MASTER'S THESIS 2026

Connecting Language Models to Data Systems in the Energy Sector

Jona Waldfogel, Jonathan Giegold

ISSN 1650-2884
LU-CS-EX: 2026-07

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2026-07

Connecting Language Models to Data Systems in the Energy Sector

Att koppla Språkmodeller till Datasystem inom Energisektorn

Jona Waldfogel, Jonathan Giegold

Connecting Language Models to Data Systems in the Energy Sector

(An Evaluation of the Model Context Protocol)

Jona Waldfogel
jo5720wa-s@student.lu.se

Jonathan Giegold
jo5515gi-s@student.lu.se

March 13, 2026

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Marcus Klang, marcus.klang@cs.lth.se
Christer Friberg, christer.friberg@eon.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

Energy networks are critical infrastructure, and leveraging recent advances in AI can support efficient access to operational data. Conversational agents can streamline access to data, but reliability depends on how they use enterprise tools. This thesis evaluates the Model Context Protocol, a standard for connecting language models to external systems, in an energy-data setting at E.ON. We implement three MCP servers (metadata, grid infrastructure, metering) and benchmark Claude Sonnet 4, GPT-4o-mini, along with a locally hosted Phi-4-mini. Using mirrored private (E.ON) and public (UK Power Networks) datasets, we adapt a protocol-aware benchmark combining rule-based checks of tool invocations with rubric-based judging of task completion, grounding, and planning by o4-mini. We extended the benchmark with a deterministic check comparing expected vs generated answers. Generated tasks require either one server (1-server) or cross-server coordination (2-server). In 1-server tasks, the cloud models show more than 0.99 schema understanding and high task completion (0.958 for Claude Sonnet 4 vs 0.940 for GPT-4o-mini), with Claude Sonnet 4 producing more factually correct answers with a higher Expected vs generated score (0.968 vs 0.907). The cloud models arrive at similar answers, but use different strategies. Claude Sonnet 4 plans more efficiently, and GPT-4o-mini is faster but runs more rounds and is 16 times cheaper per task. The 2-server setting decreased task completion to around 0.52 for both cloud models, and tool calls increased, resulting in higher execution time and token consumption. The local Phi-4-mini is unreliable beyond lookups with low schema understanding (0.776 and 0.628) and task completion (0.355 and 0.138) in 1-server and 2-server settings, respectively. Overall, domain-based MCP servers proved easy to extend with additional tools. Still, our results show that 2-server workflows induce substantial coordination overhead, making server boundaries and tool discovery critical design choices.

Keywords: tool-augmented agents, MCP, energy networks, benchmarking, modular architecture

Acknowledgements

We want to thank our academic supervisor, Marcus Klang, for his guidance throughout this thesis, especially his support in shaping the structure, methodology, and overall direction of the work.

We also thank our supervisor at E.ON, Christer Friberg, for his continuous technical support, help with configurations and access to required technologies and data, and for sharing practical ideas that improved the solution.

Finally, we thank Jacek Malec for reviewing our work and providing feedback, as well as E.ON and the Data Analytics Team for their trust and support. We are also grateful to everyone at E.ON who met with us and helped us understand the energy network context and the relevant data systems.

Contents

Chapter 1
Introduction

This thesis was conducted at E.ON, a large European energy company that operates extensive energy infrastructure and distribution networks across Europe. E.ON serves around 47 million customers and manages approximately 1.6 million kilometres of energy distribution grid, generating large volumes of operational data from metering systems, grid assets, outages and geospatial information systems (E.ON, 2025). In practice, this data is distributed across multiple platforms and schemas, making it difficult and time-consuming for engineers and analysts to locate the right information, interpret it correctly, and apply it to day-to-day operational and analytical tasks.

At the same time, the rapid adoption of Artificial Intelligence (AI) and conversational agents has transformed how people interact with information systems in everyday contexts (Bick et al., 2024; Chatterji et al., 2025). Conversational agents built on Large Language Models (LLMs) can interpret natural language queries, retrieve information, and generate fluent responses (Chang et al., 2024; Qin et al., 2024). When extended with agent capabilities, it can decompose user goals, interact with external tools and take sequences of actions rather than returning a single static answer. However, generic LLMs do not inherently know an organisation's specific data, systems or terminology, and they are prone to hallucinations and misinterpretations when context is missing or incomplete. For enterprise use, the value of conversational agents therefore depends on how effectively they can be grounded in reliable, up-to-date operational data.

The Model Context Protocol (MCP) is an open standard that defines how conversational agents connect to external tools and data sources through a common, protocol-based interface. In MCP, back-end systems expose tools, resources and prompts via MCP servers, while a host application and MCP clients manage connections and pass structured JSON payloads between the LLM and these servers. This separation allows the conversational agent to query databases or call domain-specific operations without changing the underlying model, and supports modular integration of new data sources over time. In domains like energy networks, where data is heterogeneous and distributed, MCP offers a principled way to provide conversational agents with structured, machine-readable context drawn directly from oper-

ational systems.

