Evaluating Context Provisioning Strategies for LLM-Based Systems: An Empirical Study with the Lattes Platform

Michel A. Vasconcelos
Universidade de Fortaleza
Fortaleza, CE, Brazil
michel.vasconcelos@gmail.com

Nabor C. Mendonça
Universidade de Fortaleza
Fortaleza, CE, Brazil
nabor@unifor.br

Abstract

Software systems that integrate Large Language Models increasingly depend on external context, but the mechanism used to provide that context can strongly affect answer quality, token consumption, latency, and system design. This paper studies context provisioning as an architectural decision through a benchmark for question answering over Lattes curricula, which are large semi-structured documents describing academic and professional trajectories. The baseline compares five effective strategy-format configurations derived from four strategies: inline HTML, inline JSON, local function calling, local MCP, and remote MCP. The experiment comprises 1,200 answer-generation runs over five curricula, twelve questions, and four answering models, followed by 3,600 LLM-as-a-judge assessments from three independent judges. Inline JSON achieved the highest majority accuracy (45.8%), followed by inline HTML (42.1%), indicating that broad prompt-level visibility can improve answer quality. Operation-mediated strategies were less accurate overall (35.0–35.8%), but substantially more token-efficient; remote MCP achieved the lowest token cost per majority-correct answer, although with higher median latency and lower direct observability. Question characteristics also strongly mediated the results: broad inferential questions were easier, whereas counting, temporal filtering, entity matching, and supervision-related questions were consistently harder. Overall, the study shows that context provisioning should be treated not only as a prompting choice, but as an architectural trade-off involving quality, cost, latency, modularity, and auditability in LLM-based systems.

Keywords

Large Language Models, Context Engineering, Context Provisioning, Model Context Protocol, Empirical Software Engineering

1 Introduction

Large Language Models (LLMs) are increasingly being integrated into software systems as components for question answering, decision support, information extraction, summarization, and interactive assistance [21]. In these systems, output quality depends not only on the model itself, but also on how the surrounding software provides the external context required to solve a task. This context may include documents, database records, source code, logs, user profiles, domain-specific resources, or tool outputs. As a result, context provisioning has become a central design concern in LLM-based systems [10].

When task-relevant information lies outside the model, the application must decide how that information is exposed at inference time. The most direct strategy is to embed the context in the prompt. This inline approach is simple and often effective because the model receives broad visibility over the available evidence. However, large prompts increase token consumption and latency, are constrained by context-window limits, and do not necessarily guarantee robust use of the provided information [9, 18]. Alternative strategies decouple context access from prompt construction. For example, RAG retrieves selected information before generation [7], function calling exposes application-defined operations that the model may invoke [5, 12], and MCP provides a protocol-based mechanism for connecting LLM applications to external tools and data sources [1, 11]. These alternatives may improve modularity, reuse, and separation of concerns, but they also introduce new design questions about operation granularity, execution control, latency, observability, and operational complexity [6, 16].

Despite growing interest in operation-mediated context provisioning, there is still limited empirical evidence on how these architectural alternatives affect LLM-based systems in realistic scenarios. It is not yet clear when broad prompt-level visibility should be preferred over operation-mediated access, how much overhead MCP introduces relative to local function calling, or how these strategies behave across questions with different evidence requirements. These are software engineering questions as much as artificial intelligence questions, because they concern system architecture, control flow, monitoring, deployment, and evolution.

This paper addresses these questions through a benchmark for question answering over curricula from the Brazilian Lattes Platform [2]. Lattes curricula are large semi-structured documents describing academic and professional trajectories, including education, publications, projects, supervision activities, professional experience, and other scholarly outputs. They provide a suitable domain for this study because information is distributed across multiple sections, questions require different access patterns, and the same curriculum can be exposed either as a serialized document or through structured operations over parsed data [3].

The baseline compares five effective strategy-format configurations derived from four strategies: inline HTML, inline JSON, local function calling, local MCP, and remote MCP. Together, these configurations cover a design space that ranges from direct prompt-based integration to local operation-mediated execution and provider-managed remote protocol access. Across 1,200 answer-generation runs and 3,600 LLM-as-a-judge assessments, we analyze answer quality, token consumption, latency, call behavior, and the influence of question characteristics.

The results show no universally best context provisioning strategy. Inline JSON achieved the highest majority accuracy (45.8%), followed by inline HTML (42.1%), indicating that broad prompt-level visibility can improve answer quality. However, this advantage came at a substantial token cost. Operation-mediated strategies

were less accurate overall (35.0–35.8%), but were far more token-efficient; remote MCP achieved the lowest token cost per majority-correct answer, although with higher median latency and lower direct observability. Question characteristics also strongly mediated the results: broad inferential questions were easier, whereas counting, temporal filtering, entity matching, and supervision-related questions were consistently harder. Taken together, these findings show that context provisioning should be treated as an architectural design decision rather than only as a prompting choice.

In summary, this paper makes the following contributions:

- We frame context provisioning as an architectural design decision in LLM-based systems, distinguishing direct prompt-based access, benchmark-controlled local function calling, local MCP, and provider-managed remote MCP.
- We present a benchmark for LLM-based question answering over Lattes curricula, including curriculum artifacts, question categories, five strategy-format configurations, execution specifications, traces, and efficiency metrics.
- We empirically compare these configurations over 1,200 answer-generation runs and 3,600 judge assessments in terms of answer quality, token consumption, latency, call behavior, and question characteristics.
- We derive practical implications for software engineers designing LLM-based systems that must balance answer quality, token budget, latency, modularity, observability, and operational complexity.

The remainder of this paper is organized as follows. Section 2 provides the required conceptual background. Section 3 presents the study design, including the research questions, evaluated strategies, and experimental infrastructure. Section 4 reports the results and discusses their implications. Section 5 examines threats to validity. Section 6 reviews related work, and Section 7 concludes the paper.

2 Background

This section defines the concepts used throughout the paper. We first position the study within context engineering, then clarify the notion of context provisioning, and finally summarize the role of operation-mediated access, MCP, and Lattes curricula in our experimental benchmark.

2.1 Context Engineering

LLMs generate outputs conditioned on the context available at inference time. In modern LLM-based systems, this context may include not only user instructions, but also retrieved documents, structured records, tool descriptions, prior interaction history, execution results, and domain-specific resources. The term *context engineering* refers to the systematic design, construction, optimization, and management of these information payloads [10].

In this paper, we focus on *context provisioning*: the architectural mechanism through which task-relevant external information is exposed to the model at inference time. A context provisioning strategy determines whether information is embedded directly in the prompt, retrieved by an intermediate component, accessed through callable operations, or served through a protocol-based context service. Thus, context provisioning is not only a prompting issue, but also a software engineering decision: it affects where context is stored, how it is accessed, which component controls access, and how observable and reusable the resulting integration is.

