QVAL: Cheaply Evaluating Dense Supervision Signals for Long-Horizon LLM Agents

Sergio Hernández-Gutiérrez[1] **Matteo Merler**[2*] **Ilze Amanda Auzina**[1*]
Joschka Strüber[1] **Ameya Prabhu**[1†] **Matthias Bethge**[1†]

[1]Tübingen AI Center, University of Tübingen [2]Fondazione Bruno Kessler

Abstract

LLM agents increasingly act over long horizons, where a single trajectory can contain hundreds or thousands of actions. In these settings, outcome-only rewards provide too sparse guidance, failing to inform the model about the goodness of intermediate actions. Dense supervision methods aim to solve this problem by scoring intermediate steps, from intrinsic confidence to self-distillation and embedding similarities; yet, it is common practice to evaluate them by measuring the downstream performance of a training pipeline that integrates them. This is expensive, conflates supervision quality with training engineering confounders, and renders different methodological families requiring distinct training setups incomparable. As a result, dense supervision methods are rarely evaluated on common ground. We introduce QVAL, a training-free testbed for directly evaluating dense supervision signals. Given a state-action pair, QVAL measures how well a method's score is *value-aligned*: whether it orders actions according to reference-policy Q-values. This lets us compare signals before any training run and separate signal quality from design choices therein. We instantiate QVAL with four diverse environments and benchmark 21 dense supervision methods spanning seven methodological families, comprising over 1.2K evaluation experiments across six open-weight backbones. We find that simple prompting baselines consistently outperform recent dense supervision methods from the literature, and that performance clusters strongly by method family. These findings hold across model sizes, environments, and observation modalities. QVAL is designed to be easily extensible to new environments, enabling researchers to iterate on dense supervision methods before launching a single training run.

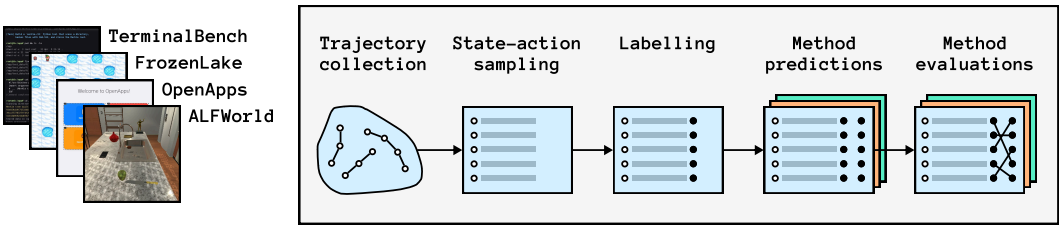


Figure 1: QVAL **design pipeline**. We collect trajectories in multi-turn environments, sample candidate state-action pairs, and label them with Q-values with respect to a reference policy. We perform method prediction and measure alignment between the predicted scores and the labels. This training-free testbed isolates learning-signal quality before any downstream RL training.

*Equal Contribution †Equal Advising Correspondence to: `sergio.hernandez@bethgelab.org`

Preprint.

1 Introduction

Autonomous agents are the next frontier for Large Language Models (LLMs): they can write codebases, operate graphical interfaces, and navigate simulated environments. These tasks require models to act over long horizons, with a single trajectory spanning hundreds or thousands of actions. Sparse rewards make learning intractable as horizons grow: an outcome-based reward gives little guidance about the goodness of individual steps and, more critically, they may never be observed if the task is beyond the frontier of the model's capabilities. Current Reinforcement Learning (RL) algorithms for LLM post-training primarily rely on sampling-based value estimation. For example, GRPO (Shao et al., 2024) estimates the value of a completion by comparing samples within a group. This works best when trajectories are short, requiring the model a reasonable number of samples to attribute which actions caused the outcome. However, agent trajectories increasingly involve multi-step tool use, recursive decomposition, and context compaction. As these trajectories grow in length, the trajectories diverge, outcome rewards become insufficient, and group-based estimators discard most of the information contained in intermediate actions. This limitation has motivated methods that produce denser supervision signals. For example, intrinsic-curiosity-based supervision methods, which use token probabilities along reasoning traces (Yoon et al., 2026), tool calls (Xie et al., 2026), or interaction feedback (Auzina et al., 2026); and self-distillation methods, which derive intermediate supervision from model-generated judgments or targets (Hübotter et al., 2026; Shenfeld et al., 2026; Song et al., 2026). These methods share a common goal: assign useful scores to intermediate states or actions. Yet they have largely been developed and evaluated in isolation. Thus, we can view them as instances of a broader set of *dense supervision* methods, and a taxonomy derived from their methodological families naturally arises.

The primary bottleneck towards comprehensive evaluation is that we lack a cheap and direct way to compare dense supervision signals. Today, a method is evaluated by inserting it into a post-training pipeline and measuring the downstream accuracy improvement. This is expensive, often unachievable due to the distinct setups required by different methods (e.g., algorithms or data modalities), and it makes the results hard to interpret: the measured gain conflates the supervision quality and the engineering choices used for RL training, such as algorithmic or optimization features, normalization techniques, interactions with other training signals, etc. We pose the following research question:

Can we evaluate dense supervision signals in isolation,
before expensive post-training runs?

We introduce QVAL, a cheap, training-free *testbed* for dense supervision methods. This enables researchers to iterate on new dense supervision methods and their variants quickly. Practitioners can vet candidate signals before integrating them into post-training pipelines, where post-training costs and confounds are higher. Dense signals also matter beyond training: they can guide search at test time, including tree search or MCTS-style rollouts.

QVAL has a simple design, shown in Figure 1. Each sample is a state-action pair (s, a) labelled with a *reference Q-value*: the expected return of the trajectory that continues from (s, a) under an expert *reference policy*. We obtain this from either a scripted optimal policy or an expert planner where one exists, or a frontier model where it does not, estimating the value from the best of several rollouts (Section 3). We evaluate a method by scoring every sample by its *value-alignment*: how well its predicted dense scores order the samples relative to the reference Q-values. Because every method sees the same inputs and is judged against the same targets, with fixed models and prompt context, QVAL isolates the quality of the signal itself from the confounders of a training pipeline, giving an indication of whether methods provide a viable learning signal before any training runs.

At release, QVAL covers four multi-turn environments across text and visual domains: programming and agentic terminal interaction, computer and application use, embodied reasoning, and goal-directed navigation. For each environment, we collect state-action pairs and provide reference Q-values. We use these four datasets to evaluate 21 dense supervision methods spanning seven methodological families: direct prompting (Liu et al., 2023; Ma et al., 2024), intrinsic signals (Auzina et al., 2026; Kwok et al., 2026), code generation (Ma et al., 2023b; Li et al., 2024), self-distillation (Hübotter et al., 2026), ranking-based prediction, pre-trained models (Ma et al., 2022, 2023a), and embedding similarity (Rocamonde et al., 2024; Baumli et al., 2023). We find large differences in signal quality across methods. Simple prompting baselines outperform several more complex methods from the literature. Methodological family predicts performance: direct-prompting methods perform

consistently well across environments, while code-based methods work best in smaller action and state spaces and do not scale as well to more complex settings. We find that adding complexity within a family rarely helps. We also find that signal quality does not degrade monotonically with task difficulty, and that text observations and Q-value targets tend to produce stronger signals.

In summary, we make three main contributions. First, we introduce QVAL, a training-free testbed that evaluates dense supervision methods by their *value-alignment*, how well their scores order actions according to reference Q-values, across text and visual domains. It provides a common comparison ground, makes signal quality cheap to measure, and separates that measurement from later integration into training pipelines. Second, we explore how to annotate multimodal datasets of state-action pairs from four diverse environments, i.e., how to generate the reference Q-values. Third, we benchmark 21 dense supervision methods, grouped into seven families, across six open-weight backbones from 9B to 122B parameters, for more than 1.2K experiments. Our analysis shows that simple baselines are strong, methods cluster by family, added intra-family complexity rarely helps, and the choice of learning signal remains a major bottleneck for agentic post-training.

QVAL is built to grow. The same collection and labelling procedure can be easily extended to new environments. A new method only needs to provide one score per state-action pair to allow direct comparisons. We will continue expanding QVAL as new agentic benchmarks emerge. Practitioners can also use our framework to build evaluation datasets for their own tasks and environments.

2 QVAL: A Training-Free Testbed for Dense Supervision Methods

Dense supervision should primarily predict the overall success of trajectories. A single long-horizon trajectory contains many actions, and one that looks reasonable in isolation can still make the final goal harder to reach. Dense supervision methods score each intermediate decision, so a score is only useful if it reflects *where a decision leads* rather than how good it looks locally. QVAL asks exactly this: does a method assign higher scores to the actions that make eventual success more likely?

Setup. We consider an environment modelled as a Markov Decision Process (Bellman, 1957), with state space $\mathcal{S}$, action space $\mathcal{A}$, transition distribution $T(s' \mid s, a)$, reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1]$. At each step the agent observes a state $s \in \mathcal{S}$, takes an action $a \sim \pi(\cdot \mid s)$, receives reward $r(s, a)$, and the environment transitions to $s' \sim T(\cdot \mid s, a)$. A trajectory $\tau = (s_0, a_0, r_0, s_1, \dots)$ is a realization of this process, with return $G(\tau) = \sum_{t \geq 0} \gamma^t r_t$. A *policy* $\pi(a \mid s)$ maps states to distributions over actions. The *Q-value function* of a policy π,

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi}[G(\tau) \mid s_0 = s,\, a_0 = a]\,, \tag{1}$$

gives the expected return from state s after committing to action a, and afterwards continuing to behave following π (Sutton and Barto, 2018). The analogous *state-value function* $V^\pi(s) = \mathbb{E}_{\tau \sim \pi}[G(\tau) \mid s_0 = s]$ scores a state without committing to a first action.

Reference policies. Q^π is only defined once a *reference* continuation policy π is fixed. QVAL uses Q^π similarly to how a supervised dataset uses labels, annotating each decision point with a reference value, and evaluating a method based on how well its scores reproduce their ordering. Importantly, π is the policy we label (s, a) pairs with, not the policy that will ultimately be trained with the signal (i.e., the LLM). In fact, π should be as close to optimal as the environment allows, so that a high Q^π denotes a genuinely high-value action, and there is no risk of a good action receiving a bad score due to a sub-optimal continuation by π. Outcome rewards say nothing about intermediate behaviour; $r_t = 0$ at every non-terminal step, so an action's value is determined entirely by how π rolls out the trajectory. We do not assume a reference policy is available when a dense signal is later deployed, only that we can construct one here, in a controlled setting, to obtain trustworthy labels. Section 3 describes how we obtain the reference policy π for each chosen environment.

Value-aligned signals. A dense supervision method assigns a scalar score to each intermediate action, i.e. $k : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. QVAL measures whether they correlate with the reference value, ordering decisions the same way their return does. We call k *value-aligned* under π if

$$k(s, a) = \phi\big(Q^\pi(s, a)\big) \quad \text{for some strictly increasing } \phi, \tag{2}$$

so that a perfectly value-aligned signal ranks all decision points exactly as Q^π does.

We argue that value-alignment is a cheap proxy for a signal's downstream usefulness, as long as the reference policy π used to compute Q^π is a close approximation of an optimal policy. A signal that orders actions by their return will provide meaningful supervision at every step a policy takes during training, and one that orders them poorly must rely on other mechanisms to be useful, so alignment is a cheap early indicator of whether a signal carries the information needed for successful supervision.

Evaluating value-alignment. Value-alignment (Eq. 2) is a theoretical property: a signal either is a strictly increasing transform of Q^π, or it is not. In practice, we want to measure the *degree* to which a method is value-aligned under our reference policies π. We thus evaluate each method by the rank correlation between its predicted scores and the reference labels. Predictions live on incompatible scales (e.g., raw LLM scores, code-generated outputs, token log-probabilities, embedding distances) that cannot be placed under a common loss, so we compare methods by the ordering they induce rather than by absolute values. This is consistent with recent evidence that LLM and VLM judges order candidates reliably even when their absolute scores are poorly calibrated (Kumar et al., 2026). We report Spearman's ρ (Spearman, 1904) as our main metric, standard for meta-evaluating an automatic scorer against reference judgments (Liu et al., 2023), and Kendall's τ (Kendall, 1938) in Appendix C. Both lie in $[-1, 1]$ and measure monotonic agreement; they differ in outlier sensitivity, with Spearman dominated by a few badly-ordered pairs and Kendall weighting every inversion equally. For methods that output a permutation over the candidate actions at a state, rather than a value per point, we compute the rank correlation between the predicted and the label-induced permutation within each state and average across states. Appendix C gives the full definitions, including tie and NaN handling.

3 QVAL-v1.0: Benchmarking Dense Supervision Methods

We instantiate QVAL-v1.0, a benchmark built with the QVAL methodology described above, initially employing four environments and evaluating 21 dense supervision methods. QVAL is designed to be extensible: practitioners can add environments and methods without changing the core construction. We provide an overview of environments and methods in QVAL-v1.0 next, with detailed descriptions of each environment in Appendix B and each method in Appendix D.

3.1 Datasets

Environments. We choose four environments that stress different aspects of the problem: task difficulty, action space size, observation modality, and the context needed to judge an action. The suite covers goal-directed navigation in FrozenLake (Towers et al., 2025), embodied reasoning in ALFWorld (Shridhar et al., 2020), browser-based computer use in OpenApps (Ullrich et al., 2025), and terminal-based problem solving in TerminalBench (Merrill et al., 2025). These environments also span different state and action spaces. FrozenLake has a small discrete action space. TerminalBench requires open-ended shell commands over rich textual states. TerminalBench is text-only, while the other environments support both textual and visual observations.

Data collection. For each environment, we collect trajectories and sample N state-action pairs for each (Table 2, Appendix B). We do not aim to maximize task success during collection; instead, we aim to obtain a wide coverage, including both high- and low-value states and actions. For OpenApps and FrozenLake, we use scripted policies, designed to be sub-optimal and maximize coverage. For TerminalBench and ALFWorld, we generate trajectories with DeepSeek v3.2 (DeepSeek-AI et al., 2025). To further improve coverage, we select a limited set of state-action pairs from a range in the middle of each trajectory. This heuristic removes data-points that carry little signal for value prediction: very early states often occur before meaningful task progress, while very late states are close to termination, so most actions have similar value. We also sample three additional alternative actions for each state, which allows methods to rank candidate actions under the same context.

Data labeling. Our primary label is the estimated $Q^\pi(s_t, a_t)$ under the reference policy π of an action a_t taken in the state s_t. To label a point, we restore the environment to s_t, force the first continuation step to take the dataset action a_t, then follow π and record the discounted return. We perform this process several times if π is non-deterministic and choose as our label for the pair (s_t, a_t) the maximum observed return as an approximation to near-optimal continuation; this corresponds to a Max-Value Monte Carlo (MVMC) sampling strategy. The reference policy π is environment-specific. For OpenApps and FrozenLake, we use scripted optimal policies. For ALFWorld, we use an expert planner. In TerminalBench, however, an optimal policy is intractable to find. We therefore

estimate TerminalBench values with MVMC rollouts ($k = 16$) with GPT-5.5 (OpenAI, 2026) as a backbone. We verify that GPT-5.5 is a strong continuation policy: it reaches 100% Pass@16 on our TerminalBench subset, and we further compare it with Claude Opus 4.7 (Anthropic, 2026) in Section 4.2. We also provide a reference state value $V(s_t)$ for FrozenLake, ALFWorld, and OpenApps, estimated with the same procedure but without forcing the first action. This lets us test whether our results are robust to the choice of target value (Section 4.2).

For TerminalBench, we use a subset of tasks from TBLite (OpenThoughts-Agent team, 2026). Appendix D specifies model instantiations and sampling parameters. Appendix B gives full environment parameterization and prompts. Appendix F reports complete per-model results.

3.2 Dense Supervision Methods

Dense supervision methods can induce different signals and employ available information in varied ways. Some ask a model for a value directly, others compare actions or embeddings, generate code, or distill from feedback. Using QVAL-v1.0, we evaluate 21 dense supervision methods and group them into seven families by the information they use to score state-action pairs, summarized in Table 1. The methods include direct implementations of prior work, adaptations of prior methods to our setting, as well as additional baselines introduced in this paper. We next describe each family briefly.

