Verifying Agents in Rubric-Graded Environments*

Markus Dücker Vaibhav Kumar Yi Liu

Ronak Chaudhary Andreas Plesner Francisco Guzmán

Anish Athalye

Abstract

As AI agents take on open-ended tasks, verifying their outputs is increasingly difficult. *Rubric-graded agent environments*—where a verifier judges multiple natural-language criteria against the agent's deliverables and environment state—have emerged as a popular paradigm. We conduct the first systematic study of verifiers for such environments. First, we create BankerVerifierBench (BVB), a meta-evaluation dataset of 3,204 human-judged criteria across 21 investment-banking tasks. Next, we derive verifier requirements directly from the rubric corpus, yielding a nine-capability taxonomy that we distill into three design principles—reactive verification, environment alignment, and domain guidance—which we implement in Gandalf, an open-source verifier. Finally, we evaluate Gandalf on BVB: its cheapest configuration (F1 0.633, $42) is Pareto optimal, exceeding the most expensive baseline (F1 0.538, $414) by 9.5 points at one-tenth the cost.

Keywords

agentic verification, agents-as-a-judge, LLM-as-a-judge, rubric-based evaluation, agent benchmarks, verifiers

1 Introduction

Modern AI agents use large language models (LLMs) to interact with tools and environments in pursuit of a goal [10, 51, 69]. Early efforts focused on *verifiable domains* where deterministic automated checks suffice: SWE-bench [20] uses unit tests for software engineering, HotpotQA [67] uses exact-match scoring for multi-hop question answering, and WebArena [76] uses programmatic checks for browser use. Such environments trade verifiability against scope, open-endedness, and realism.

Recently, agent evaluation has expanded into *semi-verifiable domains* where judgment is required. Benchmarks such as BankerToolBench [24], APEX-Agents [57], CoreCraft [40], and ClawBench [73] use natural-language *rubrics*—fine-grained binary criteria that are objectively gradable—to define and capture correctness [12, 16, 58].

Verifying agent rollouts in such environments is itself challenging. Agents produce multi-file deliverables (e.g., an Excel spreadsheet plus a PowerPoint presentation) and invoke tools with side effects (e.g., sending an email); judging a single criterion may require joint reasoning across heterogeneous artifacts. Verifiers for rubric-graded environments span a broad design space, from LLM-as-a-judge over the trajectory [40] to fully agentic judges [71, 79]. Despite growing interest, existing benchmarks treat the verifier as a means to grade rollouts or supply training rewards, not as an object of study.

Contributions. We conduct the first systematic study of verifiers for agents in rubric-graded environments. We (i) release BankerVerifierBench (BVB), a meta-evaluation dataset of 3,204 human-judged criteria across 21 BankerToolBench [24] tasks; (ii) develop a methodology that derives the capabilities a verifier needs from any rubric corpus, yielding a nine-capability taxonomy on BVB; (iii) distill the taxonomy into three design principles—*reactive verification*, *environment alignment*, and *domain guidance*—which we implement in Gandalf[1], an open-source verifier; and (iv) show that Gandalf is Pareto-optimal on BVB and generalizes to a structurally different productivity-domain benchmark.

2 Background

In a *rubric-graded agent environment*, an agent acts within a structured environment (filesystem, tools, external data) to produce one or more files together with environment-state changes; a verifier then assigns a binary met/unmet judgment to each criterion in the task's natural-language rubric. Unlike deterministically graded environments, criteria are open-ended and often require artifact inspection, domain knowledge, and qualitative judgment.

BankerToolBench (BTB). BTB [24] is a benchmark of 100 end-to-end investment-banking workflows—discounted-cash-flow analyses, leveraged-buyout models, trading comparables, mergers—each taking a junior banker ~5 hours. Agents receive a task description, a data room, and a fixed set of tools (a market-data platform, an SEC EDGAR API, a company-profile API, plus a filesystem and pre-installed libraries such as `openpyxl` and `python-pptx`), and produce multi-file deliverables in standard office formats (`.xlsx`, `.pptx`, `.docx`, `.pdf`). Each task ships with an expert-authored rubric of binary, weighted criteria. We adopt BTB as the basis of our study.

Baseline verifiers. We study three open-source verifiers spanning the design space: *AutoRubric* [48] grades each criterion from the agent trajectory only, with no artifact access; *Archipelago* [57] grades over a filesystem diff and Reducto-extracted visuals; *Agent-as-a-Judge* (AAAJ) [79] uses a workflow-based agent that plans and executes file-reading steps.

Appendix E contains a more detailed discussion of related work.

*All authors are affiliated with Handshake AI Research; Andreas Plesner is additionally affiliated with ETH Zürich. Correspondence: {markus, anish.athalye, paco}@joinhandshake.com.

[1] https://github.com/Handshake-AI-Research/gandalf-the-grader

Table 1: The nine verifier capabilities induced from BVB and the design principles that address each. P1 (reactive verification) applies universally. P2 (environment alignment) addresses capabilities that require native artifact inspection. P3 (domain guidance) addresses capabilities that depend on conventions not present in the criterion text. The % column reports each capability's share of the 3,204 rubric criteria as a primary tag.

Capability	%	P1	P2	P3
Audit quantitative metric calculation	23.0	✓		
Verify literal data point accuracy	15.5	✓	✓	
Reconcile cross-component data linkage	14.9	✓	✓	
Verify structural component inventory	12.1	✓	✓	
Inspect visual presentation and style	11.6	✓	✓	
Audit accounting logic and convention	7.6	✓		✓
Validate document labeling and metadata	6.5	✓	✓	
Audit governance and strategic narrative	5.4	✓		✓
Validate temporal and sequential integrity	3.3	✓	✓	

3 BankerVerifierBench: A meta-evaluation dataset

We introduce BankerVerifierBench (BVB), the first meta-evaluation dataset for verifiers in rubric-graded agent environments. BVB extends prior meta-evaluation work for LLM judges [41, 56] and agent trajectories [36] to settings where criteria require inspecting multi-file binary deliverables and tool-mediated environment state rather than text alone. The dataset consists of 21 BTB tasks each paired with one agent rollout (full trajectory plus final environment state including deliverable files in native binary formats) and an expert-authored rubric of 47–286 binary criteria per task (3,204 total, met-class prevalence 78.7%). An initial 6 tasks (493 criteria) were graded by two reviewers independently with a senior reviewer issuing a master verdict (inter-annotator agreement 89.5%); the remaining 15 tasks (2,711 criteria) were graded by a single reviewer plus senior master verdict.

Deriving verifier requirements from the corpus. Rather than fixing a verifier architecture in advance and determining its capabilities, we start from the rubric and ask what capabilities a verifier needs. A three-stage pipeline—open-ended free-text tagging of each criterion, clustering these tags into canonical capabilities, then re-tagging each criterion against that fixed label set—maps each criterion to one or more *capabilities*: kinds of checks a verifier must perform. Applied to BVB, this yields the nine-capability taxonomy in Table 1. The labels are verifier-side (they describe what a verifier must check, not what the agent must produce) and architecture-agnostic (no specific runtime, library, file format, or data source), so the methodology carries to other rubric corpora. Capability descriptions, worked examples, and the full construction pipeline are in Appendix A.

4 Design principles for verifiers

Reading the taxonomy as a specification surfaces three structural patterns: (i) the verification path varies across criteria; (ii) judging requires inspecting artifact-internal structure that text serialization erases; (iii) judging requires domain knowledge not in the criterion text. Each pattern follows from the nature of natural-language rubrics over rich agent environments rather than from the financial-modeling setting of BVB. We address each with a design principle.

P1: Reactive verification. We cannot predict the verification path from the criterion text alone: two criteria within a single capability can require different files to open, different tools to invoke, and different orders of operation. The verifier should be a *reactive agent* [69] that, given a criterion, decides at inference time which artifacts to open and which tools to invoke, terminating once it reaches a verdict.

P2: Environment alignment. Several capabilities reduce to inspecting properties of the agent's artifacts (formula structure, layout metadata, cross-file references) that do not survive flattening to text or rendering as images. The verifier should run in the *same environment* as the rollout agent, with access to the same filesystem, libraries, and tools (including domain-specific Model Context Protocol (MCP) [17] tools). This also makes the verifier domain-agnostic by construction: deploying it in a new RL environment requires only granting the same tool access as the rollout.

P3: Domain guidance. Several capabilities require knowledge not in the criterion text—accounting sign conventions, disclosure standards for an investment memo, clinical guidelines for a medical workflow. The verifier should accept domain context at deployment time through a dedicated *guidance channel*, separate from the criterion text and from the architecture.

5 Gandalf: a verifier for rubric-graded environments

We implement P1–P3 in *Gandalf*, a portable Python application. P1 builds on the OpenHands SDK [63], an open-source agent harness. P2 is realized by running Gandalf directly inside the rollout agent's environment, connecting to the same MCP tools. P3 is realized through a configuration system that accepts natural-language judge guidance and selects the backing LLM, letting a benchmark designer trade off accuracy and cost. As a performance optimization, Gandalf supports *batching*—verifying multiple criteria within one agent session—which amortizes file-reading and tool-call cost across the typical ~150 criteria per BTB rubric.

6 Evaluation

We evaluate Gandalf against the three baselines from Section 2 on BVB. Unless otherwise noted, Gandalf runs at batch size b=16 with Gemini 3 Flash, with guidance and environment-aligned MCP tools enabled. We report F1 on the unmet class (the minority class; 21.3% prevalence is more discriminating than raw accuracy) and total dollar cost across the entire dataset.

