Reducing Domain Engineering Effort with Autonomous Agent Learning

Anonymous ACL submission

Abstract

Building large language model (LLM) systems for specialized knowledge bases (KBs) typically demands custom components such as ontology parsers, embedding pipelines, and retrieval modules tailored to the target schema. This paper asks whether autonomous LLM agents can substitute for these hand-crafted components, given only a small set of examples, read access to raw KB files, and a code sandbox for exploration. We compare tool-oriented agents that rely on domain-specific infrastructure against autonomous agents that replace all custom components with a Python sandbox, evaluating on a drilling-domain ontology across multiple model scales. An LLM-based user simulator drives multi-turn refinement, and soft RDF graph metrics quantify output quality. Tool-oriented agents with explicit validation consistently outperform autonomous agents, though the gap narrows at larger model scales. A scaling analysis further shows that annotating additional examples can narrow but not close this gap, suggesting that annotation effort can partially substitute for engineering effort. Code is available at `https://anonymous.4open.science/r/DWIS-Copilot-CDFE`

1 Introduction

Specialized knowledge bases such as ontologies and knowledge graphs encode rich structured vocabularies for domains including drilling engineering, biomedical curation, and industrial interoperability. Translating natural-language requests into valid expressions over such bases requires familiarity with the KB structure that many practitioners lack. LLMs provide a natural interface (Brown et al., 2020), but integrating a KB into an LLM system typically demands custom components like schema parsers, embedding indices, and domain-specific retrieval logic, together with substantial maintenance as the KB evolves.

The central question this paper addresses is *whether a KB copilot can be built without domain-specific infrastructure, by supplying an LLM agent with only (i) a small set of examples, (ii) read access to the raw KB files, and (iii) a general-purpose code sandbox.*

These inputs are deliberately minimal. The examples are pairs of natural-language requests and target configurations, and the agent receives the unprocessed KB with no schema parser, embedding index, or retrieval module. Autonomous agents must discover and exploit KB structure through their own code. Related work on zero-shot task-oriented dialogue, such as AutoTOD (Xu et al., 2024), shares the motivation of reducing engineering but assumes a fixed API schema and operates on slot-filling dialogues. Our setting differs, as the agent must discover ontology structure from raw RDF files with no predefined API.

To answer this question, we compare seven architectures spanning a spectrum from heavily engineered to fully autonomous. We evaluate on the D-WIS ontology (D-WIS Industry Group, 2025; Cayeux et al., 2019, 2023) using an LLM-based user simulator (Mosqueira-Rey et al., 2023) for multi-turn feedback. A full example and trace are in Appendix F.

Our contributions are:

1. A framework comparing engineered and autonomous KB copilots across a spectrum of infrastructure requirements, evaluated across three model scales.
2. Empirical evidence that tool-oriented validation retains a consistent advantage over autonomous agents on D-WIS, with honest characterization of the gap.
3. A scaling analysis showing that annotating $\sim$50 examples can narrow but not close the gap, with the Prep agent exhibiting a non-monotonic dip at $n = 10$ that recovers at $n = 20$.
4. An analysis of multi-turn correction benefit

Tool-Oriented: CaG, CaG+val., RAG+val., GraphRAG+val.	
Custom	Ontology parser, embedding index, retrieval pipeline
Optional	Validation module (for +val. variants)
Autonomous: PaL, Prep, Prep+Code	
Generic	Training sheet (examples), raw KB files, code sandbox

Figure 1: Infrastructure requirements by agent category. Tool-oriented agents require domain-specific components, while autonomous agents use only generic resources and the unprocessed KB.

showing autonomous agents gain 3 to 7 pp Soft Triple F1 from simulator feedback vs. less than 2 pp for tool-oriented agents, demonstrating that the multi-turn protocol is especially valuable when domain-specific validation tooling is absent.

A survey of related work is provided in Appendix A.