Ranking methods prompt an LLM to directly compare a set of candidate actions from the same state. This group contains `ranking`, a direct LLM action-ranking baseline (details in Appendix D.7).

Table 1: **Methods covered in QVAL-v1.0**. We group methods by the signal used to score state–action pairs and report each method source.

Method group	Method name (source)
Ranking	`ranking` (Baseline)
Direct	`direct-16` (Baseline) `direct-batched` (Baseline) `direct-sequential` (Baseline) `direct-single` (Baseline) `gvl` (Ma et al., 2024)
Intrinsic scoring	`verifier` (Kwok et al., 2026) `Δbelief` (Auzina et al., 2026)
Self-Distillation	`sdpo` (Hübotter et al., 2026) `sdpo-gt` (Extension)
Pre-trained	`vip` (Ma et al., 2022) `liv-cos` (Ma et al., 2023a) `liv-l2` (Ma et al., 2023a) `liv-txt` (Ma et al., 2023a)
Embedding	`vlm-sor-softmax` (Baumli et al., 2023) `vlm-sor` (Baumli et al., 2023) `vlm-rm-cos` (Rocamonde et al., 2024) `vlm-rm` (Rocamonde et al., 2024)
Code	`eureka` (Ma et al., 2023b) `codegen` (Li et al., 2024) `codegen-avg` (Extension)

Direct methods prompt an LLM or VLM to output a numeric value for a data-point. This is reminiscent of the LLM-as-judge numeric-scoring paradigm (Liu et al., 2023). The simplest variant uses one data-point per prompt (`direct-single`). We also test variants that predict multiple data points at once, with points from the same environment but not necessarily from the same trajectories. The `direct-batched` variant provides multiple data-points in a single prompt, while `direct-sequential` provides multiple data-points in a multi-turn format, appending new data-points after the previous answer. The purpose of both variants is to test whether scoring multiple points at once helps to ground each score in a common scale. Another variant averages 16 independent estimates for the same data-point (`direct-16`). We also adapt GVL (Ma et al., 2024) to text-based environments and value prediction (`gvl`) by giving the model a shuffled full-trajectory around the target transition before asking for a value. Appendix D.2 gives details for these methods.

Intrinsic scoring methods derive scores from the model's own confidence rather than from an explicit value estimate. We adapt ΔBelief (Auzina et al., 2026) (`Δbelief`), which scores an action by the change in the probability the model assigns to eventual success once the action's outcome is observed. We also adapt LLM-as-a-Verifier (Kwok et al., 2026) (`verifier`), which prompts the model to score a (s, a, s') tuple on a rubric of per-environment quality criteria such as correctness, efficiency, and error-freeness, using the probabilities it assigns across an ordered grading scale and averaging them into a scalar score. Appendices D.3 and D.4 give details on these methods.

Self-distillation methods score a candidate action by how much more likely the model is to have produced that action once it sees additional privileged information about the action's outcome. The

intuition is that a good action becomes more probable in hindsight when its favorable outcome is shown, whereas a poor action does not. The family contains two offline re-ranking signals: `sdpo` (Hübotter et al., 2026), where the additional information is the candidate's immediate next state, and `sdpo-gt`, an oracle ablation that additionally reveals the next expert action and a summary of how the trajectory ends. Appendix D.6 gives more details for these methods.

Pre-trained methods use fixed representations from value-pretrained vision-language models. VIP (Ma et al., 2022) (`vip`) scores an image state by its negative distance to a goal-image embedding. LIV (Ma et al., 2023a) contributes three variants: `liv-cos`, which uses the paper's original cosine similarity score with an image goal; `liv-l2`, which uses a VIP-style negative Euclidean distance to an image goal; and `liv-txt`, which compares the state image embedding to a textual goal embedding. Appendix D.9 provides additional details about these methods.

Embedding methods use frozen vision-language encoders to score image states by similarity to a text goal. VLM-RM (Rocamonde et al., 2024) contributes `vlm-rm-cos`, which scores a state by cosine similarity between its image embedding and the goal-text embedding, and `vlm-rm`, which projects out baseline visual features before measuring progress toward the goal. VLM-SOR (Baumli et al., 2023) contributes `vlm-sor-softmax`, which uses the softmax probability assigned to the target goal among negative goals, and `vlm-sor`, which thresholds that probability into a binary success reward. Appendix D.8 provides additional details for these methods.

Code methods prompt an LLM to generate executable Python code for a scoring function with state-action pairs as input. The code is generated once and then executed to produce predicted scores for all data-points. The family contains three methods that differ in how code is generated. `codegen`, our adaptation of Auto MC-Reward (Li et al., 2024), generates k candidate functions independently (reporting per-function correlation), and `codegen-avg` averages their predictions first, and then reports correlation. `eureka` (Ma et al., 2023b) instead refines the function iteratively: across several rounds, an LLM judge scores the previous round's candidates and the best is fed back as the seed for the next. Appendix D.5 provides additional details for these methods.

Most methods output one score per data point. However, some methods instead output only a permutation over the candidate actions for a state (these are `ranking`, Δ`belief`, `sdpo`, and `sdpo-gt`). For these methods, we compute the rank correlation between the predicted permutation and the label-induced permutation within each state, then average across states. In our figures, we separate these two categories of methods wherever applicable; we refer to the metric employed for the earlier group as *global Spearman*, and for the latter as *state-local Spearman*.

3.3 Experimental Details

Backbones. We fix the LLM/VLM backbone across methods wherever possible, so that differences in value-alignment reflect the scoring methods themselves rather than the underlying models' performance. We evaluate the Qwen3.5 family (Qwen Team, 2026) at 9B, 27B, 35B-A3B, and 122B-A10B parameter scales, and the Gemma 4 family (Google DeepMind, 2026) at 26B-A4B and 31B parameter scales. The `vip` and `liv` methods use their corresponding pre-trained models. The `vlm-rm` and `vlm-sor` methods use CLIP (Radford et al., 2021) and SigLIP (Zhai et al., 2023).

Contextual information. To ensure a fair comparison, we give all methods the same context: a high-level task description, the same environment dynamics, a textual reward specification, and descriptions of the state and action spaces. Appendix D provides full details.

4 Results

Using QVAL-v1.0, we evaluate all aforementioned methods with Qwen3.5 and Gemma 4 models on our multi-environment datasets. We additionally complement these results by ablations that probe the robustness of our conclusions and the effect of observation modality and signal type.

4.1 Main Results

Simple methods align best. Figure 2 shows that ranking and direct prompting achieve the highest degree of value-alignment across environments and backbones, consistently outperforming the other families. Direct value prediction is thus a surprisingly strong baseline for dense supervision. Methods

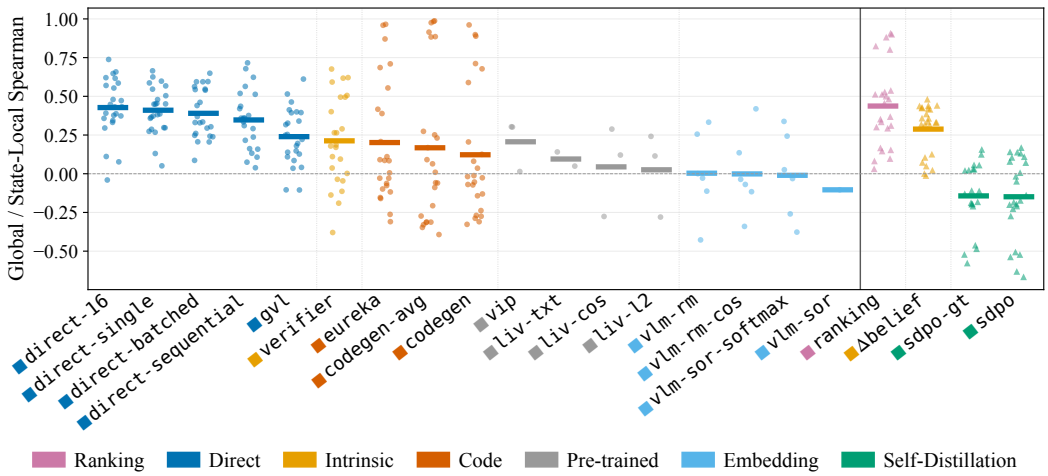


Figure 2: **Distribution of Spearman correlations by dense supervision method.** Each point is one environment-model evaluation pair; horizontal bars show means across evaluations. The two groups separated by the vertical divider report different metrics: on the left, global Spearman correlations, and on the right, per-state Spearman correlations averaged across states. Ranking and direct-prompting methods align best with reference policy values on average. Methods in the same family show similar value-alignment patterns, supporting our method taxonomy.

also cluster clearly by family, obtaining similar correlation ranges within each, which suggests our taxonomy captures real differences in the signal each family extracts. Code-based methods are the exception, with the largest variance (Figure 3), as their effectiveness depends heavily on the complexity of the state and action spaces and on how readily those can be captured in code.

Complexity does not help. Within a family, more elaborate variants do not reliably improve value-alignment (Figure 2). In the direct family, the multi-estimate and batched/sequential variants do not clearly beat the simpler `direct-single`. In the code family, averaging over generated functions (`codegen-avg`) lifts the mean correlation over `codegen` slightly but still leaves substantial variance. In self-distillation, giving the teacher privileged target-policy information (`sdpo-gt`) does not improve over `sdpo`. These results highlight the value of measuring signal quality directly: QVAL reveals whether added complexity translates into a better dense feedback signal.

Difficulty does not predict performance. Figure 3 reports correlations per environment, ordered roughly from simpler closed-action settings (FrozenLake) to open-ended ones (TerminalBench). Value-alignment does not decline monotonically with task difficulty. Direct-prompting methods stay positive everywhere, including TerminalBench, while other families behave differently for specific environments rather than following difficulty. Code and ranking methods weaken in open-ended environments like TerminalBench. In contrast, self-distillation shows stronger on TerminalBench, possibly because it benefits from richer intermediate feedback. A method's value-alignment thus depends on the interaction with the environment's unique characteristics.

4.2 Robustness Analyses

Input modality. We compare value-alignment under text and image representations of the same state, on the environments that provide both. The results shown in Figure 4 (left) indicate that the evaluated methods recover reference values more reliably from text than from images. This suggests that, in our setting, parsing visual information is more challenging.

Reference value type. QVAL provides state-value labels $V(s_t)$ alongside Q-values for OpenApps, ALFWorld, and FrozenLake, letting us test whether method rankings depend on the choice of target value. Figure 4 (right) compares the two across the direct, code, and pre-trained families, averaged over models and environments. The relative ordering of methods is largely preserved, so our conclusions do not rely on a particular value target. Absolute correlations do change: code and pre-trained methods align better with state values, direct prompting with Q-values. This likely reflects differences in how methods consume the input: code-based methods may more naturally express

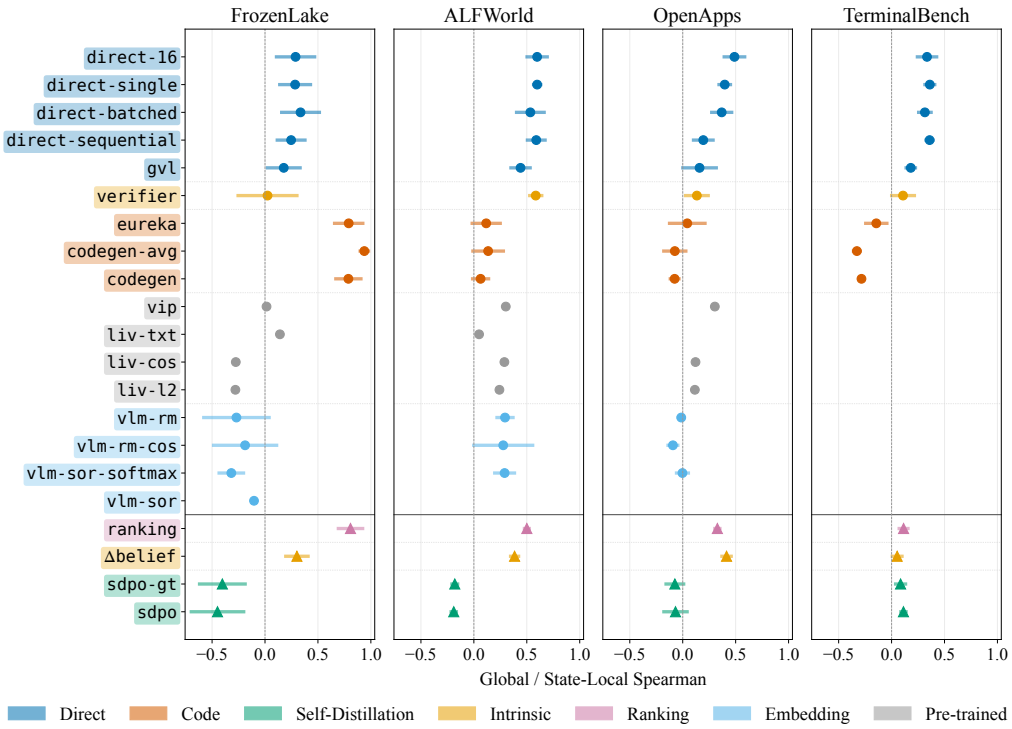


Figure 3: **Per-method Spearman correlation by environment.** Correlations are computed against reference values and averaged across model backbones within each environment; error bars show 95% confidence intervals. Visual methods are not evaluated on TerminalBench. The two groups separated by the horizontal divider report different metrics: on the top, global Spearman correlations, and on the bottom, per-state Spearman correlations averaged across states. Signal quality does not degrade uniformly with task complexity: direct-prompting methods remain consistently positive across environments, while other method families show stronger environment-specific behaviour.

state-level heuristics as executable functions, while direct-prompting can explicitly prompt for a target action. Nevertheless, the broad trends remain consistent across targets.

MVMC backbones. We evaluate whether the choice of backbone used for Max-Value Monte-Carlo rollout collection affects the TerminalBench labels. Figure 5 compares labels estimated using GPT-5.5 (OpenAI, 2026) and Claude Opus 4.7 (Anthropic, 2026). Reference values from the two models result in closely matching method correlations, with methods with positive correlations under one backbone obtaining similar positive correlations under the other, and viceversa. This shows that our TerminalBench results are robust across two frontier models with independent training recipes.

4.3 Discussion

Direct prompting as a strong baseline. Our results suggest that progress in dense feedback methods is conflated with evaluation recipes. When methods are compared directly through value alignment, simple direct value prediction is surprisingly competitive, outperforming more specialized mechanisms. This does not imply that complex dense-feedback methods are unnecessary, but rather shows that current improvements over sparse rewards or end-to-end training baselines also reflect changes in data, compute, exploration, prompting, or optimization rather than a better dense signal.

Information across modalities. At the same time, our results must be interpreted through the information available to each method. Text-based prompting often receives compact state and task abstractions, whereas vision-only methods only receive pixels and a goal specification, making them more versatile but less performant. Their weaker alignment therefore does not show that visual feedback is intrinsically inferior; rather, it highlights that value estimation often requires the right abstraction, especially when progress depends on symbolic, relational, or hidden state information.

