PROTEA: Offline Evaluation and Iterative Refinement for Multi-Agent LLM Workflows

Kazuki Kawamura, Satoshi Waki, Kei Tateno
Sony Group Corporation
{kazuki.kawamura,satoshi.waki,kei.tateno}@sony.com

Abstract

Multi-agent LLM workflows—systems composed of multiple role-specific LLM calls—often outperform single-prompt baselines, but they remain difficult to debug and refine. Failures can originate from subtle errors in intermediate outputs that propagate to downstream nodes, requiring developers to inspect long traces and infer which agent to modify. We present PROTEA, a unified interface for offline, test-driven improvement of multi-agent workflows. PROTEA executes a workflow, scores intermediate node outputs with configurable rubrics, and overlays per-node states and rationales on the workflow graph to localize likely bottlenecks. To support complex systems where final-answer references are the primary supervision, PROTEA performs backward node evaluation: it generates candidate node-level expectations from final-answer references and graph context, then compares them with observed node outputs. For selected nodes, PROTEA presents targeted prompt revisions as editable before/after comparisons, then automatically reruns and re-evaluates the workflow to show output changes and score trajectories within the same interface. In two production-adjacent workflows, PROTEA improved document-inspection accuracy from 64.3% to 83.9% and recommendation Hit@5 from 0.30 to 0.38. In a formative study with six experienced LLM developers, participants valued graph-level localization, per-node rationales, and editable before/after prompt revisions.

1 Introduction

LLM applications are increasingly built as agentic and multi-agent systems that compose multiple role-specific LLM calls into a workflow graph. By splitting responsibilities across nodes (e.g., intent analysis, retrieval, planning, ranking, and response generation), such graph-based workflows can be more controllable and produce higher-quality outputs than single-prompt approaches. This design pattern is now widely supported by orchestration frameworks (e.g., AutoGen, LangGraph) and decomposition-based prompting paradigms (Wu et al., 2024; LangChain, 2024a; Yao et al., 2023).

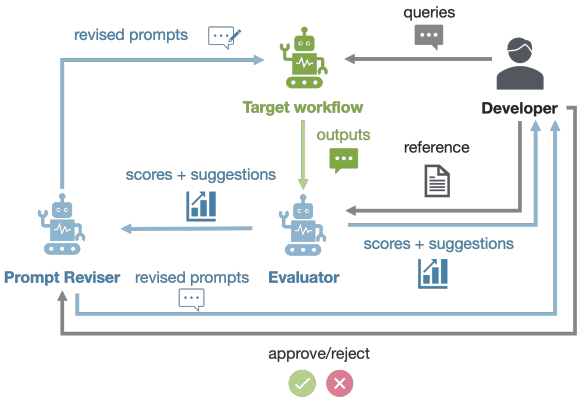

Figure 1: Overview of PROTEA: an interactive, evaluation-driven framework for developer-steered, AI-assisted refinement of multi-agent LLM workflows. Developers identify bottlenecks, inspect evidence and prompt revisions, edit or approve changes, and compare behavior within one loop.

However, this decomposition makes refinement a systems-level debugging task. When the final answer needs improvement, a small omission, misunderstanding, or formatting drift in an upstream artifact can propagate to downstream nodes. Two questions dominate day-to-day iteration: which node is a useful starting point for inspection, and what minimal prompt revision would improve the workflow while preserving desired behavior.

A growing body of work provides evaluation resources for LLMs and agents, but these typically emphasize end-to-end success rather than graph-aware diagnosis. Agent benchmarks for tool use and multi-step behavior (e.g., AgentBench and ToolLLM/ToolBench) report task-level outcomes and aggregated statistics (Liu et al., 2024; Qin et al., 2024). Similarly, practical evalua-

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 27–35

tion frameworks and libraries—including OpenAI Evals, lm-evaluation-harness, OpenEvals, DeepEval, and promptfoo—support offline testing and regression suites for LLM applications (OpenAI, 2023; Gao et al., 2023; LangChain, 2025; Ip and Vongthongsri, 2026; Webster et al., 2025). For specific architectures such as RAG, frameworks like RAGAs and ARES evaluate components such as retrieval relevance and answer faithfulness (Es et al., 2024; Saad-Falcon et al., 2024). These tools are useful for measurement and comparison, yet they often evaluate the application as an end-to-end system: even when traces are available, the developer must manually determine where in a workflow the error was introduced and how to revise a particular node prompt.