2.2 Operation-Mediated and Protocol-Based Context Access

The most direct context provisioning strategy is *inline context*, in which the application serializes relevant information and embeds it directly in the prompt. Other strategies introduce intermediate mechanisms between the model and the underlying information source. RAG systems retrieve selected information and add it to the model input [7]; memory systems preserve information across interactions; and operation-mediated approaches expose external operations that the model may invoke during execution [15].

Operation-mediated context access is commonly implemented through function calling or tool calling in current provider APIs: the application declares available functions or tools, the model selects an operation and proposes arguments, the application executes it, and the result is returned to the model [5, 12]. In this paper, we use *operation-mediated* as the generic term for this family of approaches, reserving *function* for local function calling and *tool* for MCP-based access. From a software engineering perspective, operation-mediated access shifts part of the context-selection burden from prompt construction to the interaction between the model and externally implemented operations. This can reduce upfront context transmission, but it also introduces failure modes such as missing function or tool calls, inappropriate operation selection, invalid arguments, or incomplete evidence gathering.

The Model Context Protocol (MCP) extends this idea by turning externally exposed operations into a protocol-based integration mechanism [1]. In MCP terminology, these operations are surfaced as tools. Instead of implementing callable operations directly inside the LLM application, a system can place them behind an MCP server, creating an explicit boundary between the LLM-facing client and the component responsible for accessing and serving external context [6]. This boundary may improve separation of concerns, interoperability, and reuse, but it also introduces engineering concerns such as serialization, error handling, deployment, observability, and, in remote settings, network latency and availability [16]. This motivates evaluating MCP not only by answer quality, but also by efficiency, observability, and operational implications.

2.3 Lattes Curricula as Semi-Structured Context

We use curricula from the Brazilian Lattes Platform as the source of external context for our benchmark [2]. Lattes curricula are suitable for evaluating context provisioning strategies because they combine recurring structure with heterogeneous content [3]. They are organized into sections such as profile, education, projects, publications, professional experience, and supervisions, which makes them amenable to parsing and section-oriented access. At the same time, their content varies substantially across researchers, with sections that may be absent, repeated, long, sparse, or unevenly populated.

This structure allows the same curriculum to be exposed to the model in different ways. Inline strategies can embed a serialized

artifact, such as HTML or JSON, directly in the prompt. Operation-mediated strategies can instead operate over a parsed JSON representation and expose domain-specific operations over sections such as profile data, expertise, education, projects, publications, supervisions, academic activities, technical output, and artistic output. Consequently, the Lattes setting lets us compare broad prompt-based visibility with structured operation-mediated access over the same underlying semi-structured documents.

3 Study Design

This section describes the empirical study used to compare context provisioning strategies for LLM-based question answering over Lattes curricula. The study is implemented as a benchmark in which the question set and curriculum instances are kept fixed while the mechanism used to expose curriculum information to the model is varied.

3.1 Research Questions

The study is guided by the following research questions:

RQ1: Effect on Answer Quality How do different context provisioning strategies affect the quality of LLM-generated answers over Lattes curricula?

RQ2: Execution Efficiency How do the strategies differ in terms of execution efficiency, including token consumption, latency, and call behavior?

RQ3: Influence of Question Characteristics How do question characteristics affect answer quality, judge disagreement, token usage, latency, and operation-call behavior across context provisioning strategies?

RQ1 focuses on answer quality, comparing whether direct prompt-based context leads to different outcomes from context access through local functions, local MCP, or remote MCP. **RQ2** focuses on execution efficiency and behavior, considering token usage, latency, model calls, function calls, tool calls, MCP calls, and loop steps. RQ3 investigates whether the response variables analyzed in RQ1 and RQ2 vary with the type of information access required by each question, such as factual lookup, temporal reasoning, quantitative aggregation, cross-section matching, or broader synthesis.

3.2 Benchmark Dataset and Questions

The dataset is organized around Lattes curriculum instances. For the experiment reported in this paper, we selected five instances from senior computer science researchers affiliated with different Brazilian universities. These instances were selected to provide variation in curriculum size, as well as in the structure and distribution of information across curriculum sections. Each instance is represented by a directory in the benchmark repository containing multiple context artifacts, including the original HTML page, a cleaned HTML version, a parsed JSON representation, and block-level context files. The alternative representations aim to mitigate the challenges of long contexts in LLM-based systems by balancing token cost with the quality of the information provided to the model. The cleaned HTML is a reduced version of the original page, with scripts and other nonessential structures removed so that only content relevant to the experiment is preserved. The parsed JSON provides a structured representation of the original HTML and is used by local functions and MCP servers. Finally, the block-level context files serve as the reference basis for answer evaluation.

Table 1: Selected Lattes curriculum instances.

Instance	HTML Size	Cleaned HTML	JSON Size	Pub.	Proj.	Tech.
1	908 KB	383 KB	356 KB	482	52	7
2	358 KB	114 KB	124 KB	227	5	0
3	672 KB	334 KB	294 KB	323	20	50
4	387 KB	180 KB	175 KB	197	19	94
5	684 KB	336 KB	304 KB	290	28	85

Table 2: Selected benchmark questions.

Question ID	Description
`q_phd`	Where and how long ago the researcher completed the PhD.
`q_field`	The main field of knowledge in which the researcher works.
`q_coauth`	The researcher's most frequent co-authors.
`q_advpub`	Whether the researcher usually publishes with advisees.
`q_pubyear`	The most relevant bibliographic production in a given year.
`q_projfit_2`	How the candidate's background aligns with a given project.
`q_indexed`	Whether publications are concentrated in indexed or highly ranked venues.
`q_sup`	The number of supervised students by level and completion status.
`q_techprod`	The number of patents, software registrations, prototypes, cultivars, or other technological products.
`q_admin`	Administrative positions held by the researcher, such as course coordination or graduate-program leadership.
`q_tcc5y`	The number of undergraduate thesis or undergraduate research supervisions in the last five years.
`q_en`	Whether the researcher speaks English.

Table 1 summarizes the selected instances. The five curricula differ substantially in size and content distribution. For example, the largest raw HTML artifact is approximately 908 KB, while the smallest is approximately 358 KB. The number of parsed publications ranges from 197 to 482, projects from 5 to 52, and technical outputs from none in the parsed representation to 94 items. This variation is important for the benchmark because different strategies may be affected differently by document size, section sparsity, and the concentration of information in specific curriculum sections. The selected curricula should be interpreted as controlled benchmark instances rather than as statistical representatives of Lattes curricula. The goal of this baseline is to compare how the same semi-structured source behaves under different context provisioning mechanisms while keeping the curricula, questions, models, operation surface, and execution pipeline fixed.

The benchmark repository also contains a catalog of 28 questions over Lattes curricula, derived from a previous work on LLM-based question answering over Lattes semi-structured data [3]. For the current experiment, we selected the 12 questions summarized in Table 2. The selected questions cover different information access patterns, including direct factual lookup, temporal filtering, quantitative aggregation, ranking, cross-section matching, and broader inferential synthesis over semi-structured curriculum data.

