AGENT-AS-A-COACH: TOWARDS FULLY AGENTIC, STATEFUL, AND TOOL-AUGMENTED PROCESS REWARDS

Ed Li[*†] **Junyu Ren**[*] **Cat Yan** **Kerem Goksel**

ABSTRACT

Training multiagent LLM systems with reinforcement learning requires solving credit assignment: determining which agent's actions led to success or failure in long, multi-turn trajectories. Process rewards from external LLM judges address this by scoring each agent's actions rather than only the final outcome. However, existing judges are *stateless*: each evaluation is an independent LLM call with no memory of prior scores, no awareness of scoring drift, and no mechanism for self-calibration—even as thousands of evaluations accumulate over a training run. We propose upgrading the stateless *LLM-as-a-Judge* to *agent-as-a-coach*: a tool-augmented LLM agent with persistent per-coach memory, rolling evaluation statistics, and a multi-turn reasoning loop. The coach queries its own scoring history, detects task-type biases, writes qualitative notes for self-calibration, and passes messages via external memory. In preliminary experiments on DSBench with a three-agent data science pipeline (Qwen3-4B), our agent-as-a-coach approach produces adaptive rebalancing between classification and regression tasks: anti-correlated performance oscillations that a stateless judge lacking cross-evaluation awareness cannot generate.

1 INTRODUCTION

Training multiagent LLM systems with reinforcement learning requires solving credit assignment: which agent's actions led to success or failure? Outcome-only rewards assign a single success/failure signal to an entire trajectory (Lightman et al., 2023; DeepSeek-AI, 2025), leaving most agent actions without gradient information in long-horizon, multi-turn settings. Recent work addresses this by using an external LLM judge to provide dense, per-action process rewards on a 0–10 scale (Li et al., 2026), enabling implicit credit assignment when combined with policy gradient methods such as REINFORCE++ (Hu et al., 2025). We study this setting on DSBench (Jing et al., 2025), an end-to-end data science benchmark where a three-agent pipeline—Data Engineer, Modeler, and Analyst—collaborates to solve classification and regression tasks by executing Python code in a shared sandbox (Figure 1). This setting is particularly challenging: failures propagate across agents (e.g., a preprocessing error by the Data Engineer causes the Modeler to fail), making per-action evaluation essential. However, the judge in Li et al. (2026) treats each evaluation independently—a stateless LLM call with no memory of prior evaluations, no ability to detect scoring drift or task-type bias, and no mechanism for self-calibration over the hundreds of evaluations that occur during a training run. This limitation is shared by other recent approaches that co-optimize reward models alongside policies (Wang et al., 2026).

We propose upgrading this passive judge to an *agentic coach*: an LLM agent equipped with persistent per-coach memory, analytical tools, and a multi-turn reasoning loop (Figure 1). We use the term "coach" rather than "judge" to emphasize this shift—while a judge evaluates outputs against fixed criteria, a coach actively investigates context, reflects on its own history, and adapts its feedback over the course of training. The coach operates in a tool-augmented loop of up to K turns, querying its own evaluation statistics, detecting scoring biases, writing persistent notes for self-calibration, and passing messages between per-coach instances for cross-agent credit attribution. In preliminary