Figure 1 plots cost against F1 for all four verifiers across the Gemini and GPT model families. The headline finding is that *verifier architecture matters more than model choice*: Gandalf with any backing model outperforms every other verifier with any backing model. In the GPT family, the cheapest Gandalf configuration (GPT-5.4-Nano, $42) achieves F1 0.633, exceeding the most expensive Archipelago configuration (GPT-5.4, $414, F1 0.538) by 9.5 points at

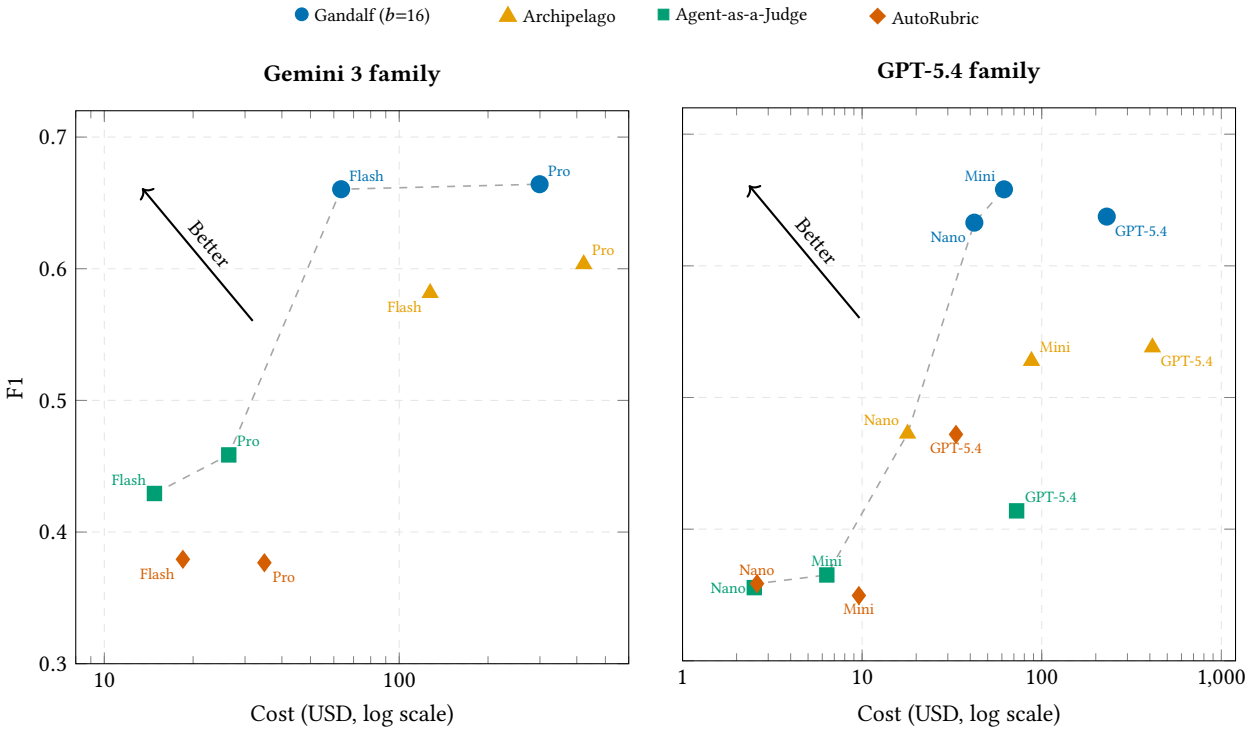

Figure 1: Cost vs. F1 across verifiers on BVB, split by model family. Left: Gemini 3 (Flash, Pro). Right: GPT (Nano, Mini, 5.4). Gandalf (b=16) dominates across both families. Costs computed from public API pricing: Gemini 3 Flash \$0.50/\$3.00, Gemini 3 Pro \$2.00/\$12.00, GPT-5.4-Nano \$0.20/\$1.25, GPT-5.4-Mini \$0.75/\$4.50, GPT-5.4 \$2.50/\$15.00 per 1M input/output tokens.

one-tenth the cost. Within each verifier, scaling to a larger model yields diminishing returns: Gandalf spans 0.633–0.664 across a 7× cost range, while AAAJ and AutoRubric cluster below F1 0.47 regardless of model.

Why each baseline fails. A per-capability error breakdown (full table in Appendix D) explains the gap by the principles each baseline lacks. *AutoRubric* satisfies none of P1–P3 and shows high, undifferentiated errors (32.8–61.5% across capabilities on BVB). *AAAJ* is workflow-based and inspects files through its own toolset rather than the rollout environment; errors concentrate on the inspection-bound capabilities P2 covers (e.g., *verify literal data point accuracy* 38.2%, *audit quantitative metric calculation* 35.1%). *Archipelago* grades from a serialized snapshot, giving it a partial form of P2; errors split bimodally along P2's fault line—false negatives (unmet criteria passing) dominate on capabilities requiring native artifact inspection (deficient outputs pass when serialization erases the disqualifying signal), while false positives (met criteria getting rejected) dominate on capabilities that reduce to judgment over rendered evidence. Gandalf's only two narrow losses (1–2 percentage points in error rate) are on capabilities where the bottleneck is the LLM applying conventions or judging coherence over text both verifiers can see, not artifact-grounded inspection.

Ablations within Gandalf. Toggling Gandalf's principles isolates the cost of substituting for P2 and P3 within a P1-equipped verifier. P2's contribution surfaces in a joint *No-Guidance + No-MCP* run (the pure-MCP run is uninformative because our guidance references MCP-exposed tools): F1 unchanged but cost rises 60% (\$64 → \$102), as the agent compensates by writing ad-hoc Python scripts to parse binary files. Removing guidance alone (P3) drops F1 by 1.1 points (0.660 → 0.649) and raises cost 14% (\$64 → \$73). Within Gandalf, both principles are primarily cost optimizations; their full impact—which non-P1 architectures cannot substitute around—surfaces in the cross-verifier comparison above. Full ablations and sensitivity to batch size and model choice are in Appendix B.

Generalization. We replicate the analysis on an internal OpenClaw benchmark of persona-driven personal-productivity workflows—a structurally different domain (consumer rather than financial) with a different tool surface (messaging clients, calendar and booking services) and evidence type (many tools are stateful, so verification depends on whether the environment changed). Table 2 reports F1 and cost across all four verifiers on 24 tasks. Gandalf achieves F1 0.951, leading the next-best verifier (Archipelago with Gemini 3 Flash, 0.876) by 7.5 points and again Pareto-optimal: the cheaper verifiers sacrifice 7–10 F1 points. The cross-verifier ranking and the architecture-over-model finding replicate; the non-agentic verifiers cluster in a narrow band (0.849–0.876), suggesting that once serialization captures most of the evidence, a stronger model yields diminishing returns. Further analysis is in Appendix C.

Table 2: Verifier F1 and cost on OpenClaw (24 episodes, 396 criteria). Gandalf leads the next-best verifier by 7.5 F1 points and is Pareto optimal on this second domain.

Verifier	Model	F1	Cost
Gandalf	Gemini 3 Flash	**0.951**	$8.20
Archipelago	Gemini 3 Flash	0.876	$1.77
Archipelago	GPT-5.4-Mini	0.866	$0.59
AutoRubric	Gemini 3 Flash	0.849	$0.83
AAAJ	Gemini 3 Flash	0.757	$5.66
AAAJ	GPT-5.4-Mini	0.698	$1.09

7 Conclusion

Our results motivate the adoption of guidable, environment-aligned reactive agents as the Pareto-optimal design for verifying agents in rubric-graded environments. We publicly release BVB to facilitate research on verifiers and open-source Gandalf to unlock more realistic agent benchmarks.

Limitations. BVB contains a single rollout per task; we did not measure verifier sensitivity to the rollout distribution. Our study measures verifier F1 against human labels; we did not evaluate whether the F1 advantage translates to better-trained agents through RL. Our evaluation covers two domains (BVB and OpenClaw); we expect Gandalf to generalize further and hope this is validated by future benchmarks.

References

[1] Afra Feyza Akyürek, Advait Gosai, Chen Bo Calvin Zhang, Vipul Gupta, Jaehwan Jeong, Anisha Gunjal, Tahseen Rabbani, Maria Mazzone, David Randolph, Mohammad Mahmoudi Meymand, Gurshaan Chattha, Paula Rodriguez, Diego Mares, Pavit Singh, Michael Liu, Subodh Chawla, Pete Cline, Lucy Ogaz, Ernesto Hernandez, Zihao Wang, Pavi Bhatter, Marcos Ayestaran, Bing Liu, and Yunzhong He. 2025. PRBench: Large-Scale Expert Rubrics for Evaluating High-Stakes Professional Reasoning. arXiv:2511.11562 [cs.CL]. https://arxiv.org/abs/2511.11562

[2] Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Qui nonero Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. 2025. HealthBench: Evaluating Large Language Models Towards Improved Human Health. arXiv:2505.08775 [cs.CL]. https://arxiv.org/abs/2505.08775

[3] Elad Ben Avraham, Changhao Li, Ron Dorfman, Roy Ganz, Oren Nuriel, Amir Dudai, Aviad Aberdam, Noah Flynn, Elman Mansimov, Adi Kalyanpur, and Ron Litman. 2026. DREAM: Deep Research Evaluation with Agentic Metrics. arXiv:2602.18940 [cs.AI]. https://arxiv.org/abs/2602.18940

[4] Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024. T-Eval: Evaluating the Tool Utilization Capability of Large Language Models Step by Step. In *Proceedings of the 62nd Meeting of the Association for Computational Linguistics (ACL)*. Bangkok, Thailand, 9510–9529.

[5] Jonathan Cook, Tim Rocktäschel, Jakob Foerster, Dennis Aumiller, and Alex Wang. 2024. TICKing All the Boxes: Generated Checklists Improve LLM Evaluation and Generation. arXiv:2410.03608 [cs.AI]. https://arxiv.org/abs/2410.03608

[6] Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. 2026. Agentic Reward Modeling: Verifying GUI Agent via Online Proactive Interaction. arXiv:2602.00575 [cs.RO]. https://arxiv.org/abs/2602.00575

[7] Gaole Dai, Shiqi Jiang, Ting Cao, Yuqing Yang, Yuanchun Li, Rui Tan, Mo Li, and Lili Qiu. 2026. ProRe: A Proactive Reward System for GUI Agents via Reasoner-Actor Collaboration. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*. Rio de Janeiro, Brazil.