The questions include tag metadata to support stratified analysis and facilitate the interpretation of model behavior. These tags help identify whether poor answers are associated with specific question characteristics, such as temporal filtering, evidence integration across sections, ranking, or incomplete evidence. For

Table 3: Operations exposed by the Lattes service.

Operation	Description
`get_profile`	Returns the researcher's identification data.
`get_expertise`	Returns expertise, research lines, and awards.
`get_education`	Returns academic background.
`get_projects`	Returns research and professional projects.
`get_supervisions`	Returns supervision activities grouped by level and status.
`get_experience`	Returns professional experience.
`get_academic_activities`	Returns academic service, boards, events, and related activities.
`get_publications`	Returns bibliographic production.
`get_technical_output`	Returns technical output.
`get_artistic_output`	Returns artistic and cultural output.

example, `factual` denotes questions answerable from explicit curriculum statements, such as `q_en`; `temporal` marks questions constrained by dates or time windows, such as `q_phd` and `q_tcc5y`; `inferential` indicates that the model must synthesize evidence rather than extract a single field, as in `q_field`; `ranking` identifies questions that require selecting the most frequent or most relevant item, as in `q_coauth` and `q_pubyear`; `quantitative` refers to counting or aggregation tasks, such as `q_sup`; `matching` indicates that entities must be aligned across records, as in `q_coauth` and `q_advpub`; `cross-block` denotes questions whose evidence is distributed across multiple curriculum sections, as in `q_projfit_2`; and `weak-data` marks cases in which the available evidence is incomplete, indirect, or noisy, as in `q_indexed` and `q_techprod`. These tags do not affect benchmark execution. Instead, they provide an analytical layer for comparing results across question characteristics, in addition to comparing context provisioning strategies.

3.3 Context Access Modes and Strategies

The benchmark evaluates context provisioning strategies that differ along three architectural dimensions: how the curriculum context is represented and exposed to the model, who controls the interaction loop, and whether context access crosses a remote boundary. Figure 1 summarizes these alternatives.

The first access mode is *inline*. In this mode, a serialized curriculum artifact is inserted directly into the prompt, requiring the model to inspect the provided context and identify the evidence needed to answer the question. As shown in Figure 1(a), there is no operation loop in this configuration: the benchmark builds the prompt, sends it to the model provider through the LLM SDK, and receives a final answer. We instantiate this mode with two configurations: `inline_html`, which uses the cleaned HTML artifact, and `inline_json`, which uses the parsed JSON artifact. Thus, inline access gives the model broad visibility over the curriculum, but makes input-token cost dependent on the size of the serialized artifact.

The second access mode is *operation-mediated*. In this mode, the curriculum is not provided directly in the prompt. Instead, the model receives access to read-only operations over semantically defined curriculum sections. The operation surface, summarized in Table 3, is shared across all operation-mediated configurations. The operations are organized around the main Lattes sections, including author profile, expertise, education, projects, supervisions, professional experience, academic activities, publications, technical output, and artistic output. Each operation requires a `lattes_id`; operations associated with temporal sections may also receive optional `start_year` and `end_year` arguments. Keeping the operation surface constant allows us to attribute differences primarily to loop control, protocol boundaries, and context placement, rather than to differences in the information exposed to the model. Concrete examples of operation calls and abridged outputs are provided in the replication package.

Table 4: Evaluated context provisioning configurations.

Configuration	Context handling	Loop control	Boundary
`inline_html`	Prompt	No operation loop	Local
`inline_json`	Prompt	No operation loop	Local
`local_function`	Local functions	Benchmark	Local
`local_mcp`	In-memory MCP	Benchmark	Local
`mcp`	Remote MCP	Provider	Remote

We evaluate three operation-mediated configurations. In `local_function`, shown in Figure 1(b), the benchmark controls the loop: it receives function-call requests from the model, executes local Python functions over the parsed JSON artifact, returns their results, and repeats this process until a final answer is produced. In `local_mcp`, shown in Figure 1(c), the benchmark still controls the loop, but the same operations are exposed through a local MCP server and accessed by an MCP client. This isolates the effect of introducing MCP as a local protocol boundary. In `mcp`, shown in Figure 1(d), the MCP server is remote and the operation loop is delegated to provider-native MCP support. In this case, the benchmark configures the interaction and receives the final answer, but context access is mediated by the provider and a remote MCP service.

Table 4 summarizes the five evaluated strategy-format configurations according to the three architectural dimensions used in the study. The comparison between `inline_html` and `inline_json` isolates the effect of representation format under direct prompt-based access. Comparing inline with operation-mediated configurations captures the effect of moving context from the prompt to explicit operations. Comparing `local_function` with `local_mcp` isolates the local MCP boundary, while comparing `local_mcp` with `mcp` examines the transition from benchmark-controlled local MCP to provider-managed remote MCP. The difference between local MCP and remote MCP is intentional and reflects the control-flow variable evaluated in the study. In `local_mcp`, the benchmark drives the operation loop: it receives model-requested operations, calls the local MCP client/server boundary, records the intermediate steps, and returns tool results to the model. In (remote) `mcp`, the operation loop is delegated to provider-native MCP support and the MCP server is accessed remotely. As a consequence, using a remote MCP server better represents a provider-managed protocol integration, but gives the benchmark less direct visibility into intermediate tool orchestration than the locally controlled strategies.

3.4 Benchmark Pipeline and Baseline Experiment

All configurations are executed through the same benchmark pipeline, in four phases. First, the planning phase expands the experiment configuration into query runs, each corresponding to one combination of curriculum instance, question, model, strategy-format