Overall, QVAL suggests that new dense supervision methods should justify their added complexity by improving the underlying signal, not just downstream performance. By design, it measures signal

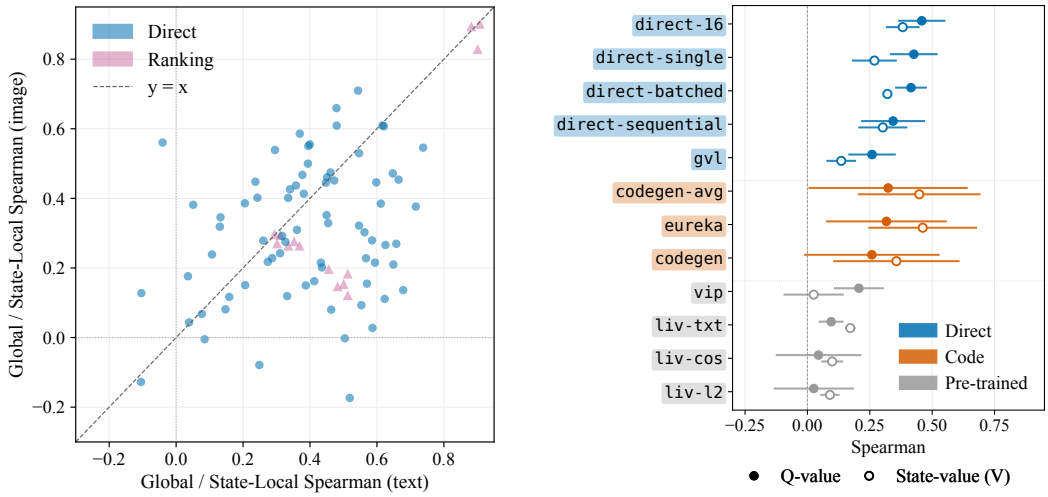


Figure 4: (Left) **Text vs. image observation.** Each point is an evaluation of method, environment, and model combinations. Points above the diagonal benefit from visual input; points below are hurt by it. (Right) **Q-value (filled points) vs. state-value (shallow points).** Signal type averaged over models & environments (OpenApps, ALFWorld, FrozenLake). Points show means; bars show ±1 SE.

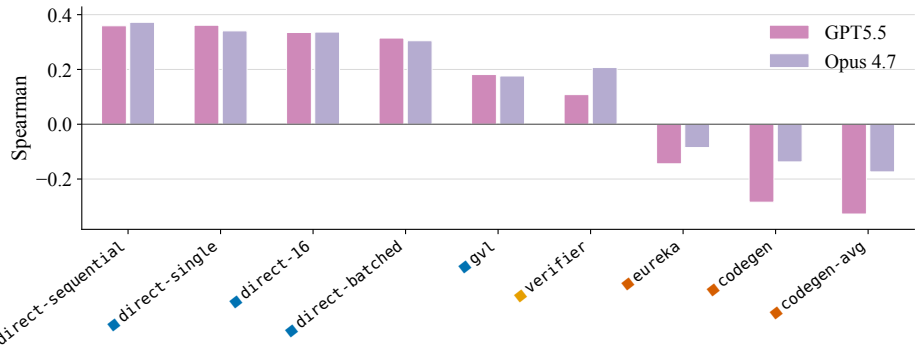


Figure 5: **Backbone ablation for TerminalBench value labels.** We compare method correlations under different reference policies: GPT-5.5 or Claude Opus 4.7, both using MVMC. The relative ordering of methods is largely preserved across the labelling backbones, suggesting that the TerminalBench labels capture a stable notion of downstream task progress under strong model policies.

quality on its own and leaves integration as a separate downstream question: a high score does not guarantee training gains, but a low score is strong evidence against a method's viability under the assumed reference policy. QVAL is thus a cheap diagnostic that filters candidate signals based on value alignment before expensive training runs, rather than replacing them.

5 Conclusion

We introduced QVAL, a training-free methodology that evaluates dense supervision methods by their *value-alignment*, how well their scores rank an agent's intermediate actions according to reference values, turning a question that once required a training run into a cheap evaluation. We instantiated it as QVAL-v1.0, benchmarking 21 methods spanning seven families from the literature across four environments and six open-weight backbones. We found that simple direct prompting provides the strongest signal and that methods cluster reliably by family, robustly across model sizes, environments, modalities, and target types. QVAL is built to grow: we will keep extending it with new state-of-the-art environments and domains, new methods plug in by emitting a single score per state-action pair, and practitioners can apply our methodology to build datasets for their own tasks. We hope QVAL supports faster, cheaper iteration on the dense signals needed to train long-horizon agents.

References

Anthropic. Claude Models Overview. https://platform.claude.com/docs/en/about-claude/models/overview, 2026.

Ilze Amanda Auzina, Joschka Strüber, Sergio Hernández-Gutiérrez, Shashwat Goel, Ameya Prabhu, and Matthias Bethge. Intrinsic credit assignment for long horizon interaction. In *ICLR 2026 Workshop on Lifelong Agents: Learning, Aligning, Evolving*, 2026. URL https://openreview.net/forum?id=7lD91CE4IU.

Kate Baumli, Satinder Baveja, Feryal Behbahani, Harris Chan, Gheorghe Comanici, Sebastian Flennerhag, Maxime Gazeau, Kristian Holsheimer, Dan Horgan, Michael Laskin, et al. Vision-language models as a source of rewards. *arXiv preprint arXiv:2312.09187*, 2023. URL https://arxiv.org/abs/2312.09187.

Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5): 679–684, 1957. ISSN 0095-9057. URL https://www.jstor.org/stable/24900506.

Paweł Budzianowski, Emilia Wiśnios, Michał Tyrolski, Gracjan Góral, Igor Kulakov, Viktor Petrenko, and Krzysztof Walas. OpenGVL – Benchmarking Visual Temporal Progress for Data Curation, February 2026. URL http://arxiv.org/abs/2509.17321. arXiv:2509.17321 [cs].

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, et al. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Google DeepMind. Gemma 4 model card, April 2026. URL https://ai.google.dev/gemma/docs/core/model_card_4.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement Learning via Self-Distillation, January 2026. URL http://arxiv.org/abs/2601.20802. arXiv:2601.20802 [cs].

M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938. ISSN 0006-3444. doi: 10.2307/2332226. URL https://www.jstor.org/stable/2332226.

M. G. Kendall. The Treatment of Ties in Ranking Problems. *Biometrika*, 33(3):239–251, November 1945. ISSN 0006-3444. doi: 10.1093/biomet/33.3.239. URL https://doi.org/10.1093/biomet/33.3.239.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. URL https://arxiv.org/abs/1712.05474.

Divake Kumar, Sina Tayebati, Devashri Naik, Ranganath Krishnan, and Amit Ranjan Trivedi. VLM Judges Can Rank but Cannot Score: Task-Dependent Uncertainty in Multimodal Evaluation, April 2026. URL http://arxiv.org/abs/2604.25235. arXiv:2604.25235 [cs.LG].

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. Llm-as-a-verifier: A general-purpose verification framework, 2026. URL https://llm-as-a-verifier.notion.site. Notion Blog.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, L. J. Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating Reward Models for Language Modeling, June 2024. URL http://arxiv.org/abs/2403.13787. arXiv:2403.13787 [cs].

Hao Li, Xue Yang, Zhaokai Wang, Xizhou Zhu, Jie Zhou, Yu Qiao, Xiaogang Wang, Hongsheng Li, Lewei Lu, and Jifeng Dai. Auto MC-Reward: Automated Dense Reward Design with Large Language Models for Minecraft, March 2024. URL http://arxiv.org/abs/2312.09238. arXiv:2312.09238 [cs].

Lei Li, Yuancheng Wei, Zhihui Xie, Xuqing Yang, Yifan Song, Peiyi Wang, Chenxin An, Tianyu Liu, Sujian Li, Bill Yuchen Lin, Lingpeng Kong, and Qi Liu. VL-RewardBench: A Challenging Benchmark for Vision-Language Generative Reward Models, June 2025. URL http://arxiv.org/abs/2411.17451. arXiv:2411.17451 [cs].

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. CriticBench: Benchmarking LLMs for Critique-Correct Reasoning, June 2024. URL http://arxiv.org/abs/2402.14809. arXiv:2402.14809 [cs].

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG Evaluation using Gpt-4 with Better Human Alignment. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.153. URL https://aclanthology.org/2023.emnlp-main.153/.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. RM-Bench: Benchmarking Reward Models of Language Models with Subtlety and Style, October 2024. URL http://arxiv.org/abs/2410.16184. arXiv:2410.16184 [cs].

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. AgentRewardBench: Evaluating Automatic Evaluations of Web Agent Trajectories, October 2025. URL http://arxiv.org/abs/2504.08942. arXiv:2504.08942 [cs].

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*, 2022. URL https://arxiv.org/abs/2210.00030.

Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. Liv: Language-image representations and rewards for robotic control. In *International Conference on Machine Learning*, pages 23301–23320. PMLR, 2023a. URL https://dl.acm.org/doi/10.5555/3618408.3619378.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-Level Reward Design via Coding Large Language Models. October 2023b. URL https://openreview.net/forum?id=IEduRUO55F.

Yecheng Jason Ma, Joey Hejna, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, Osbert Bastani, Dinesh Jayaraman, Wenhao Yu, Tingnan Zhang, Dorsa Sadigh, and Fei Xia. Vision Language Models are In-Context Value Learners. October 2024. URL https://openreview.net/forum?id=friHAl5ofG.

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah A. Smith, Hannaneh Hajishirzi, and Nathan Lambert. RewardBench 2: Advancing Reward Model Evaluation, June 2025. URL http://arxiv.org/abs/2506.01937. arXiv:2506.01937 [cs].

Mike A. Merrill, Alexander Glenn 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, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Kwesi Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt.

Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces. October 2025. URL https://openreview.net/forum?id=a7Qa4CcHak&referrer=%5Bthe%20profile%20of%20Alex%20Dimakis%5D(%2Fprofile%3Fid%3D~Alex_Dimakis1).

OpenAI. GPT-5.5 System Card, April 2026. URL https://deploymentsafety.openai.com/gpt-5-5.

Bespoke Labs OpenThoughts-Agent team, Snorkel AI. OpenThoughts-TBLite: A High-Signal Benchmark for Iterating on Terminal Agents. https://www.openthoughts.ai/blog/openthoughts-tblite, February 2026.

Karl Pearson. VII. Mathematical contributions to the theory of evolution.—III. Regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London, Series A: Containing Papers of a Mathematical or Physical Character*, (187):253–318, December 1896. ISSN 0264-3952. doi: 10.1098/rsta.1896.0007. URL https://doi.org/10.1098/rsta.1896.0007.

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

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8748–8763. PMLR, July 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=N0I2RtD8je.

Simon Roy, Samuel Barbeau, Giovanni Beltrame, Christian Desrosiers, and Nicolas Thome. Revisiting the Learning Objectives of Vision-Language Reward Models, December 2025. URL http://arxiv.org/abs/2512.20675. arXiv:2512.20675 [cs].

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

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-Distillation Enables Continual Learning, January 2026. URL http://arxiv.org/abs/2601.19897. arXiv:2601.19897 [cs].

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. October 2020. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Nishad Singhi, Hritik Bansal, Arian Hosseini, Aditya Grover, Kai-Wei Chang, Marcus Rohrbach, and Anna Rohrbach. When To Solve, When To Verify: Compute-Optimal Problem Solving and Generative Verification for LLM Reasoning. In *Conference on Language Modeling (COLM)*, October 2025. doi: 10.48550/arXiv.2504.01005. URL http://arxiv.org/abs/2504.01005.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. PRMBench: A Fine-grained and Challenging Benchmark for Process-Level Reward Models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25299–25346, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1230. URL https://aclanthology.org/2025.acl-long.1230/.

Yuda Song, Lili Chen, Fahim Tajwar, Remi Munos, Deepak Pathak, J. Andrew Bagnell, Aarti Singh, and Andrea Zanette. Expanding the Capabilities of Reinforcement Learning via Text Feedback, February 2026. URL http://arxiv.org/abs/2602.02482. arXiv:2602.02482 [cs].

C. Spearman. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, 15(1):72, January 1904. ISSN 00029556. doi: 10.2307/1412159. URL https://www.jstor.org/stable/1412159?origin=crossref.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, Cambridge, MA, USA, 2 edition, November 2018. ISBN 978-0-262-03924-6. URL https://mitpress.mit.edu/9780262039246/reinforcement-learning/.

Mark Towers, Ariel Kwiatkowski, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Kallinteris Andreas, Markus Krimmel, Arjun Kg, Rodrigo De Lazcano Perez-Vicente, J. K. Terry, Andrea Pierré, Sander V. Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A Standard Interface for Reinforcement Learning Environments. October 2025. URL https://openreview.net/forum?id=qPMLvJxtPK&referrer=%5Bthe%20profile%20of%20John%20U.%20Balis%5D(%2Fprofile%3Fid%3D~John_U._Balis1).

Karen Ullrich, Jingtong Su, Claudia Shi, Arjun Subramonian, Amir Bar, Ivan Evtimov, Nikolaos Tsilivis, Randall Balestriero, Julia Kempe, and Mark Ibrahim. OpenApps: Simulating Environment Variations to Measure UI Agent Reliability. October 2025. URL https://openreview.net/forum?id=cj1MAx7lKs.

Yutao Xie, Nathaniel Thomas, Nicklas Hansen, Yang Fu, Li Erran Li, and Xiaolong Wang. Tips: Turn-level information-potential reward shaping for search-augmented llms. *arXiv preprint arXiv:2603.22293*, 2026.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-Evaluation Guided Beam Search for Reasoning. *Advances in Neural Information Processing Systems*, 36:41618–41650, December 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/81fde95c4dc79188a69ce5b24d63010b-Abstract-Conference.html.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023. URL https://arxiv.org/abs/2310.11441.

Michihiro Yasunaga, Luke Zettlemoyer, and Marjan Ghazvininejad. Multimodal RewardBench: Holistic Evaluation of Reward Models for Vision Language Models, February 2025. URL http://arxiv.org/abs/2502.14191. arXiv:2502.14191 [cs].

Eunseop Yoon, Hee Suk Yoon, Jaehyun Jang, SooHwan Eom, Qi Dai, Chong Luo, Mark A. Hasegawa-Johnson, and Chang D. Yoo. PACR: Progressively ascending confidence reward for LLM reasoning, 2026. URL https://openreview.net/forum?id=jKAqtb63Bl.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid Loss for Language Image Pre-Training. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11941–11952, Paris, France, October 2023. IEEE. ISBN 979-8-3503-0718-4. doi: 10.1109/ICCV51070.2023.01100. URL https://ieeexplore.ieee.org/document/10377550/.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: Identifying Process Errors in Mathematical Reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1009–1024, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.50. URL https://aclanthology.org/2025.acl-long.50/.

Part I
Appendix

We now provide thorough details about the environment, metrics, methods, and model configuration, along with complete results, which were summarized in tables.

Contents

A Related Works **15**

B Environment Details **16**

B.1 TerminalBench 16
B.2 OpenApps 17
B.3 ALFWorld 18
B.4 FrozenLake 19

C Evaluation Metrics **20**

D Method Details **21**

D.1 Shared method context 21
D.2 Direct prediction family 21
D.3 LLM-as-a-Verifier (`verifier`) 21
D.4 ΔBelief (Δ`belief`) 22
D.5 Code-generation family 22
D.6 Self-Distillation 22
D.7 Ranking (`ranking`) 23
D.8 Embedding-similarity (VLE) methods 23
D.9 Pre-trained value methods 24
D.10 Hyperparameter summary 24
D.11 Prompt templates 24

E Model Details **34**

F Complete Results **34**

A Related Works

Evaluation through training. Most dense feedback methods are evaluated by using the proposed signal inside a downstream training or selection pipeline and reporting task return, or pass-rate improvements. For instance, self-evaluation signals that guide reasoning search are typically validated only by final task accuracy (Xie et al., 2023), and generative verifiers are assessed by downstream selection accuracy under a fixed inference budget (Singhi et al., 2025). Although this demonstrates end-to-end utility, it makes signal quality hard to isolate: measured gains depend on the policy model, optimizer, exploration process, environment distribution, amount of generated data, and implementation details of the training loop. Some work evaluates progress or reward-model quality more directly in restricted settings, especially in robotics and visual progress estimation (Ma et al., 2024; Budzianowski et al., 2026; Roy et al., 2025). QVAL is complementary: as a training-free testbed, rather than asking whether a full training recipe improves, it fixes datasets, model backbones, and environment context, then directly measures whether a proposed dense signal orders states or actions consistently with value labels derived from reference continuations.