Another line of tools focuses on the development and operation of LLM applications by providing tracing, datasets, evaluations, and prompt management in the interface. For example, LangSmith aims to support tracing and evaluation alongside prompt testing and deployment management, while platforms such as Arize Phoenix, Weave, Langfuse, and PromptLayer provide complementary combinations of observability, evaluation, and prompt/version management (LangChain, 2024b; Arize AI, 2024; Weights & Biases, 2024; Langfuse, 2024; PromptLayer, 2024). These platforms substantially improve visibility, but the iteration-and-comparison loop for multi-agent graphs still requires substantial manual effort: developers translate evaluation evidence into a candidate fix, edit prompts (often in a separate view or in code), rerun the workflow, and then reconstruct what changed. Moreover, in realistic product settings, teams often have labels only for the terminal output; defining reference outputs for every intermediate node is costly, and such references are rarely maintained.

Automatic prompt optimization and self-improvement methods address a different part of the problem. DSPy compiles modular LM programs and can optimize prompts/modules against an end metric; OPRO and APE use LLMs to search for improved prompts; Self-Refine and Reflexion iteratively improve generations via self-feedback and reflection (Khattab et al., 2024; Yang et al., 2024; Zhou et al., 2023; Madaan et al., 2023; Shinn et al., 2023). These approaches are complementary to PROTEA: they optimize program behavior against an objective, whereas PROTEA focuses on the developer workflow for a concrete multi-agent graph by surfacing candidate bottleneck nodes and proposing localized, inspectable prompt revisions that a developer can adopt, edit, or revert in context.

We present PROTEA, a unified interface for offline evaluation and iterative refinement of multi-agent LLM workflows that is explicitly designed for workflow designers, prompt engineers, and planners. PROTEA is built around three ideas. First, it surfaces node-level evidence rather than only terminal scores: intermediate node outputs are evaluated with configurable rubrics using LLM-as-a-judge techniques (Liu et al., 2023; Zheng et al., 2023; Gu et al., 2024), and pass/warn/fail states plus rationales are overlaid directly on the workflow graph. Second, PROTEA reduces labeling effort via backward node evaluation: from final-answer references, it generates node-level expectations using the desired final output and downstream requirements, evaluates nodes in reverse topological order, and highlights upstream outputs as candidate bottlenecks for developer inspection. Third, PROTEA provides localized, adoptable fixes: for a developer-selected node, it proposes targeted prompt revisions in editable before/after comparisons grounded in the evaluator rationale and node rubric. After the developer adopts (or edits) the prompt revision, PROTEA automatically reruns and re-evaluates the workflow and shows before/after comparisons and score histories in the same interface, enabling rapid iteration without tool switching. We evaluate PROTEA through two production-adjacent workflow studies, an automatic iteration stress test, and a formative user study with six experienced LLM developers. The workflow studies improved document-inspection accuracy from 64.3% to 83.9% and recommendation Hit@5 from 0.30 to 0.38; the user study suggests that developers value graph-level localization, node-level rationales, and editable before/after prompt revisions for offline debugging and refinement.