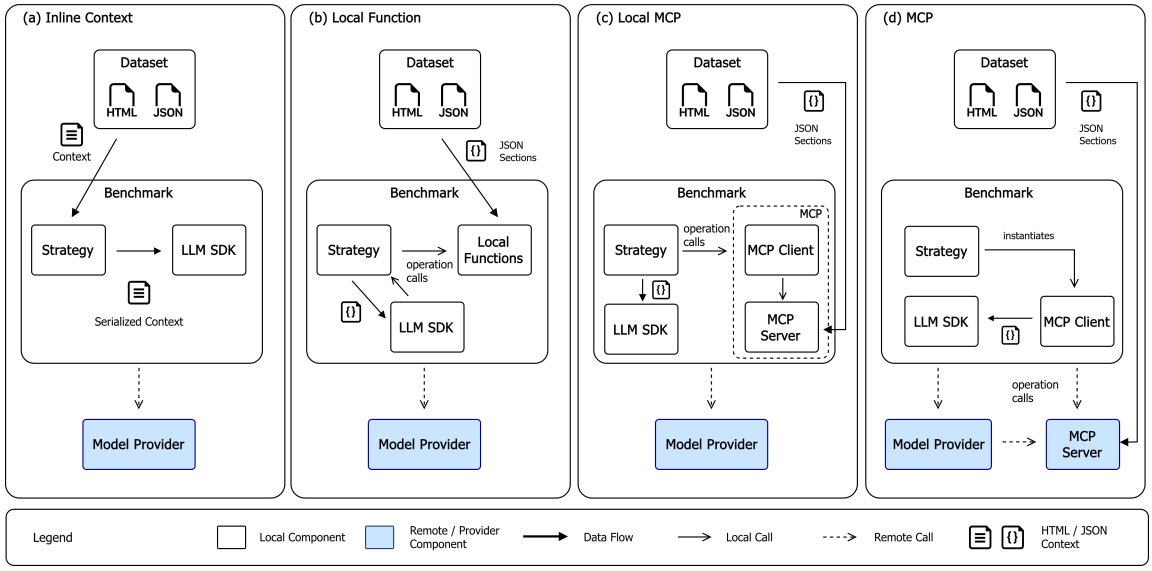


Figure 1: Architectural view of the evaluated context provisioning strategies.

configuration, and repetition. Second, the query-execution phase renders the question, configures the target model, applies the selected context access strategy, and records the generated answer, token usage, duration, calls, errors, and traces. Inline runs embed the selected artifact in the prompt, whereas operation-mediated runs provide the model with the curriculum identifier and available operations.

Third, the evaluation phase applies an LLM-as-a-Judge protocol [22]. Each judge receives the question, candidate answer, and question-specific evidence blocks, independent of the context originally used to generate the answer. This provides a common reference for comparing answers produced by different strategies while avoiding the cost of sending full curricula to every judge. Judges assess two criteria: *correctness*, whether the answer is factually supported and avoids unsupported claims, and *completeness*, whether it addresses all answerable parts of the question. Each criterion is rated as `meets`, `partial`, or `misses`. Finally, the export phase combines planned runs, answers, judge assessments, aggregate outcomes, and traces into analysis-ready files.

The baseline experiment instantiates this pipeline with five curriculum instances, twelve questions, four answer-generation models, five strategy-format configurations, and one repetition per condition, yielding 1,200 answer-generation runs. The answer-generation models are OpenAI's `gpt-5.4-nano` and `gpt-5.4-mini`, and Google's `gemini-2.5-flash-lite` and `gemini-2.5-flash`. These models were selected to keep the experiment affordable while covering two providers and two model sizes per provider. All 1,200 answer-generation runs completed successfully.

Each answer was evaluated by three stronger judge models: OpenAI's `gpt-5.4`, Google's `gemini-3.1-flashlite-preview`, and Anthropic's `claude-sonnet-4-6`. Thus, the evaluation phase produced 3,600 judge assessments. We aggregate assessments into two quality outcomes. An answer is *majority-correct* when at least two judges rate both correctness and completeness as `meets`; it is *unanimously correct* when all three judges do so. Evaluation-phase tokens are reported separately and are not included in answer-generation cost metrics.

In addition to quality outcomes, the benchmark records query-phase execution metrics, including duration, model and operation calls, function calls, MCP tool calls, loop steps, retry counts, token usage, cached input tokens when available, context size, prompt size, and rate-limit or retry waiting time. Remote MCP calls are extracted from provider-native MCP evidence when available, since this strategy is not fully controlled by the benchmark engine. For Gemini remote MCP runs, we also compute an adjusted input-token metric that adds provider-reported native tool-use prompt tokens to direct prompt tokens.

Full prompt templates for answer generation and judge assessment are available in the replication package.

4 Results

This section reports the results of the experiment designed to answer the three research questions. The experiment produced 1,200 successful answers and 3,600 judge assessments. Answer generation consumed 43.95 million query-phase tokens, while the three-judge evaluation consumed 28.50 million tokens. We keep these quantities separate because evaluation cost is a property of the benchmark procedure, whereas query cost is a property of the context provisioning strategy. Unless otherwise stated, token metrics refer only to the answer-generation phase and exclude evaluation tokens.

4.1 RQ1: Effect on Answer Quality

Table 5 shows that inline context produced the highest answer quality in the aggregate. `inline/json` reached 45.8% majority accuracy, followed by `inline/html` with 42.1%. The three operation-mediated configurations were lower and close to each other: local

Table 5: Effectiveness, cost of agreement, latency, and observed calls by context provisioning configuration.

Configuration	Strategy	Runs	Maj.	Unan.	Tok./run	Tok./Maj.	Tok./Unan.	Sec.	Obs. calls
I-HTML	Inline HTML	240	42.1	26.3	79.3k	188.4k	302.0k	4.09	0.00
I-JSON	Inline JSON	240	45.8	24.6	70.6k	153.9k	287.0k	4.01	0.00
Func.	Local function calling	240	35.0	18.8	11.7k	33.5k	62.6k	4.61	1.23
L-MCP	Local MCP	240	35.4	17.5	12.6k	35.4k	71.7k	4.45	1.29
R-MCP	Remote MCP	240	35.8	14.6	9.0k	25.1k	61.7k	5.99	1.15

function calling reached 35.0%, local MCP reached 35.4%, and remote MCP reached 35.8%. The same ordering is mostly preserved under the stricter unanimous metric: inline HTML and inline JSON reached 26.3% and 24.6%, while operation-mediated configurations ranged from 14.6% to 18.8%. Because the baseline uses one repetition per condition, marginal differences should not be interpreted as definitive rankings. This cost-aware design already required 1,200 answer-generation runs and 3,600 judge assessments. As a robustness check, a block bootstrap over curriculum–question blocks estimated the inline versus operation-mediated majority-accuracy difference at +8.5 percentage points, with a 95% interval from +1.9 to +15.5; a paired analysis over curriculum–question–model cells yielded the same qualitative result. Thus, the main contrast is supported, while the three operation-mediated configurations should be regarded as close alternatives.

This result indicates that broad prompt-level visibility remains advantageous for answer quality in this baseline. Inline strategies expose the entire serialized curriculum to the model, reducing the need for the model to select operations, choose parameters, and sequence calls before answering. However, the advantage is moderate rather than absolute. Operation-mediated strategies still produced non-trivial majority agreement: approximately one third of their answers were majority-correct even though they used far less context per query. This suggests that structured access through operations can provide useful answers, but also that operation selection and downstream aggregation become part of the task difficulty.

The difference between inline HTML and inline JSON is also informative. Inline JSON achieved the highest majority accuracy and consumed fewer tokens than inline HTML. This suggests that, when full-context prompting is used, a parsed representation can be more effective than cleaned HTML because it removes part of the markup burden and exposes the curriculum in a more regular form. Nevertheless, both inline variants remain expensive compared with operation-mediated access.

