Domain Adaptation with LLM Study Agents

Anonymous ACL submission

Abstract

Deploying large language models (LLM) over specialized domain tasks typically demands domain-specific parsers, embedding pipelines, and retrieval modules. In this paper, we explore whether autonomous LLM agents can replace this engineering, given only examples, optional raw data access, and a code sandbox. We compare domain-specific baselines against three autonomous agents sharing a single architecture across four tasks spanning configuration generation, question answering, document retrieval, and mathematical reasoning. Autonomous agents surpass baselines on three of four tasks but fall short on configuration generation where domain-specific validation tooling retains an advantage. However, scaling analysis shows that additional training examples can substitute these tools, suggesting that annotation effort can often replace engineering effort. Code available at https://anonymous.4open.science/r/Study-Agents-B65E.

1 Introduction

Specialized domains (ontologies, document corpora and structured datasets) encode knowledge in formats that require considerable expertise to exploit programmatically. Translating natural-language queries into correct domain-specific outputs over such resources typically demands custom infrastructure. Domain-specific parsers, retrieval pipelines, and validation modules must be built and maintained for the target domain (Lewis et al., 2020; Han et al., 2024; Reimers and Gurevych, 2019). As the underlying knowledge evolves, this infrastructure must be maintained in step, creating a persistent engineering burden.

Recent advances in LLM capabilities have reduced the cost of individual pipeline stages (Brown et al., 2020; Chen et al., 2021), but the overall architecture remains a human design decision. For each new domain, engineers must decide *what* information to retrieve from the data source, *how* to parse and represent the domain-specific format, and *when* to validate outputs against domain constraints. We propose to delegate these decisions to the LLM itself.

The central question this paper addresses is whether an LLM agent, equipped with (i) a small set of input-output examples, (ii) read access to any available raw data files (e.g., ontologies, document corpora, or structured tables), and (iii) a general-purpose code sandbox, can learn domain structure autonomously and produce outputs competitive with hand-crafted baselines.

These three constraints are deliberately minimal. The examples are pairs of natural-language queries and expected outputs. If the task involves external data, the agent receives the unprocessed files with no schema parser, embedding index, or retrieval module provided. Some tasks provide no external data at all. The sandbox is a persistent Python environment with standard data-science libraries. Any domain-specific logic must be discovered and implemented by the agent's own code.

To test this, we introduce a *Study Agent* that operates in two phases. During *preparation*, the agent studies a set of training examples, explores any available data files through sandbox code execution, and distils a natural-language knowledge summary capturing domain patterns, data structures, and task conventions. During *testing*, the agent answers new queries using this stored summary without repeating exploration. Because preparation is not focused on any single user query, the agent can explore the data broadly, discovering structural patterns that may benefit many downstream queries. Some of this exploration may be unproductive, but the insights that do transfer are captured in the knowledge summary and reused across all test queries.

We evaluated four tasks spanning configuration generation (D-WIS), MultiHop question answering (MultiHopRAG), document retrieval (BRIGHT),

and mathematical reasoning (LiveBench-Math). For each task, we compare domain-specific baselines against three autonomous agents sharing the same generic architecture.

Our contributions are:

1. We propose a two-phase autonomous Study Agent architecture (preparation + testing) that replaces domain-specific infrastructure with sandbox-based exploration for the purpose of domain knowledge extraction.
2. A multi-task evaluation spanning four datasets and four task types, comparing domain-specific baselines against generic autonomous agents.
3. Empirical evidence characterizing the trade-off between domain-specific engineering and autonomous exploration across diverse tasks.

2 Related Work

2.1 Retrieval and Context Augmentation

Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) grounds LLM outputs in retrieved external knowledge, using dense passage retrieval (Karpukhin et al., 2020). Self-RAG (Asai et al., 2023) extends this with self-reflection to decide when retrieval is necessary. GraphRAG (Han et al., 2024) augments retrieval with graph-structural information, improving coherence for structured knowledge tasks by favoring connected subgraphs over isolated matches. Cache-Augmented Generation (CaG) (Chan et al., 2025) takes the opposite approach and places the entire knowledge base in the context window, trading retrieval complexity for prompt length. These methods all require domain-specific infrastructure (embedding models, indexing pipelines, or parsers) that must be designed and maintained for each new domain.

2.2 Prompting Strategies for Complex Reasoning

Chain-of-Thought (CoT) 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 CoT by having the model first devise a plan and then execute each step. Tree-of-Thought (ToT) (Yao et al., 2023a) enables deliberate multi-path problem solving by exploring and evaluating multiple reasoning branches. Decomposed prompting (Khot et al., 2023) decomposes tasks into sub-problems handled by specialized modules.

2.3 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. ReAct (Yao et al., 2023b) interleaves reasoning traces with tool-use actions, providing an influential paradigm for agentic tool use. ReWOO (Xu et al., 2023) decouples reasoning from observation to improve efficiency. Wang et al. (2024) survey LLM-based autonomous agents, identifying planning, memory, and tool use as core capabilities. Code generation with LLMs (Chen et al., 2021) provides conceptual foundations for sandbox-based approaches.

A related line of work replaces modular task-oriented dialogue (TOD) pipelines with zero-shot LLM agents. AutoTOD (Xu et al., 2024) deprecates the conventional dialogue state tracker, policy, and NLG modules in favour of a single instruction-following LLM that autonomously decides what to do at each dialogue turn given a predefined API schema. While sharing the motivation of reducing domain-specific engineering, TOD agents operate in a narrower setting. They assume a fixed set of APIs with known slot schemas and are evaluated on slot-filling dialogues (MultiWOZ, SGD).

2.4 Program-Aided and Code-Generating Approaches

Program-aided Language models (PaL) (Gao et al., 2023) and Program-of-Thought (Chen et al., 2022) offload structured reasoning and computation to generated Python code. PaL generates code for every query independently, with no mechanism to accumulate domain knowledge across queries.

2.5 Agent Knowledge and Prompt Optimization

Saveliev et al. (2025) propose human-guided, data-centric LLM co-pilots that leverage human feedback to iteratively improve model behavior. Tang et al. (2025) introduce Agent KB, which leverages cross-domain experience for agentic problem solving by maintaining a shared knowledge repository. Vasilopoulos (2026) describe codified context, an infrastructure layer that provides structured domain knowledge to AI agents operating in complex codebases. These works share the motivation of reducing manual engineering through agent autonomy. A concurrent work explores a similar question in

a different setting, using an LLM-based user simulator for multi-turn refinement on a single ontology (Anonymous, 2026).