[8] 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. 2026. ARM-Thinker: Reinforcing Multimodal Generative Reward Models with Agentic Tool Use and Visual Reasoning. In *Proceedings of the 2026 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Denver, CO.

[9] Arduin Findeis, Floris Weers, Guoli Yin, Ke Ye, Ruoming Pang, and Tom Gunter. 2025. Can External Validation Tools Improve Annotation Quality for LLM-as-a-Judge?. In *Proceedings of the 63rd Meeting of the Association for Computational Linguistics (ACL)*. Vienna, Austria, 15997–16020.

[10] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*. Honolulu, HI, 10764–10799.

[11] Xinyan Guan, Yanjiang Liu, Xinyu Lu, Boxi Cao, Ben He, Xianpei Han, Le Sun, Jie Lou, Bowen Yu, Yaojie Lu, and Hongyu Lin. 2024. Search, Verify and Feedback: Towards Next Generation Post-training Paradigm of Foundation Models via Verifier Engineering. arXiv:2411.11504 [cs.AI]. https://arxiv.org/abs/2411.11504

[12] Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2026. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*. Rio de Janeiro, Brazil.

[13] Jiuzhou Han, Wray Buntine, and Ehsan Shareghi. 2025. VerifiAgent: a Unified Verification Agent in Language Model Reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Suzhou, China, 16410–16431.

[14] Harbor Framework Team. 2026. Harbor: A framework for evaluating and optimizing agents and models in container environments. https://github.com/harbor-framework/harbor

[15] Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. 2024. LLM-Rubric: A Multidimensional, Calibrated Approach to Automated Evaluation of Natural Language Texts. In *Proceedings of the 62nd Meeting of the Association for Computational Linguistics (ACL)*. Bangkok, Thailand, 13806–13834.

[16] Yun He, Wenzhe Li, Hejia Zhang, Songlin Li, Karishma Mandyam, Sopan Khosla, Yuanhao Xiong, Nanshu Wang, Xiaoliang Peng, Beibin Li, Shengjie Bi, Shishir G. Patil, Qi Qi, Shengyu Feng, Julian Katz-Samuels, Richard Yuanzhe Pang, Sujan Gonugondla, Hunter Lang, Yue Yu, Yundi Qian, Maryam Fazel-Zarandi, Licheng Yu, Amine Benhalloum, Hany Awadalla, and Manaal Faruqui. 2025. AdvancedIF: Rubric-Based Benchmarking and Reinforcement Learning for Advancing LLM Instruction Following. arXiv:2511.10507 [cs.CL]. https://arxiv.org/abs/2511.10507

[17] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv:2503.23278 [cs.CR]. https://arxiv.org/abs/2503.23278

[18] Yuzhen Huang, Weihao Zeng, Xingshan Zeng, Qi Zhu, and Junxian He. 2025. From Accuracy to Robustness: A Study of Rule- and Model-based Verifiers in Mathematical Reasoning. arXiv:2505.22203v2 [cs.LG]. https://arxiv.org/abs/2505.22203v2

[19] Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, Xijun Gu, Peiyi Tu, Jiaxin Liu, Wenyu Chen, Yuzhuo Fu, Zhiting Fan, Yanmei Gu, Yuanyuan Wang, Zhengkai Yang, Jianguo Li, and Junbo Zhao. 2025. Reinforcement Learning with Rubric Anchors. arXiv:2508.12790 [cs.AI]. https://arxiv.org/abs/2508.12790

[20] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. Vienna, Austria.

[21] Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. 2024. Prometheus: Inducing Fine-grained Evaluation Capability in Language Models. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. Vienna, Austria.

[22] Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. 2024. Prometheus 2: An Open Source Language Model Specialized in Evaluating Other Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Miami, FL, 4334–4353.

[23] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. 2025. RewardBench: Evaluating Reward Models for Language Modeling. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*. Albuquerque, NM, 1755–1797.

[24] Elaine Lau, Markus Dücker, Ronak Chaudhary, Hui Wen Goh, Rosemary Wei, Vaibhav Kumar, Saed Qunbar, Guram Gogia, Yi Liu, Scott Millslagle, Nasim Borazjanizadeh, Ulyana Tkachenko, Samuel Eshun Danquah, Collin Schweiker, Vijay Karumathil, Asrith Devalaraju, Varsha Sandadi, Haemi Nam, Punit Arani, Ray Epps, Abdullah Arif, Sahil Bhaiwala, Curtis Northcutt, Skyler Wang, Anish Athalye, Jonas Mueller, and Francisco Guzmán. 2026. BankerToolBench: Evaluating AI Agents in End-to-End Investment Banking Workflows. arXiv:2604.11304 [cs.AI]. https://arxiv.org/abs/2604.11304

[25] Yukyung Lee, Joonghoon Kim, Jaehee Kim, Hyowon Cho, Jaewook Kang, Pilsung Kang, and Najoung Kim. 2025. CheckEval: A reliable LLM-as-a-Judge framework for evaluating text generation using checklists. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Suzhou, China, 15771–15798.

[26] Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. 2025. From Generation to Judgment: Opportunities and Challenges of LLM-as-a-judge. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Suzhou, China, 2757–2791.

[27] Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. 2026. The Tool Decathlon: Benchmarking Language Agents for Diverse, Realistic, and Long-Horizon Task Execution. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*. Rio de Janeiro, Brazil.

[28] Keyu Li, Junhao Shi, Yang Xiao, Mohan Jiang, Jie Sun, Yunze Wu, Dayuan Fu, Shijie Xia, Xiaojie Cai, Tianze Xu, Weiye Si, Wenjie Li, Dequan Wang, and Pengfei Liu. 2026. AgencyBench: Benchmarking the Frontiers of Autonomous Agents in 1M-Token Real-World Contexts. arXiv:2601.11044 [cs.AI]. https://arxiv.org/abs/2601.11044

[29] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Singapore, 3102–3116.

[30] Xiangyi Li, Kyoung Whan Choe, Yimin Liu, Xiaokun Chen, Chujun Tao, Bingran You, Wenbo Chen, Zonglin Di, Jiankai Sun, Shenghan Zheng, Jiajun Bao, Yuanli Wang, Weixiang Yan, Yiyuan Li, and Han chung Lee. 2026. ClawsBench: Evaluating Capability and Safety of LLM Productivity Agents in Simulated Workspaces. arXiv:2604.05172 [cs.AI]. https://arxiv.org/abs/2604.05172

[31] Bill Yuchen Lin, Yuntian Deng, Khyathi Chandu, Faeze Brahman, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. 2025. WildBench: Benchmarking LLMs with Challenging Tasks from Real Users in the Wild. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*. Singapore.

[32] Lanbo Lin, Jiayao Liu, Tianyuan Yang, Li Cai, Yuanwu Xu, Lei Wei, Sicong Xie, and Guannan Zhang. 2026. JADE: Expert-Grounded Dynamic Evaluation for Open-Ended Professional Tasks. arXiv:2602.06486 [cs.AI]. https://arxiv.org/abs/2602.06486

[33] Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. 2025. OpenRubrics: Towards Scalable Synthetic Rubric Generation for Reward Modeling and LLM Alignment. arXiv:2510.07743 [cs.CL]. https://arxiv.org/abs/2510.07743

[34] 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, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. AgentBench: Evaluating LLMs as Agents. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. Vienna, Austria.

[35] 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 (EMNLP)*. Singapore, 2511–2522.

[36] Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. 2025. AgentRewardBench: Evaluating Automatic Evaluations of Web Agent Trajectories. In *Proceedings of the 2nd Conference on Language Modeling (COLM)*. Montreal, Canada.

[37] Zeyao Ma, Jing Zhang, Xiaokang Zhang, Jiaxi Yang, Zongmeng Zhang, Jiajun Zhang, Yuheng Jing, Lei Zhang, Hao Zheng, Wenting Zhao, Junyang Lin, and Binyuan Hui. 2026. Scaling Agentic Verifier for Competitive Coding. arXiv:2602.04254 [cs.CL]. https://arxiv.org/abs/2602.04254

[38] Shiva Krishna Reddy Malay, Shravan Nayak, Jishnu Sethumadhavan Nair, Sagar Davasam, Aman Tiwari, Sathwik Tejaswi Madhusudhan, Sridhar Krishna Nemala, Srinivas Sunkara, and Sai Rajeswar. 2026. EnterpriseOps-Gym: Environments and Evaluations for Stateful Agentic Planning and Tool Use in Enterprise Settings. arXiv:2603.13594 [cs.AI]. https://arxiv.org/abs/2603.13594

[39] Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah A. Smith, Hannaneh Hajishirzi, and Nathan Lambert. 2026. RewardBench 2: Advancing Reward Model Evaluation. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*. Rio de Janeiro, Brazil.

[40] Sushant Mehta, Logan Ritchie, Suhaas Garre, Ian Niebres, Nick Heiner, and Edwin Chen. 2026. EnterpriseBench Corecraft: Training Generalizable Agents on High-Fidelity RL Environments. arXiv:2602.16179 [cs.AI]. https://arxiv.org/abs/2602.16179

[41] Tianjun Pan, Xuan Lin, Wenyan Yang, Qianyu He, Shisong Chen, Licai Qi, Wanqing Xu, Hongwei Feng, Bo Xu, and Yanghua Xiao. 2026. RubricEval: A Rubric-Level Meta-Evaluation Benchmark for LLM Judges in Instruction Following. arXiv:2603.25133 [cs.AI]. https://arxiv.org/abs/2603.25133

[42] Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. Vancouver, Canada, 48371–48392.

