FROM TRACES TO JUDGES: AUTOMATIC DESIGN OF EVALUATION PIPELINE FOR LLM-BASED MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating multi-agent systems from their execution traces is hard: traces are long, failure modes are diverse, and existing methods rely on fixed, benchmark-specific judges or manual prompt engineering, so each new benchmark needs a new judge.

We introduce **AutoJudge**, a framework that constructs an adaptive pool of LLM judges inside a fixed evaluation graph. Given a trace, an evaluation taxonomy, and an output schema, a MetaAgent writes the roles and instructions of a set of intermediate judges and a final aggregation judge, enabling input-adaptive evaluation without task-specific training or judge design. To handle long traces, we propose a dual strategy combining full-trace evaluation with a summary-based mode and controlled step-level access to the original trace.

Across seven benchmarks with heterogeneous taxonomies, AutoJudge matches or exceeds the published benchmark-specific evaluator on 14 of 19 within-benchmark metrics with a fixed, low-cost judging backbone and no benchmark-specific judge design, while specialized judges, trained or run on stronger backbones, remain ahead on several metrics. Results are stable across repeated runs, and the framework runs at inference time without training.

Our code is available at https://anonymous.4open.science/r/AutoJudge-B0E0

1 INTRODUCTION

Multi-agent systems (MAS) are increasingly deployed for complex problem-solving, yet their evaluation remains challenging Li et al. (2024); Ma et al. (2024); Liu et al. (2023). Unlike single-agent systems and standalone LLM applications, MAS exhibit behaviors that emerge from how agents coordinate and split the task, so end-to-end task metrics alone do not reveal whether a final outcome reflects robust planning and tool use or fragile intermediate decisions. In the setting we study here, the practical evaluation problem is broader still: for a new benchmark, trace format, target output schema, or agent architecture, one must decide not only how to score trajectories, but also how to design the judge itself. Our premise is that the judge pool should not be written once per benchmark: inside a fixed evaluation graph, the judges' roles and instructions can be constructed for each trace from the task, the system structure, and the failure modes the trace exposes.

Recent trace-based methods like TRAIL Deshpande et al. (2025), Who&When Zhang et al. (2025), Aegis Kong et al. (2026), AgentErrorBench Zhu et al. (2025), PIPA Kim et al. (2026) and AgentRewardBench Lù et al. (2025) address specific failure attribution and categorization in agent traces. Benchmark suites including AgentBench Liu et al. (2024), TravelPlanner Xie et al. (2024), and τ-bench Yao et al. (2024) span diverse agent tasks and interaction patterns. Yet these judge designs remain tied to particular benchmarks, architectures, or annotation schemes, requiring manual redesign for new evaluation settings.

We introduce AutoJudge, a practical, inference-time framework for automatic generation of LLM-based judges from execution traces. It needs no training and adapts to new taxonomies and output schemas. Given a trace (or summary), taxonomy, and output schema, a MetaAgent writes the roles and instructions of several intermediate judges, which run in parallel and feed one aggregation judge in a fixed graph, adapting the judge pool to the input rather than reusing static templates. The framework supports full-trace evaluation, summary mode with step-level retrieval, and both per-trace

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

and dataset-level generation. Per-trace pipelines outperform dataset-level reuse on Who&When. AutoJudge matches or exceeds the published benchmark-specific evaluators on 14 of 19 metrics across attribution, localization, error categorization, and reward benchmarks, motivating per-trace judge-pool construction over fixed judges, while domain-specific optimized judges remain stronger in several settings.

2 Related Work

2.1 Benchmarks and Production Evaluation Frameworks

Standard benchmarks such as SWE-bench Jimenez et al. (2024), GAIA Mialon et al. (2024), and MMLU-Pro Wang et al. (2024) report end-to-end success rates that mask intermediate failures, coordination issues, and error modes in multi-agent traces. Production frameworks (Arize Phoenix Arize AI (2026), DeepEval Confident AI (2026), PydanticAI Pydantic Services Inc. (2026), Maxim AI Maxim AI (2026), Langfuse Langfuse (2026)) provide LLM-as-judge metrics such as faithfulness, correctness, toxicity, and tool-call checks through configurable rubrics, but they lack multi-agent system-level metrics such as inter-agent attribution, and all operate through static, user-specified pipelines.

2.2 Trace Benchmarks and Judge Architectures

Recent work increasingly evaluates multi-agent systems through execution traces rather than only end-to-end task success (Table 6). Most benchmarks are coupled with their own task-specific judge architectures, evaluation taxonomies, and output protocols.

Who&When studies ReAct-style tool-augmented reasoning pipelines and introduces prompt-based judges for agent-level and step-level failure attribution. TRAIL focuses on web navigation, coding, and environment-interaction agents, using hierarchical prompting to localize planning, environment, and tool-use failures. AgenTracer Zhang et al. (2026) targets agent- and step-level localization of the decisive error: it annotates failed trajectories through counterfactual replay and programmatic fault injection, and trains a dedicated 8B tracer with reinforcement learning.

Several benchmarks emphasize coordination and multi-agent interaction quality. Aegis evaluates heterogeneous MAS settings spanning QA, mathematical reasoning, coding, and debate systems through rubric-decomposed judges assessing competence, coordination, and validation correctness. AgentFail Ma et al. (2025) similarly studies system-level failure attribution using hierarchical evaluators and specialized judge ensembles. AgentErrorBench frames evaluation as taxonomy-driven error classification over memory, planning, reflection, action, and tool/API failures in interactive agents.

Another line of work formulates evaluation as reward-style trajectory assessment. AgentRewardBench evaluates web and workstation agents using scalar or binary reward-style judges over complete trajectories, while PIPA proposes unified evaluation protocols for interactive planning agents across heterogeneous tasks. Pumpkin Zhidkovskaya et al. (2026) explores ensemble-based judging for heterogeneous multi-agent pipelines, combining 10 independent LLM judges with a final aggregation judge for binary trajectory-level classification.

Despite this diversity, existing methods rely on fixed prompts or benchmark-specific pipelines, tied to one trace format and taxonomy and hard to transfer. They also struggle with ultra-long traces, where truncation loses intermediate reasoning and full-context evaluation becomes prohibitively expensive. None adapts its evaluation logic to failure modes it was not written for.

Unlike automatic prompt-engineering methods such as APE Zhou et al. (2023), OPRO Yang et al. (2024), and DSPy Khattab et al. (2024), which iteratively optimize instructions or pipeline parameters against a metric and labeled development examples, AutoJudge performs trace-conditioned judge-pool construction at inference time, with no optimization loop and no labeled development data required. It also differs from benchmark-specific judges such as Who&When and TRAIL, which fix both the evaluation criteria and the judge architecture to one benchmark: AutoJudge instead takes the taxonomy and schema as input, so the same graph serves a new benchmark without being redesigned by hand.