Table 6 shows that model family affects the magnitude of the quality differences. OpenAI models produced higher accuracies overall, and their operation-mediated configurations remained competitive with inline configurations. For example, GPT-Nano with local function calling reached 53.3% majority accuracy and 33.3% unanimity, while consuming 16.1k tokens per run. GPT-Mini with local MCP matched inline HTML in majority accuracy (53.3%) while consuming only 20.5k tokens per run. By contrast, Gemini operation-mediated configurations were more strongly penalized, especially with Gemini-Lite. This suggests that the effectiveness of operation-mediated access depends not only on the operation surface, but also on the model's ability to decide when and how to use the exposed operations.

Table 6: Model-level effectiveness and execution efficiency by context provisioning configuration.

Family	Model	Configuration	Maj.	Unan.	Tok.	Sec.	Calls
OpenAI	GPT-Nano	I-HTML	48.3	30.0	78.7k	8.50	0.00
		I-JSON	60.0	30.0	67.9k	13.50	0.00
		Func.	53.3	33.3	16.1k	7.00	1.42
		L-MCP	43.3	28.3	18.6k	5.00	1.40
		R-MCP	48.3	28.3	17.4k	8.00	1.35
	GPT-Mini	I-HTML	53.3	28.3	78.8k	4.00	0.00
		I-JSON	50.0	23.3	68.1k	4.00	0.00
		Func.	45.0	25.0	20.2k	6.00	1.83
		L-MCP	53.3	30.0	20.5k	6.00	1.88
		R-MCP	51.7	20.0	17.2k	9.00	1.62
Gemini	Gemini-Lite	I-HTML	33.3	23.3	79.2k	3.00	0.00
		I-JSON	33.3	18.3	72.5k	3.00	0.00
		Func.	15.0	5.0	4.4k	3.00	0.87
		L-MCP	21.7	5.0	4.7k	3.00	1.05
		R-MCP	15.0	3.3	0.3k	3.00	0.78
	Gemini-Flash	I-HTML	33.3	23.3	80.5k	5.50	0.00
		I-JSON	40.0	26.7	73.7k	5.00	0.00
		Func.	26.7	11.7	6.1k	4.00	0.78
		L-MCP	23.3	6.7	6.4k	4.00	0.82
		R-MCP	28.3	6.7	1.1k	5.00	0.85

4.2 RQ2: Execution Efficiency

Table 5 also summarizes the efficiency dimensions. The results show a clear trade-off: inline configurations achieve higher answer quality, but this quality comes at a substantially higher token cost. Inline HTML and inline JSON consume, on average, 79.3k and 70.6k query tokens per run, respectively. In contrast, operation-mediated configurations reduce the average query cost to 11.7k tokens for local function calling, 12.6k for local MCP, and 9.0k for remote MCP. Thus, operation-mediated context access uses roughly one sixth to one eighth of the tokens required by inline context provisioning.

The difference becomes more pronounced when token cost is normalized by agreement. Inline JSON, the most accurate configuration, requires 153.9k query tokens per majority-correct answer and 287.0k tokens per unanimously correct answer. Inline HTML requires 188.4k and 302.0k tokens, respectively. By comparison, remote MCP requires only 25.1k tokens per majority-correct answer and 61.7k tokens per unanimously correct answer. Local function calling requires 33.5k and 62.6k tokens, while local MCP requires 35.4k and 71.7k. Therefore, the best inline configuration requires approximately 6.1 times more tokens per majority-correct answer than remote MCP and approximately 4.7 times more tokens per unanimously correct answer.

These results indicate that inline context provisioning is preferable when maximizing answer quality is the primary goal, but

operation-mediated strategies provide a more cost-efficient path to acceptable agreement. In particular, remote MCP has the lowest token cost per majority-correct answer, while local function calling has the lowest token cost per unanimously correct answer among the locally controlled operation-mediated strategies. This suggests that operation-mediated context access is attractive in scenarios where token budget is constrained and moderate reductions in answer quality are acceptable.

Call behavior further distinguishes the strategies. Inline configurations do not perform operation calls because the serialized curriculum is embedded directly in the prompt. In contrast, local function calling and local MCP average 1.23 and 1.29 observed calls per run, respectively. Remote MCP averages 1.15 observed calls per run. However, this metric must be interpreted according to the execution mechanism. For local function calling and local MCP, calls are counted from benchmark-controlled traces. For remote MCP, calls are counted from provider-native MCP evidence recorded in query traces. Thus, remote MCP exposes less direct observability to the benchmark, even though native MCP activity can still be detected.

Latency follows a different pattern from token consumption. Although remote MCP is the most token-efficient configuration, it has the highest median query duration, 5.99s. Inline HTML and inline JSON have lower median latencies, 4.09s and 4.01s, respectively, but their distributions exhibit long upper tails. Local function calling and local MCP remain close to the inline configurations in the median case, with 4.61s and 4.45s. Figure 2a complements these medians by combining a violin plot, which shows the distribution shape, with an overlaid boxplot for median and interquartile range. The figure indicates that reducing prompt size does not necessarily reduce wall-clock latency, especially when context access depends on operation execution and provider-mediated tool orchestration.

Overall, **RQ2** shows that token efficiency and latency do not move together. Operation-mediated strategies drastically reduce token consumption and the cost of obtaining agreement, but remote MCP introduces higher median latency. Local function calling and local MCP provide a middle ground: they preserve most of the token-efficiency advantage of operation-mediated access while keeping median latency closer to the inline configurations. Therefore, the choice between direct and operation-mediated context access depends on whether the system prioritizes answer quality, token budget, latency, or trace observability.

4.3 RQ3: Influence of Question Characteristics

Question characteristics strongly affected the relative behavior of direct and operation-mediated context access. Table 7 reports per-question accuracy, full judge disagreement, and a short interpretation of the dominant difficulty. Full disagreement means that, for at least one evaluation criterion, the three judges assigned three different ratings (`meets`, `partial`, and `misses`) to the same answer. This metric identifies questions whose evidence or expected answer form is especially ambiguous for the judges.