2 System Overview

PROTEA supports offline refinement of multi-agent LLM workflows represented as directed acyclic graphs (DAGs). Each node corresponds to a role-specific LLM call (or a tool-augmented agent) with its own prompt and evaluation criteria, and edges represent intermediate outputs passed between agents. The system is organized around a fixed-suite iteration loop: run the current workflow, evaluate intermediate node outputs, inspect high-

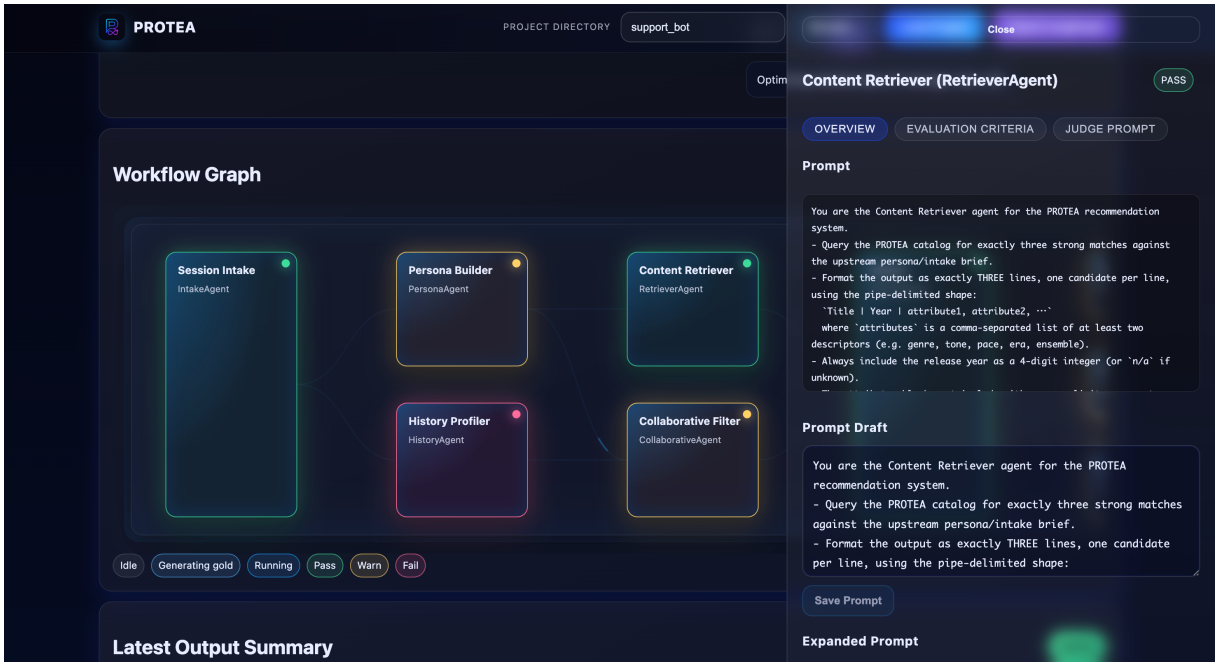

Figure 2: Representative PROTEA interface for inspecting and debugging a multi-agent workflow. The main view shows the workflow graph and node-level evaluation states. Red/yellow nodes indicate FAIL/WARN states and guide inspection. The inspection panel shows details for the selected node, including its prompt, evaluation result, suggested revision, and expanded prompt. Developers can inspect connections, review node behavior, and revise prompts while monitoring workflow-level effects.

lighted nodes, revise the selected node's prompt, and rerun the workflow to check regressions.

2.1 Workflow setup and saved runs

A PROTEA project combines (i) a workflow specification (loaded from a saved project or imported from LangGraph), (ii) an offline test suite of queries with optional final-answer references, and (iii) per-node evaluator settings (rubrics, judge prompts, and thresholds). When human-provided intermediate references are available, they can be attached to the corresponding node. Otherwise, PROTEA can generate a candidate node-level expectation for each node from the final-answer reference and graph context using the backward mode described in Section 3.1. This generated expectation is stored and shown in the interface, so developers can inspect, edit, or override it as needed.

For each run, PROTEA saves node traces (inputs/outputs and metadata), evaluator scores and rationales, any generated node-level expectations, and the current set of node prompts. These saved records support the interface shown in Figure 2 and make iterations comparable on the same test suite.