3 PROPOSED APPROACH

Given an execution trace, a MetaAgent generates a structured set of judges that evaluate different aspects of the trajectory, and an aggregation judge combines their outputs into a final decision.

3.1 PROBLEM SETUP AND DESIGN PRINCIPLES

We consider the problem of evaluating execution traces produced by LLM-based agent systems. Each trace is a sequence of steps representing agent actions, tool calls, and intermediate reasoning.

Formally, the input to our system consists of: **Trace** T (full execution trace or summarized representation), **Taxonomy** $\mathcal{C}$ (evaluation dimensions such as task completion, reasoning quality, or tool usage), and **Output schema** $\mathcal{S}$ (structured specification of the desired output format, e.g., JSON fields and labels).

The goal is to construct an evaluation function

$$f : (T, \mathcal{C}, \mathcal{S}) \to y,$$

where y is a structured evaluation output conforming to schema $\mathcal{S}$.

Instead of manually designing this function, we generate its judges automatically: a MetaAgent writes the roles and instructions of multiple LLM-based judges, which are then composed in a fixed evaluation graph.

Multi-agent scope. The orchestration layer itself is agent-count-agnostic. What makes the evaluation multi-agent is the input and the schema: the trace preserves agent identities, roles, messages, tool interactions, and task transfers, and the output schema can request coordination quality, role consistency, or culprit attribution. Single-agent traces can be processed the same way, but every evaluation in this paper targets a MAS.

Design Principles. Our framework follows four principles: (1) **minimal user input**: only the evaluation taxonomy $\mathcal{C}$ and output schema $\mathcal{S}$ are required; (2) **automatic pipeline construction**: evaluation logic is generated dynamically as a multi-judge pipeline; (3) **schema-driven evaluation**: $\mathcal{S}$ defines the final decision structure, keeping the evaluation logic unchanged across benchmarks; (4) **MetaAgent-driven judge construction**: judge prompts and roles are created dynamically after analyzing the input trace.

3.2 SYSTEM OVERVIEW

The system consists of three core components: (1) an input preparation layer that processes the trace, (2) a MetaAgent that generates the evaluation pipeline, and (3) an execution layer that instantiates and runs the generated judges.

The input trace is optionally preprocessed; the MetaAgent then produces a structured specification of several intermediate judges followed by a final aggregation judge; during execution the intermediate judges emit structured outputs, which the aggregation judge synthesizes into the final decision.

3.3 INPUT PREPARATION LAYER (FULL TRACE VS SUMMARY MODE)

Execution traces can exceed millions of tokens, beyond standard context windows. In full trace mode, the raw trace is passed directly to the MetaAgent and all judges, preserving maximal fidelity at high cost. In summary mode, each step is compressed while the full trace is stored externally; judges operate on the summary and can retrieve specific steps through a retrieval tool by step identifier, while unrestricted access to the full trace is disallowed. Unlike Who&When and TRAIL, which assume full-context availability or use static truncation, this combines structured compression with on-demand retrieval.

3.4 METAAGENT: JUDGE-POOL GENERATION

The MetaAgent is responsible for constructing the judge pool, i.e., generating a structured set of LLM-based judges. Their interaction pattern is fixed and is the same in every reported experiment:

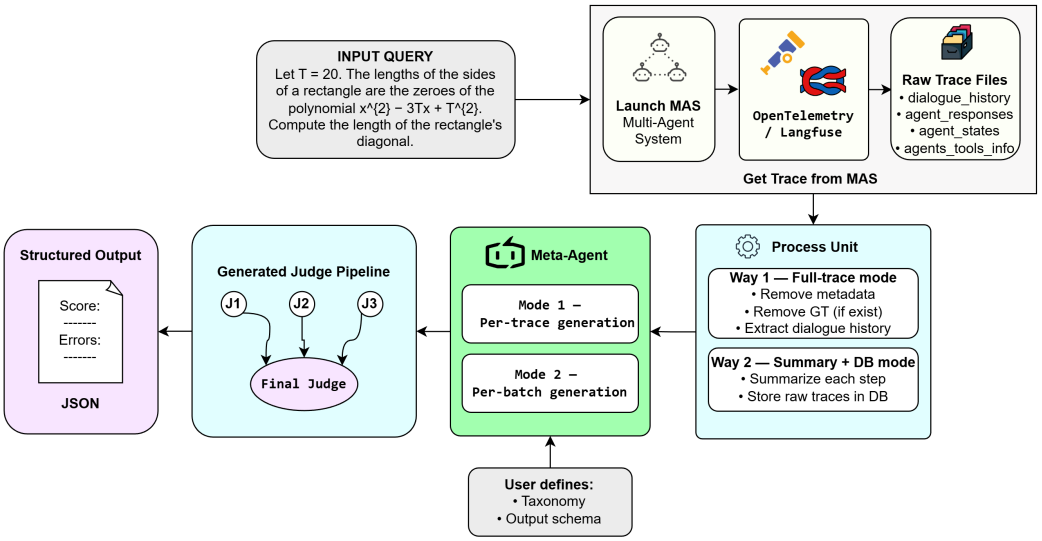


Figure 1: Overview of the AutoJudge pipeline.

intermediate judges run in parallel and feed a single aggregation judge, which outputs the final evaluation. The MetaAgent does not search over or change this graph.

Pipeline Structure. The generated pipeline includes: (1) intermediate judges, each responsible for evaluating a specific aspect of the trace, and (2) an aggregation judge, which consumes intermediate outputs and produces the final decision.

Operating Modes. We consider two operating modes: (1) **per-trace judge-pool generation (default)**: the MetaAgent generates a separate pool for each trace (using full or summarized traces), so the judges adapt to that trace; (2) **dataset-level judge-pool generation (budget mode)**: a single reusable pool is built from a small random subset of traces (three in our implementation) to capture coarse dataset characteristics, reducing cost but potentially lowering evaluation quality.

A schematic comparison is shown in Figure 2.

Prompting Interface. The MetaAgent operates using a fixed prompt template with two user-defined components: (1) **taxonomy**: evaluation dimensions such as correctness, reasoning quality, or tool usage; and (2) **output schema**: the desired structure and format of the final output.