Because E.ON's operational data cannot be disclosed publicly, this thesis uses two parallel datasets to study MCP-based conversational agents in a realistic but reproducible way. We define a Private Dataset, consisting of E.ON's internal operational data accessed in a secure environment, and a Public Dataset constructed from UK Power Networks' open data with similar schema characteristics. The Public Dataset is used for all reportable experiments, while the Private Dataset is used to validate whether observed patterns transfer to real enterprise data under comparable MCP server configurations.

Within this setting, the thesis investigates how conversational agents connected via MCP can streamline access to operational energy data, how different LLMs perform when acting through MCP servers over the Private and Public Datasets, and how an MCP architecture can be designed to remain modular as new datasets and schemas are added. Building on MCP-Bench by Wang et al. (2025), a protocol-aware benchmarking framework that evaluates LLM agents through real MCP servers, we adapt its methodology to our energy-network context in order to assess protocol adherence, factual correctness and planning behaviour in MCP-based conversational agents.

1.1 Aims and Research Questions

The overall aim of this thesis is to assess whether MCP, together with conversational agents, is a suitable choice for scalable, context-aware access to enterprise energy data, and to understand how an MCP architecture should be designed to support scalable expansion with additional datasets. In this context, modular refers to a design that can easily scale and adapt in terms of adding datasets and changing schemas without interrupting service.

RQ1: Which evaluation metrics are most suitable for assessing MCP-generated responses?

RQ2: Which models are suitable for generating responses with the Model Context Protocol (MCP)?

RQ3: How well do chosen models perform in terms of response quality, information accuracy and cost?

RQ4: How can a modular MCP architecture be designed?

1.2 Scope

This thesis investigates how conversational agents can extract information from datasets in the electrical energy domain. Specifically, we evaluate two datasets in parallel to handle confidentiality concerns. These datasets are, publicly available UK Power Networks data and E.ON's internal operational data. We investigate both a local and two cloud-based models to assess cost trade-offs. We explicitly exclude larger local models, as we do not have access to hardware that can run them. We were able to choose models from different providers that offer distinct architectures and pricing tiers to investigate their respective trade-offs. The three models are Claude Sonnet 4, GPT-4o-mini and Phi-4-mini-instruct.

1.2.1 Delimitations

The study is delimited in several ways. Only tools and queries that can be mapped consistently between the Public and Private Datasets are implemented, limiting the range of tasks to domains where comparable schemas exist (metadata, network information and metering). We evaluate two specific cloud-hosted models and one very small local model. Larger local models and additional model families are excluded due to hardware and time constraints, so the results do not cover the full spectrum of possible LLM architectures.

The MCP servers are restricted to read-only operations. We do not study write actions or tools with side-effects, which means that issues such as transactional safety and rollback strategies fall outside the scope of this thesis. The evaluation is based on a finite QA corpus with manually authored gold-standard answers and Swedish fuzzy prompts, and on an adapted MCP-Bench configuration with a limited number of 2-server tasks. This design supports controlled comparison across models and servers, but it also constrains task diversity and means that 2-server results should be interpreted as indicative rather than exhaustive.

1.3 Division Of Work

This thesis was carried out as a close collaboration between the two authors. Most project decisions were made jointly, including the formulation of the research questions, the selection of datasets and models, and the overall methodological direction.

In terms of implementation, both authors contributed to developing the MCP-based benchmarking environment and tool ecosystem. Jonathan Giegold took the main responsibility for the API management, including request construction, execution parameters, and caching. He also led the adaptation and integration work related to the MCP-Bench evaluation pipeline, and was responsible for setting up and running the local Phi-4-mini experiments. Jona Waldfogel took the main responsibility for tool design decisions, such as tool granularity, as well as mapping and tailoring tools to match the different datasets and domains.

For benchmark content creation, both authors participated in identifying suitable public datasets, but Jona Waldfogel was responsible of the QA corpus with gold answers. Jona also led the development of the Swedish fuzzy prompt variants used in the benchmark. Evaluation decisions were made jointly, including how to treat expected-answer scoring across settings, while Jona implemented the expected-vs-generated scoring. Both authors were involved in running experiments, troubleshooting failures, and performing manual checks to ensure that the evaluation behaved as intended.

The analysis and presentation of results were also shared. Jona Waldfogel took the lead in producing plots and tables, while the interpretation of findings and the discussion of implications were developed collaboratively. The writing process was a parallel effort where both authors reviewed and revised all parts of the manuscript. Overall, Jonathan Giegold contributed more to the Introduction, LLM background, evaluation metrics, Method Sections 3.2–3.5 and 3.7, and the Conclusion, while Jona Waldfogel contributed more to Sections 2.2–2.4, Section 3.6, and Sections 5.9–5.12. The remaining sections were written and refined jointly to ensure consistency in structure, terminology, and style.

