with $\alpha \in [0, 1]$ selected via cross-validation. We refer to the convex combination as IRT + `trubric`. For `trubric`, raw trace, and generic we use ridge regression as the regressor, with the regularization chosen via cross validation. Implementation details for all estimators, including a technical description of IRT, are provided in Appendix B.

Task selection methods. For each task budget m, we consider two methods for selecting the target's m executed tasks. *Random* selection draws the m tasks uniformly. Reported results average over 50 randomly sampled task subsets. *Adaptive* selection chooses tasks greedily using the fitted item-response model. In particular at each step, the task with the highest expected Fisher information under the current ability posterior is executed and the posterior is updated (Appendix B).

4.1 Results

Rubric-based preprocessing produces more useful trace representations. Figure 1 shows that both generic and task-specific rubrics reduce recoverable information about system identity, model family, and harness while retaining task and correctness information. As discussed above, these attributes can introduce nuisance variation and obscure similarities in how systems execute tasks. Task-specific rubrics produce a larger reduction in identity-related signals compared to generic rubrics. These qualitative patterns persist across all seven embedding models.

The differences in attributes (i)–(v) are also reflected in the quality of the system-level geometry for predicting benchmark score. Figure 2 shows a stronger relationship between system position in the embedding space and benchmark score for `trubric` than for raw-trace embeddings. In Figure 3, leave-one-system-out prediction error decreases more rapidly with additional tasks when using `trubric` representations than when using raw-trace embeddings. On SWE-bench Verified under leave-one-family-out evaluation, increasing the task budget from 1 to 20 reduces geometry-only MAE from 0.081 to 0.060 for `trubric`, compared with 0.099 to 0.089 for raw traces. Table 1 further shows that generic rubrics are competitive on SWE-bench at small budgets, though task-specific rubrics provide larger improvements on Terminal-Bench 2.0. For the remainder of the paper "trace geometry" refers to a `trubric`-based geometry.

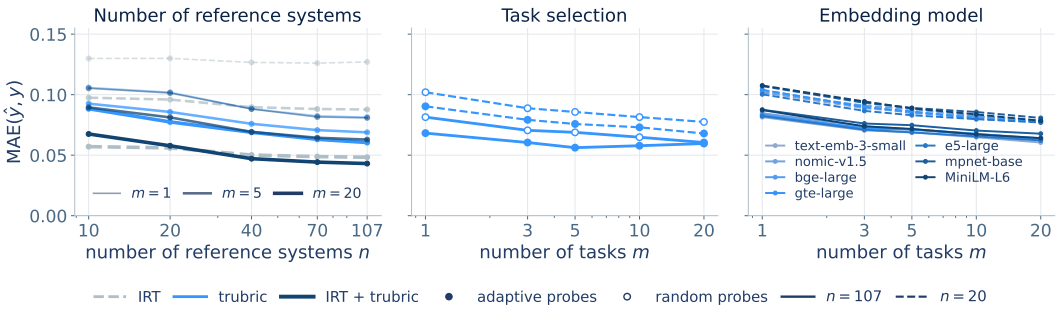


Figure 4: MAE of benchmark score estimators with access to different number of reference systems (left), under different task selection procedures (center), and using different text encoders (right) under the leave-one-family-out evaluation protocol on SWE-bench Verified. The embedding model analysis (right) only investigates the effect of the embedding model on $\hat{y}_{geom}$.

Trace geometry supports task-efficient benchmark prediction. Table 1 and Figure 3 show that `trubric`-based geometry predicts full benchmark performance more accurately than the sample score for small task budgets ($m \leq 10$) and the population mean for all budgets. The `trubric`-based geometry can also outperform IRT, which exploits the correctness patterns of reference systems, at small task budgets. On SWE-bench, the geometry produces a more accurate estimate from 1 through 10 random tasks (Figure 3). On Terminal-Bench, $\hat{y}_{geom}$ outperforms IRT at one task, but $\hat{y}_{IRT}$ is more accurate from three tasks onward. We emphasize that the geometric methods *only* require the target's trace content, and *not* its correctness labels, which means that these methods are applicable in settings where grading target tasks is infeasible (e.g., due to cost, lack of expertise, etc.), provided scored reference systems are available. $\hat{y}_{geom}$ and can be interpreted as a high bias, low variance estimate of the full benchmark score.

Combining trace geometry with IRT is best. The trace-based predictions and correctness-based predictions exhibit different behavior as the task budget grows: trace-based predictions are informative with very few executions, while IRT improves substantially as the number of tasks increases. Figure 3 shows that combining their predictions improves prediction across all evaluated budgets and benchmarks. At $m = 5$ on SWE-bench, IRT + `trubric` achieves 0.063 MAE, compared with 0.069 for geometry alone (Wilcoxon's $p < 10^{-4}$) and 0.088 for IRT (Wilcoxon's $p < 10^{-12}$). On Terminal-Bench – where IRT is relatively stronger – the combination achieves 0.090, compared with 0.107 for geometry (Wilcoxon's $p < 10^{-3}$) and 0.097 for IRT (Wilcoxon's $p < 10^{-2}$). These results suggest a complementary predictive value of execution artifacts and correctness outcomes. The broad pattern persists across the system-, family-, and harness-exclusion protocols in Figure 3, although the size of the improvement varies.

Performance improves with larger reference sets. The left panel of Figure 4 shows that increasing the number of reference models generally improves both IRT and geometry-based estimators. At $m = 5$ random tasks on SWE-bench, expanding from $n = 20$ to $n = 107$ reduces geometry-only MAE from 0.086 to 0.069 and IRT + `trubric` MAE from 0.081 to 0.063. The contribution of geometry depends on reference availability: given a small number of reference models and a large task budget, IRT can outperform geometry-only methods. The medium budget ($m = 20$) and small set of reference model ($n = 20$) regime is the setting where IRT outperforms the convex combination.

Adaptive task selection composes with trace-based estimation. Adaptive task selection provides a further improvement. With $m = 5$ adaptive tasks, IRT + `trubric` achieves 0.048 MAE on SWE-bench, compared with 0.063 for five random tasks (Table 1). Together, Table 1 and the center panel of Figure 4 show that adaptive selection improves both IRT and geometry-based prediction and can further improve evaluation efficiency.