[43] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large Language Model Connected with Massive APIs. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada, 126544–126565.

[44] Hao Peng, Yunjia Qi, Xiaozhi Wang, Zijun Yao, Bin Xu, Lei Hou, and Juanzi Li. 2025. Agentic Reward Modeling: Integrating Human Preferences with Verifiable Correctness Signals for Reliable Reward Systems. In *Proceedings of the 63rd Meeting of the Association for Computational Linguistics (ACL)*. Vienna, Austria, 15934–15949.

[45] Andreas Plesner, Francisco Guzmán, and Anish Athalye. 2026. An Imperfect Verifier is Good Enough: Learning with Noisy Rewards. arXiv:2604.07666 [cs.LG]. https://arxiv.org/abs/2604.07666

[46] 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 *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. Vienna, Austria.

[47] Mohit Raghavendra, Anisha Gunjal, Bing Liu, and Yunzhong He. 2026. Agentic Rubrics as Contextual Verifiers for SWE Agents. arXiv:2601.04171 [cs.LG]. https://arxiv.org/abs/2601.04171

[48] Delip Rao and Chris Callison-Burch. 2026. Autorubric: Unifying Rubric-based LLM Evaluation. arXiv:2603.00077 [cs.CL]. https://arxiv.org/abs/2603.00077

[49] MohammadHossein Rezaei, Robert Vacareanu, Zihao Wang, Clinton Wang, Bing Liu, Yunzhong He, and Afra Feyza Akyürek. 2025. Online Rubrics Elicitation from Pairwise Comparisons. arXiv:2510.07284 [cs.CL]. https://arxiv.org/abs/2510.07284

[50] Jon Saad-Falcon, Rajan Vivek, William Berrios, Nandita Shankar Naik, Matija Franklin, Bertie Vidgen, Amanpreet Singh, Douwe Kiela, and Shikib Mehri. 2025. LMUnit: Fine-grained Evaluation with Natural Language Unit Tests. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Suzhou, China, 3303–3324.

[51] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*. New Orleans, LA, 68539–68551.

[52] Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G. Finlayson, David Sontag, Tyler Murray, Sewon Min, Pradeep Dasigi, Luca Soldaini, Faeze Brahman, Wen tau Yih, Tongshuang Wu, Luke Zettlemoyer, Yoon Kim, Hannaneh Hajishirzi, and Pang Wei Koh. 2025. DR Tulu: Reinforcement Learning with Evolving Rubrics for Deep Research. arXiv:2511.19399 [cs.CL]. https://arxiv.org/abs/2511.19399

[53] Ved Sirdeshmukh, Kaustubh Deshpande, Johannes Mols, Lifeng Jin, Ed-Yeremai Cardona, Dean Lee, Jeremy Kritz, Willow Primack, Summer Yue, and Chen Xing. 2025. MultiChallenge: A Realistic Multi-Turn Conversation Evaluation Benchmark Challenging to Frontier LLMs. In *Proceedings of the 63rd Meeting of the Association for Computational Linguistics (ACL)*. Vienna, Austria, 18632–18702.

[54] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. 2025. PaperBench: Evaluating AI's Ability to Replicate AI Research. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. Vancouver, Canada, 56843–56873.

[55] Rui Sun, Zuo Bai, Wentao Zhang, Yuxiang Zhang, Li Zhao, Shan Sun, and Zhengwen Qiu. 2025. FinResearchBench: A Logic Tree based Agent-as-a-Judge Evaluation Framework for Financial Research Agents. In *Proceedings of the 6th ACM International Conference on AI in Finance (ICAIF)*. Singapore, 656–664.

[56] Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y. Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. 2025. JudgeBench: A Benchmark for Evaluating LLM-based Judges. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*. Singapore.

[57] Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, Neel Venugopal, Alannah Hsia, Isaac Robinson, Calix Huang, Olivia Varones, Daniyal Khan, Michael Haines, Austin Bridges, Jesse Boyle, Koby Twist, Zach Richards, Chirag Mahapatra, Brendan Foody, and Osvald Nitski. 2026. APEX-Agents. arXiv:2601.14242 [cs.CL]. https://arxiv.org/abs/2601.14242

[58] Vijay Viswanathan, Yanchao Sun, Shuang Ma, Xiang Kong, Meng Cao, Graham Neubig, and Tongshuang Wu. 2025. Checklists Are Better Than Reward Models

For Aligning Language Models. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS)*. Mexico City, Mexico, 114728–114754.

[59] Manya Wadhwa, Zayne Sprague, Chaitanya Malaviya, Philippe Laban, Junyi Jessy Li, and Greg Durrett. 2025. EvalAgent: Discovering Implicit Evaluation Criteria from the Web. In *Proceedings of the 2nd Conference on Language Modeling (COLM)*. Montreal, Canada.

[60] Jize Wang, Xuanxuan Liu, Yining Li, Songyang Zhang, Yijun Wang, Zifei Shan, Xinyi Le, Cailian Chen, Xinping Guan, and Dacheng Tao. 2026. GTA-2: Benchmarking General Tool Agents from Atomic Tool-Use to Open-Ended Workflows. arXiv:2604.15715 [cs.CL]. https://arxiv.org/abs/2604.15715

[61] Jize Wang, Zerun Ma, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. 2024. GTA: A Benchmark for General Tool Agents. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada, 75749–75790.

[62] Xinchen Wang, Ruida Hu, Pengfei Gao, Chao Peng, and Cuiyun Gao. 2025. An Agent-based Evaluation Framework for Complex Code Generation. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Seoul, South Korea, 2427–2439.

[63] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. 2025. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*. Singapore.

[64] Ran Xu, Jingjing Chen, Jiayu Ye, Yu Wu, Jun Yan, Carl Yang, and Hongkun Yu. 2026. Incentivizing Agentic Reasoning in LLM Judges via Tool-Integrated Reinforcement Learning. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*. Rio de Janeiro, Brazil.

[65] Ran Xu, Tianci Liu, Zihan Dong, Tony Yu, Ilgee Hong, Carl Yang, Linjun Zhang, Tao Zhao, and Haoyu Wang. 2026. Alternating Reinforcement Learning for Rubric-Based Reward Modeling in Non-Verifiable LLM Post-Training. arXiv:2602.01511 [cs.CL]. https://arxiv.org/abs/2602.01511

[66] Zhangchen Xu, Yuetai Li, Fengqing Jiang, Bhaskar Ramasubramanian, Luyao Niu, Bill Yuchen Lin, and Radha Poovendran. 2025. TinyV: Reducing False Negatives in Verification Improves RL for LLM Reasoning. arXiv:2505.14625 [cs.LG]. https://arxiv.org/abs/2505.14625

[67] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Brussels, Belgium, 2369–2380.

[68] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2025. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*. Singapore.

[69] 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 *Proceedings of the 11th International Conference on Learning Representations (ICLR)*. Kigali, Rwanda.

[70] Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. 2026. Claw-Eval: Toward Trustworthy Evaluation of Autonomous Agents. arXiv:2604.06132 [cs.AI]. https://arxiv.org/abs/2604.06132

[71] Runyang You, Hongru Cai, Caiqi Zhang, Qiancheng Xu, Meng Liu, Tiezheng Yu, Yongqi Li, and Wenjie Li. 2026. A Survey on Agent-as-a-Judge. arXiv:2601.05111 [cs.CL]. https://arxiv.org/abs/2601.05111

[72] Qiyuan Zhang, Junyi Zhou, Yufei Wang, Fuyuan Lyu, Yidong Ming, Can Xu, Qingfeng Sun, Kai Zheng, Peng Kang, Xue Liu, and Chen Ma. 2026. RubricBench: Aligning Model-Generated Rubrics with Human Standards. arXiv:2603.01562 [cs.AI]. https://arxiv.org/abs/2603.01562

[73] Yuxuan Zhang, Yubo Wang, Yipeng Zhu, Penghui Du, Junwen Miao, Xuan Lu, Wendong Xu, Yunzhuo Hao, Songcheng Cai, Xiaochen Wang, Huaisong Zhang, Xian Wu, Yi Lu, Minyi Lei, Kai Zou, Huifeng Yin, Ping Nie, Liang Chen, Dongfu Jiang, Wenhu Chen, and Kelsey R. Allen. 2026. ClawBench: Can AI Agents Complete Everyday Online Tasks? arXiv:2604.08523 [cs.CL]. https://arxiv.org/abs/2604.08523

[74] 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. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*. New Orleans, LA, 46595–46623.

[75] Karen Zhou and Chenhao Tan. 2026. AutoChecklist: Composable Pipelines for Checklist Generation and Scoring with LLM-as-a-Judge. arXiv:2603.07019 [cs.CL]. https://arxiv.org/abs/2603.07019

[76] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. Vienna, Austria.

[77] Fengbin Zhu, Xiang Yao Ng, Ziyang Liu, Chang Liu, Xianwei Zeng, Chao Wang, Tianhui Tan, Xuan Yao, Pengyang Shao, Min Xu, Zixuan Wang, Jing Wang, Xin Lin, Junfeng Li, Jingxian Zhu, Yang Zhang, Wenjie Wang, Fuli Feng, Richang Hong, Huanbo Luan, Ke-Wei Huang, and Tat-Seng Chua. 2025. FinDeepResearch: Evaluating Deep Research Agents in Rigorous Financial Analysis. arXiv:2510.13936 [cs.CL]. https://arxiv.org/abs/2510.13936

[78] Lianghui Zhu, Xinggang Wang, and Xinlong Wang. 2025. JudgeLM: Fine-tuned Large Language Models are Scalable Judges. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*. Singapore.

[79] Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. 2025. Agent-as-a-Judge: Evaluate Agents with Agents. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. Vancouver, Canada, 80569–80611.