Reward and critic benchmarks. There is also a growing literature on evaluating reward models and critic models for language and vision-language systems. RewardBench, RM-Bench, and RewardBench 2 evaluate reward models with response-comparison or accuracy-based tasks spanning chat, reasoning, safety, subtle errors, and style biases (Lambert et al., 2024; Liu et al., 2024; Malik et al., 2025). VL-RewardBench and Multimodal RewardBench extend this style of evaluation to multimodal reward models and VLM judges (Li et al., 2025; Yasunaga et al., 2025). ProcessBench and PRMBench evaluate models' ability to identify erroneous reasoning steps (Zheng et al., 2025; Song et al., 2025), CriticBench evaluates models' ability to critique and correct solutions (Lin et al., 2024), while AgentRewardBench studies automatic evaluation of complete web-agent trajectories (Lù et al., 2025). These benchmarks are closely related in spirit, but they primarily assess final responses, pairwise preferences, critiques, step-level correctness labels, or whole-trajectory judgments. QVAL instead provides a testbed for the intermediate signal needed for multi-turn agent training, measuring per-state and per-action alignment with value labels in interactive environments.

B Environment Details

This Appendix details the four environments comprising QVAL at release. For each, we describe the source, observation and action spaces, reward function, episode horizon, and the parameters used for trajectory collection and ground-truth labelling. Table 2 summarises the per-environment configuration; Figure 6 shows a representative state for each environment.

Table 2: QVAL environment summary. *Eval pts.* and *Rank. pts.* are the number of (s, a, s') triples and ranking points retained after filtering and used in the experiments of Section 4. k is the maximum number of candidate actions per ranking point. Sampling sources: *LLM* = candidate actions sampled from the collection actor at higher temperature; *manual* = enumerated by an environment-specific sampler. Citations for each environment appear in the corresponding subsection.

	TerminalBench	**OpenApps**	**ALFWorld**	**FrozenLake**
Source	TBLite (easy)	BrowserGym suite	ALFWorld OOD	Gymnasium 8×8
Modalities	text	text, image	text, image	text, image
# distinct tasks	19	8	24 scenes (1 type)	8 maps
# trajectories	118	40	40	50
# eval points	100	94	100	100
# ranking points	100	94	100	100
Max ranking k	4 (LLM)	4 (manual)	4 (LLM)	4 (manual)
Episode horizon	40 steps	45 steps	40 steps	30 steps
Collection actor	DeepSeek v3.2	scripted (ε=0.25)	DeepSeek v3.2	scripted (ε=0.1)
Reference policy	GPT-5.5, k=16 MC	scripted optimal	expert planner	scripted optimal

B.1 TerminalBench

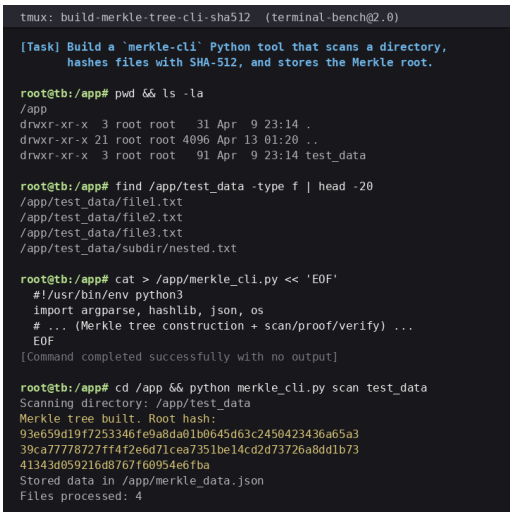
We use the *easy* difficulty split of TBLite (OpenThoughts-Agent team, 2026), a curated subset of TerminalBench (Merrill et al., 2025) designed for iteration on terminal agents. We specifically choose TBLite over the full TerminalBench as we find many tasks to be unsolvable even by the best current models, mostly due to timeouts cancelling long operations (e.g., package installs in R). The split contains 19 tasks spanning system administration, cryptography, log processing, data engineering, and ML pipeline tasks. The 19 tasks are: `amuse-install`, `broken-python`, `build-merkle-tree-cli-sha512`, `convolutional-layers`, `cosign-keyless-signing`, `cryptographic-protocol-verifier`, `hydra-debug-slurm-mode`, `image-tile-identification`, `jq-data-processing`, `jsonl-aggregator`, `log-summary`, `mlflow-register`, `pandas-etl`, `playing-card-recognition`, `protein-sequence`, `raft-log-repair-concurrent-access`, `schedule-vacation`, `smiles-data-lab`, `systemd-log-monitoring`.

Observation/Action. The agent operates a persistent `tmux`-backed shell inside an Apptainer container sandbox for safety. Observations are the trailing terminal scrollback after each command; valid actions are any valid shell command (in text) wrapped in `<action>` tags. Shell state (current directory, exported variables, background processes) persists across commands within a trajectory.

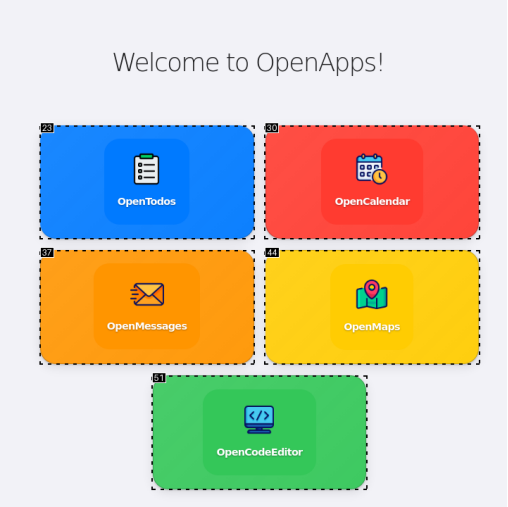
Reward. Binary verifier output: 1.0 if the task verifier passes on submission or step truncation, 0.0 otherwise.

Collection. Trajectories are collected with DeepSeek v3.2 (DeepSeek-AI et al., 2025) as the actor, with a maximum of 40 steps per episode, 180 s per command, and a 900 s wall-clock cap per trajectory. We collect 118 successful trajectories across the 19 tasks, retain one (s, a, s') triple per trajectory uniformly at random (excluding the first and last 3 turns to avoid trivially-early and policy-saturated states), and sample 4 ranking-candidate actions per state from the same actor at elevated temperature (T=1.2, top-p=0.9).

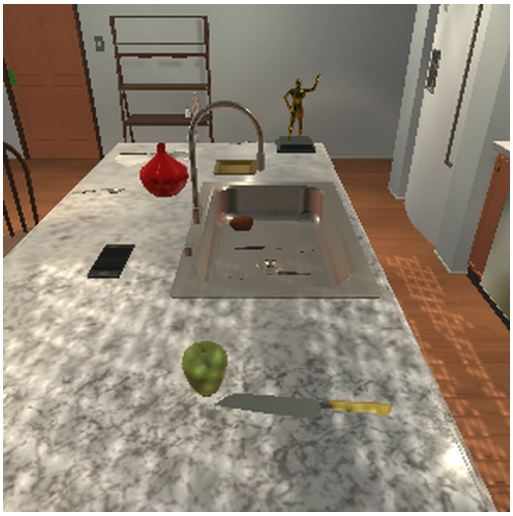
Reference policy. For TerminalBench, an optimal policy would solve every task using a single, most often convoluted command; approximating such a policy would not be desirable for practical applications, as it would result in actions which are harder to interpret, less human-aligned, more error-prone, and could yield trained models that would lack the compositional skills to generalize



(a) TerminalBench



(b) OpenApps



(c) ALFWorld

(d) FrozenLake

Figure 6: Representative state observations for each environment. TerminalBench is text-only and the image is for presentation only; the remaining three environments support either text or image observations, with the rendered image shown.

to longer and more complex terminal-based tasks. Therefore, we choose to employ a Max-Value Monte Carlo approach: using a more desirable reference policy, we complete k trajectories from the given state and estimate the value of the data point as the highest discounted cumulative reward attained by any trajectory. In order to ensure the high quality of our labels, we constrain the reference policy to solve every task in the dataset in at most k attempts, where k is the number of rollouts sampled via Monte Carlo. For TerminalBench, we use GPT-5.5 (OpenAI, 2026) and sample $k = 16$ rollouts. We observe that GPT-5.5 obtains a Pass@16 of 100% on our subset of TerminalBench tasks. Additionally, we further validate our reference policy by showing that different models serving as the Monte Carlo backbones do not alter the results of our benchmark in Section 4.2.

B.2 OpenApps

OpenApps (Ullrich et al., 2025) is a suite of synthetic web applications (calendar, todo, messenger, map, code editor) built with FastHTML and exposed through BrowserGym. The 8 tasks are: `add_meeting_with_dennis` (calendar), `add_christmas_shopping_event`

(calendar), add_paper_reading_meeting_with_einstein (calendar), remove_wacv_abstract_deadline (calendar), add_call_mom_to_my_todo (todo), mark_water_plants_as_done (todo), message_bob_to_meet (messenger), save_paris_to_my_favorite_places (map).

Observation/Action. Each step the agent sees either an accessibility-tree (AXTree) text representation of the page or a 1280×720 screenshot with Set-of-Marks bid annotations drawn on visible interactive elements (Yang et al., 2023). Actions are BrowserGym primitives: `click(bid)`, `fill(bid, text)`, `select_option(bid, option)`, `scroll(x, y)`, `press(key)`, `hover(bid)`, `go_back()`, and `noop()`.

Reward. The OpenApps server checks the application state against the task-specific target after every step and returns 1.0 once the goal state is reached, 0.0 otherwise.

Collection. The collection actor is a hand-crafted scripted policy that consults the AXTree to perform each task; we mix in random BrowserGym actions with ε=0.25 to produce a balanced mixture of successful and sub-optimal trajectories. We collect 40 trajectories (5 per task, uniform over the 8 tasks), with max_steps=45 and a 180 s BrowserGym call timeout. We sample up to 3 evaluation points per trajectory uniformly, drop the first and last turn, and retain 94 (s, a, s') triples and 94 ranking points. The 4 ranking candidates per state are enumerated deterministically from the live AXTree by a sampler.

Reference policy. The same scripted policy without ε-noise serves as the optimal reference. Because the policy is deterministic and the environment is deterministic, we use a single rollout per state.

B.3 ALFWorld

ALFWorld (Shridhar et al., 2020) aligns the embodied THOR simulator (Kolve et al., 2017) with a TextWorld interface, exposing household tasks through both natural-language observations and rendered images of the agent's egocentric view.

Task scope. ALFWorld defines six task types (`pick_and_place`, `cool_and_place`, `heat_and_place`, `clean_and_place`, `examine_and_place`, `pick_two_and_place`). Producing reliable expert-policy ground-truth under the visual modality requires a planner that operates on the underlying simulator state, provided by ALFWorld. Empirically, we find the planner to be robust on `pick_and_place` but to degrade on the other five types when paired with the visual rendering pipeline; to keep ground-truth quality uniformly high we therefore restrict our QVAL dataset to `pick_and_place_simple`. Nevertheless, task diversity within this single type is substantial: the dataset spans 24 distinct game files drawn from 4 distinct rooms (kitchen, office, bedroom-vault, bathroom) with 8 distinct (target object, destination receptacle) pairings, including kitchen utensils placed in drawers and cabinets, writing implements on desks and shelves, valuables in safes, and toiletries in bathroom fixtures.

Observation/Action. Text observations include the current scene description and a list of admissible commands (navigation, object pickup, placement, container open/close, object-state changes such as cleaning). In the visual modality the agent sees the egocentric THOR render instead of the text environment description. Actions are the raw text commands wrapped in `<action>` tags; the adapter constrains action selection to the admissible-command set provided by the simulator.

Reward. Binary outcome reward: 1.0 when the simulator declares the task goal reached, 0.0 otherwise.

Collection. We collect 40 trajectories with DeepSeek v3.2, max_steps=40, sampling up to 5 evaluation points per trajectory uniformly with the first turn and last 2 turns excluded. After ranking-pruning (we keep only states with ≥ 3 valid alternative actions for ranking), the dataset retains 106 evaluation points and 106 ranking points; the experiments use the first 100. Ranking candidates (k=4) are sampled from the same actor at T=0.2.

Reference policy. We use ALFWorld's handcoded expert planner that consults the simulator's PDDL state to issue an optimal action sequence. The planner is deterministic, so a single rollout per state suffices. We additionally validated this reference against DeepSeek-v3.2 Max-Value MC (k=32): the two agree at Spearman 0.83 on the same evaluation points, supporting the use of the cheaper expert as canonical ground truth.

B.4 FrozenLake

We use the 8×8 FrozenLake variant from Gymnasium (Towers et al., 2025) with deterministic dynamics (`is_slippery`=False). To test agents on more than a single fixed layout, we generate 8 distinct random maps with seed 42 and cycle through them across trajectories, so each map serves as one "task".

Observation/Action. Text observations are an 8×8 ASCII grid in which the agent's current cell is marked with `@`, holes with `H`, and the goal with `G`; the visual modality instead provides a 512×512 pygame render of the same grid. The action space is four discrete moves: `left, down, right, up`; off-grid moves leave the agent in place.

Reward. Sparse outcome reward: 1.0 for reaching the goal, 0.0 for falling into a hole or exhausting the step budget.

Collection. The collection actor is a scripted shortest-path policy mixed with ε=0.1 uniform-random noise. ε=0.1 produces a $\sim 34\%$ trajectory success rate on 8×8, sufficient to obtain a balanced mixture of successful and failed trajectories under the 30-step horizon. We collect 50 trajectories and sample up to 5 points per trajectory, yielding 239 evaluation points; the first 100 are used for experiments (matching the prefix used by the other environments). Ranking candidates (k=4) are the four discrete actions.

Reference policy. A scripted optimal policy that follows the shortest path to the goal serves as the reference. The environment is deterministic and so is the policy; a single rollout per state is exact.

C Evaluation Metrics

The performance scores we provide for QVAL are correlation-based. For a dataset with N data points, let y_i denote the label associated with point i (either a state-value or a Q-value as obtained in Section 3) and $\hat{y}_i$ the value predicted by an evaluated method. We assess methods by the alignment of $(\hat{y}_i)_{i=1}^N$ with $(y_i)_{i=1}^N$ rather than by absolute error: predictions on different scales (e.g. raw LLM scores, code-generated functions, or token log-probabilities) cannot be compared on a common loss, and the practical use of a dense signal in RL depends on the *ordering* the signal induces, not its numerical scale. We therefore restrict ourselves to rank-based correlation metrics.

Let $r_i = \mathrm{rank}(y_i)$ and $\hat{r}_i = \mathrm{rank}(\hat{y}_i)$ denote the ranks of y_i and $\hat{y}_i$ within their respective sequences, with ties broken by averaging.

Spearman's ρ. Spearman's correlation (Spearman, 1904) is the Pearson correlation (Pearson, 1896) between the ranks:

$$\rho \;=\; \frac{\sum_{i=1}^N (r_i - \bar{r})(\hat{r}_i - \bar{\hat{r}})}{\sqrt{\sum_{i=1}^N (r_i - \bar{r})^2 \;\cdot\; \sum_{i=1}^N (\hat{r}_i - \bar{\hat{r}})^2}}, \tag{3}$$

where $\bar{r}$ and $\bar{\hat{r}}$ are the mean ranks. Equivalently, ρ is invariant under any monotonic transformation of either variable, and reduces to a sum of squared rank differences when no ties are present.

Kendall's τ. Kendall's correlation (Kendall, 1938, 1945) instead counts agreements between pairs of points. A pair (i, j) with $i \neq j$ is *concordant* when $(\hat{y}_i - \hat{y}_j)(y_i - y_j) > 0$, *discordant* when this product is negative, and tied otherwise. With C concordant and D discordant pairs out of the $\binom{N}{2}$ unordered pairs,

$$\tau \;=\; \frac{C - D}{\sqrt{(C + D + T_y)\,(C + D + T_{\hat{y}})}}, \tag{4}$$

where T_y and $T_{\hat{y}}$ are the numbers of pairs tied on the labels or on the predictions, respectively.

Comparison. Both metrics take values in $[-1, 1]$: 1 denotes perfect ordinal agreement, -1 a complete inversion, and 0 no monotonic association. They differ in what they penalise. Spearman aggregates squared rank differences, so a pair that is ordered very wrong (e.g., the largest label paired with the smallest prediction) dominates the score; Kendall counts each pairwise inversion with equal weight regardless of how far apart the points are in the ranking, making it more robust to extreme outliers but typically yielding smaller absolute values than ρ on the same data. We report both: Spearman is more sensitive to localised errors at the tails of the value distribution, while Kendall provides a stricter, scale-free reading of monotonic agreement.