[*]Equal contribution.
[†]Corresponding author: ed.li@yale.edu

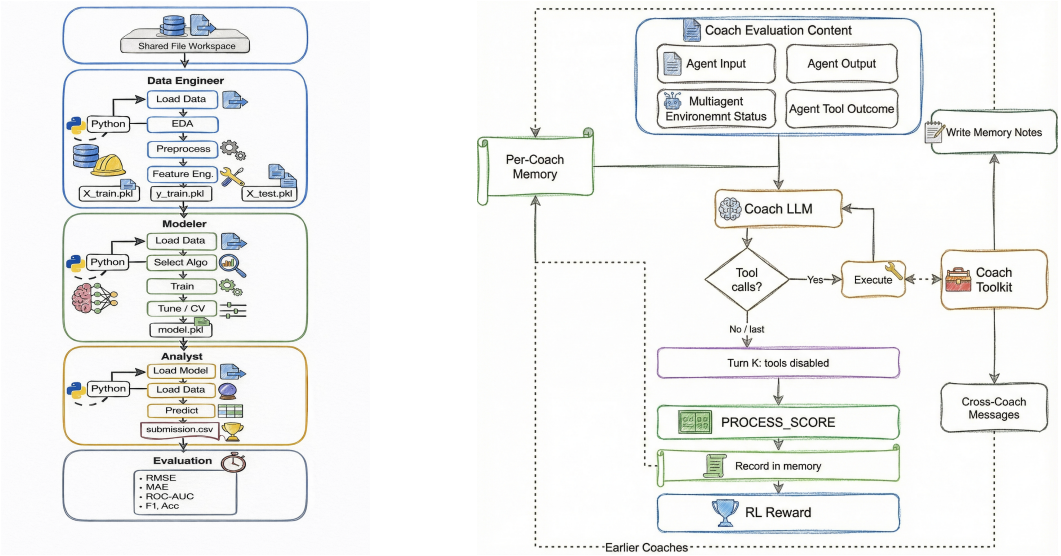

Figure 1: **(a)** DSBench three-agent pipeline: agents execute Python code via a shared sandbox, with the agentic coach evaluating each agent's actions independently. **(b)** Agentic coach evaluation loop: at each agent turn, the coach queries its toolkit and persistent memory before assigning a process reward score.

experiments on DSBench with Qwen3-4B agents, we observe the coach's memory enabling *adaptive rebalancing* between classification and regression tasks: anti-correlated oscillations in task-type performance that a stateless judge, lacking any cross-evaluation awareness, could not produce (Section 4).

2 Related Work

Zheng et al. (2023) established the LLM-as-a-Judge paradigm, showing that strong LLMs like GPT-4 can approximate human preferences at $\sim 80\%$ agreement, while also documenting systematic biases—position, verbosity, and self-enhancement—that constrain reliability (Ye et al., 2024; Chen et al., 2024; Wataoka et al., 2024). A recent survey (You et al., 2026) organizes the emerging *Agent-as-a-Judge* paradigm along three evolutionary stages—procedural, reactive, and self-evolving—and five methodological dimensions: multi-agent collaboration, planning, tool integration, memory, and optimization. Within tool-augmented evaluation, Peng et al. (2025) propose RewardAgent, which combines preference-based reward models with verifiable correctness signals from search engines and code interpreters, and Han et al. (2025) introduce VerifiAgent, a two-tiered verification agent that adaptively selects domain-appropriate tools (computer algebra, symbolic logic, knowledge bases) and outperforms process reward models at inference-time scaling with fewer samples. Ding et al. (2025) train ARM-Thinker, a multimodal reward model that autonomously invokes external tools (image cropping, document retrieval) during scoring, jointly optimizing tool-calling decisions and judgment accuracy via multi-stage reinforcement learning. For multi-agent evaluation, ChatEval (Chan et al., 2023) pioneered multi-agent debate for assessment, while Chen et al. (2025) extend this with MAJ-Eval, constructing evaluator personas that engage in group debate for multi-dimensional feedback. Our agentic coach combines elements from several of these threads—tool integration, persistent memory, and multi-turn reasoning—but is distinguished by its use as a *process reward model for RL training* rather than as an inference-time evaluator.

3 METHODOLOGY

3.1 FROM STATELESS JUDGE TO STATEFUL COACH

We introduce the *agentic coach*: a drop-in replacement for the stateless coach that equips the evaluator with persistent memory, analytical tools, and a multi-turn reasoning loop. The agentic coach conforms to the same evaluation interface, so the multiagent workflow and training loop require no modification—only the coach component changes.

Multi-turn evaluation loop. Rather than producing a score in a single LLM call, the agentic coach operates in a tool-augmented loop of up to K turns (default $K = 3$). At each turn, the coach may call internal tools to query its memory, analyze code artifacts, or check for biases before committing to a score. This mirrors how a human coach would reflect on past performance before giving feedback, rather than evaluating each action in isolation.

Per-coach memory. Each agent role (e.g., data engineer, modeler, analyst) has a dedicated coach memory instance that maintains a rolling window of the W most recent evaluations (default $W = 100$). Per-coach separation is critical: different roles naturally receive different score distributions due to task difficulty differences—a shared memory would misidentify these structural differences as scoring bias. Each memory instance tracks:

- **Rolling statistics**: count, mean, standard deviation of scores.
- **Trend metrics**: Theil-Sen slope (Theil, 1950) (robust to outliers) and Kendall's τ (Kendall, 1938) (non-parametric trend strength with p-value).
- **Task-type breakdowns**: Separate statistics for each task category (e.g., classification vs. regression), enabling bias detection.
- **Qualitative notes**: Free-text insights written by the coach LLM itself (up to 20 notes), persisted across evaluations for self-calibration.