These components allow users to control evaluation criteria and output format without adjusting to the benchmark. The MetaAgent uses them to generate judges aligned with the desired evaluation objectives. In total, the researcher supplies the trace, the taxonomy, the output schema, and the fixed prompt template with its few-shot examples; the MetaAgent supplies only the judges' roles and instructions. The prompting scheme is fixed rather than trained: a general instruction defines universal criteria for multi-agent traces (task completion, reasoning quality, tool use, environment or API failures), few-shot examples illustrate these criteria through reusable diagnostic categories, and the output schema carries all benchmark adaptation by specifying the structure and labels of the final prediction. Adapting to a new benchmark therefore requires only a new schema, with the evaluation logic left inspectable and unchanged.

An example of a full pipeline is illustrated in Figure 1. A complete run from trace to verdict, with the generated judge instructions, their structured outputs, the aggregator input, and the final verdict, is condensed in Appendix C; the full record is `example.md` in the repository.

The final evaluation is produced by an aggregation judge that jointly reasons over three sources of information: (1) the execution trace (either full or summarized with retrieval access), (2) the outputs of all intermediate judges, and (3) the user-defined output schema. Unlike intermediate judges, which

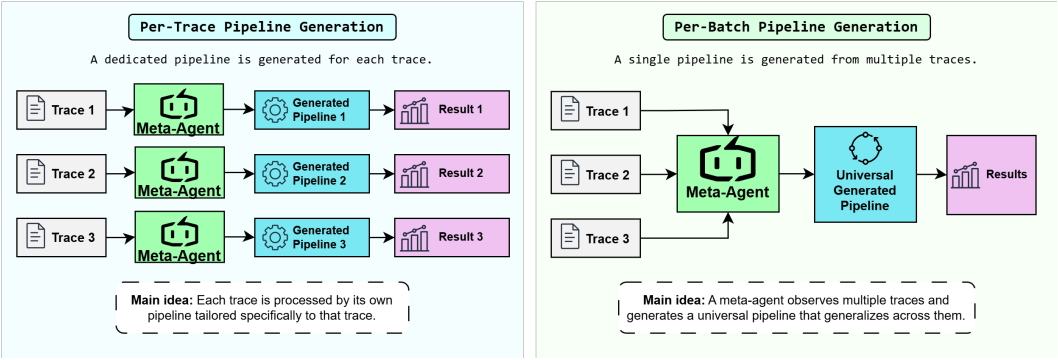


Figure 2: Comparison of per-trace and dataset-level judge-pool generation (labelled "per-batch" in the figure).

focus on localized aspects of the trajectory, the aggregation judge performs global synthesis over all available signals.

The aggregation judge does not operate as a simple voting mechanism over intermediate outputs. Instead, it performs schema-conditioned reasoning that integrates structured judgments with evidence from the original trace, so it can overrule an intermediate judge by returning to the trace, as in the legal-research case of Appendix D.

Local evidence extraction and the global decision are thus separate steps, and the final judge can still revisit the raw trajectory. Appendix C shows the aggregator's input and output for one trace.

3.5 Implementation

AutoJudge is implemented as a modular LLM-based evaluation system built on PydanticAI, which provides structured orchestration and strict schema validation for all LLM outputs. Every LLM component in the system is executed as a schema-constrained call with automatic retries on format violations, so every output is valid structured data.

We adopt a hierarchical model allocation strategy. The **MetaAgent** is instantiated using Gemini 3 Flash Preview, selected for its stronger reasoning capabilities required for judge-pool synthesis. All evaluation judges, including intermediate and aggregation components, are powered by Gemini 2.5 Flash, which keeps large-scale inference cheap.

For long execution traces, we introduce a preprocessing **summarization module**, implemented using Gemini 2.5 Flash, which compresses raw traces into structured summaries before evaluation. This module is strictly decoupled from the judging process and is not part of the evaluation logic. Full traces are stored in a **PostgreSQL** database for controlled step-level retrieval during evaluation via a constrained API. Judges operating in summary mode can query individual trace steps, while unrestricted access to the full trace is explicitly disallowed to enforce localized reasoning.

The system is executed as a runtime graph with a fixed topology: the MetaAgent first generates the judge pool, which is then instantiated and executed via PydanticAI. All judges operate as stateless LLM calls conditioned on the trace (or its summary), evaluation taxonomy, and output schema. Intermediate outputs are validated and passed to a final aggregation judge, which produces the structured evaluation result. The API returns the final verdict, the generated pool, and the graph; the intermediate judges' diagnostics are stored in the pipeline trace and can be exposed through an optional structured output.

4 Experimental Studies

We evaluate AutoJudge across multiple benchmarks of LLM-based agent execution traces, each with its own taxonomy and scoring scheme, aiming to show that it adapts to diverse evaluation

criteria without task-specific judge design or retraining. In every experiment, Gemini 3 Flash Preview generates the judge pool and Gemini 2.5 Flash runs the judges, the aggregator, and the summarizer.

We structure the study around three questions: (1) the effect of design choices on evaluation quality (Who&When), (2) cross-dataset generalization across taxonomies and metrics, and (3) robustness to variations in model configuration and sampling.

4.1 Design Space Evaluation on W&W Dataset

4.1.1 Judge Pool Generation Strategy

We compare two strategies for constructing evaluation pipelines: (1) per-trace judge generation and (2) dataset-level judge pool generation.

Per-trace generation constructs a dedicated evaluation pipeline conditioned on each individual trace, while dataset-level pooling builds a single judge set from a small subset of traces and reuses it across the dataset.

Comparison. As shown in Table 8, per-trace generation consistently outperforms dataset-level pooling across both Who&When subsets, with the largest gap on the algorithmic subset, where higher behavioral diversity makes trace-specific adaptation more critical.

The final prompting configuration is selected via a separate ablation over judge ensembles (Appendix B), optimizing for agent-level accuracy.

Dataset-level pooling is cheaper but less accurate; per-trace generation is more accurate at higher cost (cost accounting in Appendix E).

4.1.2 Selected Configuration

Based on preliminary experiments, we select two configurations of our framework: (1) **AutoJudge** (per-trace judge-pool generation) and (2) **AutoJudge (summ + DB)**, which operates in summary mode with database-backed trace retrieval.

Unless otherwise stated, these configurations are used consistently across all benchmarks.

4.2 Cross-Dataset Evaluation

We evaluate AutoJudge across multiple benchmarks of agent execution traces, each defined by its own error taxonomy and evaluation protocol. All datasets consist of LLM-based agent traces, but differ in annotation schemes and metrics.

We compare three setups: (1) **AutoJudge (full trace)**, a high-cost configuration that operates on the complete trace, (2) **AutoJudge (summ + DB)**, a more efficient variant that uses summarized traces with database-backed access to the full trace, and (3) the **benchmark-specific baseline**, i.e., the evaluator published with each benchmark, reported with that paper's own backbone.