If a method fails to produce a value at the point i (e.g., extraction failure for direct prediction, or a runtime error in code generation), we record $\hat{y}_i = \mathrm{NaN}$ and drop the pair $(y_i, \hat{y}_i)$ before computing either correlation, so the score reflects only the points the method actually attempted. Nonetheless, for every experiment, we ensure that all results we report contain a sufficient number of non-NaN points for high statistical significance.

Per-state ranking score. The ranking methods, `sdpo` and `ranking`, do not emit a value per point but a permutation σ_i over the K_i candidate actions $\{a_i^{(1)}, \ldots, a_i^{(K_i)}\}$ at each state s_i. We score it within each state and average across states. Let $\sigma_i^\star$ be the ranking induced by the per-candidate Q-value labels $\{y_{i,1}, \ldots, y_{i,K_i}\}$. We compute

$$\bar{\rho}_{\text{rank}} \;=\; \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \rho(\sigma_i^\star,\, \sigma_i)\,, \tag{5}$$

where $\mathcal{I}$ is the set of states with $K_i \geq 2$ and a non-degenerate label ranking (i.e. at least two distinct $y_{i,k}$ values). Each $\rho(\sigma_i^\star, \sigma_i)$ is a Spearman correlation computed independently among the candidates of state s_i, so the metric isolates within-state action discrimination from cross-state value calibration, something the global Spearman of Eq. 3 cannot do.

D Method Details

This Appendix details the parameterisation and execution of every method evaluated in Section 3.2. Section D.1 fixes the contextual information shared by every prompt-based method; Sections D.2–D.9 describe each method family; the full prompt templates are collected in Section D.11. The complete configuration files used to launch each experiment are released alongside the benchmark.

D.1 Shared method context

Every prompt-based method receives a uniform `MethodContext` describing the environment, populated with per-environment data. The context contains: a one-paragraph *task description*, a one-paragraph *reward description*, a description of the *observation/action spaces*, a serialised representation of the *state* (text or image), the most recent *action* (and the *next state* for Q-value and shaped-reward signals), and a bounded *interaction history* (up to `max_history_turns` turns, truncated from the oldest turn first when the prompt budget is exceeded). Methods that operate on visual observations use interleaves text and image content blocks at the positions where states and goals appear in the template. Encoder-only methods (Sections D.8–D.9) bypass the prompt and consume the same `state`/`goal` images directly.

D.2 Direct prediction family

All four direct variants share the same per-point prompt template (Appendix D.11); they differ only in how many points are scored per LLM call and how the calls are batched.

`direct-single`**.** One prompt per evaluation point. The model is asked for a single scalar inside an `<answer>` tag, with optional reasoning preceding it (`react_tags` extraction scheme).

`direct-batched` **(packed).** Up to four points are batched into a single prompt; the model is instructed to return four scalars in order, each inside its own `<answer_i>` tag. The grouping is fixed at `prompt_batch_size=4`. We use this variant to test whether co-presenting multiple states helps the model anchor its predictions to a common scale.

`direct-sequential`**.** Up to eight points are presented in a single prompt as an ordered turn-by-turn dialogue: each turn shows one state-action pair and asks for one scalar before moving to the next. The ordering follows the natural temporal order of the underlying trajectory when the points come from the same trajectory; otherwise it is arbitrary.

`direct-16`**.** Identical per-point prompt to `direct-single`, with k=16 independent samples drawn at the backbone's default decoding temperature. The reported scalar is the mean of the parsed responses; samples that fail to parse are dropped before averaging. We use this variant to disambiguate signal quality from sampling noise.

`gvl`**.** Our re-implementation of GVL (Ma et al., 2024) shows the model the entire trajectory in shuffled order (with no temporal markers) and asks for a per-state value scalar. We adapt the original shuffling protocol.

D.3 LLM-as-a-Verifier (`verifier`)

Logprob readout. For each (s, a, s') point the model is prompted with a yes/no question (*"Did the agent take a high-value action?"*) and we read the logprobs of the single tokens `Yes` and `No` at the answer position, normalising via log-sum-exp:

$$\text{score}(s, a) = \log p(\texttt{Yes} \mid C) - \text{lse}(\log p(\texttt{Yes} \mid C),\ \log p(\texttt{No} \mid C)).$$

This requires raw token logprobs from the backbone; backbones served via OpenRouter without logprob support are therefore omitted from the verifier results.

Criteria. Rather than a single "high-value" question, we issue one yes/no query per evaluation criterion and average their normalised scores. Criteria are environment-specific:

- **ALFWorld** (4): correctness, error detection, efficiency, precondition awareness.
- **TerminalBench** (3): correctness, efficiency, error detection.
- **OpenApps** (3): correctness, efficiency, error detection.
- **FrozenLake** (1): correctness.

Each criterion has its own one-paragraph definition surfaced inside the verifier prompt (Appendix D.11).

Prompt grouping. `prompt_grouping: random` — points are shuffled before being assigned to verifier batches, eliminating trajectory-order artifacts in the cached KV state.

D.4 ΔBelief (`Δbelief`)

We adapt ΔBelief (Auzina et al., 2026) to our setting using the same logprob readout as the verifier. The model is prompted twice per (s, a, s') point, once with the *pre* context (history up to and including state s) and once with the *post* context (history extended by a and s'), and asked the same yes/no question: *"Will the agent eventually succeed?"* The score is the change in the model's belief after observing the action's effect:

$$\text{score}(s, a, s') = \log p(\text{success} \mid \text{post}) - \log p(\text{success} \mid \text{pre}),$$

with $\log p(\text{success} \mid C) = \log p(\texttt{Yes} \mid C) - \text{lse}(\log p(\texttt{Yes} \mid C), \log p(\texttt{No} \mid C))$. The pre/post prompt template is in Appendix D.11.

D.5 Code-generation family

`codegen` **signature.** The model is asked to emit a single Python function whose name and signature depend on the signal type:

- State-value: `signal_function(state: str) -> float`.
- Q-value: `signal_function(state: str, action: str, next_state: str) -> float`.

The prompt provides the task description, reward description, and a small number of in-context state strings drawn from collected trajectories. We sample k=16 functions per (env, model) combination at the backbone's default decoding temperature, parse each via a strict signature validator, and execute them in a restricted Python sandbox. Functions that fail validation, raise during execution, or return a non-finite value contribute `NaN` for that point.

We report two views of the same 16 samples: (a) `codegen`, the per-sample correlation, and (b) `codegen-avg`, in which the predictions are averaged over the 16 functions before computing the correlation, yielding a single significance-tested cell.

We re-implement Eureka (Ma et al., 2023b) as an iterative search around the codegen prompt: `search_iterations` $= 16$ outer iterations, each generating `num_samples` $= 8$ candidate functions; in each iteration the previous-iteration functions are scored on a `judge_num_points`=8 held-out point set by an LLM judge and the best is carried into the next iteration's prompt as a refinement target. This matches the original method modulo the LLM judge: we use the same backbone for generation and judging.

D.6 Self-Distillation

Our offline SDPO (Hübotter et al., 2026) (`sdpo`) implementation scores candidate actions by the per-token logprob delta between a teacher context that sees per-candidate feedback f and a student context that does not:

$$\text{score}(s, a) = \frac{1}{|a|} \sum_t \left[\log p_{\text{teacher}}(a_t \mid s, f) - \log p_{\text{student}}(a_t \mid s) \right].$$

Both contexts are scored on the same backbone — what distinguishes teacher from student is the prompt content, not the model. The feedback f injected into the teacher prompt is a per-candidate

textual hint derived from the candidate's ground-truth Q-value: candidates whose GT Q-value exceeds the median in their state are marked as *good* candidates, the rest as *bad* candidates. SDPO is therefore a ranking-only method (it produces K scores per state, not a single scalar) and is reported in the result tables under the per-trajectory ranking metric.

`sdpo-gt` is an oracle-teacher ablation of our offline SDPO ranking method. Like `sdpo`, it scores the exact same candidate action tokens under two contexts and uses the mean teacher–student log-probability difference as the action score. The student sees only the original decision point. The teacher sees the same state plus candidate-specific evidence from cached ground-truth rollouts: the immediate next state, the next expert action after the candidate, and a compact rollout outcome summary such as success status, remaining expert steps, and final reward. The stored GT evidence is used only as teacher prompt context.

D.7 Ranking (`ranking`)

The ranking method asks the LLM to score the K candidate actions available at each ranking point in decreasing order of expected value. The prompt presents the current state, the candidate actions enumerated `1..K`, and asks for a comma-separated list of indices. We parse the last comma-separated permutation of $\{1, \ldots, K\}$ in the response; responses that fail to parse a valid permutation are dropped. The reported score is the per-state Spearman of the predicted permutation against the GT-Q permutation, averaged across states with ≥ 2 distinct GT values. Like SDPO, the ranking method does not produce a per-state scalar and is therefore reported only under the ranking metric.

D.8 Embedding-similarity (VLE) methods

Encoders. Two image-text encoders are used in all four VLE variants: **CLIP ViT-L/14** (Radford et al., 2021) and **SigLIP-base** (Zhai et al., 2023). State and goal-text inputs are encoded independently with the model's native preprocessor; cosine similarity is computed in the joint embedding space. Both encoders are frozen.

Goal-text source. For each environment we use a fixed target-goal description, identical across CLIP and SigLIP:

- **FrozenLake**: "the elf reaches the gift box in the lower right corner".
- **ALFWorld**: the per-task `Objective:` string surfaced by the simulator (e.g. "put a vase in safe").
- **OpenApps**: the per-task goal string surfaced by the BrowserGym wrapper.

`vlm-rm-cos`. Raw cosine similarity between the state embedding and the goal-text embedding.

`vlm-rm`. Goal-baseline projection following VLM-RM (Rocamonde et al., 2024). We additionally embed an environment-specific *baseline* text describing a generic, goal-free state (e.g. "a blank frozen lake grid" for FrozenLake), define the direction $\mathbf{d} = \mathbf{e}_{\text{goal}} - \mathbf{e}_{\text{baseline}}$ in embedding space, and score the state by its cosine similarity to a state-embedding shifted along that direction. The mixing coefficient is fixed at $\alpha = 0.5$.

`vlm-sor-softmax`. Continuous softmax variant of VLM-SoR (Baumli et al., 2023). The state image is scored against the target goal and a small set of negative-goal descriptions (three, per environment, e.g. for FrozenLake: "the agent fell into a hole", "an empty frozen lake", "a random unrelated scene"); the reported score is the softmax probability assigned to the target goal at temperature $\tau = 0.07$.

`vlm-sor`. The original thresholded VLM-SoR method: a state receives reward 1.0 if its softmax probability of the target goal exceeds $\beta = 0.5$, and 0.0 otherwise. For state-value tables this collapses to a binary reward; we keep both the continuous and thresholded variants in the result tables to make the loss-of-information explicit.

D.9 Pre-trained value methods

`vip`. VIP (Ma et al., 2022) with the original ResNet-50 checkpoint pre-trained on Ego4D. Unlike VLE, VIP is image-only: the goal is an *image* of the goal state. We use the `trajectory_end` setting: for each evaluation point, the goal image is the rendered final state of the GT-MC reference rollout from that state, so the goal carries the same visual statistics as the state under evaluation. Scores are negative L2 distances in the learned embedding space.

`liv-cos` / `liv-l2`. LIV (Ma et al., 2023a) in its image-goal mode, using the CLIP-RN50 LIV checkpoint pre-trained on EpicKitchens. Same `trajectory_end` goal-image source as VIP. `liv-cos` reports cosine similarity in the LIV embedding space; `liv-l2` reports negative L2 distance.

`liv-txt`. LIV in its text-goal mode: the goal is the same per-environment text used for VLE (above), encoded via LIV's text tower; cosine similarity in the joint embedding space.

Q-value evaluation. For all encoder methods, the Q-value of a state-action pair is scored on the *next state*, i.e. $\hat{Q}(s, a) = \hat{V}(s')$.

D.10 Hyperparameter summary

Table 3 lists the exact hyperparameters used for every method. Decoding parameters (temperature, top-p, max-tokens, thinking budgets) are model-specific and reported in Appendix E.

Table 3: Method-specific hyperparameters used throughout the benchmark. Sample counts are per evaluation point unless stated otherwise.

Method	Hyperparameter	Value
`direct-single`	samples / point	1
`direct-batched`	batch size	4 (packed)
`direct-sequential`	batch size	8 (sequential turns)
`direct-16`	samples / point	16 (averaged)
`gvl`	shuffled context size	full trajectory
`verifier`	criteria	env-specific (1–4)
`verifier`	prompt grouping	random
Δ`belief`	criterion	"will eventually succeed"
`codegen`	samples (functions)	16
`codegen-avg`	aggregator	mean over 16
`eureka`	search iterations	16
`eureka`	samples per iter	8
`eureka`	judge held-out points	8
`sdpo`	feedback type	per-candidate good/bad
`ranking`	parser	last permutation of $\{1..K\}$
`vlm-rm`	α	0.5
`vlm-sor-softmax`	τ	0.07
`vlm-sor`	β (threshold)	0.5
`vlm-sor*`	negative goals	3 per env
`vip`	backbone	ResNet-50 (Ego4D)
`vip`	goal source	`trajectory_end` image
`liv-cos`, `liv-l2`	backbone	CLIP-RN50 (EpicKitchens)
`liv-cos`, `liv-l2`	goal source	`trajectory_end` image
`liv-txt`	goal source	per-env target-goal text

D.11 Prompt templates

We list one representative prompt per method family. Templates use `{slot}` for runtime substitutions; literal tags such as `<system>`, `<user>`, `<assistant>`, `<answer>`, and `<score_1>` are emitted verbatim. The prompt boxes use the shared `promptlst` listings style: role tags are written literally, and placeholders are injected with `(*@\promptslot{slot}@*)`. The shared direct, GVL,

verifier, codegen, and ranking builders are taken from `src/value_bench/prompts.py`; method-specific wrappers are taken from `src/value_bench/methods/delta_belief_ranking.py`, `src/value_bench/methods/sdpo_ranking.py`, and `src/value_bench/methods/llm_eureka.py` at the release commit.

The prompt-driven methods in Table 1 are covered as follows: `direct-single`, `direct-16`, `direct-batched`, and `direct-sequential` share the direct template with different batching wrappers; `gvl`, `verifier`, Δ`belief`, `sdpo`, `sdpo-gt`, `ranking`, `codegen`, `codegen-avg`, and `eureka` have separate entries below. The pretrained and embedding methods (`vip`, `liv-*`, `vlm-*`) do not emit LLM prompt templates in this codebase.

Prompt C.1: `direct-single`, Q-value, optimal-policy assumption

```
<system>
You are an expert at estimating Q-value functions for reinforcement learning
environments.

The Q-value Q(s,a) represents the expected {return_phrase} when taking action a in state s
 and then following the optimal policy thereafter. In other words, Q(s,a) is the
expected return given that action a is taken in state s, assuming optimal play
thereafter. It depends on both the state and the specific action taken.

{approximation_guidance}

You will receive the episode's state-action history (if any), the current state, the
action taken, and the resulting next state. Estimate Q(s,a) for the Current Action in
the Current State -- the expected {return_phrase} when taking this action in this state,
assuming optimal play thereafter. The history provides context about how the agent
reached the current state.

{task_description_block}

{episode_configuration_block}

{reward_functions_block}

{example_trajectories_block}

After any reasoning, respond with ONLY a single numeric Q-value estimate. Do not include
 any explanation after your final number.
</system>

<user>
Estimate the Q-value for the Current Action in the Current State for the following:

{task_text}

### State-Action History

[Note: Only the last {num_shown} of {num_available} turns are shown.]

[Turn {t}/{max_steps} - State]
{history_state_text}

[Turn {t}/{max_steps} - Action]
{history_action_text}

### Current State
[Turn {step}/{max_steps}]
{state_text}

### Current Action
{action_text}

### Next State
{next_state_text}
```

```
Now, provide your Q-value estimate as a single number. Do not explain -- output ONLY the
 number.
</user>
```

`direct-batched` and `direct-16` prepend `Estimate ... for each datapoint below`, wrap each example in `## Datapoint i`, and ask for a comma-separated list of numeric Q-value estimates in datapoint order. `direct-sequential` instead sends one user turn per datapoint, headed `## Datapoint i of n`, and asks the model to keep earlier estimates fixed while returning exactly one estimate for the current turn.