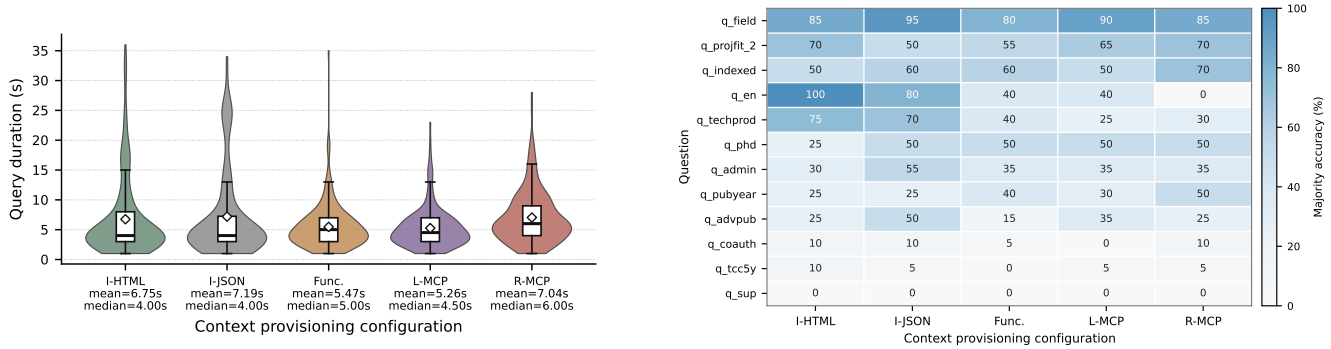
The easiest question was `q_field`, with 87% majority accuracy and 58% unanimity. It asks for a broad characterization of the researcher's field and can be answered from profile and expertise information. `q_projfit_2` also performed relatively well (62% majority accuracy), despite being cross-block, because its expected answer can often be supported by high-level project and profile evidence. These results show that inferential questions are not necessarily harder when the expected answer is qualitative and supported by salient curriculum sections.

The hardest questions were `q_sup`, `q_tcc5y`, and `q_coauth`. Their common feature is that they require aggregation rather than simple retrieval. `q_sup` obtained no majority-correct answers. This is consistent with the structure of the parsed Lattes data: supervision records are grouped by level and status, but individual entries are stored as long natural-language strings containing title, year, institution, program, funding, and role information. This contrasts with the publication section, which is more itemized, with fields such as year, category, authors, and title. As a consequence, answering supervision-count questions requires parsing and counting information embedded inside textual records, not only selecting the correct block. `q_tcc5y` adds a temporal filter to this counting problem, which explains its similarly low accuracy. `q_coauth` is difficult for a different reason: it requires identifying coauthors, aggregating across publications, and ranking or selecting names, which amplifies errors in entity matching and counting.

Figure 2b shows the strategy–question interaction in the form of a majority accuracy heatmap. Some questions are answered well across most strategies, such as `q_field`; others fail across nearly all strategies, such as `q_sup`. This pattern is important because it shows that strategy choice alone does not explain the results. In several cases, the limiting factor is the structure of the evidence and the type of reasoning required by the question. For example, `q_sup` remains at 0% majority accuracy even when operation-mediated strategies can access the supervision operation, suggesting that the output of the operation is not structured enough for reliable counting.

The results for `q_en` expose another operation-surface limitation. The tool interface was designed around the main sections of the Lattes document, following a section-oriented organization rather than a question-specific one. Because language proficiency appears as a subfield rather than as a top-level section, the interface does not provide a dedicated `get_languages` or `get_idioms` operation. Inline strategies retain direct visibility of this information, whereas operation-mediated strategies must locate it through broader operations and may therefore fail to retrieve the most relevant evidence. This case shows that operation-surface design is itself an experimental and architectural variable: operation boundaries, granularity, and alignment with expected questions can influence answer quality independently of the underlying provisioning mechanism.

Tag-level results further support this interpretation. Matching questions were the most penalized group, with 18.5% majority accuracy and only 1.0% unanimity. Quantitative questions were also difficult, with 24.7% majority accuracy and 9.2% unanimity. Temporal questions reached 28.0% majority accuracy, while factual questions reached 28.0%. In contrast, weak-data, inferential, and cross-block tags had higher majority accuracies, but these categories overlap with easier high-level questions and should not be interpreted independently. The most important pattern is that questions combining quantitative aggregation with matching or temporal filtering were



(a) (b)

Figure 2: Comparison of latency (a) and question-level accuracy (b) across context provisioning configurations.

Table 7: Question-level results and observed difficulty factors.

Question	Tags	Maj.	Unan.	Full dis.	Calls	Dominant difficulty
`q_field`	inferential, ranking	87.0	58.0	1.0	0.79	broad profile synthesis
`q_projfit_2`	inferential, cross-block	62.0	32.0	2.0	1.87	cross-section project matching
`q_indexed`	weak-data, quantitative	58.0	23.0	11.0	0.42	weak evidence and count interpretation
`q_en`	factual	52.0	52.0	0.0	0.34	simple lookup; mostly stable
`q_techprod`	weak-data, quantitative	48.0	29.0	3.0	0.60	weakly structured technical output
`q_phd`	factual, temporal	45.0	23.0	0.0	0.58	temporal education lookup
`q_admin`	factual	38.0	19.0	0.0	1.05	administrative-role evidence
`q_pubyear`	weak-data, temporal, ranking	34.0	5.0	4.0	0.38	temporal publication ranking
`q_advpub`	inferential, cross-block, quantitative, matching	30.0	2.0	5.0	0.72	supervision–publication matching
`q_coauth`	inferential, quantitative, ranking, matching	7.0	0.0	29.0	0.36	coauthor aggregation and ranking
`q_tcc5y`	factual, quantitative, temporal	5.0	1.0	4.0	1.07	temporal counting over supervision records
`q_sup`	factual, quantitative	0.0	0.0	8.0	0.61	counting over unstructured supervision records

consistently harder than questions requiring broad qualitative synthesis.

The highest full-disagreement rates occurred for `q_coauth` (29%), `q_indexed` (11%), and `q_sup` (8%). Trace inspection associates these cases with different failure modes: name normalization and ranking in `q_coauth`, incomplete venue evidence in `q_indexed`, and counting over long semi-structured records in `q_sup`. These cases suggest boundary ambiguity and evidence-structure limitations rather than arbitrary judge inconsistency.

4.4 Practical Implications for System Design

The results support four practical guidelines for designing LLM-based systems. First, inline JSON provides a useful quality baseline when the complete context fits the available token budget. It preserved broad evidence visibility, achieved the highest aggregate accuracy, and required fewer tokens than inline HTML. Teams should therefore evaluate more complex provisioning mechanisms against a structured inline baseline rather than assuming that operation-mediated access will improve answer quality.

Second, operation-mediated access is appropriate when token efficiency, controlled context exposure, modularity, or reuse outweigh the observed quality difference. This choice, however, transfers part of the problem from prompt construction to model–operation interaction. Its effectiveness depends on the model's ability to select operations, construct arguments, interpret outputs, and determine whether further calls are required. Model selection and operation-surface design should therefore be evaluated jointly.

Third, operations should be designed around the reasoning required by the expected queries, not only around the source document's sections. Section-level operations were sufficient for retrieving relevant evidence, but were less effective when questions required counting, temporal filtering, entity matching, or ranking. For these tasks, normalized records, explicit status and date fields, and deterministic aggregation operations can reduce the amount of parsing and reasoning delegated to the model.

Finally, local and remote operation-mediated configurations address different operational priorities. Local function calling and local MCP provide greater trace visibility and control over the interaction loop, supporting debugging and auditing. Remote MCP provides a provider-managed integration boundary and lower token cost per majority-correct answer, but showed higher median latency and less direct observability. The appropriate configuration should therefore be selected according to the required balance among answer quality, token cost, latency, modularity, and auditability.