3.2 COACH TOOLKIT

The agentic coach has access to six lightweight, in-process tools (Appendix B) that require no external services. Four read-only analytics tools query the coach's own memory for performance history, task-type statistics, scoring bias detection, and code quality heuristics. A memory-write tool lets the coach persist qualitative notes across evaluations. A communication tool (`send_message_to_coach`) lets an upstream coach flag issues for the downstream coach, enabling proper cross-agent credit attribution in sequential pipelines. All tools are designed to *expose raw data* rather than prescriptive conclusions—the coach LLM interprets statistics and decides what matters.

4 PRELIMINARY EXPERIMENTS

4.1 SETUP

We train a three-agent pipeline (Data Engineer, Modeler, Analyst) on DSBench (Jing et al., 2025), an end-to-end data science benchmark consisting of 64 Kaggle-style training tasks (43 classification, 21 regression) and 8 held-out evaluation tasks. All three agents share a Qwen3-4B backbone (Team, 2025) and are trained with REINFORCE++ (Hu et al., 2025). Each agent has 2 tool-use turns per round, executing Python code via a SandboxFusion (Seed, 2024) code sandbox. Training uses 6×A100 80GB GPUs with DeepSpeed ZeRO-3, vLLM inference (2 engines, prefix caching enabled), rollout batch size 16, and 2 samples per prompt. One epoch over the 64 training tasks corresponds to 4 training steps; we train for 3 epochs (12 steps) with evaluation on the held-out set every 2 steps.

The process reward coach is a Gemini-3-Flash model operating in a multi-turn agentic loop (up to $K = 3$ turns) with access to all six tools described in Appendix B. Each training step generates $\sim$580 coach evaluations (3 agents $\times$ 2 turns $\times$ 16 rollouts $\times$ 2 samples), totaling 7,016 over the full run.

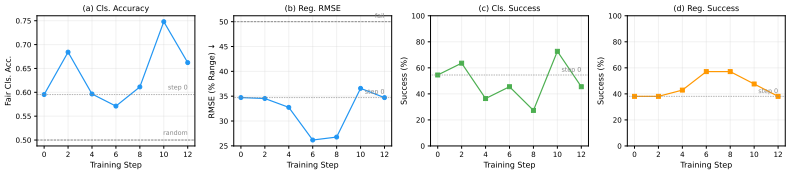

Figure 2: Evaluation metrics over 3 epochs (12 training steps) on 8 held-out DSBench tasks. Dotted lines mark step-0 (untrained) performance. **(a)** Fair classification accuracy (failed tasks penalized at 0.5). **(b)** Fair RMSE as % of target range (lower is better; failures penalized at 50%). **(c, d)** Success rates—the fraction of tasks producing valid predictions—for classification and regression respectively. The anti-correlated oscillation between task types reflects the coach's adaptive rebalancing via persistent memory.

Raw metrics computed only over tasks that produce valid predictions are misleading—a model that succeeds on 2 easy tasks can appear better than one that attempts 10 harder tasks. We report *fair* metrics that penalize failures: failed classification tasks receive 0.5 accuracy (random baseline), and failed regression tasks receive 50% RMSE-as-percentage-of-range (the failure penalty threshold). This ensures models cannot game the metric by attempting fewer tasks.

4.2 Results

Figure 2 shows fair evaluation metrics over 12 training steps on 8 held-out DSBench tasks. All metrics are evaluated against ground truth; the coach provides only process rewards during training and does not influence evaluation scoring.

Li et al. (2026) report *unintended specialization* with a stateless coach on the same DSBench pipeline: classification metrics peak early then permanently regress to baseline, while regression continues improving—the stateless coach systematically scores regression tasks higher (+0.5–1.8 points), and agents exploit this bias via gradient updates. The most interesting pattern in Figure 2 is that our agentic coach produces the opposite dynamic: an *anti-correlated oscillation* between classification and regression rather than unidirectional drift. When one task type improves, the other temporarily regresses, followed by corrective rebalancing in subsequent epochs. Classification accuracy peaks at 0.75 (step 10, +25% over baseline) before moderating; regression reaches its best fair RMSE of 26.2% (step 6) but gives back gains when classification surges. The agentic coach accumulates task-type statistics in its rolling memory and queries them via tools before scoring, enabling self-correcting reward dynamics where neither task type is permanently sacrificed—precisely the failure mode of the stateless baseline.