2 Framework

All seven systems are implemented as LangChain (Chase, 2022) Reasoning-and-Acting (ReAct Yao et al., 2023) agents that share the same evaluation protocol. They are organized into tool-oriented and autonomous categories, distinguished by infrastructure requirements (Figure 1).

2.1 Task Formulation

Given a natural-language request q and a knowledge base $\mathcal{K}$, the system produces a configuration $c = (D, T)$ consisting of:

- *declarations* $D = \{\texttt{Class}_i\texttt{:Instance}_i\}$, each instantiating a class from $\mathcal{K}$.
- *triples* $T = \{(s_j, p_j, o_j)\}$, each asserting a relationship using a predicate $p_j \in \mathcal{K}$.

The configuration must be *consistent* with $\mathcal{K}$, meaning every class and predicate URI must exist in the ontology schema.

We instantiate $\mathcal{K}$ as the D-WIS RDF/Turtle ontology (D-WIS Industry Group, 2025), which defines ∼2,200 classes and ∼250 properties for drilling-domain sensor data, signal processing, and equipment interconnection. A complete example is provided in Appendix F.

2.2 Tool-Oriented Agents

All four require a custom `rdflib`-based ontology parser. +val. variants additionally need a purpose-built evaluator for structural and semantic validation.

CaG. Places the full ontology schema and few-shot examples in the system prompt (Chan et al., 2025) and generates the configuration in a single pass.

CaG+val. Adds a `validation` tool that checks class/predicate validity and enables iterative self-correction.

RAG+val. Replaces the full-context prompt with on-demand semantic retrieval (Lewis et al., 2020) plus validation (Appendix G).

GraphRAG+val. Extends RAG by re-ranking retrieved fragments via graph-structural connectivity scores (Han et al., 2024).

2.3 Autonomous Agents

The three autonomous agents receive only a *training sheet* (examples from Section 2.1) and raw ontology access, using a code sandbox to explore the KB programmatically.

PaL. Following Gao et al. (2023), writes Python scripts per query to load the ontology, query class hierarchies, and assemble the output.

Prep. Adds a *preparation phase* before test queries that builds a *knowledge summary* (structured KB entries included in the prompt at test time) by exploring the ontology with transient code execution.

Prep+Code. Splits preparation into two stages, where a *Preparation Agent* builds the knowledge summary and a *Helper Agent* then constructs a library of reusable functions from it. The main agent calls these helpers at test time.

2.4 User Simulator

An LLM-based user simulator (Mosqueira-Rey et al., 2023) provides multi-turn natural-language feedback to guide agent corrections. Rather than comparing against a per-task gold-standard oracle, the simulator acts as an experienced domain user, receiving a *training sheet* of past configuration examples disjoint from both the agent's training set and the test set, together with the original query, and evaluating whether the agent's latest output satisfies the user's intent. When the output is insufficient, feedback consists of a concise natural-language correction requesting one concrete change at a time (≤25 words). The session is marked complete when the simulator judges

the output sufficiently correct based on its domain knowledge from the training examples.

This design requires no additional ground-truth annotation per test task, as the simulator's judgment is grounded in historical examples rather than per-task gold standards. As a proxy evaluator, it may occasionally accept approximately correct configurations or request adjustments to valid ones. Its reliability reflects the quality of the training sheet. Initial vs. corrected metrics (Figure 2) are reported in Appendix B.

3 Experiments

3.1 Evaluation

The user simulator (Section 2.4) interacts with each system for up to $R = 3$ rounds per task.

Interaction-level metrics.

- **Task completion rate**: fraction of tasks for which the simulator accepts the final configuration.
- **Mean dialog iterations**: average number of agent responses per task.

Structural metrics. Both the generated configuration $c = (D, T)$ and the gold-standard $c^* = (D^*, T^*)$ are parsed into RDF graphs. Instance IDs are normalized via bipartite matching, where instances sharing a class are paired by maximizing Jaccard similarity of predicate signatures (minimum-weight matching). All metrics use *soft* scoring with partial credit via hierarchy-distance similarity $1/(1 + d)$ using `rdfs:subClassOf`/`rdfs:subPropertyOf` paths.