Ground truth. Four of the seven benchmarks provide human-expert labels, and each labels a different target. Who&When Zhang et al. (2025) annotates the failure-responsible agent and the decisive step. TRAIL Deshpande et al. (2025) provides human labels for trace errors and their locations. AgentErrorBench Zhu et al. (2025) was annotated by researchers experienced with LLM agents. In AgentRewardBench Lù et al. (2025), an expert reviewed every trajectory for success, side effects, and repetition. The accuracy, F1, and precision values below therefore measure agreement with these existing human judgments, not with labels produced by AutoJudge or another evaluator.

Results. Across benchmarks, the better of the two AutoJudge variants beats the benchmark-specific baseline on 14 of the 19 metrics in Table 1. Three of the five losses are within 0.10 to 2.81 points; the two larger gaps, AEGIS Error MF1 and AgenTracer Step Accuracy, expose current limits in fine-grained error classification and exact step localization. These are within-benchmark comparisons that use each benchmark's own labels and metric; we do not compare or aggregate scores across benchmarks, and each row is a separate test of the same construction mechanism. The baselines'

Table 1: Comparison of AutoJudge variants across multiple benchmarks. Best results are highlighted in green.

Dataset / Metric	AutoJudge (summ + DB)	AutoJudge (full)	Benchmark-specific baseline
WW hand (Agent Acc)	**61.49**	59.19	55.17
WW hand (Step Acc)	**17.25**	12.07	7.02
WW algo (Agent Acc)	55.55	**63.23**	54.33
WW algo (Step Acc)	41.23	**43.38**	25.51
TRAIL (GAIA) (F1)	21.35	21.26	**24.16**
TRAIL (GAIA) (Loc Acc)	26.55	**27.68**	17.05
Pumpkin small (F1)	64.52	**66.13**	64.30
Pumpkin large (F1)	64.58	**72.00**	65.70
Aegis (Pair mF1)	7.0	**8.9**	6.99
Aegis (Agent mF1)	51.93	**53.31**	42.02
Aegis (Error mF1)	22.5	**24.17**	23.47
Aegis (Pair MF1)	2.51	2.66	**2.76**
Aegis (Agent MF1)	17.1	**17.67**	16.45
Aegis (Error MF1)	12.69	14.59	**19.85**
AgentErrorBench (S Acc)	8.78	**26.93**	20.33
AgentErrorBench (S+M Acc)	2.06	7.77	**8.00**
AgentRewardBench AxTree (Precision)	**77.92**	72.73	68.5
AgenTracer Code (Agent Acc)	**75.9**	74.0	72.21
AgenTracer Code (Step Acc)	9.6	10.2	**18.85**

sources and backbones are listed in Table 7 (Appendix E), which also shows that trained or specialized judges, some with stronger backbones, remain ahead on several metrics. As shown in Table 1, the full-trace configuration generally achieves the strongest performance, particularly on datasets requiring detailed reasoning over the entire trajectory (e.g., WW Algo from Who&When, Pumpkin, and Aegis).

AutoJudge offers two practical advantages: no benchmark-specific judge design when adapting to new taxonomies or schemas, and a scalable summary+DB mode handling ultra-long traces (>2M tokens). The summary-based variant (summ + DB) remains competitive and in some cases performs best (e.g., WW Hand), at lower cost.

Analysis. Access to the full trace gives the highest accuracy. The summary-based approach stays close on Who&When, TRAIL and AEGIS at a smaller context, but not on AgentErrorBench (8.78 vs. 26.93 S Acc) or Pumpkin large (64.58 vs. 72.00).

Both variants generalize across datasets with different taxonomies and metrics, indicating that AutoJudge can adapt evaluation pipelines without dataset-specific tuning. At the same time, certain metrics (e.g., some Aegis and AgentErrorBench settings) remain challenging, where specialized judges from prior work still have an advantage.

The per-benchmark pattern matches the manually examined cases in Appendix D. On TRAIL, AutoJudge has lower error F1 but higher localization accuracy than the benchmark-specific baseline: it often finds the bad location but assigns the wrong error class. On AEGIS, agent attribution is strong while rare or adjacent error classes remain difficult. On WW Hand, summary+DB beats the full trace, plausibly because summarization removes irrelevant context while retrieval preserves the evidence that is needed. Explicit, local evidence is easier for the generated judges than distributed evidence or fine-grained separation of causes.

4.3 Comparison with Baseline Judge Pipelines

We compare AutoJudge against three fixed judge pipelines on the Who & When benchmark: a direct LLM evaluator, a standard PydanticAI judge, and ChatEval Chan et al. (2024). All methods use `Gemini 2.5 Flash` as the judge backbone, isolating the effect of pipeline design rather than model choice. The two WW judge rows re-run the evaluator released with Who&When, in both of its modes, with that backbone, and are scored with the benchmark's released `evaluate.py`.

As shown in Table 2, AutoJudge leads on step accuracy in every setting and on agent accuracy on the *hand* subset. On the *algo* subset it reaches 43.38% step accuracy versus 38.90% for ChatEval and 28.57% for the released Who&When judge, while the all-at-once mode of that judge remains stronger on agent accuracy (66.67% vs. 63.23%), a result we report as-is rather than omit. On the hand subset, AutoJudge (summ + DB) achieves the best overall performance with 61.49% agent accuracy and 17.25% step accuracy.

Table 2: Baseline judge pipelines on Who&When, all with a Gemini 2.5 Flash backbone.

System	WW hand		WW algo	
	Agent	Step	Agent	Step
PydanticAI judge	51.70	5.20	57.10	8.70
ChatEval agent	55.20	15.50	54.80	38.90
WW judge (all-at-once)	55.17	3.45	**66.67**	28.57
WW judge (step-by-step)	44.83	10.34	39.68	28.57
AutoJudge (summ + DB)	**61.49**	**17.25**	55.55	41.23
AutoJudge (full)	59.19	12.07	63.23	**43.38**

The gains are especially pronounced on the algorithmic subset, where greater trace diversity makes input-adaptive decomposition particularly beneficial. Appendix B.1 additionally runs AutoJudge with a GPT-4o backbone against the published Who&When scores. Overall, per-trace judge-pool generation improves step localization over both minimal and structured fixed evaluation workflows on this benchmark, while the released judge's all-at-once mode keeps the lead on algo agent accuracy.

4.4 Choice of Model Configuration

Based on the results in Table 10, we configure AutoJudge with Gemini 3 Flash Preview as the MetaAgent and Gemini 2.5 Flash as the judge backbone. This combination achieves strong agent-level accuracy (59.19%) with moderate step-level accuracy (12.07%) at a reasonable cost (3.9 USD), while processing all samples successfully (Lost = 0).