2.2 System architecture

PROTEA consists of a browser-based interface and a backend service that orchestrates (a) workflow execution, (b) automated evaluation using rubric-based LLM judging, and (c) prompt revision assistance. At a high level (Figure 1), an automated evaluation module produces per-node results and improvement suggestions. When final-answer references are the available supervision, this module first performs backward node evaluation to generate candidate node-level expectations from the final-answer reference and dependencies between nodes, then compares them to observed node outputs to produce scores and rationales. An automatic improvement module turns these evaluation results into an editable prompt draft for a selected node. All prompts, traces, and evaluation outputs are stored with the project so that developers can compare iterations.

2.3 How a developer uses PROTEA in practice

Figure 3 illustrates the typical interaction flow. A developer first loads a project and confirms the target agent graph (Step 1), then clicks nodes to inspect and edit their metadata such as prompts, evaluation criteria, and (when available) references

(Step 2). They run the workflow with the current prompts on a selected test query (Step 3) or a batch of queries, and then trigger AUTO EVALUATE to score intermediate node outputs (Step 4). When final-answer references are the available supervision, AUTO EVALUATE can additionally generate node-level expectations in a backward manner from the final-answer reference and downstream requirements (Section 3.1); the interface exposes this stage as reference generation and stores the generated expectation for inspection. The workflow graph is annotated with discrete states (PASS/WARN/FAIL), and selecting a highlighted node opens an inspection panel that shows its outputs alongside evaluator rationales and suggested prompt updates (Step 5). Finally, the developer edits the suggested prompt draft for that node, saves it, and reruns inference and evaluation to compare the new behavior against the same offline suite (Step 6). Across iterations, the evaluation-history view summarizes score trajectories and enables replaying past runs and restoring previous prompt sets for rollback. Figure 2 shows how graph-level states and node-level inspection are combined in the interface. PROTEA also provides an automatic iteration mode (AUTO LOOP in the interface) to repeat this evaluate→revise→re-evaluate cycle for a fixed number of iterations while logging each run.

3 Key Functionalities

PROTEA supports multi-agent workflow iteration under realistic constraints—complex dependencies, final-answer-only references, and long trace histories—by coupling three core mechanisms: backward node evaluation for generating node-level expectations, node-level diagnosis support in the workflow interface, and validated prompt refinement with automatic re-evaluation.

3.1 Node-level Evaluation via Backward Inference

Agents in a workflow are coupled through the outputs they pass to downstream nodes. Developers often use final-answer references as the primary supervision signal, while node-level references are created selectively. To support this setting, PROTEA uses *Backward Node Evaluation*: starting from final-answer supervision, the system generates a candidate expectation for each node using the requirements of downstream nodes.

Formally, let the workflow be a DAG $G = (V, E)$. Each node $v \in V$ has an instruction I_v and, when provided, a required output format $\mathcal{S}_v$. For a given test query, the workflow produces an observed output y_v at each node. The reference for v, denoted y_v^{ref}, is chosen as follows. For the final node, the final-answer reference $y_{\text{final}}^{\star}$ is used when available; if it is unavailable, a human-provided node reference $y_v^{\star}$ is used when present, and otherwise PROTEA generates a fallback candidate reference. For intermediate nodes, a human-provided node reference $y_v^{\star}$ is used when available; otherwise, PROTEA generates a candidate reference $\hat{y}_v$ from the final-answer reference and graph context when possible, falling back to the node instruction and required output format.

The candidate reference is produced by a reference-generation step, denoted $\mathcal{M}_{\text{gen}}$. We use separate symbols for the reference-generation, evaluation, and prompt-revision steps to distinguish their roles in the loop. For node v, the reference-generation step receives a context containing I_v, the optional required output format $\mathcal{S}_v$, the node's local position in the graph, the needs of its immediate children, and, when available, the final-answer reference for the current query. It generates a node output that supports the downstream path toward the desired final answer.