Appendix

A Capability taxonomy: methodology and full results

Tasks in rubric-graded agent environments come paired with a rubric of natural-language criteria. Rather than fixing a verifier architecture in advance and determining its capabilities, we start from the rubric and ask what capabilities a verifier needs. We describe a general methodology for deriving verifier requirements from any rubric corpus, then apply it to BVB. The methodology produces a taxonomy of verifier-side *capabilities*: each capability names a kind of check the verifier must perform to judge a criterion correctly. The taxonomy is verifier-side (labels describe what a verifier must check, not what the agent must produce) and architecture-agnostic (labels do not name a specific runtime, library, file format, or data source), so it applies across verifier types.

Methodology summary. Our pipeline takes a corpus of natural-language rubrics and produces a taxonomy of capabilities in three stages. *Stage 1* generates open-ended natural-language tags for each criterion describing what a verifier needs to do to verify it. *Stage 2* clusters these tags and merges them into a small set of canonical capabilities. *Stage 3* re-tags every criterion with one to three of the canonical capabilities, ranked by confidence. Full pipeline details are in Appendix A.1.

Application to BVB. Applying the methodology to the BVB rubric (n = 3,204 criteria) results in the nine capabilities listed in Table A1. The largest, *audit quantitative metric calculation*, accounts for 23.0% of rubric criteria; the smallest, *validate temporal and sequential integrity*, accounts for 3.3%.

In the BVB setting, these abstract capabilities take concrete form. *Audit quantitative metric calculation* ranges over EV/EBITDA multiples, transaction fees; *audit accounting logic and convention* covers sign rules and balance-sheet identities; *audit governance and strategic narrative* covers disclaimers, regulatory boilerplate, and acquisition-rationale claims. The same capability labels carry to other domains with different instantiations: a clinical workflow would instantiate *audit accounting logic and convention* as dosage and contraindication checks against clinical guidelines; a productivity workflow would instantiate *audit governance and strategic narrative* as escalation policies and tone norms.

A.1 Construction pipeline

The taxonomy is produced in three stages: free-text discovery, canonicalization, and closed-vocabulary tagging. All LLM calls in this pipeline use Gemini 3 Flash, and embeddings use Google's `text-embedding-005`.

Stage 1: Free-text discovery. For each rubric criterion, an LLM receives the agent-facing task brief and the criterion text, and emits one to three short capability labels along with a one-sentence rationale per label. Three constraints are enforced programmatically: (i) the rationale must begin with "the verifier needs to …" and quote a phrase from the criterion text, which enforces verifier-side framing; (ii) labels must not contain tokens naming a specific runtime, library, file format, or data source, which enforces architecture-agnosticism; and (iii) labels must follow a fixed lexical form.

Stage 2: Canonicalization. Each unique Stage 1 label is embedded and partitioned by agglomerative clustering into 80 fine-grained micro-clusters. An LLM names each micro-cluster and produces a definition, a diagnostic signal, a worked example, and a set of non-examples contrasting it with its nearest siblings. A final LLM call merges the 80 named micro-clusters into the canonical taxonomy under three hard constraints: between 8 and 10 canonical capabilities, no single capability covering more than 40% of corpus criteria, and every input micro-cluster mapped to exactly one canonical capability.

Stage 3: Closed-vocabulary tagging. Every criterion is re-tagged against the closed canonical taxonomy. An LLM receives the criterion, its task brief, and the full taxonomy (canonical names, definitions, diagnostic signals, non-examples), and emits one to three canonical capability tags per criterion ranked by confidence. For the analyses in Appendix D, we use the highest-confidence tag per criterion to obtain a clean partition; the multi-tag distribution is consistent with the primary-tag results.

B Detailed evaluation: design principles and sensitivity

We extend Section 6 with ablations isolating P2 and P3 and a sensitivity analysis over batch size and model choice. The ablations measure the cost of substituting for P2 and P3 within a P1-equipped verifier rather than the principle's full impact. Because Gandalf is reactive (P1), the agent loop can partially compensate for missing tools or missing guidance through additional inference-time exploration: aggregate F1 stays high, and the principle's contribution surfaces in cost. The principle's full impact, which non-P1 architectures cannot substitute around, manifests in the cross-verifier comparison (Section 6) and the per-capability breakdown (Appendix D).

P2: Environment alignment. The cleanest isolation of P2 would hold guidance fixed and remove MCP. In our setup this run is uninformative: the guidance we authored for BVB references MCP-exposed capabilities, so without MCP it points at tools the verifier cannot reach. We therefore read P2's contribution from the joint ablation. Removing both guidance and MCP (*No Guidance + No MCP*) leaves F1 unchanged (0.661 vs 0.660) but raises cost by 60% ($64 to $102), as the verifier compensates by writing ad-hoc Python scripts to parse binary files and issuing tool calls to extract information that a single MCP call would have returned in a more structured manner. Within a P1-equipped verifier, environment alignment is a cost optimization: the agent loop is expressive enough to work around missing tools, but doing so is substantially more expensive.

P3: Judge guidance. Removing judge guidance—natural-language domain conventions injected into the verifier's prompt (e.g., sign rules in financial modeling, disclosure standards for investment memos)—hurts on both axes: F1 drops from 0.660 to 0.649 (1.1 points) and cost rises from $64 to $73 (14%). Without guidance, the verifier takes more exploratory tool calls to infer conventions that guidance would have stated directly, and the additional exploration only partially recovers the lost accuracy. Within a P1-equipped verifier, guidance is a strict improvement on both axes: it encodes domain

Table A1: The nine capabilities induced from the BVB corpus. The right-hand column reports each capability's share of the 3,204 corpus criteria as a primary tag (top-ranked in Stage 3).

Capability	Description	%
Audit quantitative metric calculation	Recompute a derived figure from its inputs and compare against the deliverable.	23.0
Verify literal data point accuracy	Confirm a constant in the deliverable matches a constant in the brief or source data.	15.5
Reconcile cross-component data linkage	Trace a value across two locations in the deliverable (across sheets, tabs, or files).	14.9
Verify structural component inventory	Confirm the deliverable contains the prescribed sections, files, and columns.	12.1
Inspect visual presentation and style	Evaluate fonts, colors, alignment, layout, and branding against the specification.	11.6
Audit accounting logic and convention	Check signs, identities, and component completeness against domain conventions.	7.6
Validate document labeling and metadata	Confirm textual labels (tab names, axis titles, headers) match the specification verbatim.	6.5
Audit governance and strategic narrative	Confirm required disclosures are present and qualitative claims cohere with the analysis.	5.4
Validate temporal and sequential integrity	Verify chronological alignment, period sequencing, and stage-boundary integrity.	3.3

Table A2: Gandalf ablations on BVB. The default configuration (⋆) uses Gemini 3 Flash with batch size b=16, guidance, and MCP tools.

Design principles			*Batch size*			*Model*		
Config	F1	Cost	Config	F1	Cost	Config	F1	Cost
⋆ Guidance + MCP	.660	$64	b=64	.635	$32	Gemini 3 Pro	.664	$299
No guidance, MCP	.649	$73	b=32	.644	$40	⋆ Gemini 3 Flash	.660	$64
No guidance, no MCP	.661	$102	⋆ b=16	.660	$64	GPT-5.4	.637	$230
			b=8	.667	$100	GPT-5.4-Mini	.658	$62
			b=4	.673	$159	GPT-5.4-Nano	.633	$42

knowledge that the verifier would otherwise rediscover through trial and error at higher cost.

Batch size. We vary batch size, the number of criteria graded by one agent trajectory. F1 rises monotonically as batch size shrinks, from 0.635 at b=64 to 0.673 at b=4, with cost rising from $32 to $159 across the same range. The curve flattens below b=16: dropping from b=16 to b=4 buys 1.3 F1 points at 2.5× the cost. At b=32 and below, each session is small enough for the verifier to attend to every criterion individually while amortizing file-reading and tool calls across the batch; below this point, additional sessions largely duplicate the same inspection work without extracting new signal. We fix b=16 for all cross-verifier comparisons as a deliberate cost-accuracy tradeoff.

Model choice. Table A2 also reports Gandalf at b=16 across five models spanning two providers. Gemini 3 Pro achieves the highest F1 (0.664), followed closely by Gemini 3 Flash (0.660) and GPT-5.4-Mini (0.658). The full spread from the weakest model (GPT-5.4-Nano, 0.633) to the strongest is 3.1 F1 points, a modest range given the diversity of model scale and cost. The ranking does not simply follow model size: GPT-5.4 (0.637) underperforms GPT-5.4-Mini (0.658), suggesting that factors beyond raw capability such as instruction-following fidelity or tool-use calibration mediate verifier performance. The 3.1-point spread across model choices and the plateau in batch size are both smaller than the gap between Gandalf and any other verifier on the cost-F1 frontier (Section 6): configuration choices move Gandalf along its frontier, but they do not move it off.

C Generalization to OpenClaw

To investigate whether Gandalf generalizes, we collected another internal dataset analogous to BVB in a new domain: an OpenClaw (OPC) benchmark of persona-driven personal-productivity workflows—managing schedules, coordinating with contacts, planning lifestyle tasks. OpenClaw differs from BVB in domain (consumer), tool surface (messaging clients, calendar and booking services, a weather API), and evidence type (many OpenClaw tools are stateful, so verification depends on determining whether the environment has changed, such as whether a message was sent or an event created). This dataset consists of 396 criteria judgments across 24 tasks.

The headline cross-verifier results on OpenClaw appear in Table 2 in the main text: Gandalf achieves F1 0.951, leading the next-best verifier (Archipelago with Gemini 3 Flash, 0.876) by 7.5 points and is again Pareto optimal. The results replicate the BVB finding that *verifier architecture dominates model choice*: Gandalf with Gemini 3 Flash outperforms every other verifier regardless of backing model. The non-agentic verifiers cluster in a narrow band (0.849–0.876), suggesting that once serialization captures most of the evidence, a stronger model or richer serialization yields diminishing returns.