3 Methodology

All systems are implemented as LangChain (Chase, 2022) tool-calling agents, relying on the model's native function-calling capability. The systems are organized into two categories, domain-specific baselines and autonomous agents, distinguished by their infrastructure requirements.

3.1 Task Formulation

Given a natural-language request q, the system produces a response r. Where the task involves external data, the agent also receives a data source $\mathcal{D}$. Each dataset defines the nature of r and $\mathcal{D}$:

- **D-WIS.** $\mathcal{D}$ is an RDF/Turtle ontology ($\sim$2 200 classes, $\sim$250 properties) for drilling-domain interoperability (D-WIS Industry Group, 2025; Cayeux et al., 2019, 2023). The response r is a configuration of class instance declarations and relationship triples that must be consistent with the ontology schema.
- **MultiHopRAG.** $\mathcal{D}$ is a corpus of 609 news articles. The response r is a natural-language answer that requires aggregating evidence from multiple articles (Tang and Yang, 2024).
- **BRIGHT.** $\mathcal{D}$ is a collection of long domain documents across multiple knowledge areas. The response r is a natural-language answer requiring implicit, multi-step reasoning to identify relevant passages (Su et al., 2024).
- **LiveBench-Math.** No data source is provided. The response r is a mathematical answer to olympiad-style problems with contamination-resistant evaluation through regularly refreshed problems (White et al., 2025).

3.2 Domain-Specific Baselines

For each dataset we have implemented a set of baselines that are appropriate to the task according to literature. All baselines require custom infrastructure tailored to the target dataset. Importantly, the design of each baseline reflects our knowledge of what matters for the task. For example, the MultiHopRAG RAG (Lewis et al., 2020) baseline deliberately encodes source, title, and publication date into each chunk because the developer knows these metadata fields are important for multi-hop reasoning. This kind of informed design choice is unavailable to autonomous agents, which must discover relevant structure from the data alone.

D-WIS baselines. Two systems exploit the ontology's RDF structure:

1. **CaG + validation** extends CaG with a validation tool that parses the generated configuration against the ontology, reports class and predicate errors, and enables iterative self-correction. Due to the ontology size, the information is compressed into a formatted hierarchical tree to fit the context window (Chan et al., 2025).
2. **RAG + validation** replaces the full-context prompt with on-demand semantic retrieval over ontology classes and properties, using the same validation tool. To avoid near misses, the results include class predecessors, immediate child nodes, and sister nodes formatted into hierarchical tree.

MultiHopRAG baselines. Two systems address the multi-document reasoning task:

1. **In-Memory** tests the lower bound of the task by testing what the model knows on its own as the news articles contain very publicized events the model may have been trained on.
2. **RAG** performs embedding-based retrieval over the article corpus to return relevant passages per query with maximal marginal relevance (MMR) to obtain more diverse evidence. The number of candidates is 20 with the λ parameter at 0.5.

BRIGHT baselines. One system addresses the reasoning-intensive retrieval task:

1. **RAG** performs embedding-based retrieval with MMR search over the long domain documents, applying a query-time instruction prefix to improve retrieval quality for implicit reasoning queries. Retains the same parameters as MultiHopRAG RAG baseline.

LiveBench-Math baselines. Two systems address mathematical reasoning:

1. **CoT** prompts the agent to produce step-by-step reasoning before the final answer (Wei et al., 2022).
2. **ToT** has the agent explore multiple reasoning paths, evaluate intermediate steps, and select the most promising branch (Yao et al., 2023a).

3.3 Autonomous Agents

The three autonomous agents require no domain-specific infrastructure. Each receives a *training sheet*, a document containing input-output examples for the target task, and, where available, access to the raw data files. All agents share a code sandbox in which they write and execute Python code to explore any available data, analyze patterns, and construct responses programmatically. The same agent architecture is applied across all four datasets without modification.

Program-aided Language model Following Gao et al. (2023), the agent delegates structured reasoning to generated Python code. Given a query, it writes scripts that access any available data files, analyze their content, and assemble the output. The training sheet provides in-context examples of the expected format.

Study Agent extends PaL with a dedicated *preparation phase* that runs once before any test queries. Whereas PaL re-explores from scratch for each query, the Study Agent performs this exploration once and reuses the result. During preparation, a *Preparation Agent* studies the training sheet examples, explores available data via sandbox code execution, and commits structured findings as named knowledge entries. The result is a **knowledge summary**, a structured natural-language document capturing domain patterns, data structures, and task conventions discovered during exploration. This summary is included in the system prompt at test time. During testing, the agent answers queries using the knowledge summary and transient sandbox code execution. Each query may trigger fresh code to access data files, but the agent does not repeat the broad exploratory work performed during preparation. Visualization of the workflow can be seen in Figure 1.

Study Agent with Delegation (Study + Code) extends the Study Agent by adding a second preparation stage after knowledge extraction. In the first stage, the same Preparation Agent explores the available data and builds the knowledge summary. In the second stage, a dedicated *Helper Agent* receives the extracted knowledge and training sheet and builds a library of reusable Python helper functions. The helper agent first prototypes candidate functions via transient execution, then exposes its implementations as persistent code in the sandbox. During testing, the agent calls these pre-built helpers to answer queries, reducing per-query exploration. This decomposition separates knowledge discovery from tool creation, and can introduce potential error accumulation through the two stages.

3.4 Evaluation Protocol

Each system receives a single query per task and its response is evaluated against the gold-standard answer. Metrics vary by dataset:

- **D-WIS** uses ontology-aware RDF graph metrics computed after bipartite instance alignment: Soft Triple F1 and Soft Class F1 (each using hierarchy-distance-based similarity $1/(1+d)$ via `rdfs:subClassOf`/`rdfs:subPropertyOf`) and Normalized Graph Edit Distance (NGED $\in [0, 1]$, lower is better).
- **MultiHopRAG** uses accuracy (exact match) and BERTScore (Zhang et al., 2020).
- **BRIGHT** uses document recall, and F1 over the gold retrieved document set.
- **LiveBench-Math** uses accuracy (exact match).