When v has multiple children, their requirements are presented together to produce a consolidated expectation for the node. In other words, the generated expectation is shaped to be useful to all immediate downstream nodes. A simple fallback reference derived from the node instruction and required output format is used as a robust default.

Given y_v and y_v^{ref}, an evaluator call $\mathcal{M}_{\text{eval}}$ scores the node output using node-specific criteria. For each evaluation criterion d, the evaluator produces a score $\sigma_d(v) \in [0, 1]$, and PROTEA computes an overall weighted score $s(v) \in [0, 1]$. The score is mapped to PASS, WARN, or FAIL using configurable thresholds; the default thresholds are 0.8 for PASS and 0.55 for WARN. The evaluator also returns a short rationale R_v and a suggested direction for improving the node prompt; PROTEA stores these with the reference used for scoring so developers can inspect the basis of each state.

3.2 Node-level Diagnosis Support

In workflows with many nodes, developers need a compact view of where to focus inspection. PROTEA surfaces node-level evidence on the workflow graph so that the developer can quickly choose a

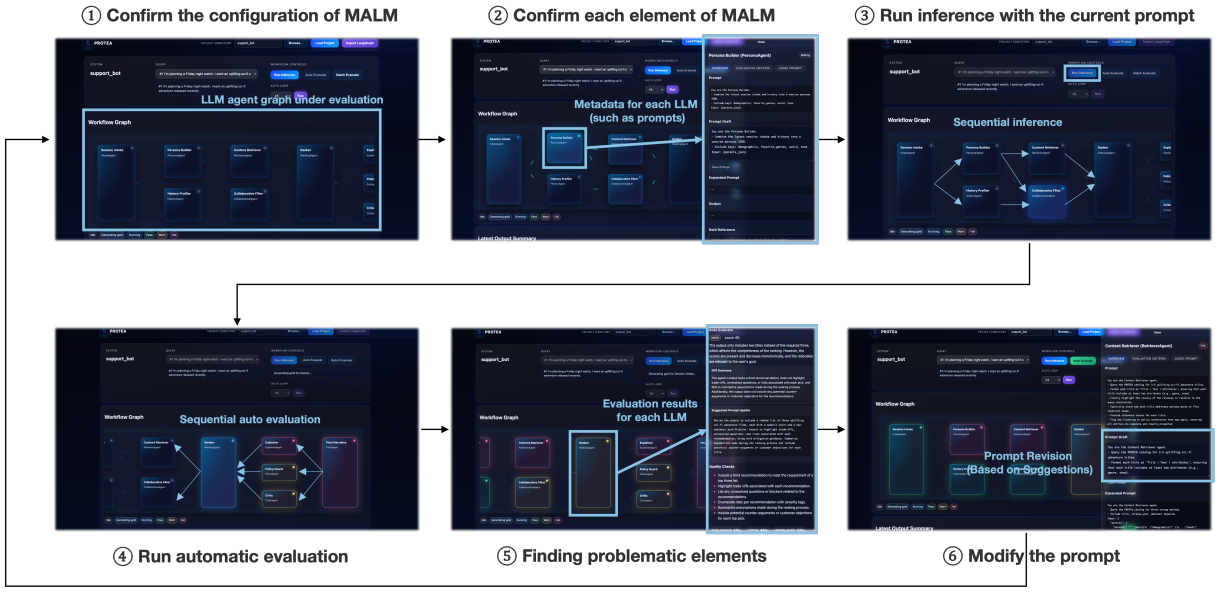

Figure 3: Six-step interface flow for improving a target multi-agent LLM workflow in PROTEA, labeled as MALM (the target multi-agent LLM workflow) in the interface: (1) load a project and confirm the agent graph, (2) inspect per-node metadata (prompts/criteria/references), (3) run inference with the current prompts on a selected test query (or a batch of queries), (4) run automatic evaluation (optionally generating node-level expectations when intermediate references are missing), (5) inspect per-node results to identify candidate bottleneck nodes, and (6) revise the selected node's prompt using the suggested draft.

node to inspect.