1.4 Report Disposition

Following the introduction, chapter 2 provides the technical background for the thesis. It covers LLMs and tool-using agents, MCP, and the datasets and evaluation concepts needed to understand the implementation. It then moves on to methodology, including the data sources, model selection, MCP design, the benchmark configuration and evaluation procedure used in the experiments. With this foundation, the empirical results are presented for both 1-server and 2-server experiments across the public and private datasets. Following the results, the discussion highlights performance patterns, failure modes, and implications for enterprise use. Finally, the thesis concludes by summarising the main takeaways and outlining directions for future work. The appendices provide supporting material such as the task prompts and example QA corpus samples used in the benchmark.

Chapter 2
Background

This chapter provides the technical foundation for the methods used in this thesis. It introduces the core concepts behind Large Language Models (LLMs), their limitations, and the techniques used to supply them with reliable context. It then outlines key developments in LLM evaluation, including the use of LLM-as-a-judge and the emergence of tool-using LLM agents. Central to this work is the Model Context Protocol (MCP), a framework that standardises how LLMs access external data and execute actions through tools. The chapter concludes with a description of the datasets, what is meant by modularity, evaluation frameworks, and metrics that support the experimental setup. Together, these elements establish the background needed to understand the design and analysis of the MCP-based system developed in this thesis.

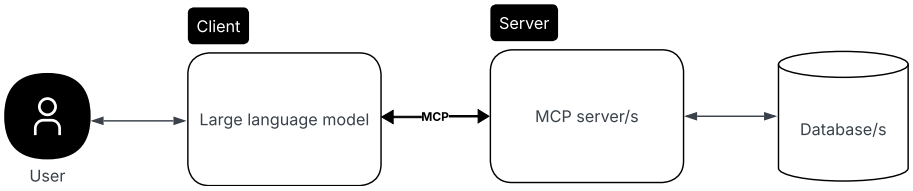

Figure 2.1: General architecture of implementation setup.

2.1 Large Language Models

LLMs are machine learning systems trained on vast corpora of text to predict and generate human-like language. They are typically based on the *Transformer architecture*, which enables the LLM to capture long-range dependencies in text, giving it better "context" (Vaswani et al., 2017). These models support applications such as summarisation, translation, and question answering, demonstrating broad linguistic capabilities. Despite their strengths, LLMs have well-known limitations. For instance, an LLM might misinterpret a prompt if it lacks clear

context, leading to incorrect or irrelevant outputs. They can also reproduce biases present in their training data or generate misleading information (Bontridder and Poullet, 2021). A particularly critical issue is **hallucination**, where an LLM confidently produces plausible but factually wrong or fabricated content (Currie, 2023). These challenges underscore the importance of careful context management when applying LLMs.

2.1.1 LLM Families

A wide range of LLMs now exists, each optimised for different capabilities, licensing models, and deployment settings (Kucharavy, 2024). Almost all modern large language models rely on large-scale self-supervised pre-training, but they differ substantially in training objectives, scale, and the capabilities that emerge from these design choices (Minaee et al., 2025).

One of the most relevant distinctions lie in how models are trained to handle multi-step reasoning. Contemporary LLMs typically undergo a multi-stage training process. Base pre-training is the prediction process over massive text corpora that was just mentioned, done to establish broad linguistic patterns and gain factual knowledge (Wei et al., 2022). Because these datasets include substantial code examples, models develop inherent understanding of function definitions and structured data formats. This foundation is then refined through supervised fine-tuning on instruction-response pairs, teaching the model to follow directives and produce coherent answers (Minaee et al., 2025). Further optimisation using reinforcement learning from human feedback (RLHF) aligns outputs with human preferences, improving response quality, safety, and reasoning structure (Minaee et al., 2025). There is also reasoning optimisation through chain-of-thought (CoT) training, which further distinguishes modern agentic models.Wei et al. (2022)showed that prompting models to generate intermediate reasoning steps significantly improves multi-step problem performance. CoT training builds on this insight. When incorporated during training, models learn to decompose complex tasks and verify intermediate steps. They can self-correct before producing final outputs.

The three models evaluated in this thesis occupy distinct positions in this training spectrum. Anthropic's Claude Sonnet 4 is a foundation model trained using Constitutional AI (Bai, 2022; Anthropic, 2025e). This approach combines RLHF with reinforcement learning from AI feedback guided by ethical principles. The result is substantial reasoning optimisation, long-context understanding exceeding 200,000 tokens, and demonstrated tool-use capabilities. GPT-4o-mini by OpenAI operates at reduced scale to balance cost and capability, undergoing supervised fine-tuning followed by RLHF optimisation (OpenAI, 2024) that enables reliable performance on moderately complex agentic tasks while maintaining faster execution and lower per-token costs. Phi-4-mini-Instruct by Microsoft represents the emerging class of small language models designed for resource-constrained deployment Minaee et al. (2025). The model with its 3.8 billion parameters was trained on high-quality web and synthetic data, with particular emphasis on reasoning-rich text and curated code datasets (Microsoft, 2025). Despite this training focus, its limited parameter count restricts reasoning depth and tool-use reliability compared to foundation-scale models.