- **Soft Triple F1**: soft F1 over full RDF triples after instance-ID normalization, requiring subject, predicate, and object to all match (with partial credit for near-miss classes/predicates).
- **Soft Class F1**: soft F1 over the multiset of class URIs in D vs. D^*.
- **NGED** (Normalized Graph Edit Distance, $\in [0, 1]$, lower is better): $(|E_\Delta| + |V_\Delta|)/(|E_{\text{exp}}| + |V_{\text{exp}}| + |E_{\text{gen}}| + |V_{\text{gen}}|)$.

3.2 Setup

All experiments use models from the Qwen 3.5 family (Team, 2025):

- **Qwen3.5-35B-A3B**: MoE, 35B total, **3B active**.
- **Qwen3.5-27B**: dense, **27B active**.
- **Qwen3.5-122B-A10B**: MoE, 122B total, **10B active**.

Table 1: D-WIS results on 326 tasks ($n = 10$, corrected Soft Triple F1 and task completion). Best per model in **bold**. Full metrics with 95% CIs in Appendix Table 2.

	35B-A3B		**27B**		**122B-A10B**	
Approach	F1↑	Cpl. %	F1↑	Cpl. %	F1↑	Cpl. %
Tool-oriented						
CaG	0.412	77.0	0.470	76.1	0.477	77.3
CaG+val.	**0.441**	72.4	**0.488**	78.5	**0.498**	77.6
RAG+val.	0.435	**81.9**	**0.488**	77.3	0.487	78.5
GraphRAG+val.	0.420	77.6	0.472	73.0	0.475	75.5
Autonomous						
PaL	0.388	80.1	0.433	81.3	0.370	77.6
Prep	0.279	57.7	0.462	89.3	0.422	81.0
Prep+Code	0.337	73.9	0.448	**93.6**	0.458	**81.6**

Because the 27B dense model activates nearly 3× more parameters per token than the 122B-A10B, both are substantially more capable than the 35B-A3B despite its larger total parameter count. We order models by effective capability as 35B < 27B ≈ 122B throughout.

Experiments are run on 326 D-WIS configuration tasks with $R = 3$ simulation rounds. Unless otherwise noted, training sheets contain 10 examples, and 95% bootstrap confidence intervals (1,000 resamples) are reported.

3.3 Main Results

Table 1 summarises corrected Soft Triple F1 and task completion for all approaches and model scales ($n = 10$). Full metrics with 95% CIs are in Appendix Table 2.

Tool-oriented agents lead. CaG+val. achieves the highest Soft Triple F1 at 35B and 122B (0.441/0.498), while at 27B CaG+val. and RAG+val. are statistically indistinguishable (both 0.488, CIs overlap). The validation tool provides direct ontology feedback within each agent turn, enabling a tight self-correction loop that the code sandbox cannot replicate.

Scale dependence and correction benefit. Prep shows marked scale sensitivity. It collapses to 0.279 at 35B (57.7% completion) when preparation fails, but recovers to 0.462 at 27B. Prep+Code mitigates 35B fragility and reaches the highest completion at 27B/122B (93.6%/81.6%). Mean dialog iterations range from 1.69 (27B PaL) to 2.21 (35B Prep). Multi-turn feedback yields 3 to 7 pp F1 gain for autonomous agents but less than 2 pp for tool-oriented agents (Figure 2). Tool-oriented agents arrive near-optimally after the first response, while autonomous agents benefit substantially from subsequent corrections.

3.4 Scaling Analysis

Figure 3 (Appendix) and Table 3 show Soft Triple F1 for the 35B model across $n \in \{1, 3, 10, 20, 50\}$ examples. All approaches improve from $n = 10$ to $n = 50$, with CaG+val. going from 0.441 to 0.693 and PaL from 0.388 to 0.676. At $n = 50$, PaL nearly matches CaG+val. (gap 0.017), suggesting that annotating ∼40 additional examples can narrow but not close the gap to engineered approaches. Prep exhibits a non-monotonic dip at $n = 10$ (0.279 vs. 0.385 at $n = 3$) that recovers at $n = 20$ (0.513), where a moderate example count encourages more ambitious preparation that fails at 35B capacity, while more examples provide enough signal for a successful albeit complex preparation.