Prediction is insensitive to the choice of text encoder. We evaluate the sensitivity of `trubric`-based prediction on SWE-bench to the choice of text encoder in the right panel of Figure 4. The effect is small (at most 0.008 MAE at any fixed setting) relative to the effect of the number of target

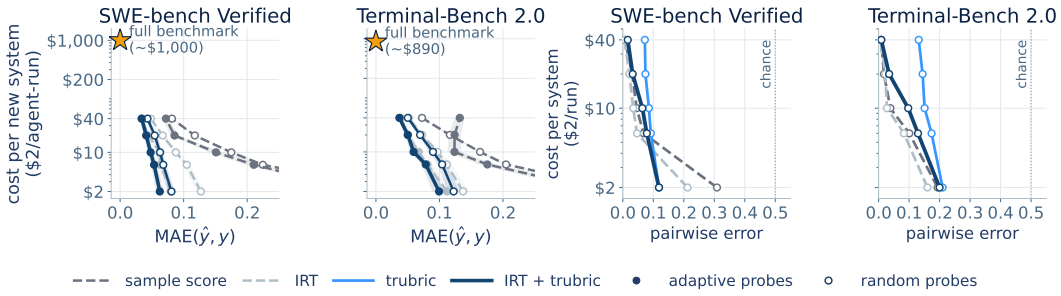


Figure 5: The cost of achieving a particular MAE (left and center left) and pairwise ranking error (center right and right) when using different benchmark score estimators on SWE-bench Verified and Terminal-Bench 2.0. Stars indicate the cost of producing the true benchmark score. An ideal method has low error (to the left) and is cheap (lower). Costs are calculated at a nominal $2 per task execution. Values are estimated using the leave-one-family-out results underlying Table 1 and, for pairwise error, leave-two-out comparisons of system pairs whose true scores differ by at least 0.05.

tasks or task selection (0.081 to 0.060 from 1 to 20 random tasks; 0.063 to 0.048 for random versus adaptive selection at $m = 5$) and the number of reference models (0.086 to 0.069 at $m = 5$).

The trace geometry reduces the cost of reaching a given prediction error. Figure 5 translates the error curves into the cost of evaluating a system. For this comparison, we assume a cost of $2 per agent execution (Jimenez et al., 2024; Merrill et al., 2026; Kapoor et al., 2026) and assume both the cost of using LLMs to do (t)rubric creation and summarization and the cost of embedding the traces are negligible. On SWE-bench, IRT + `trubric` reaches an MAE below 0.10 with one random task, compared with five tasks for IRT and twenty for the sample score. This corresponds to estimated target execution costs of $2, $10, and $40, respectively.

We also evaluate methods for comparing two systems head to head. For each pair of systems whose true benchmark scores differ by at least 0.05, we exclude both systems from the references and ask whether an estimator correctly identifies the better system from m random executions per system (right two panels of Figure 5). On SWE-bench, IRT + `trubric` decides 88% of pairs correctly from a single task per system, compared with 79% for IRT and 69% for the sample score. By five tasks per system, all estimators exceed 90% and the score-based estimators are competitive or better. On Terminal-Bench, IRT is the most accurate estimate at every tested budget. Trace information is thus most valuable for head-to-head comparisons in the lowest budget regime, where executing a single task per system ($4 for the pair) identifies the better system in nearly nine of ten pairs.

Larger rubrics improve average performance. Figure 6 varies the number of fields included in generic and task-specific rubrics, averaging performance over subsets drawn from a bank of 32 fields. For both constructions, adding fields generally improves average prediction accuracy. At $m = 5$ and a large set of reference systems, increasing a task-specific rubric from one to six fields reduces average MAE from 0.090 to 0.081 (Wilcoxon's $p < 10^{-5}$). Using all 32 fields reduces it further to 0.077 (Wilcoxon's $p < 10^{-3}$). Task-specific rubrics generally outperform generic rubrics, especially with fewer reference systems and larger number of executed tasks.

Figure 6 also reports the best performing rubric. Each point selects the lowest average error among the tested subsets separately for that setting. These *post hoc* selections demonstrate the existence of favorable small rubrics, which may be preferred in settings embeddings of the traces are stored for long periods. Importantly, the selected subsets need not be nested as field count increases, so their performance need not improve monotonically. We did not attempt to optimize the fields. The full list of the 32 field names is provided in Appendix A.4.

5 Discussion

We introduced `trubric`, a trace representation constructed by extracting a task-specific structured summary of a trace with an LLM and embedding the summary with a standard text encoder. Across seven text encoders, `trubric` embeddings preserve information about the task and the correctness

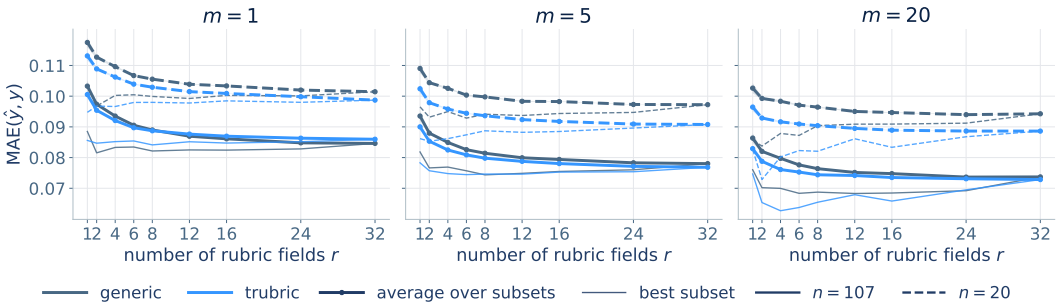


Figure 6: The effect of the number of fields present in the rubric used to extract a structured summary of the trace on MAE for different number of executed tasks and different number of reference systems for both task-specific and generic rubrics. We report both the MAE for the average (bold) and best performing (thin) sets of fields. In general, a task-specific rubric is preferred to a generic rubric when there are fewer reference systems and more executed tasks available.

of an execution while minimizing idiosyncratic information about the system, model family, and harness that produced it.

We then proposed using `trubric` embeddings to induce representations of agentic systems for task-efficient evaluation. The resulting geometry supports accurate benchmark score prediction from as little as a single executed task and, combined with an item-response model, is never worse than either baseline at any task budget.

The combined estimator reduces the cost of reaching an MAE below 0.10 on SWE-bench Verified from $10–$40 to $2 per system. These results are relatively robust to the major design choices: the text encoder, the number of rubric fields, and the task selection method. Importantly, these results support the claim that extracting more than binary correctness from cached executions can improve the efficiency of agentic evaluation as the number of systems and benchmarks continues to grow – providing a route toward improved evaluation coverage for a given system.

5.1 Limitations

We evaluated `trubric` along the five attributes relevant to efficient evaluation. We make no claims about the usefulness of the representation for other tasks one might consider with traces, such as change detection (Bridgeford & Helm, 2025), online monitoring (Helm et al., 2026b), early stopping (Ruan et al., 2026; Wang et al., 2026), or steering (Rahn et al., 2024). We leave the study of `trubric`-like representations for these tasks to future work.

As mentioned above, we did not attempt to optimize the rubric. The rubric size experiment of Section 4.1 demonstrates that favorable small rubrics exist. Developing validated procedures for selecting or learning rubric fields, such as parameterizing meta-instructions, is an open problem that we expect to have a small to medium effect on accuracy. We also note that `trubric`'s fields can be reconfigured to amplify or dampen arbitrary dimensions of variation in the input data.

In our analysis we only consider DeepSeek-V4-Flash (accessed via OpenRouter) as the rubric generator and trace summarizer. We had intended to add other frontier models as part of our ablation study, but a provider's API service banned our account and accused us of attempting to distill reasoning traces. We include this note as a warning: summarizing traces at the scale required for trace-based evaluation can resemble distillation to automated enforcement systems and may result in account bans. In any case, we leave the sensitivity analysis of the two roles to future work.