4 Experimentation

Setup. All experiments use models from the Qwen 3.5 family (Qwen Team, 2026) at four scales. Two are dense (4B and 27B active parameters) and two use Mixture-of-Experts (MoE) architectures where only a fraction of parameters are active per token[1]:

- **Qwen3.5-4B**: dense, **4B active**.
- **Qwen3.5-35B-A3B**: MoE, 35B total, **3B active**.
- **Qwen3.5-27B**: dense, **27B active**.
- **Qwen3.5-122B-A10B**: MoE, 122B total, **10B active**.

Unless stated otherwise, training sheets contain 10 examples and evaluation is performed on held-out test sets (326 examples for D-WIS, 500 for MultiHopRAG, 500 for BRIGHT, and 257 for LiveBench-Math). The external benchmarks use a random split. For D-WIS, which is our own dataset, we filter the split by pairwise similarity (fuzzy text

[1]Because the 27B dense model activates nearly 3× as many parameters per token as the 122B-A10B model, the two are comparable in overall capability (Artificial Analysis, 2026). Both substantially outperform the 35B-A3B model despite its larger total parameter count. We therefore order models by effective capability as 4B $<$ 35B $<$ 27B $\approx$ 122B throughout.

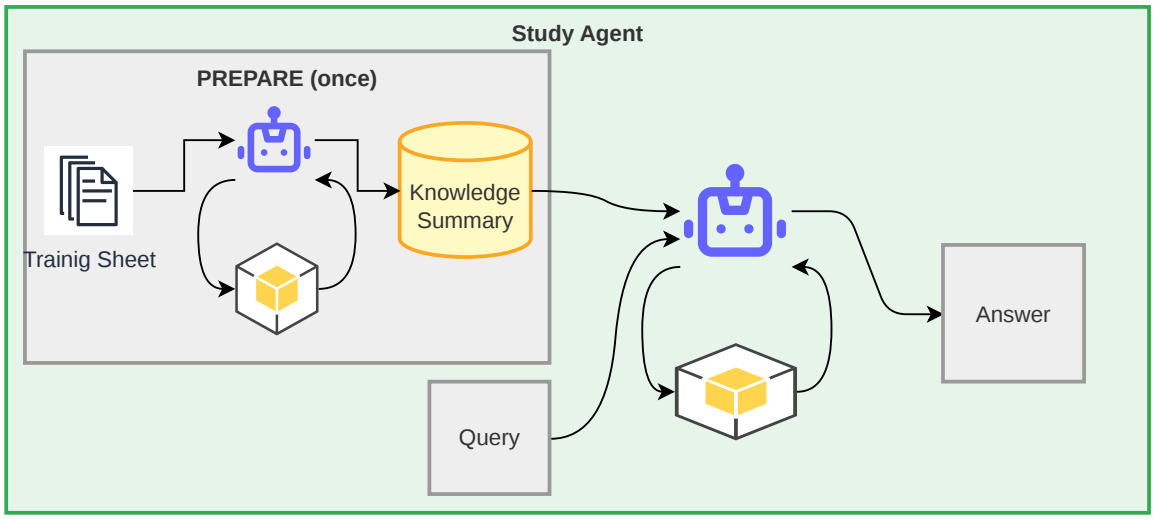

Figure 1: Workflow of Study Agent. Preparation uses sandbox to explore and mark useful knowledge about the domain. During testing, the knowledge is used to skip domain exploration.

Model	Approach	Soft Tri. F1↑	Soft Cls. F1↑	NGED↓
35B	CaG+val.	0.390	**0.606**	0.529
	RAG+val.	**0.423**	0.593	**0.467**
	PaL	0.369	0.541	0.547
	Study	0.304	0.485	0.624
	Study+Code	0.319	0.554	0.608
27B	CaG+val.	**0.492**	**0.708**	0.414
	RAG+val.	0.479	0.634	**0.411**
	PaL	0.389	0.567	0.530
	Study	0.416	0.557	0.509
	Study+Code	0.380	0.587	0.546
122B	CaG+val.	**0.499**	**0.715**	**0.407**
	RAG+val.	0.472	0.618	0.413
	PaL	0.326	0.472	0.604
	Study	0.423	0.589	0.497
	Study+Code	0.239	0.378	0.702

Table 1: D-WIS results by approach and model scale (full results in Table 7).

Model	Approach	Accuracy↑	BERT F1↑
35B	InMemory	0.642	0.741
	RAG	0.758	0.805
	PaL	0.788	0.825
	Study	**0.790**	**0.846**
	Study+Code	0.714	0.790
27B	InMemory	0.600	0.664
	RAG	0.806	0.851
	PaL	0.796	0.836
	Study	**0.864**	**0.906**
	Study+Code	0.846	0.885
122B	InMemory	0.624	0.687
	RAG	0.728	0.790
	PaL	0.730	0.773
	Study	0.802	0.854
	Study+Code	**0.826**	**0.878**

Table 2: MultiHopRAG results (full results in Table 7). The Study Agent achieves the highest accuracy (0.864) and BERT F1 (0.906) on 27B.

matching combined with graph-structural comparison of the gold RDF configurations, threshold 0.9) to prevent near-duplicates from straddling train and test[2]. Retrieval-based baselines use Qwen3-Embedding-4B as the embedding model. All autonomous agents use the same codebase without dataset-specific modifications.

4.1 D-WIS Results

Table 1 presents D-WIS results (see also Figure 4 in Appendix A.4). Domain-specific baselines lead overall. CaG+val on 122B achieves the best Soft Triple F1 (0.499), Soft Class F1 (0.715), and NGED (0.407). The validation tool provides direct feedback on ontology constraint violations, enabling iterative self-correction that autonomous agents cannot replicate.

Autonomous agents narrow the gap at larger scales. The 122B Study Agent achieves 0.423 Soft Triple F1, matching 35B RAG+val (0.423) and reducing the gap with 122B CaG+val. On 27B, Study achieves 0.557 Soft Class F1, comparable to CaG+val on 35B (0.606), suggesting that the agent identifies the correct ontology classes but struggles with the finer-grained predicate structure.

4.2 MultiHopRAG Results

Table 2 and Figure 2 show that autonomous agents surpass domain-specific baselines on this multi-hop QA task. The Study agent on 27B achieves

[2]All splits use a fixed seed.