The Artificial Analysis Intelligence Index (Artificial Analysis, 2025) aggregates results from benchmarks such as MMLU-Pro (Wang et al., 2024b), revealing consistent trade-offs between model families. Larger models with dedicated reasoning training typically achieve higher scores, while smaller models without explicit reasoning optimisation trail behind.

These patterns directly inform the cost-performance analysis presented in this thesis, where Claude Sonnet 4's superior reasoning comes at higher token costs, GPT-4o-mini offers a middle ground, and Phi-4-mini-Instruct provides minimal cost but insufficient reliability for structured agentic workflows.

Cloud vs. Local Deployment

Cloud-hosted LLMs offer access to powerful models without the need for infrastructure management, but they come with ongoing usage costs and potential regulatory concerns when sensitive data must leave an organisation's infrastructure. Local deployment provides stronger data security guarantees and predictable, hardware-based cost profiles without per-token charges. Despite these advantages, locally run models remain less common as many state-of-the-art models are too large to run efficiently on consumer hardware, and maintaining local model-serving systems requires expertise in GPU provisioning, model optimisation, and performance tuning (Dantas et al., 2025). However, these limitations are likely to diminish as hardware accelerators become more accessible and as smaller, high-performance open models continue to improve (Kachris, 2025).

2.2 LLM Agents

Recent progress in LLMs has led to a shift from viewing them purely as text predictors towards treating them as autonomous agents that can perceive an environment, decide on actions, and interact with external tools or software systems to achieve goals (Wang et al., 2024a). In contrast to a standard chatbot that answers a single prompt, an LLM agent is typically defined by three properties which are that it has a goal or task specification, it interacts with an environment, and it can take sequences of actions over time rather than producing a single response.

Researchers increasingly refer to these systems as LLM-based autonomous agents. Wang et al. (2024a) proposes a unified framework with four core modules. These modules handle a profile that encodes the agent's role, goals and capabilities, a memory that stores past interactions, a planner that decomposes high-level goals into subtasks and an actor that issues concrete actions (Wang et al., 2024a). This breakdown is useful because different agent architectures can be compared in terms of how they implement each module.

Two influential works illustrate how LLMs are turned into agents with reasoning, acting loops and tool use training.

The ReAct framework introduces a prompting method that interleaves reasoning traces with actions in a single trajectory (Yao et al., 2023). In the paper, Yao et al. (2023) explores letting LLMs generate both reasoning traces and actions and that these traces enable models to induce track and update action plans. The action, in turn, allows them to interact with external environments and obtain additional information (Yao et al., 2023). Yao et al. (2023) demonstrates that this joint reasoning and acting approach improves performance in both knowledge-intensive and interactive environments.

Complementary to prompting approaches, Toolformer explores how to train a language model to call tolls (Schick et al., 2023). Starting from a few examples of API calls, the model generates API annotations in large text corpora. These candidates are filtered based on

whether the API outputs improve the model's language modelling loss. The filtered annotations serve as self-supervised training data, resulting in an LLM that can decide on its own when and which tool to invoke during inference.

2.2.1 Challenges With LLM Agents

Despite rapid progress, current LLM agents face considerable limitations. Survey work highlights recurring failure modes such as hallucinated actions or tool calls, weak long-term planning, difficulty handling delayed rewards and sensitivity to prompt design and environment stochasticity Wang et al. (2024a). Safety and alignment concerns are amplified in the agent setting because the model not only says things but also does things in external systems, for example, running code or modifying files. Cost and latency become critical when agents must perform many tool calls or long action sequences.

One further problem is evaluation. Prior NLP and LLM evaluations rely on static, offline test sets, whereas LLM-based agents must be assessed across multiple objectives, such as task completion, latency and cost, robustness to disruption and environment changes and multi-agent collaboration behaviour (Mohammadi et al., 2025). Mohammadi et al. (2025) further conclude that robust, standardised evaluation remains an open research challenge because existing benchmarks are fragmented and mostly single objective, rarely capture realistic and dynamic deployment settings and often depend on expensive or ad hoc evaluation procedures that are difficult to scale or compare across studies.

2.2.2 Tool Calling LLMs

Several industry systems now build practical tool-using agents rather than research prototypes. Studies of these systems often describe a common workflow that includes task decomposition, tool selection, tool execution, and response synthesis (Xu et al., 2025; Chen et al., 2025). Commercial interfaces such as OpenAI's function-calling APIs directly support this workflow by letting models trigger predefined functions instead of generating free text for every action. In these systems, the model acts as a planner that decides which function to call based on the user's request. Developer frameworks such as LangChain focus on supporting this planning and tool selection process. They provide ways to route queries to the right tools and combine tool outputs into a final answer. However, recent evaluations such as ToolEyes show that even advanced LLMs still make frequent mistakes in choosing the correct tool or producing a well-structured result afterwards (Ye et al., 2025). These difficulties match the practical challenges faced by engineers building real-world agent systems.