5 Threats to Validity

Construct validity. Answer quality is assessed with LLM judges rather than human annotators, so the results depend on judge prompts, evidence selection, and judge strictness. A full human evaluation was not feasible at this baseline scale because it would

require substantial annotation effort and domain expertise, but sampled human auditing is still needed, especially for cases of strong judge disagreement. We reduce this threat by using three judges, separating correctness from completeness, and reporting both majority and unanimous agreement.

To assess the dependence on individual judges, we also analyzed judge-level behavior. Each judge alone reproduced the same inline-over-operation-mediated ordering, with majority-accuracy gaps of +8.7, +11.9, and +6.7 percentage points for the GPT, Gemini, and Claude judges, respectively. Inter-judge agreement was substantial: the mean quadratic-weighted kappa was 0.71 for correctness and 0.66 for completeness. Using Claude as a non-same-family reference, we did not observe evidence of systematic own-provider judge bias.

The study also evaluates context provisioning over benchmark representations (cleaned HTML, parsed JSON, and question-specific evidence blocks), not over the raw Lattes pages alone. These representations make strategies comparable, but they may introduce parsing errors, information loss, or uneven structure across sections. This is particularly relevant for supervision-related questions, whose evidence is less structured than publication records.

Internal validity. Differences among strategies may reflect not only the provisioning mechanism, but also implementation choices, provider SDK behavior, prompt design, operation schemas, and each model's tool-use capability. This threat is stronger for remote MCP because the provider manages part of the interaction loop and the benchmark cannot observe all intermediate steps with the same granularity available for local strategies.

We partially mitigate implementation errors through an automated test suite that reaches 67% line coverage. The tests cover benchmark infrastructure, execution flow, operation dispatch, schema and output-shape validation, and representative fixtures for the Lattes service layer. This provides evidence that the implemented tool layer behaves consistently with its expected interfaces. However, test coverage does not guarantee that the parsed Lattes data is semantically complete or that every operation output is optimal for all question types.

Token accounting is another source of variation because providers report prompt, cache, reasoning, and tool-use tokens differently. We normalize these metrics as much as possible and, for Gemini remote MCP, add provider-reported native tool-use prompt tokens to direct prompt tokens. Even so, cross-provider cost comparisons still include some measurement noise.

External validity. This study uses five Lattes curricula and twelve questions from a single domain. The curricula were selected as controlled benchmark instances rather than as statistical representatives. A per-curriculum analysis showed majority accuracy ranging from 35.0% to 44.6%; in every leave-one-curriculum-out analysis, inline JSON remained the strongest configuration, followed by inline HTML, while operation-mediated strategies remained substantially more token-efficient. Thus, the main quality–efficiency trade-off is not driven by a single curriculum. An exploratory software-repository question-answering dataset produced different absolute results but similar dependencies on representation and operation design, suggesting that concrete outcomes are domain-specific. Broader claims still require more instances, domains, and task types.

The use of SaaS LLMs further limits reproducibility because provider-side updates may change model behavior over time even when model names remain fixed. We mitigate this threat by recording model identifiers, configurations, traces, and artifacts, but exact replication still depends on provider stability.

Conclusion validity. This is a baseline experiment with one repetition per condition. The results should therefore be read as evidence of trade-offs and failure modes rather than as definitive rankings. In addition, the same curricula, questions, and models are reused across strategies, so aggregate percentages are not independent samples. Future work should add repetitions, more models and datasets, and statistical analyses that separate curriculum, question, model, and strategy effects.

6 Related Work

This section positions our study with respect to four lines of related work: (i) LLM-based question answering over Lattes curricula; (ii) context engineering, RAG, and long-context evaluation; (iii) tool use and function calling benchmarks; and (iv) MCP benchmarks.

LLM-Based Querying over Lattes Data. The closest related work to ours is LattesRex, a chatbot for question answering over Lattes curricula [3]. LattesRex adopts a modular architecture inspired by RAG. Its pipeline contains a classifier, a retriever, and a generator: the classifier maps the user query to a curriculum category, the retriever extracts the corresponding content from the structured curriculum, and the generator uses the retrieved content to produce the final answer. LattesRex evaluates this structured approach against a monolithic full-document strategy using different model families and curriculum sizes, with qualitative assessment by linguists [3].

Our work was inspired by LattesRex and reuses part of its question-answering setting. However, the focus is different. LattesRex evaluates whether a metadata-driven structured approach can support LLM-based question answering over long Lattes curricula. Our study evaluates context provisioning as an architectural design decision. Instead of comparing only monolithic and structured preprocessing, we compare direct inline context, local function calling, local MCP, and remote MCP under a common benchmark engine. In addition, our benchmark records execution traces, function calls, tool calls, MCP calls, latency, token usage, and other efficiency metrics, allowing us to analyze the trade-offs among answer quality, efficiency, observability, and architectural complexity.

Context Engineering. Our work is also related to the broader literature on context engineering for LLMs, including context retrieval and generation, context processing, context management, RAG, memory systems, tool-integrated reasoning, and multi-agent systems [10]. This framing supports our view that context provisioning is not only a prompting concern, but an architectural design decision in LLM-based systems.

RAG is one of the most widely studied mechanisms for grounding LLM outputs in external information. The original RAG formulation combines parametric model knowledge with non-parametric retrieved memory [7]. Subsequent evaluation work has examined retrieval and generation quality using dimensions such as relevance, accuracy, faithfulness, and end-to-end answer quality [20]. Our study differs from RAG evaluation because we do not evaluate

a retriever or compare retrieval algorithms. Instead, we compare different mechanisms for exposing the same semi-structured context to the model.

Long-context evaluation is also relevant because inline context strategies rely on the model's ability to use large prompts effectively. Liu et al. show that models may fail to robustly use information in long contexts, with performance depending on where relevant information appears in the input [9]. This finding motivates evaluating alternatives to simply embedding the entire document in the prompt. In our study, inline context is treated as a strong baseline, but we also evaluate whether operation-mediated access can reduce context size while preserving answer quality.

Tool Use and Function Calling Benchmarks. A growing body of work studies LLMs that interact with external tools. ReAct combines reasoning traces with actions over external environments [19], while Toolformer investigates how models can learn to use APIs as tools [15]. Gorilla evaluates API-call generation and shows how tool documentation and retrieval can improve the ability of models to invoke APIs correctly [14]. API-Bank provides a comprehensive benchmark for tool-augmented LLMs over real-world APIs, focusing on the effectiveness and limitations of LLM tool use [8]. The Berkeley Function Calling Leaderboard (BFCL) further evaluates function-calling capabilities across single, multiple, parallel, and stateful settings, including function relevance and argument construction [13].