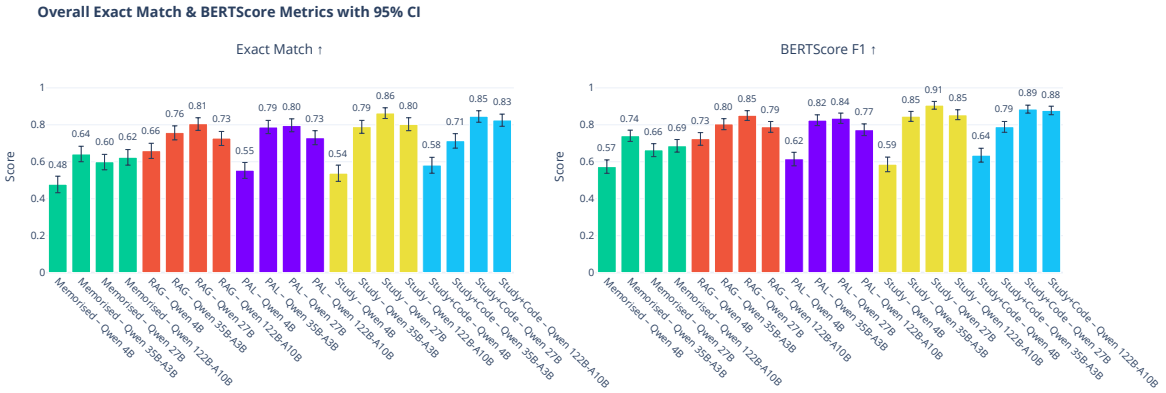

Figure 2: MultiHopRAG overall metrics by approach. Autonomous agents outperform the embedding-based RAG baseline across model scales.

Model	Approach	Recall↑	Doc F1↑
35B	RAG	0.195	0.136
	PaL	0.163	0.139
	Study	**0.504**	**0.422**
	Study+Code	0.240	0.198
27B	RAG	0.576	0.379
	PaL	0.087	0.067
	Study	**0.730**	**0.578**
	Study+Code	0.712	0.531
122B	RAG	0.350	0.235
	PaL	0.129	0.110
	Study	**0.622**	**0.521**
	Study+Code	0.565	0.448

Table 3: BRIGHT results (full results in Table 7). Document recall is the primary metric. The 27B Study Agent achieves the highest recall (0.730) and Doc F1 (0.578).

Model	Approach	Accuracy↑
35B	CoT	0.366
	ToT	0.444
	PaL	**0.537**
	Study	0.533
	Study+Code	0.510
27B	CoT	0.486
	ToT	0.436
	PaL	0.537
	Study	0.564
	Study+Code	**0.572**
122B	CoT	0.560
	ToT	0.560
	PaL	**0.580**
	Study	0.568
	Study+Code	0.498

Table 4: LiveBench-Math accuracy (full results in Table 7). Code-executing agents (PaL, Study) match or exceed reasoning-only baselines (CoT, ToT).

86.4% accuracy and 0.906 BERT F1, outperforming RAG (80.6% accuracy) by 5.8 percentage points. On 122B, Study + Code reaches 82.6% accuracy vs. 72.8% for RAG, a 9.8-point gap. PaL performs competitively (78.8% accuracy on 35B), showing that code-based data access is effective even without preparation. The delegation variant (Study + Code) improves over the plain Study Agent at 122B (82.6% vs. 80.2%) where persistent helper functions amortize cross-document exploration, but does not consistently help at smaller scales where coordination overhead may offset the benefit.

4.3 BRIGHT Results

Table 3 presents BRIGHT results (see also Figure 5 in Appendix A.4). Study Agents outperform the RAG baseline on document recall across all model scales. The 27B Study Agent achieves 0.730 recall versus 0.576 for the same-scale RAG baseline, a 15.4-point improvement, because the preparation phase allows the agent to build custom search strategies for implicit reasoning queries. Even on 35B, the Study Agent (0.504) more than doubles RAG (0.195). On 122B, Study reaches 0.622 recall and 0.521 Doc F1, substantially exceeding RAG (0.350). PaL remains below RAG across all scales despite having sandbox access, confirming that broad preparation is more beneficial than per-query exploration for this task. BRIGHT queries require implicit, multi-step reasoning to identify relevant passages, a setting where code-based exploration can implement custom search logic that surface-level embedding similarity misses.

4.4 LiveBench-Math Results

Table 4 (see also Figure 6 in Appendix A.4) shows that code-executing agents consistently match or exceed reasoning-only baselines on mathematical tasks. On 27B, Study + Code achieves 57.2% accuracy compared to 48.6% for CoT and 43.6% for ToT. PaL achieves the best overall result (58.0%) on 122B. The advantage of code execution is most pronounced at smaller scales. On 35B, PaL (53.7%) outperforms ToT (44.4%) by 9.3 percentage points. At 122B the gap narrows as the model's native reasoning ability improves. The preparation phase provides no benefit on this task, as mathematical problems are largely self-contained and do not require cross-query knowledge accumulation.

4.5 Scaling Analysis

D-WIS performance scales substantially with training examples (full data in Table 5 and Figure 3 in the Appendix). On 35B, CaG+val improves from 0.555 NGED (1 example) to 0.255 (40 examples), and Study from 0.660 to 0.327. At 40 examples all approaches converge. The key trade-off is that the Study Agent with 20 examples (0.427 NGED) surpasses CaG+val at 10 examples (0.529 NGED), meaning that annotating roughly 10 additional examples can substitute for building domain-specific validation tools.

5 Discussion

Cross-dataset patterns. The results reveal a clear dichotomy. Autonomous agents excel at question answering (MultiHopRAG), document retrieval (BRIGHT), and mathematical reasoning (LiveBench-Math) but lag behind domain-specific baselines on structured configuration generation (D-WIS). The common factor across the first three tasks is that they permit flexible, code-driven data access, where the agent can write scripts to search, filter, and cross-reference data contents, effectively reimplementing retrieval or computation logic on the fly. D-WIS, by contrast, requires generating output that conforms to a formal ontology schema with ∼2,200 classes and ∼250 properties. The domain-specific validation tool provides direct feedback on constraint violations, creating a tight correction loop that the code sandbox cannot replicate.