D Per-capability error breakdown

Table A3 breaks down errors by the capability taxonomy from Appendix A. Each baseline fails specifically on capabilities where its missing principles would have helped, while Gandalf wins everywhere except two narrow exceptions we return to at the end of this section.

AutoRubric. AutoRubric is a non-agentic verifier: it grades each criterion in an isolated LLM call over the agent's trajectory, with no artifact access and no domain guidance. It satisfies none of the three principles. Errors are high and undifferentiated across capabilities: 32.8–61.5% on BVB, where most evidence lives in the deliverable, and 9.4–31.8% on OPC, where trajectories at least carry tool-call content.

AAAJ. AAAJ inspects files through its own toolset rather than the rollout's environment, with no domain guidance. It is a workflow-based (rather than reactive) agent, lacking all of P1–P3. Errors concentrate on the inspection-bound capabilities P2 covers: on BVB, *verify literal data point accuracy* (38.2%), *audit quantitative metric calculation* (35.1%), and *reconcile cross-component data linkage* (31.4%) are its three worst rows; on OPC, *reconcile cross-component data linkage* reaches 40.0%. The single BVB row AAAJ wins, *audit governance and strategic narrative*, is the only one where neither structural inspection nor cross-component tracing applies and the criteria reduce to judging textual content. On OPC the same capability flips: Gandalf wins (error rate 6.1% vs AAAJ's 43.9%), because OPC's instances require tool-state inspection (e.g., whether a message was sent) rather than text judgment, moving the bottleneck back into P2's territory.

Archipelago. Archipelago is workflow-based and grades each criterion from a serialized snapshot of the deliverable, giving it a partial form of P2; it has neither P1 nor P3. Errors split bimodally along P2's fault line (full breakdown in Table A6; here, FP denotes over-rejection of a met criterion and FN denotes an unmet criterion passing). On the four BVB capabilities that require native artifact inspection, FN exceeds FP by 4.4–5.6% in absolute terms (e.g., *inspect visual presentation and style*: FP 10.5%, FN 16.1%): deficient outputs pass when serialization erases the disqualifying signal. On the other five capabilities, FP exceeds FN by up to 6.5% (e.g., *validate temporal and sequential integrity*: FP 12.1%, FN 5.6%): the LLM judge over-rejects on inferences from rendered evidence. The pattern reproduces and sharpens on OPC: the Gandalf–Archipelago gap on *reconcile cross-component data linkage* widens from 6.9% on BVB to 35.0% on OPC.

Where the framework does not predict. Gandalf's only losses are two BVB rows where the gap is narrow (1–2%) and the bottleneck is the LLM applying conventions or judging coherence over text both verifiers can see: *audit accounting logic and convention* (Archipelago by 2.1%) and *audit governance and strategic narrative* (AAAJ by 1.1%). The principles pay off where artifact-grounded inspection is the bottleneck, not where the bottleneck is reasoning over rendered text. The OPC result on *audit governance and strategic narrative* reinforces the point in the other direction: when its criteria require tool-state inspection, P2 reasserts itself and Gandalf wins by 37.8%.

E Related work

Agent environments. Agent benchmarks frequently rely on deterministic checks against final state, expected tool calls, or unit tests [4, 20, 27, 29, 30, 34, 38, 42, 43, 46, 61, 68]. In pursuit of greater scope and realism, recent benchmarks in semi-verifiable domains have embraced rubric-based grading. Such benchmarks include BankerToolBench [24], APEX-Agents [57], CoreCraft [40], GTA-2 [60], ClawBench [73], Claw-Eval [70], and AgencyBench [28]. Across both lines, these works treat the verifier as a means to an end, not the object of study.

Rubrics. Rubrics are a popular approach to defining correctness in semi-verifiable domains [31, 53]. Rubrics may be authored by domain experts [1, 2, 54] or generated synthetically [5, 33, 47, 59]. RubricBench [72] evaluates rubric generators themselves against human preferences. A line of work investigates rubrics in the context of training [12, 58]; some of this work investigates training with synthetic rubrics [16, 19, 49, 52]. Rubric-ARM [65] jointly optimizes a rubric generator and a judge, optimizing judgment accuracy for matching human preference ranking. Unlike these works, we take rubrics as a given and study the verifiers that consume them.

Verifier designs. Verifiers, both inside and outside the context of environments, cover a broad design space [11, 26, 71]. A line of work optimizes LLM-as-a-judge for evaluating text through prompting or training [21, 22, 35, 78], and a related line decomposes evaluation into per-criterion judgments via rubrics [15, 25, 48, 50, 75]. Benchmarks commonly apply such techniques for outcome grading in agent environments by evaluating a serialized trajectory or serialized output files [40, 70]. In contrast, some environments propose workflow-based agents as verifiers [9, 13, 44, 55, 57, 77, 79], while others propose reactive agents [3, 6–8, 32, 37, 62, 64]. Despite the broad design space, no principled account describes which design choices matter for which verification capabilities.

Meta-evaluation of verifiers. Motivated by the importance of verifiers in benchmarks, meta-evaluation benchmarks evaluate verifiers themselves, covering LLM-as-a-judge for preference rankings [23, 39, 56, 74] and LLM-as-a-judge for rubric grading [41]. Verifiers can be domain-specific, motivating domain-specific benchmarks such as AgentRewardBench [36] for web agents and meta-evaluations such as TinyV [66] and Huang et al. [18] for mathematics. BVB is the first meta-evaluation dataset that covers rubric-graded agent environments, where rubric criteria cover deliverable files and side-effecting tool calls.

Table A3: Per-capability error rates (%) for Gandalf (Gan), Archipelago (Arc), Agent-as-a-Judge (AAAJ), and AutoRubric (Aut) on BVB and OpenClaw. Bold: best verifier on each row within each corpus (ties bolded). Cells marked - denote capabilities not present in the OPC corpus.

	BVB					OPC				
Capability	*n*	Gan.	Arc.	AAAJ	Aut.	*n*	Gan.	Arc.	AAAJ	Aut.
Audit quantitative metric calculation	736	**12.0**	15.6	35.1	61.5	-	-	-	-	-
Verify literal data point accuracy	497	**12.3**	15.3	38.2	59.2	200	**4.5**	10.0	25.5	21.5
Reconcile cross-component data linkage	478	**13.4**	20.3	31.4	57.1	20	**0.0**	35.0	40.0	20.0
Verify structural component inventory	388	**10.8**	12.6	25.5	52.1	53	**1.9**	13.2	18.9	9.4
Inspect visual presentation and style	373	**17.2**	26.5	34.9	51.5	-	-	-	-	-
Audit accounting logic and convention	244	16.0	**13.9**	30.3	60.2	-	-	-	-	-
Validate document labeling and metadata	207	**13.5**	15.0	30.0	41.5	41	**7.3**	24.4	31.7	12.2
Audit governance and strategic narrative	174	19.5	21.8	**18.4**	32.8	66	**6.1**	25.8	43.9	31.8
Validate temporal and sequential integrity	107	**15.9**	17.8	20.6	45.8	12	**8.3**	25.0	16.7	16.7
Aggregate	3,204	**13.6**	17.4	31.7	54.7	396	**4.5**	16.2	28.5	20.2

F Comparison with RewardKit

RewardKit [14] is the verification component of Harbor, developed concurrently with Gandalf. Like Gandalf, RewardKit employs an agentic verifier that can interact with the agent's output artifacts, making it the most architecturally similar baseline in our study. For each criterion, RewardKit spawns an isolated Claude Code session pointed at the agent's output workspace; the judge can explore the filesystem, write scripts, and run shell commands to reach a verdict. Criteria are configured via TOML files and fall into two classes: programmatic (Python functions that inspect files or run shell commands) and judge-based (LLM or agent evaluation). We evaluate RewardKit in its agent judge mode, both with and without access to environment-aligned MCP tools (+MCP). The latter toggle is the closest analogue to Gandalf's P2 (environment alignment) within RewardKit's architecture.

The key architectural difference is in *how* the verifier accesses the environment. RewardKit operates through a general-purpose code CLI: the judge writes and executes ad-hoc scripts to inspect deliverables. Gandalf instead runs inside the rollout agent's own environment with access to the same domain-specific tools (e.g., MCP servers) the agent used during task execution. This distinction matters in practice: Gandalf can call the same market-data APIs the agent queried, inspect spreadsheet formulas through the same libraries the agent used to write them, and trace cross-file references without re-implementing extraction logic. Table A4 summarizes the result.

Gandalf's higher token count reflects the cost of environment-aligned verification: inspecting artifacts through native tools and tracing values across files requires more agentic steps than writing ad-hoc scripts in a code CLI. Despite consuming roughly 35% more tokens, Gandalf is approximately 4.5× cheaper than RewardKit because it runs on Gemini 3 Flash ($0.50/$3.00 per MTok) rather than Claude Sonnet 4.6 ($3.00/$15.00 per MTok). The combination of a 7.2-point F1 advantage over the best RewardKit variant and a substantially lower cost makes Gandalf the stronger choice for RL reward signals, where verifier accuracy impacts training quality [45].

Table A4: Comparison of RewardKit and Gandalf on BankerVerifierBench. Gandalf achieves the highest F1 while being the cheapest verifier, owing to Gemini 3 Flash's lower per-token rates. Token counts are summed across prompt and response; costs are computed at public API list prices (Gemini 3 Flash: $0.50/$3.00 per MTok in/out; Claude Sonnet 4.6: $3.00/$15.00 per MTok in/out).