After evaluation, each node is annotated with its state, score, and evaluator rationale. The interface highlights nodes requiring attention and sorts the displayed node list by status, with FAIL before WARN before PASS; within the same status, nodes with lower scores are shown earlier. This ordering gives developers a starting point for inspection while keeping the final choice under their control.

When the developer selects a node, PROTEA opens an inspection panel showing the node output, the reference used for evaluation, the evaluator rationale, and the suggested prompt revision. This design keeps the developer in control while reducing the need to read the full execution trace manually.

3.3 Prompt Refinement and Re-evaluation

To support the end-to-end iteration loop, PROTEA provides prompt refinement assistance for nodes selected by the developer. Given a node v, the prompt-revision step $\mathcal{M}_{\text{opt}}$ proposes a revised instruction I'_v.

The prompt-revision step receives the current instruction I_v, the evaluator rationale, and a short improvement suggestion derived from the node-level evaluation. It returns a rewritten prompt and a brief note explaining the change. The interface presents the result as an editable before/after comparison, so the developer can accept, revise, or discard the proposed prompt revision.

The prompt-revision step is instructed to preserve the variables used to pass the user query and parent-node outputs, keep the expected output format stable, and avoid copying test-specific content into the prompt. PROTEA also checks proposed prompts for direct copying of test questions and rejects such changes when detected. Format preservation is ultimately verified by rerunning and re-evaluating the workflow.

After a prompt revision is accepted, PROTEA reruns the full workflow on the offline test suite and evaluates the new outputs. In automatic iteration mode, PROTEA repeats the evaluate–revise–re-evaluate cycle for a fixed number of iterations. In this mode, PROTEA retains a proposed prompt revision only when repeated checks show improvement and stable behavior across the fixed suite.

4 Evaluation

We evaluate PROTEA through two offline workflow-improvement studies and a formative user study with experienced LLM developers. The workflow-improvement studies include two internal workflows close to production use, evaluated with a developer-in-the-loop protocol, and an au-

tomatic iteration stress test on independently generated workflows. For the stress test, we include a five-workflow subset with no-rewrite baselines to distinguish the effect of prompt revision from score variation caused by repeated runs. For the internal workflow studies, we report aggregate outcomes and recurring qualitative patterns while preserving project confidentiality.

4.1 Production-adjacent workflow evaluations

We report two internal case studies on workflows close to production use. Both use a developer-in-the-loop protocol: evaluation and backward expectation generation are automated, while prompt revisions are human-authored and human-approved.

Case study A: enterprise document inspection. We applied PROTEA to a high-volume document inspection workflow that must satisfy strict policy and style constraints, check dozens of criteria, and output per-criterion judgments with rationales. Initial prompts and node instructions were written by developers during prior system development (not generated for this study). The workflow is a multi-agent DAG with $N{=}5$ nodes. We evaluated on an internal manually labeled test set of several dozen documents with representative criteria, using item-level correctness (binary match to human labels). Starting from the initial multi-agent workflow, iterative refinement with PROTEA improved accuracy from 64.3% to 83.9%. In practice, many accepted revisions fell into two categories: making intermediate outputs more explicit and tightening node-specific rubrics. These changes made downstream behavior less ambiguous and evaluator rationales more actionable.

Case study B: conversational recommendation/matching. Our second case study is an internal conversational recommendation workflow that clarifies an underspecified request, generates candidate items, ranks them, and produces a final response. As in Case study A, initial prompts and node instructions were developer-written and refinement was developer-in-the-loop; the workflow DAG has $N{=}6$ nodes. Each test case includes a small set of target items, and we evaluate recommendation quality using Hit@5, defined as whether the ranked list contains at least one target item within the top five recommendations. Across iterative refinement cycles in PROTEA, Hit@5 improved from 0.30 to 0.38. This workflow uses final-answer references as the primary supervision signal; backward node evaluation and per-node rationales helped trace how constraints propagated through the graph, and localized prompt revision reduced the need to read long traces end-to-end.