Preparation fragility. Because all test queries share a single preparation run, test-time results are not independent. A flawed preparation propagates to every downstream answer. On D-WIS 122B, Study + Code degrades because the helper-building phase committed suboptimal abstractions that the testing agent could not override. This collapse does not reflect per-query difficulty but rather a structural property of the two-phase design, where preparation is a single point of failure.

The role of delegation. The Study + Code variant improves over the plain Study Agent in several settings, most notably on MultiHopRAG 122B, where persistent helper functions amortize cross-document exploration. However, delegation does not consistently help. On MultiHopRAG 27B the plain Study Agent outperforms Study + Code, and on D-WIS 122B, Study + Code underperforms PaL, as discussed above under preparation fragility.

Model scaling. The 27B dense model and the 122B-A10B MoE model have comparable overall capability despite different architectures. This parity is reflected in our results, where both models achieve similar scores across tasks. The more informative comparison is against the 35B-A3B model, which has only 3B active parameters. Here, the jump from 35B to 27B is large and consistent across all datasets and approaches. This confirms that effective model capacity (active parameters) rather than total parameter count drives performance, and that autonomous agents benefit from increased capacity at least as much as domain-specific baselines.

Designer knowledge and generalization. Domain-specific baselines benefit from the developer's prior understanding of what matters for each task. The MultiHopRAG RAG baseline encodes source, title, and publication date into each retrieved chunk because the developer knows these fields support multi-hop reasoning. The D-WIS validation tool checks generated triples against the ontology schema because the developer knows which constraints are most often violated. This informed design is effective but task-specific, and it does not transfer to new domains without additional engineering. Autonomous agents, by contrast, must discover relevant data structure on their own. That they nonetheless match or exceed baselines on three of four tasks suggests that the preparation phase can recover much of the insight that developers encode manually.

Code sandbox and generalization trade-off. Across all four datasets, the same Python sandbox replaces fundamentally different infrastructure,

including an RDF parser and ontology validator (D-WIS), an embedding index with MMR retrieval (MultiHopRAG, BRIGHT), and symbolic computation (LiveBench-Math). The agent autonomously discovers what code to write based on the data modality. This universality means deploying on a new dataset requires only curating a training sheet and providing file access, with no infrastructure development. As shown in the scaling analysis (Section 4.5), approximately 10 additional training examples for the Study Agent can match the benefit of purpose-built CaG/RAG tools, offering a path to improvement that scales with annotation effort rather than engineering effort.

6 Conclusion

We presented a two-phase autonomous Study Agent architecture that provides an alternative to domain-specific software. Instead, it relies on a LLM, sandbox-based exploration and a small set of training examples. Evaluation across four datasets shows that autonomous agents can outperform domain-specific baselines on question answering, document retrieval, and mathematical reasoning. It achieves 86.4% accuracy on MultiHopRAG and 73% document recall on BRIGHT, while matching or exceeding CoT and ToT baselines on LiveBench-Math.

The extended delegation variant of the agent (Study + Code) improves over the base Study Agent in several settings, though not universally, suggesting that separating knowledge discovery from tool creation is beneficial when query-time exploration cost is high.

Currently, the most challenging task for the agents is the D-WIS configuration generation. Domain-specific validation tooling retains a clear advantage, delineating the current boundary of the approach. However, the scaling analysis shows that both autonomous and domain-specific approaches benefit substantially from additional training examples.

Future work includes multi-agent coordination, where specialized sub-agents handle different aspects of data exploration, and scaling to larger training sets. The results suggest that for many practical domain tasks, the engineering effort is better invested in curating examples than in building custom infrastructure.

Limitations

Several limitations apply to the reported results. First, all experiments use a single model family (Qwen 3.5). While we evaluate across four scales (4B to 122B), the two largest models (27B dense and 122B-A10B MoE) have comparable effective capacity, so our results span only three distinct capability tiers. Generalization to other architectures (e.g., Llama, GPT) remains untested. Additionally, the code sandbox introduces latency. Each tool call requires Python process communication and execution, making autonomous agents slower per query than direct-generation baselines. Training examples quality may also bias results, as the agent's preparation depends on the coverage and representativeness of the provided examples. Finally, we do not provide a formal cost comparison between the engineering effort required to build domain-specific infrastructure and the annotation effort required to curate training examples. This trade-off is application-dependent.

References

Anonymous. 2026. Reducing domain engineering effort with autonomous agent learning. Concurrently submitted.

Artificial Analysis. 2026. Qwen3.5 122b a10b vs. qwen3.5 27b: Model comparison. https://artificialanalysis.ai/models/comparisons/qwen3-5-122b-a10b-vs-qwen3-5-27b. Accessed: 2026-04-20.

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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Gretchen Krueger, and 1 others. 2021. Evaluating large language models trained on code. Technical report, OpenAI.

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.

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*.

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.

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*.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

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.

Evgeny Saveliev, Jiashuo Liu, Nabeel Seedat, Anders Boyd, and Mihaela van der Schaar. 2025. Towards human-guided, data-centric LLM co-pilots. *arXiv preprint arXiv:2501.10321*.

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.

Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han-yu Wang, Haisu Liu, Quan Shi, Zachary S Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Sercan O Arik, Danqi Chen, and Tao Yu. 2024. Bright: A realistic and challenging benchmark for reasoning-intensive retrieval.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, and 1 others. 2025. Agent KB: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*.

Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *Preprint*, arXiv:2401.15391.

Aristidis Vasilopoulos. 2026. Codified context: Infrastructure for AI agents in a complex codebase. *arXiv preprint arXiv:2602.20478*.

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.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025. LiveBench: A challenging, contamination-limited LLM benchmark. In *International Conference on Learning Representations*. ArXiv:2406.19314.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. ReWOO: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*.

Heng-Da Xu, Xian-Ling Mao, Puhai Yang, Fanshu Sun, and Heyan Huang. 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 2748–2763, Bangkok, Thailand. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*. ArXiv:2305.10601.

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

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*. ArXiv:1904.09675.

A Appendix

A.1 D-WIS Scaling with Training Examples

Table 5 reports D-WIS NGED (lower is better) and Soft Triple F1 (higher is better) for the 35B model across all approaches as the number of training examples varies from 1 to 40. Figure 3 shows the corresponding heatmap. Study follows a non-monotonic trajectory, dipping at 3–5 examples before recovering at 10+. At 40 examples, PaL (0.280 NGED) nearly matches CaG+val (0.255), suggesting that sufficient examples may eliminate the need for any preparation phase.