Finally, we only consider highly curated, paired settings where every reference system has executed (nearly) every task of a single benchmark. We do not consider references with sparse or disjoint task coverage, nor signal shared across benchmarks. The PKPS formulation we use throughout supports both – Eq. (1) is defined for unpaired task sets – and we expect less curated, larger reference collections to further reduce evaluation cost.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods and simple data processing. We also used ai in the creation of artifacts (e.g., prompting a coding agent to use Matplotlib to create figures), and draft / edit parts of the manuscript with heavy user guidance. We have reviewed all AI-assisted work. For example, generated code was verified and tested for correctness by all authors and generated text was heavily edited. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The main text and appendix provide enough information to reproduce all artifacts and results.

REFERENCES

Aranyak Acharyya, Michael W. Trosset, Carey E. Priebe, and Hayden S. Helm. Consistent estimation of generative model representations in the data kernel perspective space. *arXiv preprint arXiv:2409.17308*, 2024. doi: 10.48550/arXiv.2409.17308.

Anonymous. Benchmark prediction with sparse and unpaired cached responses, 2026. Under review; citation anonymized for double-blind review.

Vinamra Benara, Chandan Singh, John X. Morris, Richard J. Antonello, Ion Stoica, Alexander G. Huth, and Jianfeng Gao. Crafting interpretable embeddings for language neuroscience by asking LLMs questions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.16714.

Allan Birnbaum. Some latent trait models and their use in inferring an examinee's ability. In Frederic M. Lord and Melvin R. Novick (eds.), *Statistical Theories of Mental Test Scores*, pp. 397–479. Addison-Wesley, 1968.

Sam Bowyer, Acyr Locatelli, and Kris Cao. Efficient benchmarking is just feature selection and multiple regression. *arXiv preprint arXiv:2605.25773*, 2026. doi: 10.48550/arXiv.2605.25773.

Eric Bridgeford and Hayden Helm. Detecting perspective shifts in multi-agent systems. *arXiv preprint arXiv:2512.05013*, 2025. doi: 10.48550/arXiv.2512.05013.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2503.13657.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026. doi: 10.48550/arXiv.2606.19348.

Brandon Duderstadt, Hayden S. Helm, and Carey E. Priebe. Comparing foundation models using data kernels. *arXiv preprint arXiv:2305.05126*, 2023. doi: 10.48550/arXiv.2305.05126.

Hayden Helm, Aranyak Acharyya, Youngser Park, Brandon Duderstadt, and Carey Priebe. Statistical inference on black-box generative models in the data kernel perspective space. In *Findings of the Association for Computational Linguistics (ACL)*, pp. 3955–3970, 2025. doi: 10.18653/v1/2025.findings-acl.204. arXiv:2410.01106.

Hayden Helm, Ben Johnson, and Carey Priebe. Query-efficient model evaluation using cached responses. In *International Conference on Machine Learning (ICML)*, 2026a. arXiv:2605.07096.

Hayden Helm, Carey Priebe, and Brandon Duderstadt. Control charts for multi-agent systems. *arXiv preprint arXiv:2605.11135*, 2026b. doi: 10.48550/arXiv.2605.11135.

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 (ICLR)*, 2024. arXiv:2310.06770.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S. Siegel, et al. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.11977.

Alex Kipnis, Konstantinos Voudouris, Luca M. Schulze Buschoff, and Eric Schulz. metabench – a sparse benchmark of reasoning and knowledge in large language models. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2407.12844.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023. doi: 10.48550/arXiv.2308.03281.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinyBenchmarks: Evaluating LLMs with fewer examples. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2402.14992.

Tural Mehtiyev and Wesley Assunção. Beyond resolution rates: Behavioral drivers of coding agent success and failure. *arXiv preprint arXiv:2604.02547*, 2026. doi: 10.48550/arXiv.2604.02547.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026. doi: 10.48550/arXiv.2601.11868.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022. doi: 10.48550/arXiv.2201.10005.

Zach Nussbaum and Brandon Duderstadt. Training sparse mixture of experts text embedding models. *arXiv preprint arXiv:2502.07972*, 2025. doi: 10.48550/arXiv.2502.07972.

Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic Embed: Training a reproducible long context text embedder. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=IPmzyQSiQE.

Hamidah Oderinwale. Agent trajectories as programs: Fingerprinting and programming coding-agent behavior. *arXiv preprint arXiv:2606.16988*, 2026. doi: 10.48550/arXiv.2606.16988.

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

Letian Peng, Yuwei Zhang, Zilong Wang, Jayanth Srinivasa, Gaowen Liu, Zihan Wang, and Jingbo Shang. Answer is all you need: Instruction-following text embedding via answering the question. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 459–477, 2024. doi: 10.18653/v1/2024.acl-long.27. arXiv:2402.09642.

Nate Rahn, Pierluca D'Oro, and Marc G. Bellemare. Controlling large language model agents with entropic activation steering. *arXiv preprint arXiv:2406.00244*, 2024. doi: 10.48550/arXiv.2406.00244.

Shauli Ravfogel, Valentina Pyatkin, Amir DN Cohen, Avshalom Manevich, and Yoav Goldberg. Description-based text similarity. In *Conference on Language Modeling (COLM)*, 2024. arXiv:2305.12517.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Conference on Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019. doi: 10.18653/v1/D19-1410. arXiv:1908.10084.

Kai Ruan, Zihe Huang, Ziqi Zhou, Qianshan Wei, Jinghao Lin, Xuan Wang, and Hao Sun. Doomed from the start: Early abort of LLM agent episodes via a recall-controlled probe cascade. *arXiv preprint arXiv:2607.06503*, 2026. doi: 10.48550/arXiv.2607.06503.

Yangjun Ruan, Chris J. Maddison, and Tatsunori Hashimoto. Observational scaling laws and the predictability of language model performance. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.10938.

Rui Shu, Chun Yong Chong, Xin Zhou, Yun Peng, Zihan Wu, Xu Han, Zeyang Zhuang, Guowen Yuan, and Yuan Wang. What resolve rate hides: Trajectory structure diagnostics for coding agents. *arXiv preprint arXiv:2607.06184*, 2026. doi: 10.48550/arXiv.2607.06184.

Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. One embedder, any task: Instruction-finetuned text embeddings. In *Findings of the Association for Computational Linguistics (ACL)*, pp. 1102–1121, 2023. doi: 10.18653/v1/2023.findings-acl.71. arXiv:2212.09741.

Mingjie Sun, Yida Yin, Zhiqiu Xu, J. Zico Kolter, and Zhuang Liu. Idiosyncrasies in large language models. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2502.12150.

Warren S Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952. doi: 10.1007/BF02288916.

Sang T. Truong, Yuheng Tu, Percy Liang, Bo Li, and Sanmi Koyejo. Reliable and efficient amortized model-based evaluation. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2503.13335.

Rajan Vivek, Kawin Ethayarajh, Diyi Yang, and Douwe Kiela. Anchor points: Benchmarking models with much fewer examples. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2024. doi: 10.18653/v1/2024.eacl-long.95. arXiv:2309.08638.