4 Discussion

Validation advantage. Even at 122B, the gap between CaG+val. (0.498) and the best autonomous agent (Prep+Code, 0.458) is 0.040, which is non-trivial and consistent with the confidence intervals. Unlike AutoTOD (Xu et al., 2024), which replaces modular TOD components given a fixed API schema, our agents must discover ontology structure from raw RDF files. Schema-free discovery requires substantially more context than slot-filling, explaining why validation tooling retains a larger advantage.

Engineering and annotation trade-off. At $n = 50$, PaL nearly matches CaG+val. at 35B (gap 0.017), suggesting annotation can partially substitute for engineering effort. However, Prep's non-monotonic scaling cautions that preparation quality must scale with example count. Discussion of preparation fragility and Prep+Code behavior is in Appendix E.

5 Conclusion

On the D-WIS ontology, tool-oriented agents with explicit validation consistently outperform autonomous agents across three Qwen 3.5 model scales. The best autonomous result (Prep, 27B, 0.462 Soft Triple F1) approaches CaG+val. (0.488) with overlapping CIs, but two-phase preparation fails significantly at 35B (0.279). At $n = 50$ examples PaL nearly matches CaG+val. at 35B (gap 0.017), suggesting annotation effort can partially substitute for engineering effort. Whether this trade-off is favorable depends on the domain and the relative cost of curation versus infrastructure development.

6 Limitations

Experiments cover a single domain (D-WIS) and one model family (Qwen 3.5), so generalization requires broader validation. The simulator evaluates based on training examples rather than per-task ground-truth, so its reliability depends on training-sheet coverage, and human evaluation on a held-out subset would provide stronger guarantees.

References

Anonymous. 2026. Can autonomous study agents replace domain engineering? Concurrently submitted.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Eric Cayeux, Benoît Daireaux, Nejm Saadallah, and Sergey Alyaev. 2019. Toward seamless interoperability between real-time drilling management and control applications. In *SPE/IADC Drilling Conf. and Exh.*, page D031S017R002.

Eric Cayeux, Carlos Damski, J Macpherson, Moray Laing, Pradeep Annaiyappa, Philip Harbidge, M Edwards, and Jonathan Carney. 2023. Connecting multilayer semantic networks to data lakes: The representation of data uncertainty and quality. *SPE Drilling & Completion*, 38(01):18–33.

Brian J Chan, Chao-Ting Chen, Jui-Hung Cheng, and Hen-Hsen Huang. 2025. Don't do rag: When cache-augmented generation is all you need for knowledge tasks. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 893–897.

Harrison Chase. 2022. LangChain. GitHub.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

D-WIS Industry Group. 2025. Drilling and wells interoperability standards (d-wis). Accessed: 2025-05-31.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International conference on machine learning*, pages 10764–10799. PMLR.

Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. 2010. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond I.I.D.: Three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, pages 3477–3488.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, and 1 others. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*.

Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. 2012. OWL 2 Web Ontology Language primer (second edition). W3C Recommendation.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 6769–6781.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *International Conference on Learning Representations*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Eduardo Mosqueira-Rey, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán, and Ángel Fernández-Leal. 2023. Human-in-the-loop machine learning: a state of the art. *Artificial Intelligence Review*, 56(4):3005–3054.

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

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992.

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 *Advances in Neural Information Processing Systems*, volume 36.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 2609–2634.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yushi Xu, Zhao Meng, and Roger Wattenhofer. 2024. Rethinking task-oriented dialogue systems: From complex modularity to zero-shot autonomous agent. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7490–7505. Association for Computational Linguistics.