Over 7,016 evaluations across 12 training steps, the agentic coach called tools an average of 3.0 times per evaluation, most frequently writing memory notes (28%), querying training phase context (25%), and checking agent performance history (24%). Bias detection was invoked in 34% of evaluations. The coach completed in 1.3 turns on average (never needing all 3 available), and fell back to stateless single-turn evaluation in only 0.17% of cases—demonstrating that the agentic loop adds capability reliably without introducing fragility.

5 Conclusion

We presented the agentic coach, a stateful process reward model that replaces single-call LLM judges with a tool-augmented agent maintaining persistent per-coach memory, evaluation statistics, and cross-agent communication. Preliminary experiments on DSBench with Qwen3-4B agents show that the coach's persistent memory enables adaptive rebalancing between classification and regression tasks, replacing the unidirectional specialization observed with a stateless coach (Li et al., 2026) with self-correcting oscillations where neither task type is permanently sacrificed.

Key directions for future work include controlled ablations isolating the contribution of memory, tools, and multi-turn reasoning; and scaling to longer training horizons where scoring drift becomes more pronounced. DSBench's small dataset (64 training tasks) prevents us from leveraging the

coach's ability to actively select or even generate training tasks (Wang et al., 2026)—a form of curriculum learning that our framework supports but that requires a larger task pool to be meaningful. Finally, we plan to scale from a single coach to a *swarm of coaches* that deliberate and vote-pool before assigning rewards, improving reward stability through ensemble agreement.

REFERENCES

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. ChatEval: Towards better LLM-based evaluators through multi-agent debate, 2023. URL https://arxiv.org/abs/2308.07201.

Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. Humans or LLMs as the judge? a study on judgement bias. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

Jiaju Chen, Yuxuan Lu, Xiaojie Wang, Huimin Zeng, Jing Huang, Jiri Gesi, Ying Xu, Bingsheng Yao, and Dakuo Wang. Multi-agent-as-judge: Aligning LLM-agent-based automated evaluation with multi-dimensional human evaluation, 2025. URL https://arxiv.org/abs/2507.21028.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Shengyuan Ding, Xinyu Fang, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiangyu Zhao, Haodong Duan, Xiaoyi Dong, Jianze Liang, Bin Wang, Conghui He, Dahua Lin, and Jiaqi Wang. ARM-Thinker: Reinforcing multimodal generative reward models with agentic tool use and visual reasoning, 2025. URL https://arxiv.org/abs/2512.05111.

Jiuzhou Han, Wray Buntine, and Ehsan Shareghi. VerifiAgent: a unified verification agent in language model reasoning, 2025. URL https://arxiv.org/abs/2504.00406.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. REINFORCE++: Stabilizing critic-free policy optimization with global advantage normalization, 2025. URL https://arxiv.org/abs/2501.03262.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. DSBench: How far are data science agents to becoming data science experts? In *International Conference on Learning Representations*, 2025.

Maurice G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

Ed Li, Junyu Ren, and Cat Yan. Scaling multiagent systems with process rewards, 2026. URL https://arxiv.org/abs/2601.23228.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Hao Peng, Yunjia Qi, Xiaozhi Wang, Zijun Yao, Bin Xu, Lei Hou, and Juanzi Li. Agentic reward modeling: Integrating human preferences with verifiable correctness signals for reliable reward systems, 2025. URL https://arxiv.org/abs/2502.19328.

ByteDance Seed. SandboxFusion: An online code execution platform for ai agents, 2024. URL https://github.com/bytedance/SandboxFusion.

Qwen Team. Qwen3 technical report, 2025. URL https://qwenlm.github.io/blog/qwen3/.

Henri Theil. A rank-invariant method of linear and polynomial regression analysis. *Indagationes Mathematicae*, 12:85–91, 1950.

Yinjie Wang, Tianbao Xie, Ke Shen, Mengdi Wang, and Ling Yang. RLAnything: Forge environment, policy, and reward model in completely dynamic RL system, 2026. URL https://arxiv.org/abs/2602.02488.

Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. Self-preference bias in LLM-as-a-judge, 2024. URL https://arxiv.org/abs/2410.21819.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, et al. Justice or prejudice? quantifying biases in LLM-as-a-judge, 2024. URL https://arxiv.org/abs/2410.02736.