Chenyu Wang, Yunbo Lyu, Junda He, Zhou Yang, Chenxing Zhong, Yaniv Harel, and David Lo. Fail-fast, restart-smart: Early failure prediction and restart for SWE agentic tasks. *arXiv preprint arXiv:2608.03222*, 2026. doi: 10.48550/arXiv.2608.03222.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022. doi: 10.48550/arXiv.2212.03533.

Xingyao Wang et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2407.16741.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed resources for general Chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 641–649, 2024. doi: 10.1145/3626772.3657878.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.15793.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. EmbedLLM: Learning compact representations of large language models. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.02223.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2410.10934.

A Trace representation details

A.1 Example task, trace, rubrics, and extractions

We illustrate the `trubric` pipeline on SWE-bench Verified instance `django__django-11099`.

Task. The problem statement, as provided to the rubric-instantiation LLM:

> UsernameValidator allows trailing newline in usernames
> Description
>
> ASCIIUsernameValidator and UnicodeUsernameValidator use the regex
> r'^[\w.@+-]+$'
> The intent is to only allow alphanumeric characters as well as ., @, +, and -. However, a little known quirk of Python regexes is that $ will also match a trailing newline. Therefore, the user name validators will accept usernames which end with a newline. You can avoid this behavior by instead using \A and \Z to terminate regexes. For example, the validator regex could be changed to
> r'\A[\w.@+-]+\Z'
> in order to reject usernames that end with a newline.
> I am not sure how to officially post a patch, but the required change is trivial - using the regex above in the two validators in contrib.auth.validators.

Instantiated `trubric`. The six task-specific field descriptions generated for this instance:

understanding Look for the agent to identify that the issue lies in the regex anchoring with $ in ASCIIUsernameValidator and UnicodeUsernameValidator, specifically that $ matches a trailing newline in Python, and that changing to \A and \Z is the proposed fix in django/contrib/auth/validators.py.

localization Check that the agent correctly locates the two validator classes (ASCIIUsernameValidator and UnicodeUsernameValidator) in django/contrib/auth/validators.py and references the existing regex r'^[\w.@+-]+$' in both.

reproduction Look for the agent to write or run a test that confirms a username ending with a newline (e.g., 'validuser\n') is currently accepted by the validators, possibly by using Django's validate_username or directly instantiating the validators.

editing Verify that the agent modifies the regex pattern in both validator classes from r'^[\w.@+-]+$' to r'\A[\w.@+-]+\Z', ensuring no other changes are made to the file.

verification The agent should run the relevant test suite (e.g., tests/auth_tests/test_validators.py or full Django auth tests) to confirm that usernames with trailing newlines are now rejected, and that existing valid usernames still pass.

final state The final state should show django/contrib/auth/validators.py with both regex patterns updated, and no other unintended modifications, with tests confirming the newline rejection behavior.

Generic rubric. For contrast, the generic baseline uses the same six fields with fixed, task-agnostic descriptions shared by every task:

understanding how the agent oriented itself before acting

localization how it searched/navigated to find the relevant code

reproduction whether/how it reproduced the issue before fixing

editing what it changed (files, nature and size of the edit)

verification whether it re-ran tests or a repro script after editing, and the OBSERVED outcome (passing/failing/errors)

final state how the run ended (clean finish, step/cost limit, submitted unverified)

Trace. An excerpt of System A's rendered trace on this task, as provided to the summarization LLM. The full rendered trace is 98,806 characters; its beginning and end are shown.

```
swe_main
First, we should locate the files that contain the `ASCIIUsernameValidator` and
`UnicodeUsernameValidator` classes to understand where the regex patterns are
defined. Since the issue mentions that these validators are in
`contrib.auth.validators`, we can start by searching for files named
`validators.py` within the `contrib/auth` directory.


action: find_file "validators.py" contrib/auth

Directory contrib/auth not found

First, we should locate the files that contain the `ASCIIUsernameValidator` and
`UnicodeUsernameValidator` classes to understand where the regex patterns are
```

```
defined. [...]

```
find_file "validators.py" contrib/auth
```
{"open_file": "n/a", "working_dir": "/django__django"}

[... roughly 97,000 characters omitted ...]

action: python regex_test.py

primary
role: user
ASCIIUsernameValidator rejected the username.
UnicodeUsernameValidator rejected the username.

(Open file: /django__django/regex_test.py)
(Current directory: /django__django)
bash-$
primary
role: assistant
The output indicates that both the `ASCIIUsernameValidator` and
`UnicodeUsernameValidator` regex patterns correctly rejected the username with
a trailing newline. This confirms that the changes made to the regex patterns
are effective.

Since the issue has been addressed and the fix has been verified with a test
script, we can now submit our changes to the code base.

```
submit
```
action: submit
```

Extractions. Structured summaries of two systems' traces on this task, produced against the rubric above. The summaries describe execution behavior without reference to the harness or underlying model that produced the trace.

SYSTEM A.

understanding The agent identified that the issue stems from Python's regex behavior where $ matches a trailing newline, and that changing the anchors to \A and \Z in both validators would fix it. It explicitly mentioned this during the initial inspection of the validator file.

localization The agent searched for validators.py, found django/contrib/auth/validators.py, opened it, and located both ASCIIUsernameValidator and UnicodeUsernameValidator classes. It acknowledged the existing regex pattern r'^[\w.@+-]+$' in both classes before making changes.

reproduction The agent attempted to reproduce the bug by creating test_validators.py and simple_test_validators.py, but these scripts failed due to missing Django settings configuration. It then created regex_test.py to directly test the regex patterns, which ran successfully and confirmed rejection of a trailing newline after the fix, though it did not demonstrate acceptance before the fix.

editing The agent edited lines 10 and 20 in django/contrib/auth/validators.py, changing the regex from r'^[\w.@+-]+$' to r'\A[\w.@+-]+\Z' in both ASCIIUsernameValidator and UnicodeUsernameValidator. The diff shows only these two regex changes to the file, with no other modifications.

verification The agent ran the custom regex_test.py script, which output that both ASCII and Unicode validators rejected a username ending with a newline. It did not run the existing Django test suite (e.g., tests/auth_tests/test_validators.py) to verify the fix against formal tests.

final state The final state includes django/contrib/auth/validators.py with both regex patterns updated to r'\A[\w.@+-]+\Z'. Three new files (test_validators.py, simple_test_validators.py, regex_test.py) were created, but no other existing files were modified. The agent submitted these changes.

SYSTEM B.

understanding The agent identified that the issue stems from Python's regex behavior where $ matches a trailing newline, causing ASCIIUsernameValidator and UnicodeUsernameValidator to incorrectly accept usernames ending with newlines. It understood that replacing ^ and $ with \A and \Z would fix the anchoring issue, as described in the issue for django/contrib/auth/validators.py.

localization The agent located the two validator classes, ASCIIUsernameValidator and UnicodeUsernameValidator, in django/contrib/auth/validators.py. It confirmed both classes use the existing regex pattern r'^[\w.@+-]+$' and identified these as the exact locations requiring modification.