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

A Related Work

A.1 Prompting Strategies for Complex Reasoning

Chain-of-thought prompting (Wei et al., 2022) demonstrated that eliciting intermediate reasoning steps markedly improves LLM performance on multi-step tasks. Plan-and-Solve (Wang et al., 2023) extends this by having the model first devise a plan and then execute each step, closely related to the preparation phase used by autonomous agents.

Program-aided language models (PAL) (Gao et al., 2023) and Program-of-Thought (Chen et al., 2022) offload computation to generated code, the core strategy of the autonomous agents here. Decomposed prompting (Khot et al., 2023) decomposes tasks into sub-problems handled by specialized modules, paralleling Prep+Code's delegation to preparation and helper sub-agents.

A.2 Retrieval-Augmented and Cache-Augmented Generation

Retrieval-Augmented Generation (Lewis et al., 2020) grounds LLM outputs in retrieved external knowledge. Dense passage retrieval (Karpukhin et al., 2020) and learned embeddings (Reimers and Gurevych, 2019) are standard components. Self-RAG (Asai et al., 2023) adds self-reflection to decide when retrieval is needed. GraphRAG (Han et al., 2024) augments retrieval with structural graph information, improving coherence for knowledge-graph tasks. Cache-Augmented Generation (Chan et al., 2025) places the entire knowledge base in-context, trading retrieval complexity for prompt length.

A.3 Tool-Augmented and Agentic LLMs

Toolformer (Schick et al., 2023) and ToolLLM (Qin et al., 2024) showed that LLMs can learn to invoke external tools at scale. ReAct (Yao et al., 2023) interleaves reasoning and tool use. Wang et al. (2024) survey LLM-based autonomous agents, identifying planning, memory, and tool use as core capabilities. AutoTOD (Xu et al., 2024) replaces conventional task-oriented dialogue modules with a zero-shot LLM agent, sharing our motivation of reducing domain-specific engineering but operating in the narrower setting of fixed-schema slot-filling dialogues.

A.4 Knowledge-Base Question Answering and Ontologies

Knowledge-base question answering has been studied in both structured and graph settings (Gu et al., 2021; Gao et al., 2010). The Web Ontology Language (OWL) (Hitzler et al., 2012) provides formal ontology semantics. The D-WIS initiative (D-WIS Industry Group, 2025; Cayeux et al., 2019, 2023) defines an RDF-based ontology for drilling-domain interoperability. A concurrent work explores a similar question in a different setting, evaluating autonomous agents across four diverse tasks without a user simulator (Anonymous, 2026).

B Additional Figures

C Full Results

Table 2 reports all metrics for all seven approaches across three model scales ($n = 10$ examples, corrected). 95% bootstrap confidence intervals (1,000 resamples) are shown for Soft Triple F1.

D Extended Scaling Results

Table 3 reports Soft Triple F1 (corrected) and NGED (corrected) for the 35B model across all approaches and training example counts $n \in \{1, 3, 10, 20, 50\}$.

E Extended Discussion

Work-shift risk. A practical concern is that the proposed simplification may primarily relocate effort rather than reduce it. Building parsers and retrieval stacks is replaced by curating representative training sheets, maintaining coverage as schemas evolve, and auditing examples for hidden leakage or bias. In domains with sparse historical configurations or frequent ontology changes, this curation burden can become a bottleneck. A fair comparison should therefore include not only runtime and output quality, but also the human cost of creating and maintaining high-quality example sets over time.

Preparation fragility at 35B. The 35B Prep failure at $n = 10$ is an important caution against evaluating autonomous agents at only a single scale or single example count. The dip is large (from 0.385 at $n = 3$ to 0.279 at $n = 10$), counterintuitive (more examples should help), and recovers at $n = 20$ (0.513). One hypothesis is that 10 examples are enough to make the preparation agent attempt a more ambitious knowledge organization, which then fails at 35B capacity. With 3 examples, simpler preparation succeeds, and with 20 examples, enough signal exists for a complex preparation to succeed. This hypothesis deserves targeted investigation.