2.2.3 Retrieval-Augmented Generation

A particularly important pattern in tool-using agents is Retrieval-Augmented Generation (RAG), where the agent queries an external knowledge base at inference time to access up-to-date or domain-specific facts (Lewis et al., 2021). Rather than relying solely on information encoded during training, the model retrieves relevant context from a structured data source and incorporates it into its response. This enables agents to answer questions about current events, organisational data, or specialised domains without retraining. Implementing RAG

at enterprise scale introduces several technical requirements. Organisations must maintain and continuously update large, indexed knowledge bases to ensure retrieved information remains current. The retrieval system must employ efficient search algorithms capable of handling high query volumes with low latency. As knowledge bases grow, infrastructure costs for storage scale proportionally. These considerations make RAG resource-intensive, particularly in environments with substantial data volumes and unpredictable query patterns.

Recent efforts have focused on standardising how agents interact with tools and data sources. One example is the Universal Tool Calling Protocol (UTCP) (UTCP Contributors, 2025a), which aims to create a common way for tools to interact with LLM agents. This reflects what Li (2025) describes as a key direction for the field, which is moving away from custom integrations and toward shared standards that make it easier to compare systems, reproduce results, and reuse tools across different platforms. This trend has paved the way for the Model Context Protocol (MCP), which proposes a common standard for how models access and communicate with tools.

2.3 Data Protocols

Our MCP servers wrap two different data-access protocols. The private dataset is accessed via Dremio, a commercial data platform, while the public dataset is accessed via Opendatasoft, an open-data platform. Dremio exposes virtualised, strongly typed enterprise schemas, providing a SQL interface with a dialect that is largely ANSI-SQL-compliant but includes engine-specific functions and extensions, as described in their Architecture (Dremio, 2025b). In the Job Results specification for Dremio, they state that queries are submitted as SQL over a REST endpoint and return a job identifier rather than results directly. Results are then fetched from a results endpoint that returns `rowCount`, the `schema` and its contents, and `rows`. The public dataset is queried only with HTTP `GET` requests where Opendatasoft Query Language (ODSQL) parameters perform filtering, selection, grouping order, limits and offsets. This makes single-call lookups and small aggregates convenient (Huwise, 2025). UK Power Networks exposes its dataset catalogue metadata using Dublin Core fields. Dublin Core is a cross-domain metadata standard that defines a small set of common elements for describing resources, such as title, creator, description, publisher, date, and rights (Dublin Core, 2026).

Both protocols are HTTP-based and support controlled retrieval through filtering and pagination. Dremio provides typed schemas and a consistent SQL interface, but queries first return a job ID and then fetch the results in a separate call. Opendatasoft returns results synchronously and is lightweight, but enforces stricter request limits for data retrieval that is not grouped. Dremio (2025b); Huwise (2025)

In this report, we refer to the public dataset platform as Opendatasoft rather than Huwise, since Opendatasoft is the platform name specific to our dataset access, and the rebranding occurred during the project.

2.4 Model Context Protocol

In late 2024, Anthropic introduced the open-source Model Context Protocol (MCP), which enables LLMs to communicate with external systems and data sources in a standardised mannerHou et al. (2025). Inspired by the Language Server Protocol (LSP), used to provide advanced language knowledge to enable features such as syntax highlighting and code completionHou et al. (2025)Ray (2025). MCP builds on the principles of LSP by extending the concept of standardised communication beyond code editors, allowing LLMs to interact with a wide range of tools and data. This created a landscape where developers can build their own MCP servers to let LLMs integrate and share data with almost any external resource, system or tool. Creating space for innovation in data sharing and integration, possibly leading to improvements in productivity and efficiency in various sectors.

MCP's ability to standardise is crucial as it removes the need for domain-specific integrations and promotes interoperability across different agents. Fundamentally, MCP defines a universal server-to-client framework that establishes a consistent connection between agents and their environment. In this framework, data sources provide agents with context required for the reasoning and decision-making, while action capabilities allow agents to perform tasks in their specific domain Brodimas et al. (2025).

2.4.1 Architecture

MCP's architecture is built around the three components Host, Client and Server. The host application initiates and supervises connections to one or more MCP servers. For every server it connects to, the host creates a dedicated client instance. Each client is paired with exactly one server, forming a one-to-one connection. An example setup is shown in Figure 2.2.

- Host - AI application responsible for managing one or more clients.
- Client - Maintains a connection to a server and retrieves contextual information and makes it available to the host.
- Server - Exposes context to connected clients.