A.2 MultiHopRAG Per-Question-Type Breakdowns

Table 6 reports per-question-type accuracy for all approaches on MultiHopRAG across four model scales. Null queries (where the answer is not in the corpus) approach perfect accuracy for Study Agents at 27B+, but domain-specific RAG struggles with these (0.31–0.87), likely because irrelevant retrieved passages mislead the model. Temporal queries requiring date reasoning remain the most difficult category across all approaches and scales. Inference queries are generally easiest for all systems.

A.3 Consolidated Results

Table 7 reports all results across all four datasets and model scales with 95% bootstrap confidence intervals (1,000 resamples). The main tables (Tables 1–4) omit 4B for space. Best result per model within each dataset is **bolded**. Models are ordered by effective capability: 4B $<$ 35B-A3B (3B active) $<$ 27B (dense) $\approx$ 122B-A10B (10B active).

A.4 Additional Figures

A.5 Generated Helper Code

During the preparation phase of the Study + Code agent, the Helper Agent autonomously builds a library of reusable Python functions persisted in the sandbox. Figure 7 shows representative excerpts from the helper library generated by the 122B model for the D-WIS task. The agent discovers the ontology structure, builds lookup mappings for quantity classes, location types, and telemetry methods, and implements configuration builders that compose RDF/Turtle triples. These functions are then called by the testing agent to answer new queries without re-exploring the ontology.

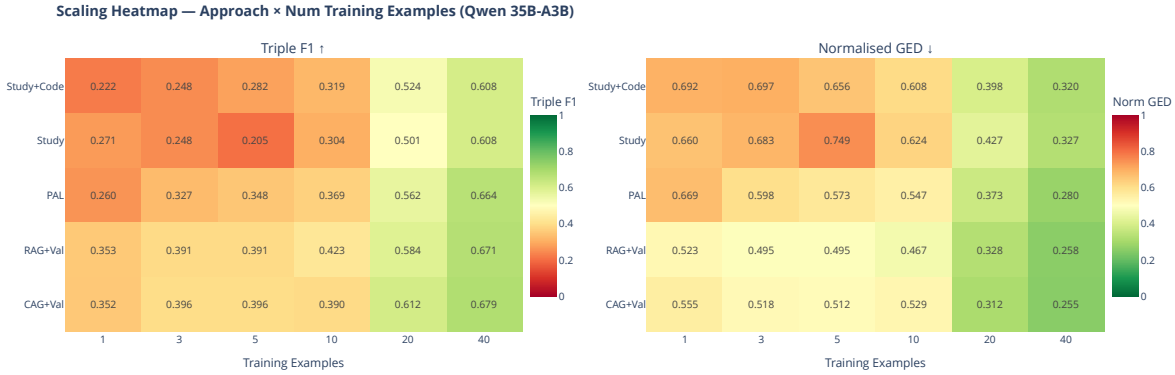

Figure 3: D-WIS Soft Triple F1 as a function of training examples (35B). Both CaG+val and Study show clear improvement with more examples.

	NGED↓					Soft Triple F1↑				
#Ex.	CaG+val	RAG+val	PaL	Study	Study+Code	CaG+val	RAG+val	PaL	Study	Study+Code
1	0.555	0.523	0.670	0.660	0.692	0.352	0.353	0.260	0.271	0.222
3	0.518	0.495	0.599	0.683	0.697	0.396	0.391	0.327	0.248	0.248
5	0.512	0.495	0.573	0.749	0.656	0.396	0.391	0.348	0.205	0.282
10	0.529	0.467	0.547	0.624	0.608	0.390	0.424	0.369	0.304	0.320
20	0.312	0.328	0.373	0.427	0.398	0.612	0.584	0.562	0.501	0.524
40	**0.255**	0.258	0.280	0.327	0.320	**0.679**	0.671	0.664	0.608	0.608

Table 5: D-WIS scaling by number of training examples (35B). NGED (lower is better) and Soft Triple F1 (higher is better) across all approaches. All approaches improve substantially from 10 to 40 examples and converge at the largest budgets.

Model	Approach	Comparison	Inference	Null	Temporal
4B	InMemory	0.405	0.656	0.279	0.450
	RAG	0.497	**0.825**	0.868	**0.532**
	PaL	**0.515**	0.719	0.397	0.468
	Study	0.423	0.588	0.926	0.394
	Study+Code	0.442	0.669	**0.956**	0.431
35B	InMemory	0.564	0.813	0.588	0.541
	RAG	0.773	**0.944**	0.338	**0.725**
	PaL	**0.779**	0.906	0.676	0.697
	Study	0.675	0.900	**1.000**	0.670
	Study+Code	0.589	0.856	0.971	0.532
27B	InMemory	0.521	0.756	0.765	0.385
	RAG	0.724	**0.950**	0.853	0.688
	PaL	0.736	0.925	0.779	**0.706**
	Study	0.828	**0.950**	**1.000**	**0.706**
	Study+Code	**0.840**	0.938	**1.000**	0.624
122B	InMemory	0.558	0.831	0.706	0.367
	RAG	0.718	**0.944**	0.309	0.688
	PaL	0.644	0.900	0.603	0.688
	Study	0.706	0.894	0.985	**0.697**
	Study+Code	**0.755**	0.919	**1.000**	0.688

Table 6: MultiHopRAG per-question-type accuracy for all approaches and model scales. Null queries test rejection ability (answer not in corpus).