Simulator reliability. The simulator evaluates agent outputs using training examples as domain knowledge, without access to per-task ground-truth configurations. This design simplifies deployment, as no labeling per test task is required, but its acceptance decisions reflect training-example coverage rather than exact ground-truth matching. Tasks that differ substantially from the training distribution may be evaluated less reliably. Future work should

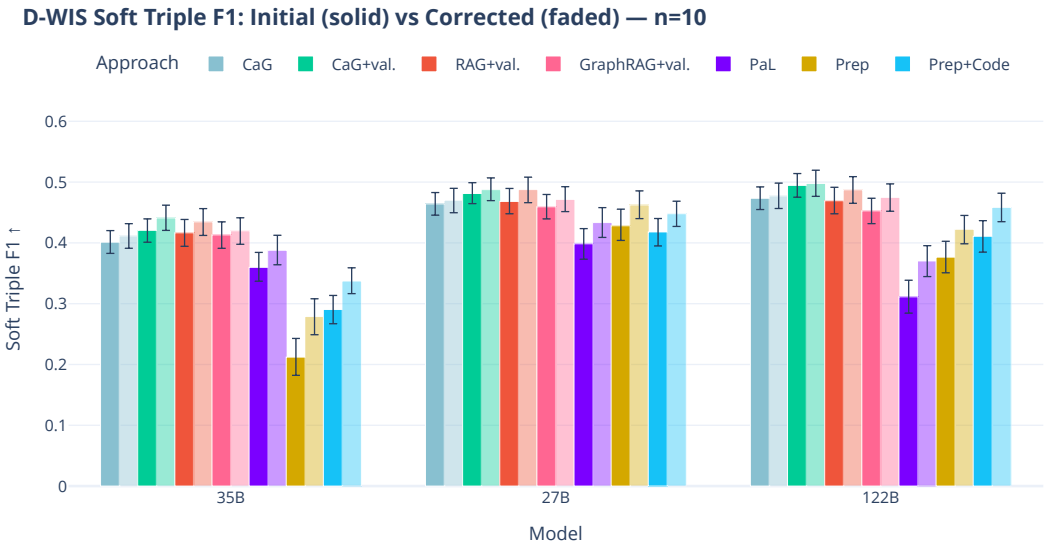

Figure 2: Initial (solid) vs. corrected (faded) Soft Triple F1 for all approaches across three model scales. Autonomous agents gain 3 to 7 percentage points from multi-turn simulator feedback, while tool-oriented agents gain at most 2 pp as their validation loop already resolves most errors within the first turn.

include human evaluation on a representative subset to validate the simulator's signal quality and assess whether any agent category benefits disproportionately from training-example-based grading.

F Task Example

Table 4 shows a representative D-WIS task. The user describes a drilling sensor configuration in natural language. The system must produce declarations D (class instantiations) and triples T (relationships) consistent with the D-WIS ontology.

In the multi-turn protocol, the user simulator compares the generated configuration against c^* and provides natural-language feedback, for example "*The pressure quantity should have a drilling component. Also, the standpipe component is missing.*" The agent then corrects its output.

G RAG Retrieval

The retrieval used by RAG+val. and GraphRAG+val. is exposed as a `retrieve` tool. This allows the agent to retrieve at its own discretion, rather than being forced to retrieve for every query, and to iteratively refine retrieval queries based on feedback. Results include class predecessors, immediate children, and sister nodes formatted into a hierarchical tree to provide context for near-miss classes.

GraphRAG extends retrieval by re-ranking retrieved ontology fragments based on graph-structural connectivity scores, favoring coherent subgraphs over isolated matches.