Table 2: Held-out balanced accuracy of linear probes for the five trace attributes of Section 3.1, for each text encoder and trace representation; these values underlie Figure 1. For evaluation-relevant attributes (i)–(ii), higher is better; for system-metadata attributes (iii)–(v), lower is better. The final row reports chance balanced accuracy.

encoder	representation	(i) task	(ii) correctness	(iii) system	(iv) family	(v) harness
`text-embedding-3-small`	raw trace	1.00	0.82	0.89	0.97	0.96
	generic	0.99	0.84	0.61	0.64	0.76
	`trubric`	0.99	0.84	0.20	0.36	0.29
`nomic-embed-text-v1.5`	raw trace	0.98	0.78	0.84	0.90	0.91
	generic	0.99	0.86	0.56	0.63	0.70
	`trubric`	0.99	0.84	0.16	0.30	0.22
`bge-large-en-v1.5`	raw trace	0.95	0.78	0.73	0.79	0.86
	generic	0.99	0.84	0.58	0.64	0.73
	`trubric`	0.99	0.85	0.16	0.33	0.23
`gte-large`	raw trace	0.95	0.77	0.77	0.83	0.90
	generic	0.99	0.85	0.56	0.59	0.70
	`trubric`	0.99	0.84	0.17	0.33	0.23
`e5-large-v2`	raw trace	0.96	0.77	0.79	0.86	0.90
	generic	0.99	0.85	0.62	0.66	0.77
	`trubric`	0.99	0.85	0.21	0.37	0.29
`all-mpnet-base-v2`	raw trace	0.93	0.77	0.68	0.76	0.82
	generic	0.99	0.84	0.43	0.52	0.57
	`trubric`	0.99	0.83	0.11	0.29	0.16
`all-MiniLM-L6-v2`	raw trace	0.91	0.75	0.70	0.75	0.86
	generic	0.99	0.84	0.45	0.53	0.59
	`trubric`	0.99	0.82	0.11	0.31	0.16
chance		0.05	0.50	0.01	0.12	0.02

reproduction The agent verified the issue by running a quick test comparing the old and new regex patterns against usernames with and without trailing newlines. The test confirmed that the old pattern matched usernames ending with a newline, while the new pattern correctly rejected them, and valid usernames still passed.

editing The agent modified the regex pattern in both validator classes from r'^[\w.@+-]+$' to r'\A[\w.@+-]+\Z' in django/contrib/auth/validators.py. The diff shows only these two regex changes, with no other modifications made to the file.

verification The agent ran the auth test suite which passed all 494 tests, including the tests in auth_tests/test_validators.py for UsernameValidatorsTests. It also ran a targeted test confirming that usernames with trailing newlines are now rejected by both validators while valid usernames still pass.

final state The final state shows django/contrib/auth/validators.py with both regex patterns updated to r'\A[\w.@+-]+\Z' in ASCIIUsernameValidator and UnicodeUsernameValidator. No other files were modified, and tests confirm the newline rejection behavior works correctly.

A.2 Linear classification tasks

For each combination of text encoder and trace representation, we assemble one embedding per trace for the executions of the 107 SWE-bench Verified systems on a fixed panel of 20 tasks (2,129 traces; a small number of executions are missing from the public archive). Rubric-based representations are the concatenated per-field embeddings of Section 3.1; raw-trace representations embed head and tail slices of the rendered log as two blocks. Each block is ℓ_2-normalized and blocks are weighted equally.

Probes are linear classifiers trained with class-balanced squared loss and a ridge penalty; the penalty is selected along a closed-form regularization path using grouped cross-validation within the training folds. Evaluation is five-fold grouped cross-validation with seeded, size-balanced group allocation: for the system-level attributes (iii)–(v) the groups are tasks, so probes are evaluated on tasks unseen during training, and for (i)–(ii) the groups are model families, so probes are evaluated on systems from families unseen during training. We report balanced accuracy averaged over folds; chance balanced accuracy for a C-class attribute is $1/C$. Figure 1 plots $(\text{accuracy} - \text{chance})/(1 - \text{chance})$ for attributes (i)–(ii) and its complement for (iii)–(v), so that better-for-evaluation is always outward. Table 2 reports the balanced accuracies underlying Figure 1.

A.3 Prompts

The rubric-instantiation prompt ({`problem`} is the task description):

```
Here is a repository issue that a software-engineering agent will be asked to fix:

{problem}

Write an instance-specific rubric for judging an agent's trace on THIS issue:
for each section below, one sentence saying what specifically to look for in
this instance (name the likely relevant modules/files/tests if inferable).

Sections: understanding, localization, reproduction, editing,
verification, final_state.
Respond with ONLY a JSON object with those keys, each a one-sentence string.
```

The extraction prompt ({`rubric`} is the instantiated `trubric` for the task, or the fixed generic rubric below for the generic baseline; {`keys`} the field names; the rendered trace is appended after the prompt):

```
Below is the (truncated) execution trace of a software-engineering agent.
Extract a factual description of the agent's behavior for EACH of the following
sections. 30-60 words per section. Describe behavior only -- do not restate the
problem. If a section did not occur, write exactly what is missing (e.g. "never
ran any test after editing").

{rubric}

Respond with ONLY a JSON object with keys {keys}, each a string.
```

The generic rubric:

```
- understanding: how the agent oriented itself before acting
- localization: how it searched/navigated to find the relevant code
- reproduction: whether/how it reproduced the issue before fixing
- editing: what it changed (files, nature and size of the edit)
- verification: whether it re-ran tests or a repro script after editing,
  and the OBSERVED outcome (passing/failing/errors)
- final_state: how the run ended (clean finish, step/cost limit,
  submitted unverified)
```

A.4 Rubric field list

The rubric-size experiment of Figure 6 draws field subsets from a bank of 32 fields. The bank consists of the six canonical fields used throughout the paper – understanding, localization, reproduction, editing, verification, and final state – together with 26 additional fields: requirements, causal diagnosis, code context, planning, test design, edge cases, regression scope, feedback response, tool recovery, patch review, exploration breadth, hypothesis testing, search strategy, dependency handling, error interpretation, incrementalism, backtracking, code reuse, instruction adherence, efficiency, state tracking, assumption checking, output quality, risk management, domain knowledge, and completion confidence.

B Efficient benchmarking details

B.1 Score-only baselines

Sample score. The mean of the target's m observed trace-level scores, $\hat{y}_{\text{sample}} = \frac{1}{m}\sum_{t\in T_0} s(f_0(t))$.

Population mean. The mean benchmark score of the reference systems, $\hat{y}_{\text{pop}} = \frac{1}{n}\sum_{i=1}^{n} y_i$, which ignores the target's executions entirely.