The stronger model is spent on the single MetaAgent call, while the judges, which consume many more tokens per trace, stay on the cheaper backbone; shifting both components to GPT-4o or Claude Sonnet 4.5 raises cost without a clear gain in evaluation quality. Qwen 3 32b and GPT-4o also lose samples to context limits in full-trace mode, which is one motivation for the summary + DB mode.

4.5 Robustness and Statistical Analysis

Over three runs per setting, judge temperatures in the 0.0–0.3 range yield similar agent-level accuracy, while 0.1 gives the highest mean step-level accuracy with lower variance than 0.0 and 0.3; MetaAgent temperatures of 0.3–0.5 perform best, with 0.3 highest on both metrics (Tables 11 and 12). Higher judge temperatures lower accuracy, and higher MetaAgent temperatures also increase run-to-run variance, so we use 0.1 for judges and 0.3 for the MetaAgent.

5 Conclusions

We introduced **AutoJudge**, a framework for automatic evaluation of LLM-based multi-agent systems in which a MetaAgent decomposes the evaluation task into specialized judges and an aggregation stage rather than relying on a fixed, benchmark-specific judge. Two design choices carry the approach: a dual input strategy that combines full-trace evaluation with a summary + DB mode for traces

beyond standard context limits, and a fixed, non-trainable prompting scheme that separates a general evaluation policy from two user-controlled components, the taxonomy and the output schema, so that a new benchmark is served by aligning with its native error definitions and output format instead of retraining.

Across seven benchmarks the experiments show that (1) per-trace judge-pool generation outperforms dataset-level reuse on Who&When, with the largest gain on the algorithm-generated subset; (2) AutoJudge generalizes across datasets with different taxonomies by directly aligning with their evaluation criteria, achieving competitive or superior results without task-specific tuning; (3) the framework remains stable under different model configurations and sampling settings, and maintains a favorable trade-off between accuracy and computational cost through summary-based processing.

LIMITATIONS

Several limitations remain. Full-trace evaluation remains challenging for extremely long execution traces, where context management without summarization can degrade reasoning over complete trajectories. The summary-based retrieval mechanism improves scalability but may occasionally omit subtle details from the original trace. In addition, highly domain-specific metrics can still benefit from manually designed judges. Finally, despite the introduced optimizations, full-trace evaluation remains computationally expensive for large-scale settings. Validation relies on benchmark ground truth: four of the seven benchmarks carry human-expert labels (Section 4.2), so the reported metrics measure agreement with existing human judgments, but we did not run an independent study with a human–human agreement ceiling.

Future work includes stronger retrieval strategies, tighter integration between intermediate judges, and hybrid approaches that pair automatic pipeline generation with lightweight adaptation to domain-specific criteria.

ETHICS STATEMENT

This work introduces AutoJudge, a framework for scalable evaluation of LLM-based multi-agent systems. It can improve benchmarking and reproducibility in AI research, but may also simplify optimization of capable agents, including in potentially risky settings. The framework inherits limitations of LLM-based judgments and summary-based trace processing, so it should be used with care, especially in high-stakes applications.

AI USE STATEMENT

AI assistants helped edit the manuscript text and its LaTeX formatting. The code, scripts and evaluation pipelines were written by the authors, with only occasional use of AI coding assistants. The LLMs that AutoJudge runs and compares against are the object of study, not writing aids. The authors reviewed all AI-assisted content and take full responsibility for the paper, its claims and its artifacts.

REFERENCES

Arize AI. Phoenix. https://phoenix.arize.com/, 2026. Source code available at https://github.com/Arize-ai/phoenix; accessed 2026-05-06.

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. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=FQepisCUWu. arXiv:2308.07201.

Confident AI. DeepEval. https://deepeval.com/, 2026. Source code available at https://github.com/confident-ai/deepeval; accessed 2026-05-06.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. TRAIL: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025. URL https://arxiv.org/abs/2505.08638.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.06770. arXiv:2310.06770.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od. arXiv:2310.03714.

Takyoung Kim, Janvijay Singh, Shuhaib Mehri, Emre Can Acikgoz, Sagnik Mukherjee, Nimet Beyza Bozdag, Sumuk Shashidhar, Gokhan Tur, and Dilek Hakkani-Tür. AURA: A diagnostic framework for tracking user satisfaction of interactive planning agents. *IEEE Transactions on Audio, Speech and Language Processing*, 34:3767–3777, 2026. doi: 10.1109/taslpro.2026.3695295. arXiv:2505.01592; version 1 was titled "PIPA: A Unified Evaluation Protocol for Diagnosing Interactive Planning Agents".

Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=zqcYoxXiN3. arXiv:2509.14295.

Langfuse. Langfuse. https://langfuse.com/, 2026. Source code available at https://github.com/langfuse/langfuse; accessed 2026-05-06.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024. doi: 10.1007/s44336-024-00009-2. URL https://doi.org/10.1007/s44336-024-00009-2.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2308.03688. arXiv:2308.03688.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2511–2522, 2023. doi: 10.18653/v1/2023.emnlp-main.153. URL https://aclanthology.org/2023.emnlp-main.153/.

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. In *Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2504.08942. arXiv:2504.08942.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=4S8agvKjle. arXiv:2401.13178.

Xuyan Ma, Xiaofei Xie, Yawen Wang, Junjie Wang, Boyu Wu, Mingyang Li, and Qing Wang. Demystifying the lifecycle of failures in platform-orchestrated agentic workflows. *arXiv preprint arXiv:2509.23735*, 2025. URL https://arxiv.org/abs/2509.23735. Introduces the AgentFail dataset. Version 1 was titled "Diagnosing Failure Root Causes in Platform-Orchestrated Agentic Systems: Dataset, Taxonomy, and Benchmark".

Maxim AI. Maxim AI. https://www.getmaxim.ai/, 2026. Accessed 2026-05-06.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2311.12983. arXiv:2311.12983.

Pydantic Services Inc. Pydantic AI. https://ai.pydantic.dev/, 2026. Source code available at https://github.com/pydantic/pydantic-ai; accessed 2026-05-06.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 37, 2024. URL https://arxiv.org/abs/2406.01574. arXiv:2406.01574.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.01622. arXiv:2402.01622.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2309.03409. arXiv:2309.03409.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=l05DseqvuD. arXiv:2509.03312.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? On automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2505.00212. arXiv:2505.00212.