Runyang You, Hongru Cai, Caiqi Zhang, Qiancheng Xu, Meng Liu, Tiezheng Yu, Yongqi Li, and Wenjie Li. A survey on agent-as-a-judge, 2026. URL https://arxiv.org/abs/2601.05111.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

A Design Principles

The agentic coach is designed around a core tenet: *let the coach LLM reason; do not override its judgment with hardcoded logic.* This leads to three concrete guidelines:

No silent post-processing. The score the coach outputs is the score used for RL training. If we want the coach to account for bias, training phase, or task difficulty, we tell it in the system prompt and provide relevant data via tools—we do not apply correction formulas after the fact. Silent post-processing creates opacity: when a reward signal is wrong, it becomes impossible to tell whether the coach's reasoning or the infrastructure is at fault.

Soft guidance over rigid rules. The system prompt provides contextual information ("you have completed N evaluations; current training phase is mid") but does not dictate specific score ranges for specific phases. The coach can use this context as it sees fit, adapting to situations that rigid rules cannot anticipate. This is what makes the coach *agentic* rather than merely *automated*.

Tools expose data, not conclusions. Analytics tools return raw statistics—means, standard deviations, trend slopes—not prescriptive quality judgments. The coach synthesizes this information through its own reasoning. This prevents the infrastructure from inadvertently becoming the reward model while the nominal "coach" merely rubber-stamps pre-computed scores.

B Coach Toolkit Details

All six tools are lightweight, in-process functions requiring no external services. Each tool returns raw data (statistics, boolean checks, bias magnitudes) rather than prescriptive conclusions—the coach LLM interprets the results and decides how they should influence scoring.

`get_agent_performance_history(agent_role).` Queries the per-coach rolling memory for a specific agent role. Returns evaluation count, mean score, standard deviation, and robust trend metrics: Theil-Sen slope (median pairwise slope, robust to outlier scores) and Kendall's τ (non-parametric trend strength with p-value). The coach can call this periodically to detect scoring drift—e.g., if its mean has risen by 0.5 over the last 50 evaluations, it may be becoming more lenient.

`get_task_type_statistics(task_type).` Returns mean, standard deviation, and count for evaluations of a specific task type (e.g., "classification" or "regression"). The coach can compare statistics across task types to check whether its scoring is calibrated consistently—a 1.5-point gap between classification and regression means might reflect genuine difficulty differences or a systematic bias.

`check_score_distribution_bias().` Computes per-task-type bias as the number of standard deviations above or below the overall mean. Returns bias magnitudes and textual recommendations when any bias exceeds 0.5σ. Requires a minimum number of evaluations (configurable via `bias_detection_window`, default 50) before reporting; returns "insufficient data" otherwise. No automatic correction is applied—the coach decides whether and how to respond.

`compute_code_quality_heuristics(code, weights={...}).` Static regex-based analysis of agent-generated Python code. Returns boolean indicators: error handling (`try/except`), imports, comments, print output, file saves, random state setting, cross-validation usage, and data leakage risk (`fit_transform` on test data). These are combined into a weighted `estimated_quality` score (0–10). Crucially, the coach can *override the default weights* via an optional `weights` parameter—for example, upweighting `saves_files` for the Data Engineer (whose primary job is producing artifacts) while upweighting `has_cross_validation` for the Modeler. Default weights are: `saves_files` (2.0), `has_error_handling` (1.5), `has_cross_validation` (1.5), `has_imports` (1.0), `sets_random_state` (1.0), `no_data_leakage` (1.0), `code_length` (1.0), `has_comments` (0.5), `has_print_output` (0.5).

`write_memory_note(key, note).` Persists a named qualitative insight across evaluations. Writing to an existing key overwrites it, enabling the coach to refine its understanding over time (e.g., updating a "regression_bias" note as the bias magnitude changes). Maximum 20 notes; when full, the oldest note is evicted. Notes are injected directly into the system prompt for subsequent evaluations, so the coach sees them without needing to call a read tool.

`send_message_to_coach(target_agent_role, message).` Passes a one-time message to the coach instance evaluating a later agent in the pipeline. Messages are consumed on first read. This enables cross-agent credit attribution: if the Data Engineer's coach observes that `X_test.pkl` was not saved, it can message the Modeler's and Analyst's coaches so they do not penalize downstream agents for an upstream failure. Deduplication prevents the same insight from being sent repeatedly across evaluations.