		D-WIS			MultiHopRAG		BRIGHT		LB-Math
Model	Approach	Tri. F1↑	Cls. F1↑	NGED↓	Acc.↑	BERT F1↑	Recall↑	Doc F1↑	Acc.↑
	CaG+val	**.34**±.02	**.57**±.02	**.57**±.02	–	–	–	–	–
	RAG+val	.24±.02	.44±.03	.66±.02	–	–	–	–	–
	InMemory	–	–	–	.48±.05	.57±.04	–	–	–
	RAG	–	–	–	**.66**±.04	**.73**±.04	.24±.04	.19±.03	–
4B	CoT	–	–	–	–	–	–	–	**.45**±.06
	ToT	–	–	–	–	–	–	–	.15±.05
	PaL	.15±.02	.27±.03	.79±.02	.55±.04	.62±.04	.06±.02	.05±.02	.41±.06
	Study	.19±.02	.35±.03	.75±.03	.54±.04	.59±.04	.34±.04	.28±.04	.43±.06
	Study+Code	.20±.02	.35±.03	.73±.02	.58±.04	.64±.04	**.40**±.04	**.33**±.04	.44±.06
	CaG+val	.39±.02	.61±.03	.53±.03	–	–	–	–	–
	RAG+val	**.42**±.02	.59±.02	**.47**±.02	–	–	–	–	–
	InMemory	–	–	–	.64±.04	.74±.03	–	–	–
	RAG	–	–	–	.76±.04	.81±.03	.20±.03	.14±.03	–
35B	CoT	–	–	–	–	–	–	–	.37±.06
	ToT	–	–	–	–	–	–	–	.44±.06
	PaL	.37±.03	.54±.03	.55±.03	.79±.04	.83±.03	.16±.03	.14±.03	**.54**±.06
	Study	.30±.03	.49±.03	.62±.03	**.79**±.04	**.85**±.03	**.50**±.04	**.42**±.04	.53±.06
	Study+Code	.32±.02	**.55**±.03	.61±.02	.71±.04	.79±.03	.24±.04	.20±.03	.51±.06
	CaG+val	**.49**±.02	**.71**±.02	.41±.02	–	–	–	–	–
	RAG+val	.48±.02	.63±.02	**.41**±.02	–	–	–	–	–
	InMemory	–	–	–	.60±.04	.66±.04	–	–	–
	RAG	–	–	–	.81±.04	.85±.03	.58±.04	.38±.03	–
27B	CoT	–	–	–	–	–	–	–	.49±.06
	ToT	–	–	–	–	–	–	–	.44±.06
	PaL	.39±.03	.57±.03	.53±.03	.80±.04	.84±.03	.09±.03	.07±.02	.54±.06
	Study	.42±.03	.56±.03	.51±.03	**.86**±.03	**.91**±.02	**.73**±.04	**.58**±.03	.56±.06
	Study+Code	.38±.03	.59±.03	.55±.03	.85±.03	.89±.02	.71±.04	.53±.03	**.57**±.06
	CaG+val	**.50**±.02	**.72**±.02	**.41**±.02	–	–	–	–	–
	RAG+val	.47±.02	.62±.02	.41±.02	–	–	–	–	–
	InMemory	–	–	–	.62±.04	.69±.03	–	–	–
	RAG	–	–	–	.73±.04	.79±.03	.35±.04	.24±.03	–
122B	CoT	–	–	–	–	–	–	–	.56±.06
	ToT	–	–	–	–	–	–	–	.56±.06
	PaL	.33±.03	.47±.03	.60±.03	.73±.04	.77±.03	.13±.03	.11±.03	**.58**±.06
	Study	.42±.03	.59±.03	.50±.03	.80±.04	.85±.03	**.62**±.04	**.52**±.04	.57±.06
	Study+Code	.24±.03	.38±.04	.70±.03	**.83**±.03	**.88**±.02	.56±.04	.45±.04	.50±.06

Table 7: Consolidated results with 95% bootstrap confidence intervals across all datasets and model scales. Values shown as mean $\pm\,\epsilon$ where $\epsilon = \max(\text{mean}-\text{lo},\ \text{hi}-\text{mean})$. Best per model is **bolded**. "–" = not applicable. Models ordered by effective capability (see Section 4).

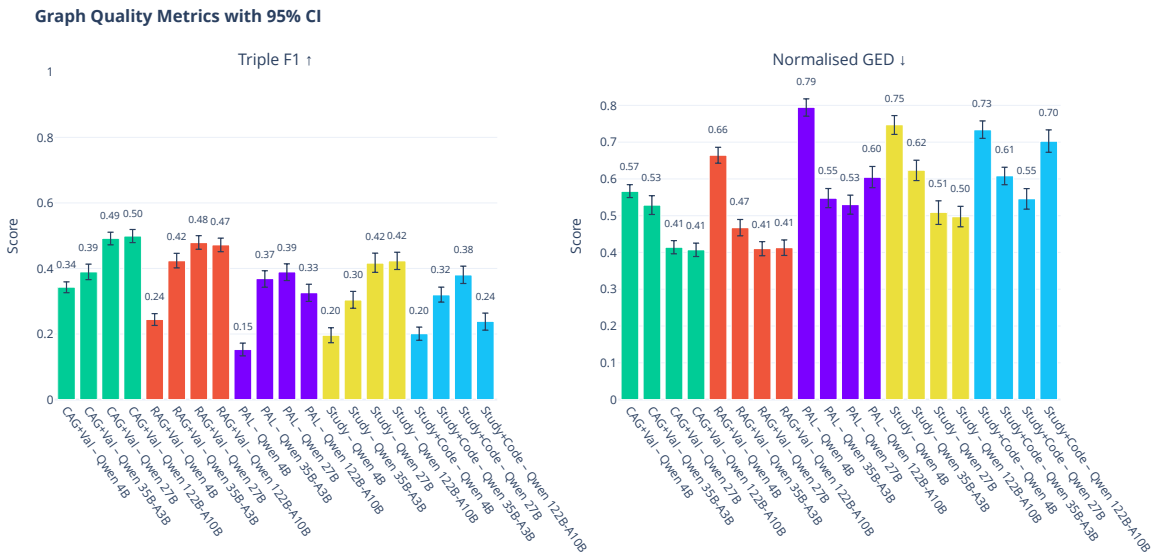

Figure 4: D-WIS quality metrics across approaches. Domain-specific baselines (CaG+val, RAG+val) consistently outperform autonomous agents on structured configuration generation.