Alina Zhidkovskaya, Kirill Rapatskikh, Jerzy Kaminski, Grigorii Barakhsin, Anna V. Kalyuzhnaya, Alexey Druzhinin, Andrey Savchenko, Julia Belikova, Konstantin Polev, and Nikolay O. Nikitin. Towards design of an automated judge for multi-agent systems. In *Trustworthy Agentic AI Workshop at the AAAI Conference on Artificial Intelligence (AAAI)*, 2026. URL https://trustagenticai.github.io/AAAI2026/AAAI-Workshop/66.pdf.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=YVozFftExy. arXiv:2211.01910.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. Where LLM agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025. URL https://arxiv.org/abs/2509.25370.

APPENDIX

A PROMPTS

MetaAgent prompt

You are an AI judge pool generator specialized in creating evaluation pipelines for multi-agent systems. Your goal is to design a team of specialized judges that detect problems, errors, and quality issues in system execution.

DESIGN PRINCIPLES:
- START SIMPLE: Create the minimum number of judges needed to detect key problems
- Prefer 3-9 specialized judges that cover different error domains
- Add more judges only when:
* Independent problem categories can be assessed in parallel (e.g., API errors vs environment setup)
* Different quality dimensions need separate evaluation (correctness, efficiency, reliability)
* Specific failure modes require dedicated detection logic
- Avoid over-engineering: one comprehensive problem detector > multiple narrow similar judges

YOU CAN FOLLOW NEXT TAXONOMY:
{taxonomy}

EXAMPLES:
{examples}

RULES:
- Ensure all judge names are unique and descriptive (end with _JUDGE)!
- You should always ask judges to use tool (get_content_tool) to recive a context (1 or 2 times per judge)!
- Instructions must include explicit scoring criteria (ideal/fair/poor)
- Each judge returns JSON with justification + score
- Focus judges on detecting specific problem categories: task failures, API errors, setup issues, tool misuse, etc.
- Include TOOL_SELECTION_JUDGE or TOOL_PERFORMANCE_JUDGE when evaluating tool-based systems
- Always include FINAL_AGGREGATOR as the final judge that synthesizes all findings into binary score (poor/ideal) and justification

CRITICAL: TOOLS:
Force the court to use tools! Be sure to specify in the prompt that they should call the tool!!! But, FINAL_AGGREGATOR should not use tools!

ATTENTION CRITICAL: FINAL_AGGREGATOR NAME:
FINAL_AGGREGATOR must be named exactly "FINAL_AGGREGATOR" (case-sensitive)

ATTENTION CRITICAL: FINAL_AGGREGATOR OUTPUT FORMAT:
FINAL_AGGREGATOR must include these exact instructions at the end (it is important that it has the same output format as indicated below):
{judge_output_format}

B FEW-SHOT PROMPTING ABLATION

To study the effect of few-shot prompting design, we conducted an ablation over six empirically selected combinations of metric-specific judges. The experiment was performed on 58 long hand-

Table 3: Ablation study of judge ensembles on handcrafted traces (58 long traces).

Ensemble	Agent Acc. (%)	Step Acc. (%)
MAS_TASK_COMPLETION_JUDGE, MAS_COMPLEXITY_JUDGE, TOOL_PERFORMANCE_JUDGE, MAS_ENVIRONMENT_SETUP_JUDGE, MAS_API_ISSUES_JUDGE, TOOL_SELECTION_JUDGE	**59.19**	3.51
MAS_TASK_COMPLETION_JUDGE, MAS_COMPLEXITY_JUDGE, TOOL_PERFORMANCE_JUDGE, TOOL_SELECTION_JUDGE	56.90	6.90
MAS_TASK_COMPLETION_JUDGE, MAS_COMPLEXITY_JUDGE	56.14	10.53
MAS_COMPLEXITY_JUDGE, MAS_ROLE_COVERAGE_JUDGE	58.62	8.77
MAS_TASK_COMPLETION_JUDGE, MAS_COMPLEXITY_JUDGE, TOOL_PERFORMANCE_JUDGE	58.62	8.62
MAS_TASK_COMPLETION_JUDGE, GUIDELINE_CONSISTENCY_JUDGE	55.17	3.51

crafted traces and evaluated using two complementary metrics: *Agent Accuracy*, which measures whether the final trajectory-level outcome is classified correctly, and *Step Accuracy*, which measures agreement at the level of individual intermediate steps.

The primary objective of this study was to identify a prompting configuration that provides reliable agent-level evaluation. Accordingly, *Agent Accuracy* was treated as the primary selection criterion, as the judge is ultimately used to determine whether an agent successfully completes the task, rather than to provide dense supervision for each intermediate step.

The results are presented in Table 3. The highest Agent Accuracy is achieved by the largest ensemble, consisting of MAS_TASK_COMPLETION_JUDGE, MAS_COMPLEXITY_JUDGE, TOOL_PERFORMANCE_JUDGE, MAS_ENVIRONMENT_SETUP_JUDGE, MAS_API_ISSUES_JUDGE, and TOOL_SELECTION_JUDGE, reaching 59.19%. Although this configuration yields relatively low Step Accuracy, this trade-off is acceptable given the focus on trajectory-level correctness.

These results suggest that broad diagnostic coverage is beneficial for evaluating long agent traces. Failures may arise from multiple sources, including incomplete task execution, excessive interaction complexity, incorrect tool usage, environment setup issues, API failures, or suboptimal tool selection. Smaller ensembles tend to capture only a subset of these failure modes, whereas larger ensembles provide a more comprehensive diagnostic signal, improving agent-level evaluation.

Based on this analysis, we select the six-judge ensemble as the final few-shot prompting configuration. While it is less suitable for fine-grained step-level annotation, it is better aligned with the primary objective of reliable trajectory-level evaluation.

B.1 AutoJudge with a GPT-4o Backbone versus the Published Who&When Judge

Both systems use GPT-4o as the judge backbone here, but unlike Table 2 the Who&When numbers are the scores published by the benchmark paper (the benchmark-specific baseline values of Table 7), not a re-run under our protocol. We further compare AutoJudge against the strongest reported configuration of the Who&When (WW) judge. Since WW achieves its best performance when using GPT-4o as the backbone, we run AutoJudge with the same GPT-4o backbone to ensure a controlled comparison under the strongest available setting for this baseline. As shown in Table 4, AutoJudge substantially outperforms WW on both evaluation levels: 67.24% vs. 55.17% in Agent Accuracy and 17.24% vs. 7.02% in Step Accuracy. These results show that AutoJudge's gains are not simply due to using a stronger LLM backbone. Even when both methods are evaluated with GPT-4o, the best-performing WW configuration remains considerably less accurate, especially at the step level. This suggests that AutoJudge's evaluation procedure provides a more effective framework for MAS evaluation, improving both agent-level judgment and fine-grained assessment of intermediate reasoning steps.