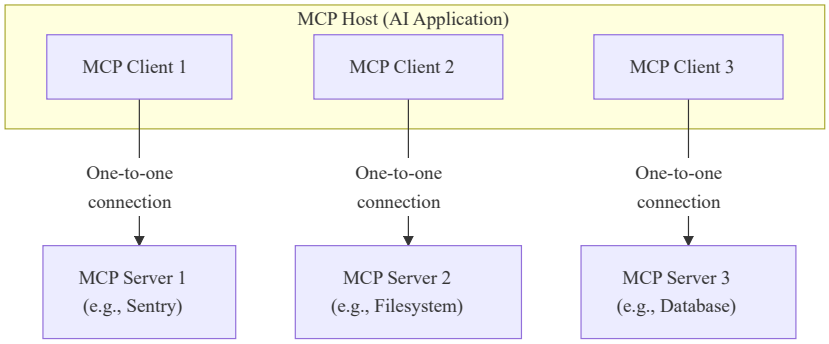

Figure 2.2: Example architecture of MCP from the Architectural overview section Anthropic (2025a).

2.4.2 Layers

In the MCP documentation's Architectural Overview, MCP is structured into two layers, a data layer and a transport layer. The data layer specifies the JSON-RPC 2.0 protocol used between clients and servers. This includes connection management, capability negotiation, and the use of primitives such as tools, resources, prompts and notifications. The transport layer describes how those messages are transported, including connection setup, framing, and authentication. MCP supports Stdio for low-latency local communication and streamable HTTP for remote access, where common web authentication, such as OAuth, can be used. Together, they separate protocol semantics from the underlying communication mechanism, which helps MCP behave consistently across environments Anthropic (2025a).

2.4.3 LLM Integration Lifecycle

As described in the Lifecycle section of the MCP documentation, MCP is a protocol with states structured around the three phases initialisation, operation, and shutdown. Lifecycle management ensures that the client and server agree on protocol versions and supported capabilities before communication begins Anthropic (2025a).

From the LLM integration view, the important behaviour is how tool outputs are returned to the model. When the model requests a tool, it produces a structured typed tool call as a JSON object. The object contains the tool name and an arguments field whose keys and types match the tool schema that the server exposes. Servers publish these schemas so the host and client can validate and serialise parameters and then forward the request as a JSON-RPC invocation over a transport such as standard input output for local runs or HTTP streaming for remote services.

The client transmits that request to the MCP server, which executes the requested operation and returns a typed JSON object. The host serialises that object as a tool response message and injects it into the model's conversation history, so the model consumes the result on its next generation step. This keeps execution and access logic on the server side while presenting the model with structured evidence to reason over.

2.4.4 Description Environment

MCP defines a set of primitives for exchanging information and actions between clients and servers, as stated in the Architecture overview of the documentation Anthropic (2025a). On the server side, three main primitives are **tools**, **resources** and **prompts**. Tools represent executable functions that clients can call to perform tasks such as file operations, API requests, or database queries. Resources supply structured contextual data, whereas prompts are reusable templates that structure interactions with the language model. Servers expose these primitives through standardised methods for discovery, retrieval and execution. This allows clients to dynamically access available capabilities.

MCP also specifies client-side primitives that enable two-way interaction. With **sampling**, a server can ask the client's language model for a completion while **elicitation** is used to request additional user input or confirmation. The **logging** primitive supports servers sending log messages to clients for debugging and monitoring Anthropic (2025a).

2.4.5 Benefits

The architecture provides benefits such as traceability, scalability and extensibility. This means that each computational step can be monitored and reproduced, and new tools and sources can be integrated without altering the models. Moreover, the MCP architecture is domain-specific, meaning that it can be applied to any system that exposes an accessible interface Avila et al. (2025).

MCP offers a standardised interface for connecting diverse tools and data sources. This reduces the need for custom-made integrations across different systems, and once tool operations are exposed, they can be reused across AI applications without further integration work. Such system compatibility is particularly important in organisations with diverse technical infrastructures and varied data representations. The protocol also benefits from a relatively mature and active ecosystem. On GitHub, MCP repositories show broad community interest, with many repositories receiving relatively high star counts (Anthropic, 2025c). By contrast, official UTCP repositories currently show more limited visibility (UTCP Contributors, 2025b), which makes the broader community support for MCP reduce technical uncertainty.

2.4.6 Modularity

A central design principle of the Model Context Protocol is modularity. MCP separates concerns across hosts, clients, and servers, allowing each component to evolve independently while maintaining a stable communication interface. This separation means that developers can introduce new data providers, computation modules, or domain-specific tools without modifying the host application or underlying agent logic. Instead, new capabilities are exposed simply by adding additional MCP servers or extending existing ones with new tools, resources, or prompts.

Modularity also enhances maintainability and system longevity. Because each MCP server encapsulates a well-defined set of functions and data sources, components can be updated, replaced, or scaled in isolation. This design enables experimentation with alternative data pipelines or model-side behaviours while ensuring compatibility with the broader ecosystem of MCP-compliant services. In practice, this results in a flexible architecture where functionality grows naturally and where changes can have minimal impact on other parts of the system.