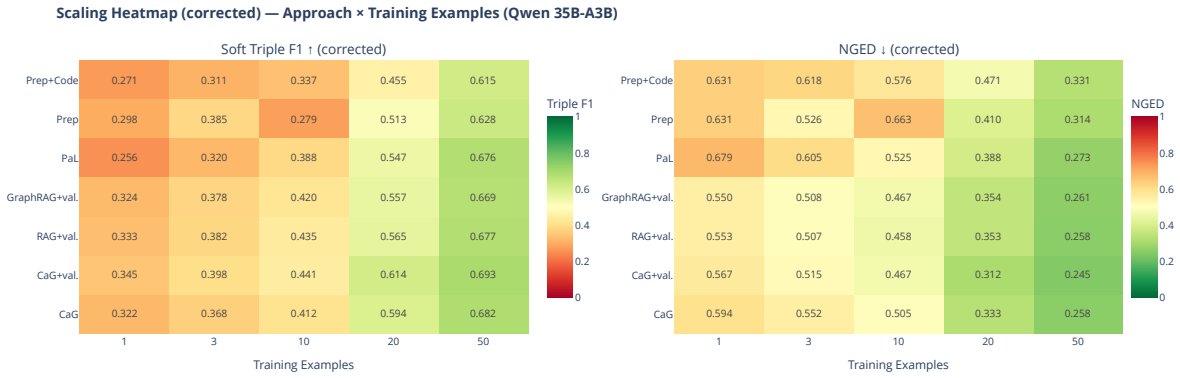

Figure 3: D-WIS Soft Triple F1 (corrected) as a function of training examples (Qwen 35B-A3B). All approaches improve substantially from $n = 10$ to $n = 50$.

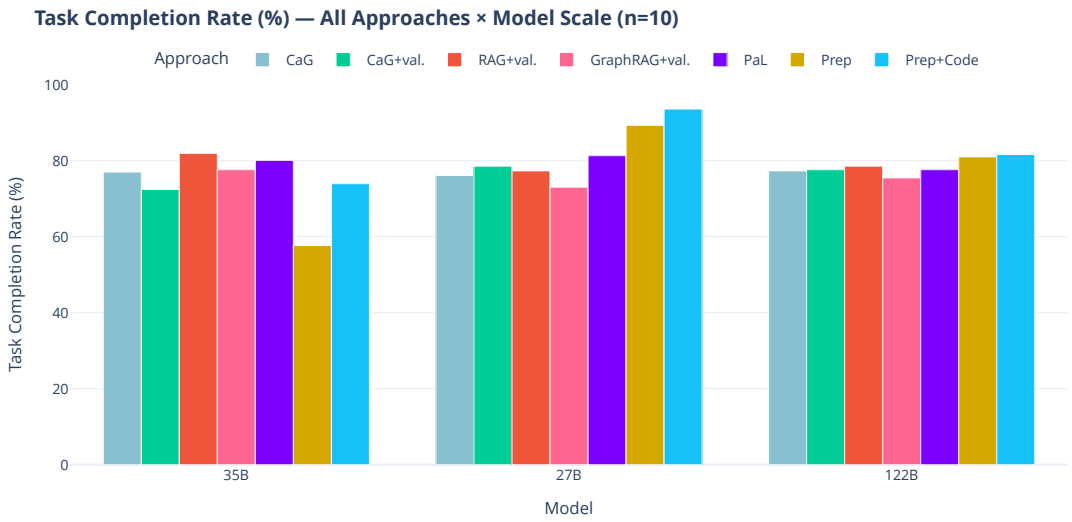

Figure 4: Task completion rate and mean dialog iterations across all approaches and model scales.

Table 2: Full D-WIS results on 326 tasks ($n = 10$, corrected metrics, 95% CIs on Triple F1). Best per model per column in **bold**.