4.2 Automatic iteration stress test

The internal workflow evaluations above test PROTEA in developer-in-the-loop settings, where humans inspect and approve prompt revisions. We also tested PROTEA's automatic iteration mode, AUTO LOOP, where the system proposes prompt revisions and retains them only when repeated offline checks show improvement.

We collected eleven multi-agent workflows that were not used during PROTEA development. Each workflow was generated independently by an LLM using only the usage documentation. All eleven workflows executed end-to-end without execution errors. For quantitative analysis, we focus on five workflows whose initial node prompts were deliberately minimal, leaving room for automatic refinement. The five workflows covered HTTP log triage, course scheduling, incident-ticket generation, multi-step word problems, and refuse-or-clarify support responses.

For each workflow, we ran AUTO LOOP for three iterations on the first query. As a no-rewrite baseline, we also ran the original workflow three times with prompt rewriting disabled. This baseline estimates score variation without prompt revisions.

Table 1 reports the no-rewrite baseline, the best score observed during automatic iteration, and the corresponding gain. Overall, four of the five minimal-prompt workflows achieved higher best scores than the no-rewrite baseline in this small stress test, three of those four achieved gains above 0.3 score units, and one workflow did not improve. The largest improvement occurred on the course-scheduling workflow, where the score increased from 0.186 to 0.800. The word-problem workflow retained its initial score because its numeric-correctness criterion was effectively exact-match: incorrect final answers received near-zero credit even when intermediate reasoning was partially useful. As a result, the optimizer received little graded signal indicating which prompt revisions moved the workflow closer to the correct answer.

4.3 Formative user study with experienced LLM developers

We ran a formative user study in which participants used PROTEA to debug and refine a con-

Workflow	No-rewrite baseline	Best score	Gain
Log triage	0.307±0.029	0.648	+0.341
Scheduling	0.186±0.001	0.800	+0.614
Incident ticket	0.333±0.110	0.840	+0.507
Refuse/clarify	0.208±0.027	0.390	+0.182
Word problem	0.000±0.000	0.000	0.000

Table 1: Automatic iteration stress test. Baseline is mean±std over three no-rewrite runs; best score is the best three-iteration AUTO LOOP score.

versational recommendation workflow and then provided feedback via a short questionnaire and open-ended comments. Six experienced LLM developers participated. The goal was to identify useful interface features and practical integration needs. We provided a pre-configured workflow and a small offline test suite with examples exhibiting failures such as missing hard constraints or incorrect item ranking; participants diagnosed the likely failing node, applied or edited a suggested prompt revision, and verified changes via before/after comparisons and score trajectories in the history view.

Participants expressed interest in PROTEA for offline iteration and debugging, especially for workflows with many nodes and intermediate outputs. They highlighted (i) graph-level localization for narrowing down candidate nodes, (ii) per-node rationales that make outcomes actionable, and (iii) the ability to adopt prompt revisions through before/after comparison views with automatic reruns.

Participants also identified priorities for adoption: evaluator calibration and generalization checks (alignment of LLM-as-a-judge scores and backward-inferred expectations across data and stochastic runs), integration with existing toolchains (importing graphs/prompts, running regression suites, exporting prompt versions for CI), team operation and reproducibility (clearer provenance and versioning), and the cost/latency of repeated offline evaluation on large suites. These priorities align with PROTEA's developer-in-the-loop use and motivate calibrated evaluation, richer regression summaries, and smoother pipeline integration.