Prompt C.2: `gvl`, shuffled-trajectory value elicitation

```
<system>
You are an expert at estimating Q-value functions for reinforcement learning
environments.

{q_value_definition}

You will receive the current state, the action taken, and the resulting next state.
Estimate Q(s,a) for the Current Action in the Current State -- the expected
{return_phrase} when taking this action in this state, assuming optimal play thereafter.

{task_and_reward_blocks}

You will receive a shuffled set of surrounding transitions from the same trajectory,
followed by one final target datapoint. The surrounding transitions are intentionally
out of chronological order and are provided only as contextual evidence. Estimate only
the final target datapoint.

Only the target datapoint includes its resulting next state. Surrounding context
transitions include only state and action.

After any reasoning, respond with ONLY a single numeric Q-value estimate. Do not include
 any explanation after your final number.
</system>

<user>
Estimate the Q-value for the Current Action in the Current State for the following:

{task_text}

## Shuffled Trajectory Context
The surrounding transitions below come from the same trajectory as the target datapoint,
 but are intentionally shuffled and not chronological.

### Context Transition {i}
#### State
{context_state_text}

#### Action
{context_action_text}

## Target Transition
### Target State
{target_state_text}

### Target Action
{target_action_text}

### Target Next State
{target_next_state_text}

Now, provide your Q-value estimate as a single number. Do not explain -- output ONLY the
 number.
</user>
```

Prompt C.3: `verifier`, **score-bin with logprob readout**

```
<system>
You are an expert verifier for reinforcement learning environments. Estimate Q-value
using an ordered discrete score scale.

{q_value_definition}

You will receive the episode's state-action history (if any), the current state, the
action taken, and the resulting next state. Estimate Q(s,a) for the Current Action in
the Current State -- the expected {return_phrase} when taking this action in this state,
assuming optimal play thereafter. The history provides context about how the agent
reached the current state.

Use an ordered 20-point score scale with the single-letter bins A through T, where A is
best and T is worst. Choose exactly one score bin per datapoint.
A = clearly and completely favorable with strong evidence of success (best)
B-D = strongly favorable with only minor remaining concerns
E-G = above average, mostly favorable with some issues
H-J = uncertain, leans favorable
K-M = uncertain, leans unfavorable
N-P = below average, significant issues remain
Q-S = poor, with only limited signs of progress
T = clearly and completely unfavorable or failed (worst)

Interpret the score bins as an ordered scale that will be converted into a scalar value
after decoding. Use higher bins for stronger evidence that the state or chosen action is
 favorable.

{task_and_reward_blocks}

Return exactly one score tag for the single datapoint.
</system>

<user>
Estimate a verifier score for each datapoint below. Estimate the Q-value for the Current
 Action in the Current State. Judge the chosen action and resulting next state using the
 ordered score bins.

### Evaluation Criterion: {criterion_name}
{criterion_description}

## Datapoint 1

{direct_datapoint_sections}

Now provide your final scores using one uppercase letter from A through T per datapoint.
Output exactly one tag per datapoint in prompt order:

<score_1>LETTER</score_1>
</user>
```

Prompt C.4: Δ`belief`, **pre/post belief query**

```
<system>
You are an expert at judging how candidate actions change an agent's probability of
eventual success in a sequential decision-making environment.

{q_value_definition}

Interpret "successful overall outcome" using the task and reward function below. For
sparse-success tasks, this means eventually completing the task. For dense-reward tasks,
 this means achieving a strong overall return.

Each candidate is scored by the change in log-probability of the one-word answer "Yes"
to a success question after the candidate's observed outcome is revealed.
```

```
{task_and_reward_blocks}
</system>

<user>
Assess the current situation before any candidate outcome is shown.

{state_action_history}

**Current State:**
{state_text}

Do not answer directly. The assistant continuation being scored is the one-word answer
to the success question below.

Question: Given the task and reward function, is the agent likely to eventually achieve
a successful overall outcome from here, assuming optimal future decisions thereafter?
Answer with exactly one word: Yes or No.
</user>

<user>
Re-assess the same situation after one specific candidate action has been attempted and
its observed outcome is available.

{state_action_history}

**Current State:**
{state_text}

**Candidate Action:**
{candidate_action_text}

**Observed Resulting State:**
{candidate_next_state_text}

Do not answer directly. The assistant continuation being scored is the one-word answer
to the success question below.

Question: Given the task and reward function, is the agent likely to eventually achieve
a successful overall outcome from here, assuming optimal future decisions thereafter?
Answer with exactly one word: Yes or No.
</user>
```

Prompt C.4 scored continuations

```
<assistant>
Yes
</assistant>

<assistant>
No
</assistant>
```

The backend scores the continuations `Yes` and `No`; the method uses the post-minus-pre log probability of `Yes`.

Prompt C.5: `codegen`, signal-function generation

```
<system>
You are an expert at writing Q-value functions for reinforcement learning environments.

{q_value_definition}

{approximation_guidance}
```

{*task_and_reward_blocks*}
</system>

<user>
Write a Python function that estimates the Q-value for a given state, action, and next state.

The function signature must be:

```python
def signal_function(state: str, action: str, next_state: str) -> float:
```

The function receives text representations of the state, the action taken, and the resulting next state. It should return a float representing the estimated Q-value.

Important constraints:
- Do NOT use recursive search, tree expansion, lookahead, or any form of simulation/rollout -- these will time out.
- Base the estimate on direct analysis of the state representation, using your understanding of what features predict good outcomes.

You may use the `collections`, `itertools`, `json`, `math`, `re`, `statistics`, and `string` standard library modules. The function will run in a restricted sandbox that does not expose the introspection builtins `locals`, `globals`, `vars`, `dir`, `eval`, or `exec` -- calling any of them raises NameError at runtime, so do not rely on them. After any reasoning, output ONLY a single Python code block containing the function definition. No explanations or commentary after the code block.
</user>

`codegen-avg` uses the same generation prompt for multiple independent samples and averages the resulting predictions over valid generated functions.

Prompt C.6: `sdpo`, asymmetric teacher / student contexts

<system>
You are an expert at comparing and ranking actions by their Q-value in reinforcement learning environments.

{*q_value_definition*}

You will evaluate one fixed candidate action at a time. The base policy scores the candidate from the original state. A self-teacher may additionally see environment feedback produced after that same action was attempted, and should use that feedback as per-turn evidence about whether the original action improved expected future success from the current state.

{*environment_specific_sdpo_guidance*}

{*task_and_reward_blocks*}
</system>

<user>
{*student_opening*}

{*state_action_history*}

Current State:
{*state_text*}

{*student_closing*}
</user>

<user>
{*teacher_opening*}

```
{state_action_history}

**Current State:**
{state_text}

{feedback_heading}
{feedback_text}

{teacher_closing}
</user>
```

Prompt C.6 scored continuation

```
<assistant>
{candidate_action_text}
</assistant>
```

For the base `sdpo` method, the feedback text is the serialized resulting state for the candidate action when next-state feedback is enabled; otherwise the prompt emits `[No explicit feedback was available for this action.]`. The same candidate action is scored in the student and teacher contexts; the method uses teacher-minus-student mean token log probability.

Prompt C.7: `sdpo-gt`, oracle teacher with stored expert evidence

```
<system>
You are an expert at comparing and ranking actions by their Q-value in reinforcement
learning environments.

{q_value_definition}

You will evaluate one fixed candidate action at a time. The base policy scores the
candidate from the original state. A self-teacher may additionally see environment
feedback produced after that same action was attempted, the next expert action from the
resulting state, and a compact stored expert rollout summary. It should use that
privileged evidence to judge whether the original action improved expected future
success from the current state.

{environment_specific_sdpo_gt_guidance}

{task_and_reward_blocks}
</system>

<user>
{student_opening}

{state_action_history}

**Current State:**
{state_text}

{student_closing}
</user>

<user>
{short_expert_teacher_opening}

{state_action_history}

**Current State:**
{state_text}

{feedback_heading}
{feedback_text}
```

```
{short_expert_heading}
The {expert_unit} below occurs after the {candidate_label} has already been executed in the
{state_label}.
Use it only as evidence about the original {candidate_label}; it is not a replacement
command to output.
{next_expert_label}: {best_next_expert_action}

Structured stored expert rollout summary:
{reached_label}: {yes_no_or_unknown}
{steps_label}: {steps_to_success}
Remaining distance proxy: {remaining_distance_proxy}
{reward_label}: {final_reward}
Stored ranking GT value ({ranking_gt_source}): {ranking_gt_value}
Stored expert rollout outcome: {success_no_success_or_unknown}

{short_expert_teacher_closing}
</user>
```

Prompt C.7 scored continuation

```
<assistant>
{candidate_action_text}
</assistant>
```

The runtime headings and labels specialize to the environment. For example, TerminalBench uses shell-response, expert-command, verifier-success, and verifier-reward labels; OpenApps, ALFWorld, and FrozenLake use the corresponding browser, household-command, or grid-move labels.

Prompt C.8: `ranking`, candidate-permutation request

```
<system>
You are an expert at comparing and ranking actions by their Q-value in reinforcement
learning environments.

{q_value_definition}

You will receive the episode's state-action history (if any), the current state, a list
of candidate actions, and the resulting next state for each action. Rank the actions by
their expected Q-value Q(s,a) -- the expected {return_phrase} when taking each action.
Output the action numbers from best to worst.

{task_and_reward_blocks}

After any reasoning, output ONLY a comma-separated list of action numbers from best to
worst. For example: 2, 1, 3, 4.
</system>

<user>
Rank the following actions by Q-value (best to worst) for the Current State:

{task_text}

{state_action_history}

## Current State
[Turn {step}/{max_steps}]
{state_text}

The following candidate actions are being evaluated in the current state:

## Actions to rank

[**Candidate Action {i}**]
{candidate_action_text}
```

[**Resulting State {*i*}**]
{*candidate_next_state_text*}

Rank all listed actions ({*k*} total here) from best (highest Q-value) to worst. Output ONLY a comma-separated list of action numbers. For example: {*example_ranking*}.
</user>

Prompt C.9: `eureka`, iterative code search with LLM judging

<system>
You are an expert at writing Q-value functions for reinforcement learning environments.

{*q_value_definition*}

{*approximation_guidance*}

{*task_and_reward_blocks*}

Iterative Search
You are participating in an iterative code-search loop. Each proposed candidate will be executed on benchmark datapoints, judged from its predicted values, and the winning code plus feedback may be shown back to you in the next iteration. Improve usefulness of the signal based on behavior, not code aesthetics.
</system>

<user>
Search Iteration
Iteration {*iteration*} of {*total_iterations*}. Write one candidate Python function for estimating the Q-value.

Function Contract
The function definition must start with:

```python
def signal_function(state: str, action: str, next_state: str):
```

The function may return either:
1. a single float, which is the actual signal value
2. a tuple `(float, dict[str, float])` where the first float is the actual signal value and the dictionary contains named additive components used only for search-time feedback.

If you return a dictionary, use short stable snake_case keys and make the scalar total equal to the sum of the component values. Build the components dictionary with explicit literal keys -- do NOT use `locals()`, `globals()`, or `vars()`.

Return Example
```python
def signal_function(state: str, action: str, next_state: str):
    progress_reward = 0.4
    safety_penalty = -0.1
    total = progress_reward + safety_penalty
    return total, {
        "progress_reward": progress_reward,
        "safety_penalty": safety_penalty,
    }
```

Current Best Code
{*previous_winner_code*}

Judge Feedback
{*reflection*}

```
{codegen_constraints_and_closing}
</user>

<system>
You are selecting the most useful dense-signal candidate for a reinforcement learning
environment.

{q_value_definition}

Judge candidate behavior from predicted values and diagnostics. Do not infer quality
from code style because candidate code is not shown.

{task_and_reward_blocks}
</system>

<user>
## Selection Rule
Choose the candidate whose predictions are the most appropriate and useful Q-value
estimates. Prefer candidates that provide discriminative, stable, non-collapsed values
and that match the intended semantics of the signal on the sampled datapoints.

## Candidate Summaries
### Candidate {candidate_label}
- Finite predictions: {finite_count}
- NaN predictions: {nan_count}
- Min: {min}
- Max: {max}
- Mean: {mean}
- Std: {std}
- Collapsed: {yes_or_no}

## Sampled Datapoints
### Datapoint {i}

{direct_datapoint_sections}

### Candidate Predictions
Candidate {candidate_label}: {prediction}

## Output Format
Respond with:
<winner>INTEGER</winner>
<rationale>Short explanation of why the winner is best.</rationale>
<feedback>Concrete guidance for improving the next iteration.</feedback>
</user>
```

E Model Details

We used six instruction-tuned LLMs. All runs used a maximum context length of 262,144 tokens, nucleus sampling with $p = 0.95$, and a thinking budget of 6,144 tokens. Standard value-prediction runs used a maximum generation length of 8,192 tokens; code-generation runs used 14,336 tokens.

Model	Temp.	Top-p	Top-k	Max tokens	Codegen max tokens
`google/gemma-4-31b-it`	1.0	0.95	64	8192	14336
`google/gemma-4-26b-a4b-it`	1.0	0.95	64	8192	14336
`qwen/qwen3.5-9b`	0.6	0.95	20	8192	14336
`qwen/qwen3.5-27b`	0.6	0.95	20	8192	14336
`qwen/qwen3.5-35b-a3b`	0.6	0.95	20	8192	14336
`qwen/qwen3.5-122b-a10b`	0.6	0.95	20	8192	14336

Table 4: LLM sampling settings used in QVAL experiments.

F Complete Results

Cells report Spearman (ρ) or Kendall (τ) correlation between predicted and ground-truth signals across evaluation points. Significance markers: $^{*}\, p < .10$, $^{**}\, p < .01$, $^{***}\, p < .001$ (no marker means not significant). Em-dash (—) marks experiments not yet run. Codegen aggregate cells show $\mathrm{mean} \pm \mathrm{std}$ on the top line and $[\min, \max]$ on the bottom over 16 sample correlations (no p-value: per-sample p's are dropped during aggregation). Ranking-style methods (`ranking`, `sdpo`, `sdpo-gt`, Δ`belief`) show $\mathrm{mean} \pm \mathrm{std}$ over per-trajectory Spearman. Rows and columns that are conceptually inapplicable to a given slice (e.g. Δ`belief` or `ranking` in V-value tables; CLIP / SIGLIP columns in V-value vision tables) are omitted from the table schema. Method rows, metric rows, and model columns with no values are omitted.

Table 5: Correlations on TerminalBench, Q-value, text modality, using Codex 5.5 Max-Value Monte Carlo for label generation.

Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B
gvl	ρ	0.196^{*}	0.127	0.222^{*}	0.109	0.180^{*}	0.254^{*}
	τ	0.166^{*}	0.108	0.168^{*}	0.074	0.118	0.185^{*}
direct-single	ρ	0.449^{***}	0.386^{***}	0.373^{***}	0.298^{**}	0.297^{**}	0.365^{***}
	τ	0.358^{***}	0.302^{***}	0.270^{***}	0.217^{**}	0.205^{**}	0.272^{***}
direct-batched	ρ	0.440^{***}	0.239^{*}	0.334^{***}	0.295^{**}	0.331^{***}	0.248^{*}
	τ	0.326^{***}	0.159^{*}	0.248^{**}	0.197^{**}	0.237^{**}	0.175^{*}
direct-sequential	ρ	0.353^{***}	0.359^{***}	0.420^{***}	0.356^{***}	0.292^{**}	0.376^{***}
	τ	0.273^{***}	0.278^{***}	0.305^{***}	0.265^{***}	0.218^{**}	0.290^{***}
direct-16	ρ	0.391^{***}	0.343^{***}	0.432^{***}	0.358^{***}	0.111	0.373^{***}
	τ	0.292^{***}	0.249^{***}	0.308^{***}	0.260^{***}	0.089	0.271^{***}
eureka	ρ	-0.117	-0.029	-0.310^{**}	0.006	-0.154	-0.263^{**}
	τ	-0.085	-0.039	-0.224^{**}	0.009	-0.115	-0.202^{**}
codegen	ρ	-0.327 ± 0.061 $[-0.432, -0.226]$	-0.269 ± 0.110 $[-0.412, -0.058]$	-0.275 ± 0.121 $[-0.395, 0.112]$	-0.310 ± 0.041 $[-0.371, -0.250]$	-0.238 ± 0.118 $[-0.412, -0.041]$	-0.288 ± 0.065 $[-0.392, -0.161]$
	τ	-0.268 ± 0.051 $[-0.345, -0.184]$	-0.219 ± 0.088 $[-0.335, -0.050]$	-0.222 ± 0.099 $[-0.308, 0.098]$	-0.244 ± 0.033 $[-0.292, -0.180]$	-0.198 ± 0.095 $[-0.329, -0.048]$	-0.224 ± 0.055 $[-0.307, -0.112]$
codegen-avg	ρ	-0.393^{***}	-0.347^{***}	-0.312^{**}	-0.313^{**}	-0.273^{**}	-0.329^{***}
	τ	-0.299^{***}	-0.267^{***}	-0.240^{**}	-0.238^{**}	-0.202^{**}	-0.246^{***}
sdpo	ρ	0.072 ± 0.598	0.139 ± 0.608	0.110 ± 0.600	0.079 ± 0.597	0.141 ± 0.591	0.132 ± 0.587
sdpo-gt	ρ	0.125 ± 0.603	0.157 ± 0.615	0.056 ± 0.579	0.030 ± 0.578	0.057 ± 0.581	—
verifier	ρ	0.265^{**}	0.179^{*}	0.037	0.095	-0.113	0.186^{*}
	τ	0.185^{*}	0.132^{*}	0.028	0.066	-0.085	0.135^{*}
Δbelief	ρ	-0.012 ± 0.647	0.073 ± 0.576	0.001 ± 0.644	0.023 ± 0.641	0.105 ± 0.634	0.126 ± 0.646
ranking	ρ	0.032 ± 0.644	0.082 ± 0.616	0.147 ± 0.629	0.098 ± 0.626	0.154 ± 0.630	0.167 ± 0.592

Table 6: Correlations on TerminalBench, Q-value, text modality, using Opus 4.7 Max-Value Monte Carlo for label generation.

Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B
`gvl`	ρ	0.207^{*}	0.175^{*}	0.149	0.060	0.255^{*}	0.212^{*}
	τ	0.170^{*}	0.150^{*}	0.108	0.044	0.184^{*}	0.149^{*}
`direct-single`	ρ	0.388^{***}	0.447^{***}	0.254^{*}	0.338^{***}	0.283^{**}	0.334^{***}
	τ	0.310^{***}	0.347^{***}	0.181^{*}	0.254^{***}	0.193^{*}	0.239^{**}
`direct-batched`	ρ	0.271^{*}	0.234^{*}	0.380^{***}	0.328^{***}	0.319^{**}	0.296^{**}
	τ	0.197^{*}	0.174^{*}	0.274^{***}	0.247^{***}	0.230^{**}	0.213^{**}
`direct-sequential`	ρ	0.335^{***}	0.449^{***}	0.407^{***}	0.334^{***}	0.315^{**}	0.393^{***}
	τ	0.258^{**}	0.344^{***}	0.315^{***}	0.250^{***}	0.231^{**}	0.293^{***}
`direct-16`	ρ	0.372^{***}	0.434^{***}	0.335^{***}	0.367^{***}	0.111	0.399^{***}
	τ	0.259^{***}	0.319^{***}	0.237^{**}	0.272^{***}	0.088	0.286^{***}
`eureka`	ρ	-0.007	0.009	-0.165	-0.115	-0.047	-0.186^{*}
	τ	-0.004	0.002	-0.119	-0.090	-0.031	-0.133^{*}
`codegen`	ρ	-0.173 ± 0.056 $[-0.240, -0.069]$	-0.134 ± 0.079 $[-0.255, 0.028]$	-0.105 ± 0.110 $[-0.234, 0.215]$	-0.150 ± 0.054 $[-0.239, -0.046]$	-0.104 ± 0.089 $[-0.250, 0.078]$	-0.159 ± 0.061 $[-0.221, -0.001]$
	τ	-0.137 ± 0.042 $[-0.185, -0.063]$	-0.108 ± 0.061 $[-0.203, 0.021]$	-0.084 ± 0.087 $[-0.176, 0.175]$	-0.116 ± 0.042 $[-0.190, -0.038]$	-0.086 ± 0.070 $[-0.191, 0.057]$	-0.123 ± 0.046 $[-0.180, -0.002]$
`codegen-avg`	ρ	-0.252^{*}	-0.230^{*}	-0.070	-0.157	-0.143	-0.192^{*}
	τ	-0.184^{*}	-0.169^{*}	-0.066	-0.115	-0.103	-0.138^{*}
`verifier`	ρ	0.292^{**}	0.318^{**}	0.139	0.214^{*}	-0.023	0.300^{**}
	τ	0.209^{**}	0.238^{**}	0.109	0.165^{*}	-0.015	0.220^{**}

Table 7: Correlations on OpenApps, Q-value, text modality, using a scripted policy with Max-Value Monte Carlo for label generation.

Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B
gvl	ρ	0.035	-0.105	0.041	0.340^{***}	0.249^{*}	0.394^{***}
	τ	0.031	-0.088	0.047	0.276^{***}	0.197^{*}	0.305^{***}
direct-single	ρ	0.382^{***}	0.449^{***}	0.358^{***}	0.451^{***}	0.454^{***}	0.287^{**}
	τ	0.307^{***}	0.372^{***}	0.283^{***}	0.364^{***}	0.355^{***}	0.233^{**}
direct-batched	ρ	0.243^{*}	0.547^{***}	0.329^{**}	0.461^{***}	0.260^{*}	0.370^{***}
	τ	0.186^{*}	0.433^{***}	0.247^{**}	0.357^{***}	0.205^{**}	0.284^{***}
direct-sequential	ρ	0.039	0.335^{***}	0.216^{*}	0.311^{**}	0.159	0.107
	τ	0.033	0.265^{**}	0.164^{*}	0.228^{**}	0.122	0.087
direct-16	ρ	0.295^{**}	0.621^{***}	0.472^{***}	0.616^{***}	0.448^{***}	0.480^{***}
	τ	0.229^{**}	0.476^{***}	0.345^{***}	0.482^{***}	0.342^{***}	0.369^{***}
eureka	ρ	0.091	0.417^{***}	-0.099	0.086	-0.160	-0.076
	τ	0.075	0.334^{***}	-0.078	0.068	-0.115	-0.049
codegen	ρ	-0.131 ± 0.077 $[-0.266, 0.023]$	-0.069 ± 0.131 $[-0.307, 0.136]$	-0.026 ± 0.252 $[-0.485, 0.434]$	-0.072 ± 0.177 $[-0.365, 0.278]$	-0.018 ± 0.248 $[-0.426, 0.430]$	-0.142 ± 0.285 $[-0.546, 0.602]$
	τ	-0.091 ± 0.060 $[-0.208, 0.029]$	-0.052 ± 0.105 $[-0.240, 0.109]$	-0.023 ± 0.206 $[-0.407, 0.354]$	-0.059 ± 0.138 $[-0.307, 0.193]$	-0.014 ± 0.202 $[-0.355, 0.358]$	-0.103 ± 0.220 $[-0.426, 0.465]$
codegen-avg	ρ	-0.058	0.009	—	-0.060	-0.089	-0.315^{**}
	τ	-0.027	0.004	—	-0.050	-0.069	-0.239^{**}
sdpo	ρ	-0.184 ± 0.593	-0.273 ± 0.641	0.105 ± 0.610	-0.049 ± 0.626	-0.016 ± 0.645	0.007 ± 0.620
sdpo-gt	ρ	-0.109 ± 0.626	-0.196 ± 0.624	0.021 ± 0.595	-0.111 ± 0.567	0.024 ± 0.597	—
verifier	ρ	0.290^{**}	0.265^{**}	0.111	0.170	-0.036	0.002
	τ	0.215^{**}	0.211^{**}	0.083	0.128^{*}	-0.026	-0.003
Δbelief	ρ	0.440 ± 0.452	0.481 ± 0.462	0.441 ± 0.478	0.359 ± 0.563	0.337 ± 0.577	0.425 ± 0.513
ranking	ρ	0.369 ± 0.571	0.311 ± 0.542	0.335 ± 0.612	0.294 ± 0.605	0.353 ± 0.602	0.302 ± 0.623

Table 8: Correlations on OpenApps, Q-value, vision modality, using a scripted policy with Max-Value Monte Carlo for label generation.

		LLM backbones						Non-LLM
Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B	Self
`gvl`	ρ	0.176^{*}	-0.128	—	0.426^{***}	-0.079	0.500^{***}	—
	τ	0.143^{*}	-0.110	—	0.338^{***}	-0.052	0.394^{***}	—
`direct-single`	ρ	0.413^{***}	0.352^{***}	0.437^{***}	0.460^{***}	0.329^{**}	0.228^{*}	—
	τ	0.336^{***}	0.302^{***}	0.356^{***}	0.386^{***}	0.265^{**}	0.184^{*}	—
`direct-batched`	ρ	0.402^{***}	0.530^{***}	—	0.475^{***}	0.279^{*}	0.586^{***}	—
	τ	0.308^{***}	0.417^{***}	—	0.379^{***}	0.217^{*}	0.463^{***}	—
`direct-sequential`	ρ	0.043	0.401^{***}	—	0.242^{*}	0.117	0.239^{*}	—
	τ	0.034	0.323^{***}	—	0.201^{*}	0.095	0.186^{*}	—
`direct-16`	ρ	0.539^{***}	0.607^{***}	0.451^{***}	0.610^{***}	0.445^{***}	0.609^{***}	—
	τ	0.406^{***}	0.472^{***}	0.353^{***}	0.469^{***}	0.355^{***}	0.485^{***}	—
`ranking`	ρ	0.264 ± 0.537	0.295 ± 0.566	0.264 ± 0.610	0.298 ± 0.539	0.277 ± 0.584	0.271 ± 0.578	—
`vip`	ρ	—	—	—	—	—	—	0.303^{**}
	τ	—	—	—	—	—	—	0.245^{**}
`liv-cos`	ρ	—	—	—	—	—	—	0.121
	τ	—	—	—	—	—	—	0.115
`liv-l2`	ρ	—	—	—	—	—	—	0.115
	τ	—	—	—	—	—	—	0.103

Table 9: Correlations on OpenApps, State-value, text modality, using a scripted policy with Max-Value Monte Carlo for label generation.

Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B
`gvl`	ρ	0.037	-0.070	-0.310^{**}	0.376^{***}	0.014	0.316^{**}
	τ	0.028	-0.046	-0.248^{**}	0.320^{***}	0.003	0.270^{***}
`direct-single`	ρ	0.187^{*}	0.139	0.271^{**}	0.152	0.110	0.188^{*}
	τ	0.147^{*}	0.116	0.209^{**}	0.118	0.091	0.158^{*}
`direct-batched`	ρ	0.363^{***}	0.509^{***}	0.046	0.423^{***}	0.330^{**}	0.255^{*}
	τ	0.287^{***}	0.391^{***}	0.029	0.332^{***}	0.259^{**}	0.197^{*}
`direct-sequential`	ρ	0.140	0.131	0.072	0.227^{*}	0.208^{*}	0.137
	τ	0.108	0.103	0.052	0.171^{*}	0.155^{*}	0.098
`direct-16`	ρ	0.296^{**}	0.578^{***}	0.385^{***}	0.478^{***}	0.210^{*}	0.419^{***}
	τ	0.219^{**}	0.446^{***}	0.279^{***}	0.363^{***}	0.154^{*}	0.313^{***}
`eureka`	ρ	0.567^{***}	0.440^{***}	-0.053	-0.120	0.429^{***}	0.528^{***}
	τ	0.453^{***}	0.341^{***}	-0.039	-0.109	0.329^{***}	0.442^{***}
`codegen`	ρ	0.277 ± 0.218 $[-0.050, 0.564]$	0.267 ± 0.167 $[-0.043, 0.555]$	0.051 ± 0.187 $[-0.208, 0.383]$	0.094 ± 0.203 $[-0.190, 0.449]$	0.246 ± 0.198 $[-0.182, 0.493]$	0.140 ± 0.206 $[-0.209, 0.482]$
	τ	0.213 ± 0.173 $[-0.047, 0.453]$	0.218 ± 0.133 $[-0.041, 0.434]$	0.037 ± 0.157 $[-0.175, 0.308]$	0.072 ± 0.156 $[-0.140, 0.347]$	0.193 ± 0.166 $[-0.161, 0.413]$	0.112 ± 0.168 $[-0.181, 0.365]$
`codegen-avg`	ρ	0.535^{***}	0.440^{***}	0.064	0.306^{**}	0.353^{***}	0.013
	τ	0.380^{***}	0.327^{***}	0.034	0.230^{**}	0.261^{***}	-0.009

Table 10: Correlations on ALFWorld, Q-value, text modality, using a scripted policy with Max-Value Monte Carlo for label generation.

Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B
gvl	ρ	0.327^{***}	0.611^{***}	0.515^{***}	0.413^{***}	0.463^{***}	0.317^{**}
	τ	0.252^{***}	0.511^{***}	0.406^{***}	0.330^{***}	0.378^{***}	0.250^{***}
direct-single	ρ	0.598^{***}	0.665^{***}	0.567^{***}	0.625^{***}	0.587^{***}	0.546^{***}
	τ	0.478^{***}	0.539^{***}	0.441^{***}	0.478^{***}	0.437^{***}	0.437^{***}
direct-batched	ρ	0.563^{***}	0.649^{***}	0.594^{***}	0.594^{***}	0.595^{***}	0.206^{*}
	τ	0.438^{***}	0.519^{***}	0.455^{***}	0.461^{***}	0.461^{***}	0.142^{*}
direct-sequential	ρ	0.678^{***}	0.716^{***}	0.564^{***}	0.623^{***}	0.519^{***}	0.436^{***}
	τ	0.544^{***}	0.572^{***}	0.422^{***}	0.483^{***}	0.392^{***}	0.335^{***}
direct-16	ρ	0.658^{***}	0.738^{***}	0.570^{***}	0.585^{***}	0.388^{***}	0.647^{***}
	τ	0.493^{***}	0.595^{***}	0.436^{***}	0.442^{***}	0.314^{***}	0.485^{***}
eureka	ρ	-0.063	-0.029	0.110	0.389^{***}	0.084	0.208^{*}
	τ	-0.047	-0.011	0.084	0.285^{***}	0.060	0.163^{*}
codegen	ρ	-0.054 ± 0.259 $[-0.313, 0.397]$	-0.012 ± 0.227 $[-0.342, 0.356]$	0.125 ± 0.192 $[-0.103, 0.488]$	0.205 ± 0.151 $[-0.020, 0.469]$	0.036 ± 0.157 $[-0.284, 0.249]$	0.080 ± 0.189 $[-0.359, 0.365]$
	τ	-0.046 ± 0.222 $[-0.267, 0.343]$	-0.009 ± 0.193 $[-0.296, 0.307]$	0.102 ± 0.157 $[-0.086, 0.417]$	0.160 ± 0.119 $[-0.017, 0.384]$	0.030 ± 0.126 $[-0.229, 0.214]$	0.054 ± 0.148 $[-0.284, 0.288]$
codegen-avg	ρ	-0.207^{*}	0.091	0.251^{*}	0.169^{*}	0.274^{**}	0.231^{*}
	τ	-0.183^{*}	0.068	0.192^{**}	0.116	0.182^{*}	0.161^{*}
sdpo	ρ	-0.138 ± 0.626	-0.195 ± 0.619	-0.226 ± 0.568	-0.172 ± 0.630	-0.200 ± 0.575	-0.209 ± 0.637
sdpo-gt	ρ	-0.129 ± 0.631	-0.209 ± 0.598	-0.181 ± 0.587	-0.195 ± 0.613	-0.187 ± 0.551	—
verifier	ρ	0.592^{***}	0.676^{***}	0.495^{***}	0.621^{***}	0.506^{***}	0.617^{***}
	τ	0.432^{***}	0.541^{***}	0.377^{***}	0.472^{***}	0.371^{***}	0.466^{***}
Δbelief	ρ	0.333 ± 0.639	0.423 ± 0.551	0.321 ± 0.674	0.429 ± 0.562	0.416 ± 0.535	0.386 ± 0.654
ranking	ρ	0.456 ± 0.492	0.513 ± 0.559	0.501 ± 0.520	0.483 ± 0.564	0.538 ± 0.555	0.513 ± 0.583

Table 11: Correlations on ALFWorld, Q-value, vision modality, using a scripted policy with Max-Value Monte Carlo for label generation.