Model	Approach	Soft Triple F1↑	Soft Class F1↑	NGED↓	Compl. (%)	Iters↓
35B	CaG	0.412 [0.391, 0.434]	0.653	0.505	77.0	2.02
	CaG+val.	**0.441** [0.421, 0.462]	**0.671**	0.467	72.4	1.90
	RAG+val.	0.435 [0.412, 0.456]	0.615	**0.458**	**81.9**	1.81
	GraphRAG+val.	0.420 [0.399, 0.441]	0.590	0.467	77.6	1.79
	PaL	0.388 [0.364, 0.413]	0.573	0.525	80.1	1.92
	Prep	0.279 [0.249, 0.308]	0.435	0.663	57.7	2.21
	Prep+Code	0.337 [0.317, 0.359]	0.569	0.576	73.9	2.18
27B	CaG	0.470 [0.449, 0.490]	0.697	0.448	76.1	2.04
	CaG+val.	0.488 [0.469, 0.507]	**0.705**	0.426	78.5	1.99
	RAG+val.	**0.488** [0.466, 0.508]	0.659	**0.411**	77.3	2.00
	GraphRAG+val.	0.472 [0.451, 0.492]	0.630	0.423	73.0	2.02
	PaL	0.433 [0.409, 0.458]	0.605	0.477	81.3	1.69
	Prep	0.462 [0.440, 0.485]	0.635	0.450	89.3	1.78
	Prep+Code	0.448 [0.427, 0.469]	0.647	0.465	**93.6**	1.71
122B	CaG	0.477 [0.456, 0.499]	0.702	0.435	77.3	1.94
	CaG+val.	**0.498** [0.477, 0.520]	**0.713**	0.414	77.6	1.98
	RAG+val.	0.487 [0.465, 0.509]	0.651	**0.411**	78.5	2.05
	GraphRAG+val.	0.475 [0.452, 0.497]	0.624	0.419	75.5	1.96
	PaL	0.370 [0.345, 0.395]	0.533	0.549	77.6	1.87
	Prep	0.422 [0.398, 0.445]	0.602	0.488	81.0	1.92
	Prep+Code	0.458 [0.435, 0.482]	0.628	0.460	**81.6**	2.02

	NGED↓ (corrected)					Soft Triple F1↑ (corrected)				
Approach	n=1	n=3	n=10	n=20	n=50	n=1	n=3	n=10	n=20	n=50
CaG	0.594	0.552	0.505	0.333	0.258	0.322	0.368	0.412	0.594	0.682
CaG+val.	0.567	0.515	0.467	0.312	**0.245**	0.345	0.398	0.441	0.614	**0.693**
RAG+val.	0.553	0.507	0.458	0.353	0.258	0.333	0.382	0.435	0.565	0.677
GraphRAG+val.	0.550	0.508	0.467	0.354	0.261	0.324	0.378	0.420	0.557	0.669
PaL	0.679	0.605	0.525	0.388	0.273	0.256	0.320	0.388	0.547	0.676
Prep	0.631	0.526	0.663	0.410	0.314	0.298	0.385	0.279	0.513	0.628
Prep+Code	0.631	0.618	0.576	0.471	0.331	0.271	0.311	0.337	0.455	0.615

Table 3: D-WIS scaling by training examples (35B, corrected metrics). Note the Prep agent's non-monotonic dip at $n = 10$ (0.279 vs. 0.385 at $n = 3$).

Table 4: Example D-WIS task for standpipe pressure monitoring.

User request q
"Set up standpipe pressure monitoring with moving-average filtering, transmitted via top-side telemetry and provided by a data flow node."
Expected configuration $c^* = (D^*, T^*)$

Declarations D^**:*
`SPP:SPP_0`
`MudStandpipe:StandPipe_0`
`PressureDrillingQuantity:SPPQuantity_0`
`TimeWindowMovingAverage:TWAvg_0`
`TopSideTelemetry:TopSideTelemetry_0`
`DataFlowNode:DataFlowNode_0`
Triples T^**:*
`SPP_0 IsHydraulicallyLocatedAt StandPipe_0`
`SPP_0 IsOfMeasurableQuantity SPPQuantity_0`
`SPP_0 IsProcessedBy TWAvg_0`
`SPP_0 IsTransmittedBy TopSideTelemetry_0`
`SPP_0 IsProvidedBy DataFlowNode_0`