These benchmarks are important for measuring whether models can select tools and construct valid calls. Our benchmark addresses a different question. We assume a fixed domain-specific operation surface over Lattes curricula and compare the architectural mechanisms used to expose that context: local function calls, local MCP, and remote MCP. Thus, the unit of comparison is not only the model's generic ability to invoke externally defined functions or tools, but the context provisioning strategy used by the surrounding software system.

MCP Benchmarks. MCP is a relatively recent protocol, and its evaluation literature is still emerging. MCP-RADAR proposes a multi-dimensional benchmark for evaluating tool-use capabilities within the MCP framework [4]. It measures answer accuracy, tool selection efficiency, computational resource efficiency, parameter construction accuracy, and execution speed across different task domains. MCP-Bench similarly evaluates tool-using LLM agents on realistic multi-step tasks using multiple live MCP servers and hundreds of tools across diverse domains [17].

Our work is complementary to these MCP benchmarks. MCP-RADAR and MCP-Bench primarily evaluate model behavior in MCP-based tool-use environments. In contrast, we evaluate MCP as one context provisioning architecture among alternatives. By comparing local MCP with local function calling, we isolate the effect of introducing an MCP protocol boundary while keeping the same Lattes operation surface. By comparing local MCP with remote MCP, we evaluate the consequences of moving from benchmark-controlled MCP execution to provider-managed remote MCP access. Finally, by comparing MCP-based strategies with inline context, we analyze the quality–efficiency trade-off between broad prompt-based visibility and structured operation-mediated context access.

7 Conclusion

This paper evaluated context provisioning strategies for LLM-based systems through a question-answering benchmark over Lattes curricula. The baseline compared inline HTML, inline JSON, local function calling, local MCP, and remote MCP across 1,200 answer-generation runs and 3,600 LLM-as-a-judge assessments. The results show that context provisioning is not only a prompting decision, but also an architectural design choice that affects answer quality, token consumption, latency, operation-call behavior, and trace observability.

Inline JSON achieved the highest majority accuracy, and inline strategies benefited from broad visibility over the curriculum. However, this quality came at a substantial token cost, especially when cost was normalized by majority agreement or unanimity. Operation-mediated strategies produced lower aggregate accuracy, but reduced token consumption by a large margin and still delivered a non-negligible level of majority and unanimous agreement. Remote MCP achieved the lowest token cost per majority-correct answer, while local function calling and local MCP provided more direct observability over the operation loop.

The results also show that strategy behavior depends on question characteristics and evidence structure. Broad profile-oriented questions were answered more reliably, whereas counting, temporal filtering, entity matching, and supervision-related questions were consistently harder. In particular, supervision data in the parsed Lattes representation is less structured than publication data, which helps explain the low accuracy of supervision-counting questions. This suggests that the design of context representations and operation outputs can be as important as the choice between inline and operation-mediated access.

Future work will focus on three immediate directions. First, we plan to repeat the experiment with more curricula, questions, models, and repetitions to assess the robustness of the observed trade-offs. Second, we will refine the operation surface for aggregation-heavy questions, with particular attention to whether operations should return more structured records for sections such as supervisions. Third, we will include human auditing of sampled answers and judge disagreements to better calibrate the LLM-as-a-judge evaluation protocol. These steps are intended to validate and refine the current baseline before broader claims are made about optimal operation design or generalization to other domains.

Artifact Availability

A replication package for this study, including the full dataset, configuration instructions and experiment results, is available at https://doi.org/10.5281/zenodo.22118922. The package is distributed under the MIT License, except for third-party materials, which remain subject to their original terms.

Acknowledgements

Generative AI tools were used to assist in structuring and revising this manuscript as well as generating code for data collection and analysis.

References

[1] Anthropic. 2024. Introducing the Model Context Protocol. https://www.anthropic.

com/news/model-context-protocol. Accessed: 2026-04-30.
[2] Conselho Nacional de Desenvolvimento Científico e Tecnológico. 2023. Plataforma Lattes. https://www.gov.br/cnpq/pt-br/acesso-a-informacao/acoes-e-programas/plataforma-lattes. Accessed: 2026-04-30.
[3] Lucas Darcio et al. 2025. LattesRex: Building ChatBots for Semi-Structured Documents. In *Simpósio Brasileiro de Tecnologia da Informação e da Linguagem Humana (STIL)*. SBC, 125–136.
[4] Xuanqi Gao et al. 2025. MCP-RADAR: A Multi-Dimensional Benchmark for Evaluating Tool Use Capabilities in Large Language Models. *arXiv preprint arXiv:2505.16700* (2025).
[5] Google. 2026. Function Calling with the Gemini API. https://ai.google.dev/gemini-api/docs/function-calling. Accessed: 2026-04-30.
[6] Xinyi Hou et al. 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. *ACM Transactions on Software Engineering and Methodology* (2025).
[7] Patrick Lewis et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. 9459–9474.
[8] Minghao Li et al. 2023. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*. 3102–3116.
[9] Nelson F. Liu et al. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173.
[10] Lingrui Mei et al. 2025. A Survey of Context Engineering for Large Language Models. *arXiv preprint arXiv:2507.13334* (2025).
[11] Model Context Protocol. 2025. Model Context Protocol Specification. https://modelcontextprotocol.io/specification/2025-11-25. Accessed: 2026-04-30.
[12] OpenAI. 2025. Function Calling in the OpenAI API. https://developers.openai.com/api/docs/guides/function-calling. Accessed: 2026-04-30.
[13] Shishir G Patil et al. 2025. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proc. of the 42nd International Conference on Machine Learning (ICML)*.
[14] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large Language Model Connected with Massive APIs. *Advances in Neural Information Processing Systems* 37 (2024), 126544–126565.
[15] Timo Schick et al. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, Vol. 36.
[16] Thoughtworks. 2025. The Model Context Protocol: Getting Beneath the Hype. https://www.thoughtworks.com/en-br/insights/blog/generative-ai/model-context-protocol-beneath-hype. Accessed: 2026-04-30.
[17] Zhenting Wang et al. 2025. MCP-Bench: Benchmarking Tool-Using LLM Agents with Model Context Protocol. *arXiv preprint arXiv:2508.20453* (2025).
[18] Peng Xu et al. 2024. Retrieval Meets Long Context Large Language Models. In *Proc. of the 12th International Conference on Learning Representations (ICLR)*.
[19] Shunyu Yao et al. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proc. of the 11th International Conference on Learning Representations (ICLR)*.
[20] Hao Yu et al. 2024. Evaluation of Retrieval-Augmented Generation: A Survey. In *CCF Conference on Big Data*. Springer, 102–120.
[21] Murong Yue. 2025. A Survey of Large Language Model Agents for Question Answering. *arXiv preprint arXiv:2503.19213* (2025).
[22] Lianmin Zheng et al. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, Vol. 36. 46595–46623.