Table 4: AutoJudge run with GPT-4o versus the published Who&When judge scores, which also use GPT-4o.

Judge	Agent Acc.	Step Acc.
AutoJudge (GPT-4o)	**67.24**	**17.24**
WW judge (GPT-4o, published)	55.17	7.02

C End-to-End Example

This appendix condenses `example.md` from the repository: one trace evaluated by AutoJudge in summary mode, from the evaluated MAS to the final verdict. The full record contains the complete trace text, every generated instruction, and every structured output.

Evaluated MAS and task. Five agents (`DataAnalysis_Expert`, `Verification_Expert`, `Computer_terminal`, `RealEstateMarket_Expert`, and a terminating `DataAnalysis_Expert` turn) produce a seven-step trace for the query "What's the smallest house (based on square footage) that has at least 2 beds and 2 baths and was sold in Prince Edward Island between June 1, 2022 and May 15 2024 according to Zillow?" At step 3 the verification script fails with a `FileNotFoundError`; at step 4 `Verification_Expert` generates a synthetic dataset instead of retrieving real data; steps 5–6 verify the function on that synthetic data; step 7 terminates without an answer to the query.

Generated judge pool. From the trace, the taxonomy, and the culprit-attribution schema, the MetaAgent wrote two intermediate judges and the aggregator. Each intermediate judge receives the query and the step-by-step summary and returns `{score, justification}` with score in {ideal, fair, poor}.

- SEARCH_INTEGRITY_JUDGE. Criteria: factuality (real Zillow data or synthetic placeholders), constraint satisfaction (date range, location, beds/baths), task completion. Output: **poor**: "The system failed to search for real data. It only verified a function on synthetic data and then terminated without providing an answer to the user's query about a real-world house on Zillow."
- EXECUTION_STRATEGY_JUDGE. Criteria: error recovery after the `FileNotFoundError` in step 3, logic soundness of substituting synthetic data, tool usage. Output: **poor**: "The agents completely bypassed the actual task of finding real-world Zillow data. When faced with a `FileNotFoundError` for a sample dataset, instead of attempting to find or generate a real dataset or use tools to access Zillow data, they pivoted to generating a synthetic dataset."
- FINAL_AGGREGATOR. Instruction: synthesize the two findings, identify the primary failure point and the responsible agent, and return `{agent, step, reason}` with the agent name exactly as it appears in the trace and the step as a plain integer. Input: the two judge outputs above (as `parent_nodes`) and the trace summary.

Final verdict. `agent = Verification_Expert`, `step = 4`: "The `Verification_Expert` agent, in step 4, made the critical decision to pivot from attempting to find real data to generating synthetic data. This decision fundamentally changed the nature of the task from answering a real-world query to verifying a function with fake data, thereby failing the user's original intent."

D Qualitative Case Analysis

We examined four traces from the Who&When hand-crafted subset by hand, reading the generated judge roles, their justifications, and the aggregator's reasoning against the expert annotation. Table 5 summarizes them. They are qualitative checks of the generated pipelines; we make no statistical agreement claim from four cases.

Table 5: Four manually examined Who&When hand-crafted traces. "Expert" is the benchmark annotation (responsible agent, decisive step); "AutoJudge" is the aggregator verdict.

Case	Task	Expert	AutoJudge	Outcome
MTG prices	Which co-banned Magic card had the largest price drop	WebSurfer, step 12: query not concise enough, run hits the time limit	WebSurfer, step 13	Agent and mechanism (failed price extraction, timeout) match; step adjacent rather than exact.
Legal research	Word deleted in the last amendment to a federal rule on the Cornell LII site	Orchestrator, step 29: concludes without enough information instead of replanning	Orchestrator, step 32	Agent and cause match. The aggregator overrode LEGAL_RESEARCH_JUDGE, which had rated the wrong answer "ideal", using MAS_EFFICIENCY_JUDGE's evidence that the answer was a guess.
Gym schedules	Gyms within 200 m of Tompkins Square Park with classes before 7 am	WebSurfer, step 4: search results miss a required gym	WebSurfer, step 21	Agent matches. Three judges independently identify the navigation loop and the terminal crash at step 21; the expert's decisive step is earlier.
Freon volume	Volume of 0.312 kg Freon-12 at the bottom of the Marianas Trench	Assistant, step 51: calculation with a fabricated density	WebSurfer, step 21	Negative case. AutoJudge blames the upstream Cloudflare access block, rates the estimated density "fair", and never names the calculating agent.

E ADDITIONAL DETAILS

Table 6: Benchmarks for evaluating LLM-based judges: MAS trace sources, judge architectures, and evaluation signals.

Benchmark	MAS architecture (trace source)	Judge architecture	Evaluation objective	Traces
Who & When	ReAct-style tool-augmented QA agents; single- and multi-agent reasoning pipelines	Single LLM judge; prompt-based rubric; no SFT; optional self-consistency sampling	Agent-level attribution of failure; step-level correctness and error localization	176
TRAIL	Web navigation agents; SWE-style coding agents; environment-interaction agents (browser / API / sandbox)	Single LLM judge; hierarchical prompting; no training; decomposition into sub-criteria	Detection of planning, environment, and tool-use failures in execution traces	148
Aegis	Heterogeneous MAS: QA, math reasoning (GSM8K/MATH), coding (HumanEval), and multi-agent debate systems	Multi-aspect prompt-based LLM judge; rubric decomposition; no SFT	Assessment of (1) agent competence, (2) inter-agent coordination, (3) validation correctness	9533
AgentRewardBench	Web agents (WebArena / VisualWebArena), workstation agents (WorkArena), assistant-style task agents	Single LLM reward-style judge; scalar/binary scoring; prompt-only (no training)	Global task completion judgment (success / partial / failure)	1302
AgentErrorBench	Goal-driven interactive agents (ALFWorld, WebShop, GAIA subsets)	LLM-based error classifier; taxonomy-driven prompting; no SFT	Fine-grained failure categorization (memory, planning, reflection, action, tool/API, environment)	200
AgenTracer	Coding, math, and general agentic MAS (MetaGPT, AutoGen, AgentPrune, AFlow, OWL) on MBPP+, KodCode, Blackjack, MATH, GSM8K, GAIA	Qwen3-8B fine-tuned with multi-granular RL (AgenTracer-8B); labels from counterfactual replay and fault injection	Agent- and step-level attribution of the decisive error	2476
AgentFail	General multi-agent systems (synthetic and real-world deployments)	Hierarchical LLM judge; optional ensemble of specialized prompt-based evaluators	System-level failure detection and categorization (agent / system / environment)	307
Pumpkin	Multi-agent pipeline traces with heterogeneous reasoning and tool-use patterns	Ensemble of 10 independent LLM judges followed by a final aggregator; prompt-only (no SFT)	Binary success/failure classification of full trajectory	328

Table 7: Full comparison including state-of-the-art published results. The benchmark-specific baseline is the score published by each benchmark paper with that paper's original backbone (GPT-4o for the WW rows; AgenTracer-8B, an RL-trained Qwen3-8B, without ground-truth access for the AgenTracer rows; Gemini 2.5 Flash for the others); the Gemini 2.5 Flash re-run of the Who&When judge appears only in Table 2. Best published shows the highest score reported in the original paper regardless of the model (WW hand and algo - GPT-4o; TRAIL - Gemini 2.5 pro-preview; Pumpkin - GPT-5; Aegis - Aegis-SFT based on Qwen2.5-14B-Instruct; AgentErrorBench - GPT-4.1; AgentRewardBench - Functional; AgenTracer - AgenTracer-8B, with ground-truth access for Agent Acc).