Item-response model. We fit the two-parameter logistic model $\mathbb{P}(s(x_{ij}) = 1) = \text{sigmoid}\left(a_j(\theta_i - b_j)\right)$ to the reference score matrix by maximum a posteriori estimation, with standard normal priors on the abilities θ_i and difficulties b_j and a $\mathcal{N}(0, 1/10)$ prior on the log-discriminations $\log a_j$ (centering the slopes at the Rasch value of one), optimized with L-BFGS. Because the benchmark score aggregates all M tasks while the model is fit on the M-task panel, we additionally fit a monotone link from ability to benchmark score, $y \approx \text{sigmoid}(c_1\theta + c_0)$, by least squares on the reference systems. For a target, we form a posterior over ability on a dense grid – the

prior is a normal with the mean and standard deviation of the reference abilities – update it with the likelihood of the target's m observed trace-level scores, and report the posterior mean of the fitted ability-to-score link as $\hat{y}_{\text{IRT}}$.

Adaptive task selection. Given the fitted item-response model, tasks are selected greedily per target: at each step, the next task is the one with the highest expected Fisher information $a_j^2\, p_j(\theta)(1-p_j(\theta))$ under the current ability posterior, the outcome is observed, and the posterior is updated. Random task selection instead draws T_0 uniformly; reported random-selection results average over 50 shared draws.

B.2 Trace-geometry baselines

The raw-trace and generic-rubric estimators replace the `trubric` representation ϕ in the pipeline of Section 3.2 and are otherwise identical. The raw-trace representation embeds the first and last portions of the rendered execution log as two blocks with the same encoder g; representations are centered by the task-level median across systems and ℓ_2-normalized. The generic-rubric representation follows the two-stage construction of Section 3.1 with the fixed field descriptions of Appendix A.3 in place of task-specific rubrics.

B.3 Hyperparameters and cross-validation

For each task draw, hyperparameters are selected by treating each reference system as a pseudo-target: its score is predicted from the remaining references (leave-one-out via the hat matrix for the ridge regressor), and the configuration minimizing the mean reference error is selected. The candidate grids are the kernel bandwidth $\sigma^2 \in \{\text{med}/2^k\}_{k=0}^{8}$ where med is the median squared distance between task embeddings, the dimension $d \in \{8, 16\}$, and the ridge penalty $\in \{0.1, 1.0\}$. The ensemble weight α is selected on a 101-point grid over $[0, 1]$ by the same reference-pseudo-target procedure.

C Systems and tasks

C.1 SWE-bench Verified systems

Table 3 lists the 107 evaluated leaderboard submissions with their model-family tag (used by the leave-one-family-out protocol; – indicates no tag could be derived from submission metadata) and official resolve rate y.

Table 3: SWE-bench Verified systems.

system	family	y
20240402_sweagent_gpt4	gpt	0.224
20240612_MASAI_gpt4o	gpt	0.326
20240620_sweagent_claude3.5sonnet	claude	0.336
20240721_amazon-q-developer-agent-20240719-dev	–	0.388
20240820_epam-ai-run-gpt-4o	gpt	0.240
20240820_honeycomb	–	0.406
20240824_gru	–	0.452
20240918_lingma-agent_lingma-swe-gpt-72b	gpt	0.250
20240918_lingma-agent_lingma-swe-gpt-7b	gpt	0.102
20240920_solver	–	0.436
20240924_solver	–	0.454
20241001_nfactorial	–	0.258
20241002_lingma-agent_lingma-swe-gpt-72b	gpt	0.288
20241002_lingma-agent_lingma-swe-gpt-7b	gpt	0.182
20241016_composio_swekit	claude	0.406
20241016_epam-ai-run-gpt-4o	gpt	0.270
20241022_tools_claude-3-5-haiku	claude	0.406
20241022_tools_claude-3-5-sonnet-updated	claude	0.490
20241023_emergent	–	0.466
20241025_composio_swekit	–	0.486
20241028_agentless-1.5_gpt4o	gpt	0.388
20241028_solver	–	0.500
20241029_OpenHands-CodeAct-2.1-sonnet-20241022	claude	0.530
20241029_epam-ai-run-claude-3-5-sonnet	claude	0.396

system	family	y
20241030_nfactorial	–	0.416
20241105_nfactorial	–	0.492
20241106_navie-2-gpt4o-sonnet	–	0.472
20241108_autocoderover-v2.0-claude-3-5-sonnet-20241022	claude	0.462
20241108_devlo	claude	0.542
20241113_nebius-search-open-weight-models-11-24	–	0.406
20241120_artemis_agent	–	0.320
20241125_enginelabs	claude	0.518
20241125_marscode-agent-dev	–	0.500
20241202_agentless-1.5_claude-3.5-sonnet-20241022	claude	0.508
20241202_amazon-q-developer-agent-20241202-dev	–	0.550
20241208_gru	–	0.570
20241212_epam-ai-run-claude-3-5-sonnet	claude	0.554
20241212_google_jules_gemini_2.0_flash_experimental	gemini	0.522
20241213_devlo	–	0.582
20241223_emergent	–	0.572
20250110_blackboxai_agent_v1.1	–	0.628
20250110_learn_by_interact_claude3.5	claude	0.602
20250117_wandb_programmer_o1_crosscheck5	o1	0.646
20250118_codeshellagent_gemini_2.0_flash_experimental	gemini	0.442
20250120_Bracket	–	0.532
20250122_autocoderover-v2.1-claude-3-5-sonnet-20241022	claude	0.516
20250203_openhands_4x_scaled	–	0.608
20250206_agentscope	claude	0.634
20250214_agentless_lite_o3_mini	o3	0.424
20250224_tools_claude-3-7-sonnet	claude	0.632
20250226_swerl_llama3_70b	llama	0.412
20250228_epam-ai-run-claude-3-5-sonnet	claude	0.628
20250316_augment_agent_v0	–	0.654
20250405_amazon-q-developer-agent-20250405-dev	–	0.654
20250410_cortexa	–	0.582
20250415_openhands	–	0.658
20250430_zencoder_ai	–	0.700
20250511_sweagent_lm_32b	–	0.402
20250514_aime_coder	claude	0.664
20250515_Refact_Agent	–	0.704
20250516_cortexa_o3	–	0.682
20250519_devlo	–	0.702
20250519_trae	–	0.706
20250520_openhands_devstral_small	devstral	0.468
20250522_sweagent_claude-4-sonnet-20250514	claude	0.666
20250522_tools_claude-4-opus	claude	0.732
20250522_tools_claude-4-sonnet	claude	0.724
20250524_openhands_claude_4_sonnet	claude	0.704
20250527_amazon.nova-premier-v1.0	nova	0.424
20250603_Refact_Agent_claude-4-sonnet	–	0.744
20250610_augment_agent_v1	–	0.704
20250611_moatless_claude-4-sonnet-20250514	claude	0.708
20250612_trae	–	0.752
20250616_Skywork-SWE-32B	qwen	0.380
20250616_Skywork-SWE-32B+TTS_Bo8	–	0.470
20250623_warp	–	0.710
20250629_deepswerl_r2eagent	–	0.422
20250629_deepswerl_r2eagent_tts	o1	0.588
20250710_bloop	claude	0.712
20250715_qodo_command	–	0.712
20250716_openhands_kimi_k2	kimi	0.654
20250720_Lingxi-v1.5_claude-4-sonnet-20250514	claude	0.746
20250725_sweagent_devstral_small_2507	devstral	0.380
20250728_zai_glm4-5	–	0.642
20250731_harness_ai	claude	0.748
20250804_codesweep_sweagent_kimi_k2_instruct	kimi	0.534
20250804_epam-ai-run-claude-4-sonnet	claude	0.768
20250806_SWE-Exp_DeepSeek-V3	–	0.420
20250807_openhands_gpt5	gpt	0.718
20250819_ACoder	–	0.764
20250901_entroPO_R2E_QwenCoder30BA3B	qwen	0.522
20250901_entroPO_R2E_QwenCoder30BA3B_tts	qwen	0.604
20250901_warp	–	0.756
20250902_atlassian-rovo-dev	–	0.768
20250915_JoyCode	–	0.746
20250924_artemis_agent_v2	claude	0.570
20250928_trae_doubao_seed_code	doubao	0.788
20250930_zai_glm4-6	–	0.682
20251014_Lingxi_kimi_k2	kimi	0.712
20251021_SalesforceAIResearch_SAGE_bash_only	–	0.730
20251103_SalesforceAIResearch_SAGE_OpenHands	–	0.738
20251103_sonar-foundation-agent_claude-sonnet-4-5	claude	0.748
20251110_frogboss-32b	–	0.536