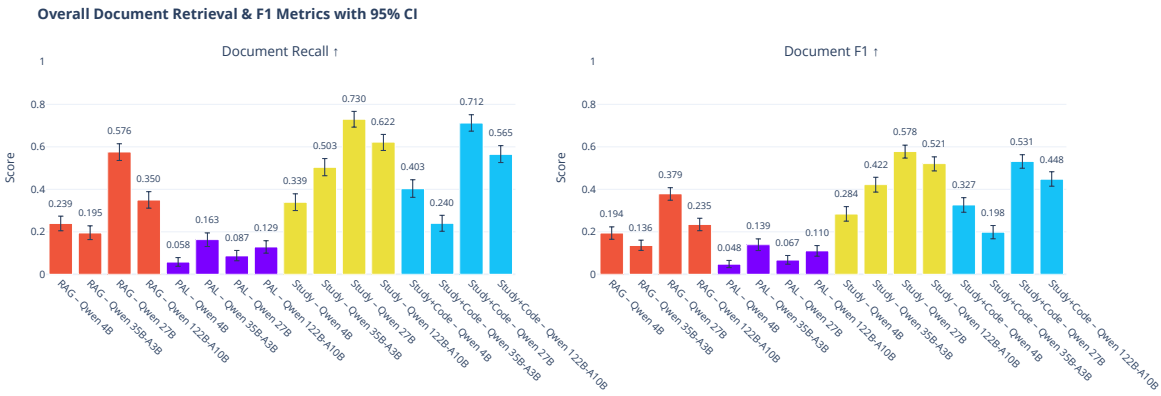

Figure 5: BRIGHT overall metrics by approach. Autonomous agents improve document retrieval over the embedding-based RAG baseline.

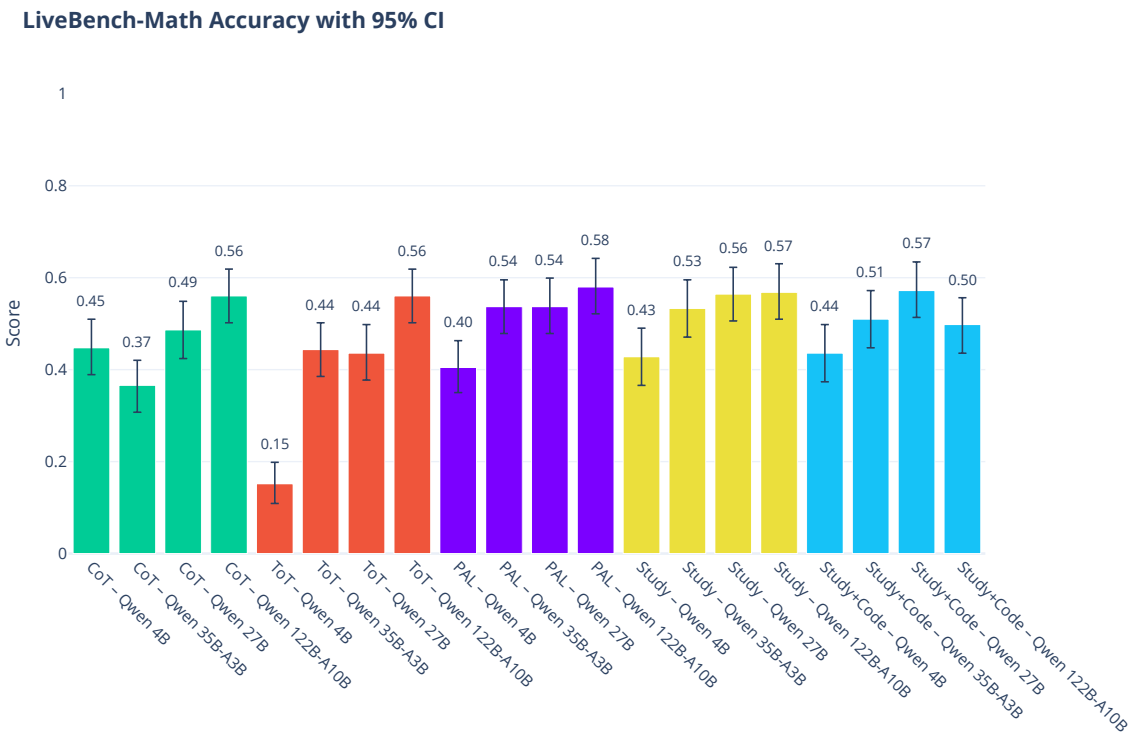

Figure 6: LiveBench-Math accuracy by approach. Code execution provides a consistent advantage over pure reasoning strategies.

```python
def find_quantity_class(quantity_name: str) -> Optional[str]:
    """Find the DWIS ontology class for a natural language quantity."""
    g = get_ontology_graph()
    direct_mappings = {
        'pressure': 'PressureDrillingQuantity',
        'acceleration': 'AccelerationDrillingQuantity',
        'torque': 'TorqueDrillingQuantity',
        'magnetic field': 'EarthMagneticFluxDensityQuantity',
        'angular velocity': 'AngularVelocityDrillingQuantity',
        'temperature': 'TemperatureDrillingQuantity',
    }
    normalized = quantity_name.lower().strip()
    if normalized in direct_mappings:
        candidate = direct_mappings[normalized]
        if DDHUB[candidate] in g.subjects(RDF.type, OWL.Class):
            return candidate
    # Fallback: search ontology classes by keyword
    for s in g.subjects(RDF.type, OWL.Class):
        name = str(s).split('/')[-1]
        if 'Quantity' in name and any(kw in name.lower()
                                      for kw in normalized.split()):
            return name
    return None

def get_location_class(location_type: str) -> Tuple[str, str]:
    """Map location description to ontology class and property."""
    mechanical = {
        'bha': ('BHAMechanicalLogicalElement', 'IsMechanicallyLocatedAt'),
        'swivel sub': ('SwivelSub', 'IsMechanicallyLocatedAt'),
        'pipe section': ('PipeSectionMechanicalLogicalElement',
                         'IsMechanicallyLocatedAt'),
    }
    hydraulic = {
        'active pit': ('ActivePitLogical', 'IsHydraulicallyLocatedAt'),
        'inlet': ('inletHydraulicBranch', 'IsAssociatedToHydraulicBranch'),
    }
    for key, val in {**mechanical, **hydraulic}.items():
        if key in location_type.lower():
            return val
    return (location_type, 'IsMechanicallyLocatedAt')

def construct_signal_name(company, location_ctx, quantity, axis=None):
    """Build signal name following DWIS naming convention."""
    parts = [company.replace(' ', '')]
    if location_ctx: parts.append(location_ctx)
    parts.append('Measured')
    parts.append(quantity.replace(' ', ''))
    if axis: parts.append(axis.upper() if len(axis)==1 else axis)
    return ''.join(parts)
```

Figure 7: Representative helper functions autonomously generated by the 122B Study + Code agent during preparation on D-WIS. The agent discovers ontology structure, builds lookup mappings, and implements reusable configuration builders.