Dataset / Metric	**AutoJudge (summ + DB)**	**AutoJudge (full)**	**Benchmark-specific baseline**	**Best published**
WW hand (Agent Acc)	61.49	59.19	55.17	55.17
WW hand (Step Acc)	17.25	12.07	7.02	7.02
WW algo (Agent Acc)	55.55	63.23	54.33	54.33
WW algo (Step Acc)	41.23	43.38	25.51	25.51
TRAIL (GAIA) (F1)	21.35	21.26	24.16	38.9
TRAIL (GAIA) (Loc Acc)	26.55	27.68	17.05	54.6
Pumpkin small (F1)	64.52	66.13	64.30	74.2
Pumpkin large (F1)	64.58	72.00	65.70	68.9
Aegis (Pair mF1)	7.0	8.9	6.99	16.62
Aegis (Agent mF1)	51.93	53.31	42.02	76.53
Aegis (Error mF1)	22.5	24.17	23.47	27.53
Aegis (Pair MF1)	2.51	2.66	2.76	9.99
Aegis (Agent MF1)	17.1	17.67	16.45	47.97
Aegis (Error MF1)	12.69	14.59	19.85	27.66
AgentErrorBench (S Acc)	8.78	26.93	20.33	42.0
AgentErrorBench (S+M Acc)	2.06	7.77	8.00	32.0
AgentRewardBench AxTree (Precision)	77.92	72.73	68.5	79.0
AgenTracer Code (Agent Acc)	75.9	74.0	72.21	72.95
AgenTracer Code (Step Acc)	9.6	10.2	18.85	18.85

Table 8: Judge pool generation strategies in summary mode (Who&When).

	Per-trace	**Dataset-level**	**WW judge**
Hand (WW subset)			
Agent Acc. (%)	61.49	57.81	55.17
Step Acc. (%)	17.25	13.50	7.02
Algo (WW subset)			
Agent Acc. (%)	55.55	42.65	54.33
Step Acc. (%)	41.23	26.59	25.51

Cost of per-trace adaptation. The complete WW Hand per-trace condition costs about $2.18 under our accounting. Accuracy per dollar is accuracy in percentage points divided by total run cost. The dataset-level cost is an estimate: it prices the single pool-generation call at ten times the average per-trace call, a conservative assumption. Table 9 gives both subsets.

Table 9: Accuracy per dollar for the two pool strategies in summary mode (Who&When). Dataset-level costs are estimated as described in the text.

Subset	Pool strategy	Agent Acc.	Step Acc.	Agent Acc./$	Step Acc./$
WW Hand	Per-trace	61.49	17.25	28.21	**7.91**
	Dataset-level (est.)	57.81	13.50	**31.78**	7.42
WW Algo	Per-trace	55.55	41.23	**23.52**	**17.46**
	Dataset-level (est.)	42.65	26.59	23.23	14.49

On WW Algo, per-trace construction gains 12.90 agent-accuracy points and 14.64 step-accuracy points and has better accuracy per dollar on both metrics. On WW Hand, reuse gives higher agent accuracy per dollar, while per-trace construction gives higher step accuracy per dollar. Pool reuse is therefore preferable when traces are similar and cost dominates; per-trace construction is preferable when heterogeneity and precise attribution matter.

Table 10: Comparison of MetaAgent and Judge model combinations.

MetaAgent model	Judge model	Agent Acc.	Step Acc.	Lost	Price ($)
Gemini 3 Flash Preview	Gemini 2.5 Flash	59.19	12.07	0	3.90
Gemini 3 Flash Preview	Qwen 3 32b	62.50	3.57	2	2.32
Gemini 3 Flash Preview	Kimi k2	53.45	**17.24**	0	5.57
Gemini 3 Flash Preview	GPT-4o	60.00	7.27	3	20.19
Gemini 3 Flash Preview	Gemini 3 Flash Preview	56.90	8.62	0	5.54
Claude Sonnet 4.5	Gemini 2.5 Flash	51.72	7.02	0	7.71
Gemini 2.5 Flash	Gemini 2.5 Flash	**63.79**	8.62	0	3.62
Qwen 3 32b	Gemini 2.5 Flash	52.38	4.76	8	3.26
Kimi k2	Gemini 2.5 Flash	42.11	7.14	0	3.90
GPT-4o	Gemini 2.5 Flash	53.45	3.45	0	6.75

Table 11: Judge temperature (MetaAgent at 0.3); accuracy (%), mean $\pm$ std over three runs.

Judge T	Agent Acc.	Step Acc.
0.0	**59.50 $\pm$ 6.01**	7.00 $\pm$ 4.66
0.1	59.19 $\pm$ 1.62	**12.07 $\pm$ 1.44**
0.3	58.09 $\pm$ 4.28	5.81 $\pm$ 2.55
0.5	54.66 $\pm$ 1.05	3.55 $\pm$ 0.08
0.7	57.60 $\pm$ 3.67	5.85 $\pm$ 1.02
1.0	52.87 $\pm$ 1.00	7.60 $\pm$ 1.01

Table 12: MetaAgent temperature (judges at 0.1); accuracy (%), mean $\pm$ std over three runs.

MetaAgent T	Agent Acc.	Step Acc.
0.0	55.17 $\pm$ 0.00	9.77 $\pm$ 4.34
0.1	56.32 $\pm$ 1.00	8.71 $\pm$ 1.66
0.3	**59.19 $\pm$ 1.62**	**12.07 $\pm$ 1.44**
0.5	58.05 $\pm$ 0.99	8.10 $\pm$ 1.04
0.7	57.47 $\pm$ 3.59	3.46 $\pm$ 1.71
1.0	57.22 $\pm$ 4.41	9.32 $\pm$ 2.10