Verifier	Model	F1	Tokens (M)	Cost ($)
RewardKit	Claude Sonnet 4.6	0.879	52.6	162.67
RewardKit +MCP	Claude Sonnet 4.6	0.905	51.7	160.76
Gandalf	Gemini 3 Flash	**0.951**	70.0	**35.76**

Capability-level breakdown. RewardKit is evaluated on a subset (n=2,634 vs. 3,204) because a fraction of grading runs failed to complete in our infrastructure. The failures arise when the list of criteria becomes too large for the verifier to handle. Gandalf handles

Table A5: Per-capability error rates (%) for all five verifiers on BVB and OPC. Bold: best verifier on each row within each corpus (ties bolded). Cells marked — denote capabilities not present in the OPC corpus. RewardKit (RK) on BTB is on its n=2,634 subset.

	BTB						OPC					
Capability	*n*	Gan.	Arc.	AAAJ	Aut.	RK	*n*	Gan.	Arc.	AAAJ	Aut.	RK
Audit quantitative metric calculation	736	**12.0**	15.6	35.1	61.5	17.0	—	—	—	—	—	—
Verify literal data point accuracy	497	**12.3**	15.3	38.2	59.2	12.8	200	**4.5**	10.0	25.5	21.5	13.5
Reconcile cross-component data linkage	478	**13.4**	20.3	31.4	57.1	16.5	20	**0.0**	35.0	40.0	20.0	35.0
Verify structural component inventory	388	**10.8**	12.6	25.5	52.1	9.6	53	**1.9**	13.2	18.9	9.4	15.1
Inspect visual presentation and style	373	**17.2**	26.5	34.9	51.5	18.3	—	—	—	—	—	—
Audit accounting logic and convention	244	16.0	**13.9**	30.3	60.2	16.8	—	—	—	—	—	—
Validate document labeling and metadata	207	**13.5**	15.0	30.0	41.5	9.7	41	**7.3**	24.4	31.7	12.2	17.1
Audit governance and strategic narrative	174	19.5	21.8	**18.4**	32.8	19.2	66	**6.1**	25.8	43.9	31.8	16.7
Validate temporal and sequential integrity	107	**15.9**	17.8	20.6	45.8	18.7	12	**8.3**	25.0	16.7	16.7	16.7
Aggregate	3,204	**13.6**	17.4	31.7	54.7	15.2	396	**4.5**	16.2	28.5	20.2	15.7

such cases by batching, but RewardKit has no functionality for this. The full results are presented in Table A5. On its subset, RewardKit's aggregate error is 15.2%, between Gandalf and Archipelago. Per-capability, RewardKit matches or beats Gandalf on three capabilities reducible to direct filesystem inspection (*verify structural component inventory* 9.6% vs. 10.8%, *validate document labeling and metadata* 9.7% vs. 13.5%, *verify literal data point accuracy* 12.8% vs. 12.3%), and trails Gandalf on the remaining six, by up to 3.6 on *audit quantitative metric calculation* and 3.1 on *reconcile cross-component data linkage*. The pattern is consistent with RewardKit's design: a general-purpose code CLI suffices when the verifier needs to read a value out of a file, but underperforms when verification requires resolving references through the rollout agent's own tool surface. RewardKit's polarity is also distinct: with an FN/FP ratio of 0.29, it is the only verifier in our study biased toward over-rejection rather than toward letting deficient outputs pass.

G Archipelago error breakdown

Table A6 reports Archipelago's per-capability FP and FN rates on BVB, sorted by FN–FP gap, supporting the bimodality claim in Appendix D. Capabilities that require native artifact inspection are FN-dominant; those that reduce to judgment over rendered evidence are FP-leaning.

H Prompts used by the capability construction pipeline

We provide the four prompts used by the pipeline below. Variables in {{ ... }} are populated at runtime with task and criterion data.

Stage 1 prompt: capability discovery. Run once per criterion to emit free-text capability labels and a verifier-side rationale.

```
You are a methodologist designing a capability taxonomy for
↪  evaluating
agentic/LLM-as-Judge verifiers (NOT for evaluating the
↪  agent). For each
rubric criterion below, name 1-3 short skills the VERIFIER
↪  must possess
to grade that criterion correctly, regardless of how the
↪  verifier is
implemented (tool-using judge, code-execution judge,
↪  multi-agent judge,
single-prompt judge, etc.).

# Verifier-side framing (read carefully)

A "capability" here is a property of the VERIFIER, not of
↪  the agent.

- Right framing: "the verifier needs to <verb a verifier
↪  does> <noun
  describing what it inspects in the deliverable or task
  ↪  spec>". Example
  verbs: inspect, locate, recompute, cross-check, recognize,
  ↪  decompose,
  trace, measure, compare, audit, parse, interpret,
  ↪  classify.
- Wrong framing (DO NOT produce): naming what the agent has
↪  to do to
  satisfy the criterion. Example forbidden
  ↪  agent-construction head-verbs:
  build, construct, produce, create, design, develop, write,
  ↪  draft,
  generate, populate, deliver, prepare, assemble, compile,
  ↪  render.
```

Table A6: Archipelago FP and FN rates (%) per capability on BVB, sorted by FN–FP gap. Capabilities that require native artifact inspection are FN-dominant; those that reduce to judgment over rendered evidence are FP-leaning.

Capability	*n*	FP rate	FN rate	FN–FP
Native inspection required (FN-dominant)				
Inspect visual presentation and style	373	10.5	16.1	+5.6
Validate document labeling and metadata	207	4.8	10.1	+5.3
Reconcile cross-component data linkage	478	7.5	12.8	+5.2
Verify structural component inventory	388	4.1	8.5	+4.4
Judgment on rendered content (FP-leaning)				
Audit accounting logic and convention	244	7.4	6.6	−0.8
Verify literal data point accuracy	497	8.2	7.0	−1.2
Audit governance and strategic narrative	174	12.1	9.8	−2.3
Audit quantitative metric calculation	736	10.6	5.0	−5.6
Validate temporal and sequential integrity	107	12.1	5.6	−6.5

```
  (Note: words like `document`, `layout`, `output`,
  `format`, `present`
  are FINE when used as nouns or adjectives, e.g.
  `document_title_content_verification_capability` or
  `present_value_audit_capability`.)
- Wrong framing (DO NOT produce): naming a specific tool,
  file format,
  or runtime. Forbidden tokens anywhere in the capability
  name: excel,
  xlsx, pptx, pdf, openpyxl, pandas, python, jinja, json,
  csv, mcp,
  sec, edgar, sql, api.
- The verifier is judge-architecture-agnostic. The
  capability MUST be
  expressible for a verifier that:
  (a) reads the deliverable file and computes things itself,
  (b) reads the agent's generation script as proxy,
  (c) calls a remote tool to fetch source-of-truth data,
  (d) reasons over a rendered preview without executing code.
  If your capability only makes sense for one of these,
  restate it more
  abstractly.

# Output format (HARD constraints)

Each capability name MUST satisfy ALL of:

- 4 to 8 words inclusive.
- lower_snake_case (only [a-z_]; no digits, no hyphens, no
  spaces).
- ENDS WITH `_capability` (counts as one of the 4-8 words).
- Reads as "<verb-or-adjective>_<thing-the-verifier-inspec
  ts>_capability".
- Distinct from sibling capabilities WITHIN this same rubric
  category.
  If two criteria in the same category map to the same
  capability, only
  emit it once per criterion, but make sure the contrasting
  one (if any)
  is genuinely different, not a paraphrase.

Good examples (illustrative archetypes -- DO NOT just copy
  them; produce
labels that fit the actual criterion):

- `numeric_value_recomputation_capability`
- `formula_dependency_tracing_capability`
- `cross_section_consistency_checking_capability`
- `external_source_factual_lookup_capability`
- `unit_and_scale_normalization_capability`
- `qualitative_intent_interpretation_capability`
- `multi_period_completeness_auditing_capability`
- `visual_layout_inspection_capability`

Bad examples (rejected by post-validation; do NOT produce
  these):

- `excel_workbook_check_capability` -> contains forbidden
  token `excel`.
- `build_dcf_model_capability` -> agent-side head-verb
  `build`.
- `formula_audit` -> missing `_capability` suffix.
- `verifier_reads_the_workbook_thoroughly_to_validate_capa
  bility` -> >8 words.
- `recompute_capability` -> <4 words; not specific.
- `accuracy_capability` -> generic virtue, not a verifier
  skill.

# Each criterion's rationale (also required)

For each criterion, include a `rationale` string of 1-2
  sentences,
explaining why the verifier needs the capabilities you
  listed. The
rationale MUST:

- Start literally with "the verifier needs to ".
- Mention something concrete from the criterion text (a
  quoted phrase,
  a number, a section name) so we can audit grounding.
- NOT mention the agent's process or how the agent should
  accomplish
  anything.

# Task context (read-only)

Task ID: {{ task_id }}
Cohort: {{ cohort }}
Rubric category: {{ rubric_category }}

## Task brief (instructions the agent received)

This is provided ONLY so you can understand WHAT deliverable
  the
verifier will inspect. The capability you name is about
  INSPECTING the
deliverable, not about the agent's process to produce it.
```

```
<task_brief>
{{ task_brief }}
</task_brief>

# Criteria to annotate ({{ n }} criterion(a) in this batch)

{% for c in criteria %}
### Criterion {{ loop.index0 }}
- **Text**: {{ c.criterion }}
- **Category**: {{ c.category }}
- **Direction (FYI only, do not let this bias you)**: {{
↪  c.direction }}

{% endfor %}

# Output

Return a JSON array, one element per criterion, in the SAME
↪  order as
above.

[
  {
    "criterion_idx": 0,
    "capabilities": ["<lower_snake_case_capability_1>",
    ↪  "<...>"],
    "rationale": "the verifier needs to ..."
  }
]

Constraints again, summarised:

1. 1-3 capabilities per criterion (never 0, never >3).
2. Each capability: 4-8 words, lower_snake_case, MUST end
↪  with
   `_capability`.
3. No agent-side head-verbs
↪  (build/produce/create/design/develop/...).
4. No tool/format tokens
↪  (excel/xlsx/python/openpyxl/pandas/json/csv/
   sec/edgar/api/sql).
5. Rationale starts with "the verifier needs to " and quotes
↪  from the
   criterion.
6. Output ONLY the JSON array. No preamble, no commentary,
↪  no markdown
   fence.
```