2.4.7 MCP Python SDK

The official MCP Python SDK by Anthropic initially referenced FastMCP but has since Jan 25, 2026, been renamed to MCPServer (Anthropic, 2026). Confusingly, a separate implementation also called FastMCP exists, developed by jlowin under the Prefect organisation, which has achieved greater community adoption as measured by GitHub stars (Prefect, jlowin, 2026). According to their README, FastMCP 1.0 was incorporated into the official MCP SDK in 2024, establishing it as the standard implementation. Also, Git history shows individual jlowin contributed to both projects. This thesis uses the official Python SDK. The functional equivalence between implementations means the architectural principles discussed apply broadly to MCP server development regardless of SDK choice.

MCPServer abstracts low-level JSON-RPC details, letting developers focus on tool registration, invocation, and input-output handling without custom networking code (Li et al., 2025). Tools, resources and prompts can be registered declaratively using decorators. MCPServer automatically generates the parameter schemas and tool metadata from the Python function signatures and their docstrings. In practice, this means that the types, required arguments, and short descriptions defined by the developer in normal Python code are directly exposed to the MCP client and the language model, without requiring any separate schema definitions. This reduces the risk of unintended changes, improving maintainability as tool inventories grow. It also makes it easier to keep server behaviour consistent across environments, as described in the "Build an MCP server" section of the documentation Anthropic (2025a). The SDK also supports structured outputs, which help expose machine-interpretable result schemas to the client and ultimately the LLM Anthropic (2025b).

2.5 Evaluation Frameworks

Systematic evaluations are essential for making sense of the rapidly growing space of LLMs and agentic systems. As models gain capabilities in reasoning, tool use, and autonomous decision-making, there is no single sufficient metric or benchmark. This is because performance must be assessed across multiple areas such as knowledge, robustness, factual reliability, efficiency and behaviour in interactive environments. Simultaneously, reproducibility and comparability become increasingly important and especially when agents are embedded in complex infrastructures such as MCP-based tool ecosystems.

This section, therefore, surveys the main strands of recent work on evaluation. First, we review LLM benchmarks for general use that target knowledge and reasoning. We then discuss the emerging LLM-as-a-judge paradigm, where models are used as a scalable substitute for human annotators. Next, we consider benchmarks for tool-using agents, with a focus on MCP-based evaluations that emphasise cross-server planning and workflows that more closely mirror operational MCP behaviour. Finally, we introduce the test standards and reproducible procedures adopted in this thesis.

2.5.1 Benchmarking of LLMs

General LLM benchmarking has progressed from knowledge tests to broader evaluations of reasoning, robustness and factual reliability. MMLU measures zero- and few-shot performance across 57 academic and professional subjects (Hendrycks et al., 2021), and they establish a standard for broad knowledge and problem-solving ability. BIG-Bench expands this to over 200 community contributed tasks designed to probe complex reasoning, linguistic nuance and scaling behaviours (Srivastava et al., 2023). HELM shifts the focus toward multimetric evaluation by comparing models across 7 metrics: accuracy, robustness, calibration, fairness, toxicity, efficiency and bias (Bommasani et al., 2023). TruthfulQA targets factual reliability by testing models on questions crafted to extract common misconceptions (Lin et al., 2022). Lin et al. (2022) revealed that larger models often generate more confident falsehoods.

2.5.2 LLM As A Judge

LLM-as-a-judge refers to the practice of LLMs not as the system under text, but as automated evaluators together with models' outputs. Instead of asking humans to rate every answer, a strong LLM is prompted with the task description, the input, one or more candidate outputs and an evaluation rubric. It is then asked to assign scores or preferences. This paradigm has emerged as a major family of evaluation methods for generative models alongside traditional benchmarks and human preference studies (Li et al., 2024; Gao et al., 2025a).

Frameworks such as G-Eval standardise prompts and force the judge to output a structured rating that can be parsed automatically. Chiang and Lee (2023) shows that the specific design of this evaluation prompt is important. For example, asking the judge to explain its rating tends to improve the correlation with human judgment compared to simply eliciting a score.

Mt-Bench and Chatbot Arena, introduced by Zheng et al. (2023), systematically explored LLM-as-a-judge for open-ended dialogue evaluation. Zheng et al. (2023) report that strong judges such as GPT-4 can reach over 80% agreement with both controlled and crowdsourced human preferences, which is roughly the same chance of agreement amongst humans. The prompts had to be carefully designed and unbiased, for example, by randomising the order of answers.

The main practical advantages of using LLMs as judges are that once a judge model and a clear evaluation rubric are set up, large batches of model outputs can be assessed automatically making the approach highly scalable and drastically reducing human annotation time compared to traditional manual evaluation. Moreover, LLM judges provide semantically rich feedback in natural language and can reason about aspects such as fluency, logical coherence and creativity. Traditional automatic metrics like BLEU and ROUGE are insufficient for open-ended NLG tasks because they often fail to capture the qualities that human annotators care about (Li et al., 2024). Finally, the framework is flexible. The same judge model can be reused with different prompts and rubrics to evaluate different criteria.