C COACH SYSTEM PROMPTS

Below are typical system prompts seen by the coach LLM at two points during training. The static sections (scoring guidance, tool descriptions) are identical across evaluations; the dynamic sections—training phase, bias alert, evaluation history, and coach-written notes—are populated from the coach's persistent memory. Shared sections are abbreviated in the late-training version.

TYPICAL EARLY-TRAINING PROMPT (NO ACCUMULATED HISTORY)

```
You are an expert AI coach evaluating agents in a multi-agent data science
pipeline. You are currently evaluating the **data_engineer** agent.

## Why Your Scores Matter | You Are Training These Agents

Your PROCESS_SCORE is used as a reward signal for reinforcement learning.
The agents are neural networks being trained with policy gradient methods |
your score directly drives gradient updates that shape how these agents
behave in the future. This means:

1. Calibration is critical. If you consistently score too high (e.g.,
   giving 8 for mediocre work), the agents learn that mediocre work is
   good enough and stop improving. Use the full 0-10 range meaningfully.
2. Score variance enables learning. RL algorithms learn from *differences*
   between scores (advantages). If you give every response a 7, the agents
   learn nothing. Differentiate clearly: broken pipeline 1-3, competent
   work 5-7, excellent work 8-9, and 10 is truly exceptional.
3. Bias causes misaligned specialization. If you systematically score one
   task type higher, agents will learn to specialize toward the
   higher-scored type at the expense of the other. This was observed in
   prior training runs. Use check_score_distribution_bias() to detect this.
4. Consistency across evaluations matters. Avoid score drift (gradually
   becoming more lenient or strict). Use get_agent_performance_history()
   to check your scoring trend.

## Your Memory & Tools

You have persistent memory and tools that make you a *stateful* coach |
unlike a stateless evaluator, you can learn from past evaluations.

Available tools:
- get_agent_performance_history(agent_role) | Rolling stats and trends.
- get_task_type_statistics(task_type) | Per-task-type score distribution.
- check_score_distribution_bias() | Detect scoring biases across types.
- compute_code_quality_heuristics(code) | Static analysis of agent code.
- get_training_phase_context() | Current training phase + guidelines.
- write_memory_note(key, note) | Save an insight for future evaluations.
- send_message_to_coach(target, message) | Pass context to downstream coach.

When to use tools vs skip them:
Not every evaluation needs tool calls. For straightforward cases, score
directly. Use tools when: you're unsure about calibration, the task type
is one you've seen bias in before, or you notice a pattern worth recording.

Why memory notes matter for training:
You will evaluate hundreds of agent actions. Without notes, you have no way
to remember insights from earlier evaluations | you might discover that
regression RMSE looks deceptively good on normalized targets, correct for
it, then forget and drift back 50 evaluations later. Notes are your
mechanism for maintaining consistent scoring across the entire training run.

## Training Phase: EARLY (data_engineer)
Agent is still learning basics. Focus on task completion. Be lenient on
methodology. Reward executable code that saves required files.

## Instructions
1. Read the evaluation context below
2. Call tools if useful | check stats, check bias if you suspect drift
3. If you notice a pattern or calibration insight, write a note
4. Provide your final score as: PROCESS_SCORE: [0.0 to 10.0]
```

TYPICAL LATE-TRAINING PROMPT (WITH ACCUMULATED STATE)

```
You are an expert AI coach evaluating agents in a multi-agent data science
pipeline. You are currently evaluating the **analyst** agent.

[...same "Why Your Scores Matter" and "Your Memory & Tools" as above...]
```

```
## Training Phase: LATE (analyst)
Agent should produce high-quality code. High standards: hyperparameter
tuning, feature engineering, ensemble methods, proper error handling.

## Bias Alert (analyst)
- classification: 0.21 std devs lower than average
- regression: 0.21 std devs higher than average
No automatic correction is applied | it is your responsibility to account
for this bias.

## Evaluation History (analyst)
90 evaluations so far. avg=7.7, std=1.2, trend=→

## Your Notes (3/20)
- regression_scoring: Regression RMSE looks deceptively good on normalized
  targets | adjust expectations downward
- scoring_calibration: I've been scoring too high in early training |
  reserve 9-10 for truly exceptional work
- analyst_pattern: Analyst frequently copies modeler code verbatim instead
  of adding analysis | penalize this

## Instructions
1. Read the evaluation context below
2. Call tools if useful | check stats, check bias if you suspect drift
3. If you notice a pattern or calibration insight, write a note
4. Provide your final score as: PROCESS_SCORE: [0.0 to 10.0]
```