system	family	y
20251110_frogmini-14b	–	0.450
20251120_livesweagent_gemini-3-pro-preview	gemini	0.774
20251127_openhands_claude-opus-4-5	claude	0.776
20251205_sonar-foundation-agent_claude-opus-4-5	claude	0.792

C.2 TERMINAL-BENCH 2.0 SYSTEMS

Table 4 lists the 65 evaluated submissions with their harness, underlying-model tags, and official score y (mean over five trials). Truncated entries are spelled out below the table.

Table 4: Terminal-Bench 2.0 systems.

system	harness	model tags	y
Ante__Gemini-3-Pro-Preview	ante	gemini-3-pro-preview	0.647
Ante__Gemini-3.1-Pro-Preview	ante	gemini-3-pro-preview	0.742
BashAgent__TermiGen-32B	bash-agent	termigen-32b	0.197
Capy__GPT-5.5	capy	gpt-5-5	0.834
ClaudeCode__GLM-4.7	claude-code	glm-4-7	0.333
CodeBrain-1.5__GPT-5.3-Codex	codebrain-1	gpt-5-3-codex	0.816
CodeBrain-1__GPT-5.3-Codex	codebrain-1	gpt-5-3-codex	0.703
CodeBrain-1__Gemini-3-Pro-Preview	codebrain-1	gemini-3-pro-preview	0.622
Codelia__GPT-5.3-Codex	codelia	gpt-5-3-codex	0.757
Deep-Agents__GPT-5.2-Codex	deep-agents	gpt-5-2-codex	0.667
Dirac__Gemini-3-Flash-Preview	dirac	gemini-3-flash-preview	0.652
Forge__GPT-5.4	forgecode	gpt-5-4	0.818
Forge__Gemini-3.1-Pro-Preview	forge-code	gemini-3-1-pro-preview	0.787
Forge__Opus-4.6	forgecode	claude-opus-4-6	0.818
Gemini_CLI__Gemini-3-Flash-Preview	gemini-cli	gemini-3-flash-preview	0.480
Gemini_CLI__Gemini-3.1-Pro-Preview	gemini-cli	gemini-3-1-pro-preview	0.608
IndusAGICodingAgent__MiniMax-M2.7	indusagi-coding-agent	minimax-m2-7	0.452
IndusAGICodingAgent__gpt-5.3-codex	indusagi-coding-agent	gpt-5-3-codex	0.691
JJAgent__Multiple	jjagent	–	0.871
Judy__Claude-Opus-4.6	judy	claude-opus-4-6	0.721
Judy__Gemini-3.1-Pro-Preview	tongagents	gemini-3-1-pro-preview	0.805
Junie_CLI__Gemini-3-Flash-Preview...[a]	junie-cli	claude-opus-4-6/gemini-3-1-pr...[b]	0.710
LemonCode__GPT-5.3-Codex	lemon-code	gemini-3-1-pro-preview/gpt-5-...[c]	0.799
LemonHarness_GPT-5.3-CodeX	lemonharness	gemini-3-1-pro-preview/gpt-5-...[d]	0.849
MAYA__Claude-4.5-sonnet	maya	claude-sonnet-4-5	0.427
MAYA__Claude-4.6-opus	maya-v2	claude-opus-4-6	0.725
Meta-Harness__Claude-Opus-4.6	meta-harness	claude-opus-4-6	0.764
Mux__Claude-Opus-4.5	mux	claude-opus-4-5	0.584
Mux__Claude-Opus-4.6	mux	claude-opus-4-6	0.665
Mux__GPT-5.2	mux	gpt-5-2	0.621
Mux__GPT-5.3-Codex	mux	gpt-5-3-codex	0.746
NexAU-AHE__gpt-5.5	nexau-ahe	gpt-5-5	0.847
OB-1_GPT-5.3-Codex-Claude-Opus-4....[e]	ob-1	claude-opus-4-5/claude-opus-4...[f]	0.727
OB-1_GPT-5.4-GPT-5.3-Codex-Claude...[g]	ob-1	claude-opus-4-5/claude-opus-4...[h]	0.825
OpenCode__Claude-Opus-4.5	opencode	claude-opus-4-5	0.523
OpenSage__GPT-5.3-Codex	sageagent	gpt-5-3-codex	0.794
OpenSage__Gemini-3-Pro-Preview	sageagent	gemini-3-pro-preview	0.652
Polaris__Claude-Opus-4.7-GPT-5.5-...[i]	polaris	claude-opus-4-7/gemini-3-1-pr...[j]	0.822
Simplai-Agent__Claude-Sonnet-4.6	simplai-agent	claude-sonnet-4-6	0.535
Simple-Codex__GPT-5.3-Codex	simple-codex	gpt-5-3-codex	0.752
Terminus-KIRA__Claude-Opus-4.6	terminus-kira	claude-opus-4-6	0.747
Terminus-KIRA__Gemini-3.1-Pro-Pre...[k]	terminus-kira	gemini-3-1-pro-preview	0.748
Terminus2__Claude-Opus-4.6	terminus2	claude-opus-4-6	0.629
Terminus2__DeepSeek-V3.2	terminus2	deepseek-v3-2	0.396
Terminus2__GLM-4.7	terminus2	glm-4-7	0.334
Terminus2__GLM-5	terminus2	glm-5	0.538
Terminus2__GPT-5.3-Codex	terminus2	gpt-5-3-codex	0.647
Terminus2__Kimi-k2.5	terminus2	kimi-k2-5	0.433
Terminus2__Minimax-m2.5	terminus2	minimax-m2-5	0.426
WOZCODE__Claude-Opus-4-7	wozcode	claude-opus-4-7	0.802
Wecode__GPT-5.5	wecode	gpt-5-5	0.881
WozCode__Claude-Opus-4.6	wozcode	claude-opus-4-6	0.681
cchuter__minimax-m2.5	cchuter	minimax-m2-5	0.427
clnkr__GPT-5.5	clnkr	gpt-5-5	0.661
copilot-cli__claude-opus-4.6	copilot-cli	claude-opus-4-6	0.607
dakou__qwen3-coder-480b	dakou-agent	qwen3-coder-480b-a35b-instruc...[l]	0.272
grok-cli__grok-4.20-0309-reasoning	grok-cli	grok-4-20-0309-reasoning	0.573
logos-latest__claude-opus-4.7	logos	–	0.852
logos-ts__claude-opus-4.7	logos	–	0.807
logos__claude-opus-4.6	logosagent	claude-opus-4-6	0.686
pilot-real__claude-opus-4-6	pilot	claude-opus-4-6	0.831
spoox-o-m__GPT-5-Nano	spoox-o-m	gpt-5-nano	0.218
spoox-o-m__GPT-5.3-Codex	spoox-o-m	gpt-5-3-codex	0.721
terminus-2__AfterQuery-GPT-OSS-20B	terminus2	gpt-oss-20b-rl	0.172
vix__claude-opus-4-7	vix	claude-opus-4-7	0.903