		LLM backbones						Non-LLM
Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B	Self
gvl	ρ	0.275^{**}	0.385^{***}	—	0.162	0.080	0.292^{**}	—
	τ	0.216^{**}	0.298^{***}	—	0.130^{*}	0.060	0.207^{**}	—
direct-single	ρ	0.446^{***}	0.454^{***}	0.228^{*}	0.266^{**}	0.028	0.322^{**}	—
	τ	0.326^{***}	0.343^{***}	0.168^{*}	0.186^{*}	0.024	0.242^{**}	—
direct-batched	ρ	0.302^{**}	0.210^{*}	—	0.215^{*}	—	0.151	—
	τ	0.220^{**}	0.128^{*}	—	0.171^{*}	—	0.119	—
direct-sequential	ρ	0.136	0.377^{***}	—	0.111	-0.173	0.202^{*}	—
	τ	0.101	0.278^{***}	—	0.087	-0.121	0.140^{*}	—
direct-16	ρ	0.269^{**}	0.546^{***}	0.155	0.279^{**}	0.150	0.472^{***}	—
	τ	0.191^{**}	0.395^{***}	0.119^{*}	0.204^{**}	0.096	0.334^{***}	—
ranking	ρ	0.197 ± 0.630	0.122 ± 0.630	0.154 ± 0.662	0.147 ± 0.663	—	0.184 ± 0.636	—
vip	ρ	—	—	—	—	—	—	0.302^{**}
	τ	—	—	—	—	—	—	0.215^{**}
liv-cos	ρ	—	—	—	—	—	—	0.288^{**}
	τ	—	—	—	—	—	—	0.201^{**}
liv-l2	ρ	—	—	—	—	—	—	0.241^{*}
	τ	—	—	—	—	—	—	0.179^{*}
liv-txt	ρ	—	—	—	—	—	—	0.049
	τ	—	—	—	—	—	—	0.035

Table 12: Correlations on ALFWorld, State-value, text modality, using a scripted policy with Max-Value Monte Carlo for label generation.

Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B
gvl	ρ	0.185^{*}	0.436^{***}	0.142	0.326^{***}	0.161	0.211^{*}
	τ	0.145^{*}	0.373^{***}	0.113	0.266^{***}	0.118	0.164^{*}
direct-single	ρ	0.560^{***}	0.469^{***}	0.386^{***}	0.410^{***}	0.480^{***}	0.335^{***}
	τ	0.430^{***}	0.391^{***}	0.287^{***}	0.320^{***}	0.370^{***}	0.264^{***}
direct-batched	ρ	0.329^{**}	0.458^{***}	0.443^{***}	0.318^{**}	0.196^{*}	0.332^{***}
	τ	0.241^{**}	0.353^{***}	0.340^{***}	0.233^{**}	0.145^{*}	0.252^{***}
direct-sequential	ρ	0.500^{***}	0.449^{***}	0.394^{***}	0.590^{***}	0.460^{***}	0.463^{***}
	τ	0.391^{***}	0.342^{***}	0.304^{***}	0.479^{***}	0.349^{***}	0.358^{***}
direct-16	ρ	0.423^{***}	0.628^{***}	0.457^{***}	0.459^{***}	0.453^{***}	0.492^{***}
	τ	0.340^{***}	0.498^{***}	0.333^{***}	0.353^{***}	0.341^{***}	0.371^{***}
eureka	ρ	0.095	0.158	0.275^{**}	0.356^{***}	0.083	0.243^{*}
	τ	0.082	0.128	0.206^{**}	0.266^{***}	0.052	0.171^{*}
codegen	ρ	-0.223 ± 0.112 $[-0.302, -0.144]$	0.029 ± 0.190 $[-0.302, 0.181]$	0.161 ± 0.188 $[-0.172, 0.398]$	0.022 ± 0.147 $[-0.255, 0.327]$	0.108 ± 0.192 $[-0.262, 0.363]$	0.160 ± 0.171 $[-0.175, 0.443]$
	τ	-0.184 ± 0.106 $[-0.260, -0.109]$	0.024 ± 0.163 $[-0.259, 0.155]$	0.120 ± 0.138 $[-0.134, 0.326]$	0.020 ± 0.120 $[-0.189, 0.273]$	0.085 ± 0.160 $[-0.225, 0.312]$	0.134 ± 0.143 $[-0.140, 0.381]$
codegen-avg	ρ	-0.239^{*}	0.015	0.361^{***}	0.075	0.335^{***}	0.275^{**}
	τ	-0.192^{*}	0.026	0.244^{***}	0.062	0.233^{**}	0.176^{*}

Table 13: Correlations on FrozenLake, Q-value, text modality, using a scripted policy with Max-Value Monte Carlo for label generation.

Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B
gvl	ρ	0.395^{***}	0.399^{***}	0.139	0.085	-0.104	0.147
	τ	0.287^{***}	0.349^{***}	0.103	0.079	-0.073	0.152^{*}
direct-single	ρ	0.131	0.479^{***}	0.504^{***}	0.051	0.268^{**}	0.274^{**}
	τ	0.100	0.412^{***}	0.376^{***}	0.061	0.201^{**}	0.231^{**}
direct-batched	ρ	—	0.544^{***}	0.540^{***}	—	0.086	0.205^{*}
	τ	—	0.428^{***}	0.412^{***}	—	0.064	0.184^{*}
direct-sequential	ρ	0.075	0.361^{***}	0.513^{***}	0.237^{*}	0.162	0.133
	τ	0.059	0.331^{***}	0.367^{***}	0.183^{*}	0.121^{*}	0.107
direct-16	ρ	0.077	0.377^{***}	0.433^{***}	-0.040	0.553^{***}	0.332^{***}
	τ	0.058	0.337^{***}	0.348^{***}	-0.029	0.414^{***}	0.224^{**}
eureka	ρ	0.710^{***}	0.959^{***}	0.965^{***}	0.554^{***}	0.870^{***}	0.686^{***}
	τ	0.574^{***}	0.885^{***}	0.875^{***}	0.413^{***}	0.710^{***}	0.513^{***}
codegen	ρ	0.888 ± 0.258 $[0.132, 1.000]$	0.898 ± 0.328 $[-0.318, 1.000]$	0.589 ± 0.496 $[-0.631, 0.983]$	0.712 ± 0.344 $[-0.001, 0.972]$	0.678 ± 0.374 $[-0.207, 0.983]$	0.961 ± 0.053 $[0.776, 0.985]$
	τ	0.850 ± 0.259 $[0.115, 1.000]$	0.877 ± 0.308 $[-0.232, 1.000]$	0.527 ± 0.442 $[-0.494, 0.934]$	0.619 ± 0.309 $[0.000, 0.903]$	0.606 ± 0.350 $[-0.179, 0.934]$	0.896 ± 0.064 $[0.694, 0.934]$
codegen-avg	ρ	0.983^{***}	0.988^{***}	0.883^{***}	0.886^{***}	0.915^{***}	0.976^{***}
	τ	0.918^{***}	0.937^{***}	0.730^{***}	0.727^{***}	0.776^{***}	0.896^{***}
sdpo	ρ	-0.505 ± 0.414	-0.631 ± 0.301	-0.537 ± 0.425	-0.522 ± 0.474	0.169 ± 0.374	-0.666 ± 0.379
sdpo-gt	ρ	-0.485 ± 0.400	-0.577 ± 0.370	-0.521 ± 0.466	-0.462 ± 0.478	0.030 ± 0.357	—
verifier	ρ	0.401^{***}	0.495^{***}	-0.046	-0.380^{***}	-0.190^{**}	-0.137^{*}
	τ	0.314^{***}	0.377^{***}	-0.041	-0.280^{***}	-0.140^{**}	-0.097^{*}
Δbelief	ρ	0.320 ± 0.529	0.438 ± 0.491	0.050 ± 0.594	0.313 ± 0.513	0.358 ± 0.515	0.331 ± 0.513
ranking	ρ	0.825 ± 0.250	0.907 ± 0.118	0.802 ± 0.298	0.901 ± 0.136	0.528 ± 0.575	0.882 ± 0.240

Table 14: Correlations on FrozenLake, Q-value, vision modality, using a scripted policy with Max-Value Monte Carlo for label generation.

		LLM backbones						Non-LLM
Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B	Self
`gvl`	ρ	0.551^{***}	0.555^{***}	—	-0.005	0.128	0.081	—
	τ	0.416^{***}	0.500^{***}	—	-0.010	0.090	0.093	—
`direct-single`	ρ	0.319^{*}	0.659^{***}	-0.002	0.382^{***}	—	0.218^{*}	—
	τ	0.257^{*}	0.541^{***}	0.007	0.292^{***}	—	0.193^{**}	—
`direct-batched`	ρ	—	0.710^{***}	—	0.522^{***}	—	0.386^{***}	—
	τ	—	0.588^{***}	—	0.400^{***}	—	0.323^{***}	—
`direct-sequential`	ρ	—	0.309^{**}	—	0.448^{***}	—	0.346^{***}	—
	τ	—	0.242^{**}	—	0.372^{***}	—	0.280^{***}	—
`direct-16`	ρ	0.068	0.467^{***}	0.215^{*}	0.560^{***}	0.093	0.119	—
	τ	0.063	0.415^{***}	0.157^{*}	0.401^{***}	0.066	0.083	—
`ranking`	ρ	—	0.902 ± 0.130	—	0.829 ± 0.269	—	0.894 ± 0.141	—
`vip`	ρ	—	—	—	—	—	—	0.014
	τ	—	—	—	—	—	—	0.015
`liv-cos`	ρ	—	—	—	—	—	—	-0.276^{***}
	τ	—	—	—	—	—	—	-0.178^{***}
`liv-l2`	ρ	—	—	—	—	—	—	-0.280^{***}
	τ	—	—	—	—	—	—	-0.176^{***}
`liv-txt`	ρ	—	—	—	—	—	—	0.141^{*}
	τ	—	—	—	—	—	—	0.096^{*}

Table 15: Correlations on FrozenLake, State-value, text modality, using a scripted policy with Max-Value Monte Carlo for label generation.

Method	Metric	Gemma4 26B-A4B	Gemma4 31B	Qwen3.5 9B	Qwen3.5 27B	Qwen3.5 35B-A3B	Qwen3.5 122B-A10B
`gvl`	ρ	0.170	0.001	0.236^{*}	-0.038	0.007	0.240^{*}
	τ	0.104	0.030	0.174^{*}	-0.026	-0.003	0.205^{**}
`direct-single`	ρ	0.116	0.065	0.503^{***}	0.022	0.178^{*}	0.263^{**}
	τ	0.077	0.069	0.379^{***}	0.024	0.134^{*}	0.206^{**}
`direct-batched`	ρ	—	0.508^{***}	0.558^{***}	0.303^{*}	-0.087	0.193^{*}
	τ	—	0.449^{***}	0.399^{***}	0.257^{**}	-0.050	0.159^{*}
`direct-sequential`	ρ	0.082	0.453^{***}	0.457^{***}	0.211^{*}	0.226^{*}	0.241^{*}
	τ	0.068	0.390^{***}	0.340^{***}	0.157^{*}	0.169^{*}	0.189^{**}
`direct-16`	ρ	0.017	0.150	0.608^{***}	-0.007	0.653^{***}	0.180^{*}
	τ	0.011	0.162^{*}	0.492^{***}	-0.002	0.500^{***}	0.117^{*}
`eureka`	ρ	0.805^{***}	0.911^{***}	0.986^{***}	0.873^{***}	0.803^{***}	0.940^{***}
	τ	0.655^{***}	0.787^{***}	0.928^{***}	0.764^{***}	0.658^{***}	0.822^{***}
`codegen`	ρ	0.831 ± 0.177 $[0.431, 1.000]$	0.899 ± 0.092 $[0.668, 0.987]$	0.810 ± 0.216 $[0.296, 0.991]$	0.847 ± 0.172 $[0.384, 0.973]$	0.825 ± 0.190 $[0.421, 0.985]$	0.877 ± 0.122 $[0.543, 0.973]$
	τ	0.741 ± 0.195 $[0.372, 1.000]$	0.791 ± 0.117 $[0.530, 0.935]$	0.702 ± 0.223 $[0.228, 0.946]$	0.738 ± 0.177 $[0.307, 0.891]$	0.726 ± 0.197 $[0.320, 0.935]$	0.771 ± 0.132 $[0.442, 0.894]$
`codegen-avg`	ρ	0.874^{***}	0.942^{***}	0.937^{***}	0.921^{***}	0.932^{***}	0.933^{***}
	τ	0.714^{***}	0.824^{***}	0.809^{***}	0.786^{***}	0.810^{***}	0.806^{***}

Table 16: Correlations on OpenApps, State-value, vision modality, using a scripted policy with Max-Value Monte Carlo for label generation.

		Non-LLM		
Method	Metric	Self	CLIP	SigLIP
`vlm-rm`	ρ	—	-0.028	0.000
	τ	—	-0.031	0.007
`vlm-rm-cos`	ρ	—	-0.069	-0.116
	τ	—	-0.044	-0.089
`vlm-sor-softmax`	ρ	—	-0.031	0.026
	τ	—	-0.007	0.030
`vip`	ρ	0.232^{*}	—	—
	τ	0.184^{*}	—	—
`liv-cos`	ρ	0.178^{*}	—	—
	τ	0.138^{*}	—	—
`liv-l2`	ρ	0.155	—	—
	τ	0.120	—	—

Table 17: Correlations on ALFWorld, State-value, vision modality, using a scripted policy with Max-Value Monte Carlo for label generation.

		Non-LLM		
Method	Metric	Self	CLIP	SigLIP
`vlm-rm`	ρ	—	0.256^{*}	0.333^{***}
	τ	—	0.187^{**}	0.221^{**}
`vlm-rm-cos`	ρ	—	0.419^{***}	0.136
	τ	—	0.297^{***}	0.090
`vlm-sor-softmax`	ρ	—	0.338^{***}	0.243^{*}
	τ	—	0.238^{***}	0.177^{*}
`vip`	ρ	0.014	—	—
	τ	0.003	—	—
`liv-cos`	ρ	0.055	—	—
	τ	0.034	—	—
`liv-l2`	ρ	0.036	—	—
	τ	0.024	—	—
`liv-txt`	ρ	0.189^{*}	—	—
	τ	0.128^{*}	—	—

Table 18: Correlations on FrozenLake, State-value, vision modality, using a scripted policy with Max-Value Monte Carlo for label generation.

		Non-LLM		
Method	Metric	Self	CLIP	SigLIP
`vlm-rm`	ρ	—	-0.113^{*}	-0.427^{***}
	τ	—	-0.069	-0.297^{***}
`vlm-rm-cos`	ρ	—	-0.036	-0.340^{***}
	τ	—	-0.026	-0.237^{***}
`vlm-sor`	ρ	—	-0.104	—
	τ	—	-0.088	—
`vlm-sor-softmax`	ρ	—	-0.259^{***}	-0.377^{***}
	τ	—	-0.182^{***}	-0.270^{***}
`vip`	ρ	-0.171^{**}	—	—
	τ	-0.120^{**}	—	—
`liv-cos`	ρ	0.065	—	—
	τ	0.053	—	—
`liv-l2`	ρ	0.081	—	—
	τ	0.068	—	—
`liv-txt`	ρ	0.154^{*}	—	—
	τ	0.102^{*}	—	—