5 Conclusion

We presented PROTEA, a unified interface for offline, test-driven refinement of multi-agent LLM workflows, combining node-level evaluation (including backward reference generation), graph-based diagnosis support, and editable prompt revisions with automatic re-evaluation. In our case studies, PROTEA improved document inspection accuracy from 64.3% to 83.9% and increased Hit@5 from 0.30 to 0.38. In an automatic iteration stress test, eleven independently generated workflows executed end-to-end without execution errors; in the five-workflow subset with deliberately minimal initial prompts and no-rewrite baselines, four workflows achieved higher best scores than the no-rewrite baseline in this small stress test, and three achieved gains above 0.3 score units. A formative user study with six experienced LLM developers indicated interest in the approach and highlighted priorities for calibrated evaluation, smoother integration with existing toolchains, and support for richer workflow structures and structured outputs.

Limitations

PROTEA is designed primarily for developer-in-the-loop iteration on fixed workflow graphs, with AUTO LOOP supporting repeated prompt refinement over a fixed regression suite. Rubric-based LLM evaluation benefits from careful evaluator design; judge calibration, multi-judge agreement, and targeted human review loops can further strengthen score stability.

The automatic stress test also showed that near-binary criteria can provide limited feedback to AUTO LOOP. When a numeric task gives near-zero credit to all incorrect final answers, the optimizer has limited ability to distinguish partially better prompt revisions. This can be mitigated by adding partial-credit criteria for intermediate facts, reasoning setup, constraint satisfaction, or format validity, while keeping exact final correctness as only one component of the rubric.

The current prototype targets DAG-based workflows, with cyclic control flow, supervisor-based coordination, and long-running interactive agents as natural extensions of the execution and visualization model. PROTEA currently focuses on localized prompt revisions within a fixed workflow; future versions can support architectural edits such as adding nodes and rewiring dependencies, together with cost-aware prompt compression.

Larger controlled studies measuring time-to-fix and regression rates, as well as broader stress tests across backbone models and workflow suites, will further characterize the developer workflow and its relation to automated prompt-optimization methods.

References

Arize AI. 2024. Arize Phoenix. https://github.com/Arize-ai/phoenix. Accessed: 2026-05-14.

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158, St. Julians, Malta. Association for Computational Linguistics.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2023. A framework for few-shot language model evaluation. Zenodo. https://zenodo.org/records/10256836. Version v0.4.0; Accessed: 2026-05-14.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2024. A survey on LLM-as-a-judge. *arXiv preprint arXiv:2411.15594*.

Jeffrey Ip and Kritin Vongthongsri. 2026. deepeval. https://github.com/confident-ai/deepeval. Version 4.0.2; Accessed: 2026-05-14.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations (ICLR)*.

LangChain. 2024a. LangGraph. https://docs.langchain.com/oss/python/langgraph/overview. Accessed: 2026-05-14.

LangChain. 2024b. LangSmith. https://docs.langchain.com/langsmith/home. Accessed: 2026-05-14.

LangChain. 2025. OpenEvals. https://github.com/langchain-ai/openevals. Accessed: 2026-05-14.

Langfuse. 2024. Langfuse. https://github.com/langfuse/langfuse. Accessed: 2026-05-14.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-EVAL: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, pages 46534–46550.

OpenAI. 2023. OpenAI evals. https://github.com/openai/evals. Accessed: 2026-05-14.

PromptLayer. 2024. PromptLayer. https://promptlayer.com/. Accessed: 2026-05-14.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*.

Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. 2024. ARES: An automated evaluation framework for retrieval-augmented generation systems. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 338–354, Mexico City, Mexico. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8634–8652.

Ian Webster, Michael D'Angelo, Steven Klein, Guangshuo Zang, and Faizan Minhas. 2025. promptfoo. https://github.com/promptfoo/promptfoo. Version 0.119.11; Accessed: 2026-05-14.

Weights & Biases. 2024. Weave. https://github.com/wandb/weave. Accessed: 2026-05-14.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2024. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Conference on Language Modeling*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *International Conference on Learning Representations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*.