[a] Junie_CLI__Gemini-3-Flash-Preview-Gemini-3.1-Pro-Preview-Claude-Opus-4.6-GPT-5.3-Codex
[b] claude-opus-4-6/gemini-3-1-pro-preview/gemini-3-flash-preview/gpt-5-3-codex
[c] gemini-3-1-pro-preview/gpt-5-3-codex
[d] gemini-3-1-pro-preview/gpt-5-3-codex
[e] OB-1_GPT-5.3-Codex-Claude-Opus-4.5-Claude-Opus-4.6
[f] claude-opus-4-5/claude-opus-4-6/gpt-5-3-codex
[g] OB-1_GPT-5.4-GPT-5.3-Codex-Claude-Opus-4.5-Claude-Opus-4.6
[h] claude-opus-4-5/claude-opus-4-6/gpt-5-3-codex/gpt-5-4
[i] Polaris__Claude-Opus-4.7-GPT-5.5-Gemini-3.1-Pro
[j] claude-opus-4-7/gemini-3-1-pro-preview/gpt-5-5
[k] Terminus-KIRA__Gemini-3.1-Pro-Preview
[l] qwen3-coder-480b-a35b-instruct-fp8

C.3 TASK PANELS

The SWE-bench Verified panel consists of the following $M = 100$ instances (a fixed 20-task panel united with a seeded random draw of 80 further instances):

astropy__astropy-14096, astropy__astropy-7336, django__django-10973, django__django-11099, django__django-11206, django__django-11333, django__django-11477, django__django-11490, django__django-11532, django__django-11880, django__django-12276, django__django-12419, django__django-13012, django__django-13121, django__django-13128, django__django-13297, django__django-13346, django__django-13410, django__django-13590, django__django-13807, django__django-13810, django__django-13933, django__django-14034, django__django-14053, django__django-14122, django__django-14238, django__django-14315, django__django-14539, django__django-14559, django__django-14725, django__django-14999, django__django-15037, django__django-15098, django__django-15315, django__django-15525, django__django-15554, django__django-15695, django__django-15814, django__django-15916, django__django-15930, django__django-15957, django__django-16100, django__django-16145, django__django-16333, django__django-16429, django__django-16454, django__django-16527, django__django-16560, django__django-16642, django__django-16801, django__django-16877, django__django-9296, matplotlib__matplotlib-20859, matplotlib__matplotlib-23299, matplotlib__matplotlib-24149, matplotlib__matplotlib-24177, matplotlib__matplotlib-25287, matplotlib__matplotlib-25332, matplotlib__matplotlib-25479, matplotlib__matplotlib-25775, matplotlib__matplotlib-26113, pallets__flask-5014, psf__requests-1724, pydata__xarray-3305, pydata__xarray-4075, pydata__xarray-4094, pydata__xarray-4966, pydata__xarray-6744, pydata__xarray-7229, pydata__xarray-7233, pydata__xarray-7393, pylint-dev__pylint-6386, pylint-dev__pylint-6528, scikit-learn__scikit-learn-10908, scikit-learn__scikit-learn-13439, sphinx-doc__sphinx-10435, sphinx-doc__sphinx-10466, sphinx-doc__sphinx-10614, sphinx-doc__sphinx-8120, sphinx-doc__sphinx-8475, sphinx-doc__sphinx-8593, sphinx-doc__sphinx-8595, sphinx-doc__sphinx-8638, sphinx-doc__sphinx-9258, sphinx-doc__sphinx-9367, sphinx-doc__sphinx-9461, sphinx-doc__sphinx-9658, sphinx-doc__sphinx-9673, sphinx-doc__sphinx-9698, sympy__sympy-13551, sympy__sympy-13647, sympy__sympy-13974, sympy__sympy-14248, sympy__sympy-14531, sympy__sympy-15875, sympy__sympy-17655, sympy__sympy-20154, sympy__sympy-21596, sympy__sympy-23950, sympy__sympy-24661

The Terminal-Bench 2.0 panel uses all 89 tasks:

adaptive-rejection-sampler, bn-fit-modify, break-filter-js-from-html, build-cython-ext, build-pmars, build-pov-ray, caffe-cifar-10, cancel-async-tasks, chess-best-move, circuit-fibsqrt, cobol-modernization, code-from-image, compile-compcert, configure-git-webserver, constraints-scheduling, count-dataset-tokens, crack-7z-hash, custom-memory-heap-crash, db-wal-recovery, distribution-search, dna-assembly, dna-insert, extract-elf, extract-moves-from-video, feal-differential-cryptanalysis, feal-linear-cryptanalysis, filter-js-from-html, financial-document-processor, fix-code-vulnerability, fix-git, fix-ocaml-gc, gcode-to-text, git-leak-recovery, git-multibranch, gpt2-codegolf, headless-terminal, hf-model-inference, install-windows-3.11, kv-store-grpc, large-scale-text-editing, largest-eigenval, llm-inference-batching-scheduler, log-summary-date-ranges, mailman, make-doom-for-mips, make-mips-interpreter, mcmc-sampling-stan, merge-diff-arc-agi-task, model-extraction-relu-logits,

modernize-scientific-stack, mteb-leaderboard, mteb-retrieve, multi-source-data-merger, nginx-request-logging, openssl-selfsigned-cert, overfull-hbox, password-recovery, path-tracing, path-tracing-reverse, polyglot-c-py, polyglot-rust-c, portfolio-optimization, protein-assembly, prove-plus-comm, pypi-server, pytorch-model-cli, pytorch-model-recovery, qemu-alpine-ssh, qemu-startup, query-optimize, raman-fitting, regex-chess, regex-log, reshard-c4-data, rstan-to-pystan, sam-cell-seg, sanitize-git-repo, schemelike-metacircular-eval, sparql-university, sqlite-db-truncate, sqlite-with-gcov, torch-pipeline-parallelism, torch-tensor-parallelism, train-fasttext, tune-mjcf, video-processing, vulnerable-secret, winning-avg-corewars, write-compressor