Stage 2 prompt (a): per-cluster canonicalization. Run once per micro-cluster to assign a canonical name, definition, diagnostic signal, example, and contrastive non-examples.

```
You are a methodologist consolidating short,
↪  machine-tractable
verifier-capability labels for an agentic/LLM-as-Judge
↪  evaluation
taxonomy.

# What is a "capability"?

A capability is a property of the VERIFIER (the
LLM-as-Judge/Agent-as-a-Judge/Any-other-judge), not of the
↪  agent. It names a
verification skill the verifier needs in order to grade a
↪  rubric
criterion correctly, regardless of whether the verifier uses
↪  tools,
executes code, or just reasons over a rendered preview.
↪  Capability
labels MUST NOT mention specific tools, file formats, or
↪  runtimes.

# Input

Below is a single CLUSTER of {{ n_members }} member-labels
↪  emitted by a
Phase 1 free-text discovery pass. They have been grouped by a
deterministic embedding + clustering step. Most members are
paraphrases of the same underlying verifier skill; some
↪  members may be
drift.

Cluster id: `{{ cluster_id }}`
Member-label frequency in the corpus (ranked):

{% for m in member_labels %}- `{{ m.label }}`  (frequency:
↪  {{ m.freq }})
{% endfor %}

A few representative example criteria from this cluster
↪  (verbatim from
the rubric):

{% for ex in example_criteria %}- *(task: {{ ex.task_id }},
↪  category:
{{ ex.category }})* {{ ex.criterion }}
{% endfor %}

# Task

Pick ONE canonical name for this cluster, plus a short
↪  description, a
diagnostic signal, an example pattern, and 1-3 contrastive
↪  non-examples
("if it's about X, prefer the `<sibling>` capability
↪  instead").

## Naming constraints (HARD)

The canonical `name` MUST satisfy ALL of:

- 4 to 8 underscore-separated words (counting `_capability`
↪  as one
  word).
- lower_snake_case, only [a-z_].
- Ends with `_capability`.
- Reads as "<verb-or-adjective>_<thing-the-verifier-inspec⌋
↪  ts>_capability".
- No agent-side construction verbs
↪  (build/construct/produce/create/
  design/develop/write/draft/generate/populate/deliver/pre⌋
  ↪  pare/
  assemble/compile/output/render).
- No tool/format tokens
↪  (excel/xlsx/pptx/pdf/openpyxl/pandas/python/
  jinja/json/csv/mcp/sec/edgar/sql/api).
- Distinct enough from common siblings
(cross-section consistency, external lookup, completeness/
  scope, semantic intent, unit normalization,
  ↪  structural-grading) that
  it could not be paraphrased into one of them.

## Output

Return a single JSON object (NOT an array):

{
  "name": "<lower_snake_case_capability>",
  "description": "<1-2 sentences: what observable
  ↪  verification skill
    this is>",
  "diagnostic_signal": "<1 sentence: what is visible in the
  ↪  trace if
    the verifier exercised this capability vs. failed to>",
  "example_pattern": "<1-2 sentences: a concrete narrative
  ↪  of the
```

```
    verifier exercising this skill on a deliverable>",
  "non_examples": [
    "if the criterion is about <adjacent skill X>, prefer the
    `<sibling_capability>` label.",
    "..."
  ]
}

No preamble, no commentary, no markdown fence -- only the
↪  JSON object.
```

Stage 2 prompt (b): final merge. Run once over all canonicalized clusters to produce the final taxonomy of 8–10 capabilities under the size and coverage constraints.

```
You are finalising a small, distinct verifier-capability
↪  taxonomy
that will help characterize what a verifier running in
a Reinforcement Learning evironment needs to do.
You are given a slate of {{ n_in }}
candidate capabilities already produced by a clustering
↪  step, and you
must merge near-duplicates and produce a final list of {{
↪  min_caps }}-
{{ max_caps }} canonical capabilities.

# Background

A capability is a property of the VERIFIER, not of
the agent. Each canonical capability must be expressible for
↪  verifiers
of any architecture (tool-using, code-executing, or
↪  pure-reasoning).
Capability names MUST NOT mention specific tools, file
↪  formats, or
runtimes.

# Candidate slate

Each candidate has its current name, description, and the
↪  share of
corpus criteria (out of {{ n_total }}) where it was the
↪  dominant tag at
the cluster level.

{% for c in candidates %}- name: `{{ c.name }}`
  share: {{ c.share }}%  (n_criteria_dominated: {{ c.n }})
  description: {{ c.description }}
  diagnostic_signal: {{ c.diagnostic_signal }}
{% endfor %}

# Merge rules (HARD)

1. **Final count.** Output between {{ min_caps }} and {{
↪  max_caps }}
   canonical capabilities (inclusive).
2. **No giant bucket.** No single canonical capability may
↪  dominate
   more than {{ max_share_pct }}% of the corpus. If a
   ↪  candidate
   currently exceeds this share, propose a SPLIT: emit two
   ↪  or more
   sub-capabilities whose union covers the same members but
   ↪  each
   respects the cap.
3. **No singletons.** No canonical capability may dominate
↪  fewer than
   2% of the corpus. Merge such candidates into the nearest
   ↪  semantic
   neighbor.
4. **Pairwise distinctness.** For every pair of canonical
↪  capabilities
   (A, B) that could plausibly be confused, A's
   ↪  `non_examples` or B's
   `non_examples` MUST mention the other and explain the
   ↪  boundary.
5. **Coverage.** Every input candidate must map to exactly
↪  one
   canonical capability via the `member_input_names` field
   ↪  below. No
   silent drops.

# Naming constraints (HARD)

Same as in the upstream prompt:

- 4 to 8 underscore-separated words (counting `_capability`
↪  as one
  word).
- lower_snake_case, only [a-z_].
- Ends with `_capability`.
- Reads as "<verb-or-adjective>_<thing-the-verifier-inspec⌋
↪  ts>_capability".
- No agent-side head-verbs
↪  (build/produce/create/design/develop/write/
  draft/generate/populate/deliver/prepare/assemble/compile⌋
  ↪  /output/
  render).
- No tool/format tokens
↪  (excel/xlsx/pptx/pdf/openpyxl/pandas/python/
  jinja/json/csv/mcp/sec/edgar/sql/api).

# Output

Return a JSON array:

[
  {
    "name": "<lower_snake_case_capability>",
    "description": "<1-2 sentences>",
    "diagnostic_signal": "<1 sentence>",
    "example_pattern": "<1-2 sentence narrative>",
    "non_examples": ["if <pattern>, prefer
    ↪  `<sibling_capability>`",
      "..."],
    "member_input_names": ["<candidate name 1>", "<candidate
    ↪  name 2>",
      "..."]
  }
]

No preamble, no commentary, no markdown fence -- only the
↪  JSON array.
```

Stage 3 prompt: closed-vocabulary tagging. Run once per criterion to assign one to three taxonomy capabilities, ranked by confidence.

```
You are a methodologist tagging rubric criteria with the
↪  verifier
capabilities required to grade them. The taxonomy below is
↪  fixed. Pick
1-3 capabilities per criterion from the closed vocabulary --
↪  do NOT
invent new names.

# Verifier-side framing (read carefully)

A "capability" is a property of the VERIFIER, not of
the agent. It names a verification skill the verifier must
↪  possess to
grade the criterion correctly, regardless of the verifier's
↪  internal
```

```
architecture (tool-using, code-executing, single-prompt, multi-agent,
etc.).

This is judge-architecture-agnostic on purpose: the same tag applies
whether the verifier reads the deliverable file directly, parses the
agent's generation script, calls a remote tool, or reasons over a
rendered preview.

# Capability taxonomy (closed vocabulary -- pick from these only)

{% for cap in capabilities %}
### `{{ cap.name }}`
- **Description**: {{ cap.description }}
- **Diagnostic signal**: {{ cap.diagnostic_signal }}
- **Example pattern**: {{ cap.example_pattern }}
{% if cap.non_examples %}- **Boundaries**:
{% for ne in cap.non_examples %}    - {{ ne }}
{% endfor %}{% endif %}
{% endfor %}

# Task context

Task ID: {{ task_id }}
Cohort: {{ cohort }}
Rubric category: {{ rubric_category }}

## Task brief (instructions the agent received)

This is provided ONLY so you can understand what deliverable the
verifier will inspect. The capability you assign is about INSPECTING
the deliverable, not about how the agent might produce it.

<task_brief>
{{ task_brief }}
</task_brief>

# Criteria to tag ({{ n }} criteria in this batch)

{% for c in criteria %}
### Criterion {{ loop.index0 }}
- **Text**: {{ c.criterion }}
- **Category**: {{ c.category }}

{% endfor %}

# Tagging rules

1. Each criterion MUST be assigned 1-3 capability names, in
   MOST-RELEVANT-FIRST order.
2. Capability names MUST exactly match a `name` from the taxonomy above
   (case-sensitive, including the `_capability` suffix). No paraphrasing.
3. Tag based on what the criterion REQUIRES the verifier to do, not on
   whether any specific verifier implementation actually does it.
4. Reserve very-broad / structural-grading capabilities for criteria
   where the grading process itself is the focus (e.g. a self-consistency
   check), not as a default top-up tag.

# Output

Return a JSON array, one element per criterion, in the SAME order as
above. Each element MUST set `criterion_idx` to the criterion's index
above.

[
  {
    "criterion_idx": 0,
    "capabilities": ["<capability_name_1>",
    "<capability_name_2>",
      "..."]
  }
]

Output ONLY the JSON array. No preamble, no commentary, no markdown
fence.
```