Meta evaluations on LLM-as-a-judge highlight both its promises and limits. Chiang and Lee (2023) shows that seemingly small design choices such as requiring explanations, using chain och thought or changing answer format can significantly affect how well LLM ratings correlate with human judgment. JudgeBench by Tan et al. (2025) finds that even string models perform only slightly above chance when judging difficult knowledge, reasoning, maths and coding tasks with clear ground truth. It indicates that reliably evaluating hard problems remains challenging. Bavaresco et al. (2025) shows that LLM judges can approximate human raters on some text quality and safety datasets, but that performance is highly domain- and rubric dependent, which means that human checks are still needed. Finally, Dorner et al. (2025) results show that when the evaluated model is stronger than the judge, LLM-as-a-judge can at best reduce the need for ground truth labels by about a factor of two. This suggests that there are benefits, but they are limited.

2.5.3 Using LLMs for benchmarking

Recent work has popularised the LLM-as-a-judge paradigm, where a strong reference model is prompted to rate the outputs of other systems instead of relying solely on human annotators. This paper by Zhu et al. (2025) show that when the model GPT-4 was used as a judge on

MT-Bench and Chatbot Arena, it achieves 80% agreement with human ratings, suggesting that LLM-based judges can provide a scalable proxy for costly human evaluation. Subsequent evaluations provide a more nuanced picture, showing that LLM judges align well with humans under constrained conditions but can become unreliable for more open-ended or cross-lingual tasks (Doostmohammadi et al., 2024). They showed that the validity of automatic judges is context-dependent. This survey of LLMs-as-judges by Li et al. (2024) highlights their advantages, such as scalability, flexibility and ability to handle tasks without a clear reference answer, as well as their limitations in handling position order and verbosity, biases and the risk of models preferentially scoring outputs that resemble their own style.

2.5.4 Benchmarking of Agents

Before MCP, several benchmarks already targeted tool-using agents. ToolLLM by Qin et al. (2023) builds a large corpus of roughly 16,000 real-world APIs and associated tasks to study tool selection and parameter grounding. AgentBench evaluates LLMs as agents across eight interactive environments spanning operating systems, databases, games and web tasks. Liu et al. (2025a) showed that there are substantial differences between API-based advanced models and smaller open source models in long-term reasoning and decision-making. WebArena provides a representative web environment with full browser interaction for autonomous agents, emphasising multistep web tasks (Zhou et al., 2024). τ-Bench goes further by modelling conversations between an agent that has domain-specific API tools and language models simulating a user (Yao et al., 2024). Where they introduced the metric pass^k, which measures robustness over repeated runs against the same scenario.

2.5.5 Benchmarking of Agents Using MCP

A more niche line of work involves evaluating agents directly in MCP integration. MCP-Universe constructs tasks across multiple MCP servers and domains, exposing how models fail on long, dynamic workflows that require chaining tools across servers (Luo et al., 2025). MCP-Radar defines 507 tasks over a diverse set of servers and measures answer correctness, accuracy and efficiency (Gao et al., 2025b). MCPToolBench++ scales further by mining thousands of marketplace MCP servers and automatically generating single- and multistep tool-call queries over 40 categories, enabling analysis on schema understanding, parameter reasoning and errors in MCP tasks (Fan et al., 2025). LiveMCPBench assembles 70 live MCP servers and 527 tools into 95 everyday tasks. They combine it with an LLM-as-a-judge framework (Mo et al., 2025). MCPEval automates both task generation and evaluation through LLM-as-a-judge, but it targets just five MCP servers (Liu et al., 2025b). A benchmark that adds a security perspective is MCPSecBench (Yang et al., 2025). It is done by constructing MCP tasks that probe prompt injection, unsafe tool invocation and other security risks in MCP-based agents. Finally, MCP-Bench by Wang et al. (2025) and MCPMark by Wu et al. (2025) design workflows that more closely reflect real MCP usage. MCP-Bench connects agents to dozens of live MCP servers and 250 tools to test cross-server planning, tool retrieval from fuzzy instructions and long multistep tasks. MCPMark, on the other hand, focuses on stress testing CRUD-heavy workflows in isolated environments. Both apply LLM-as-a-judge to check for correctness.

2.5.6 MCP-Bench

This thesis adopts MCP-Bench by Wang et al. (2025) as the primary evaluation framework, extending it for the energy-network context. MCP-Bench provides a standardised protocol-aware benchmarking system specifically designed to evaluate LLM agents interacting with real MCP servers.

MCP-Bench is a benchmark designed to evaluate tool-using LLM agents in realistic MCP ecosystems. Requirements for solving a user request are not only to use correct tool calls but also robust multistep reasoning, cross-tool coordination and evidence-grounded synthesis. Conceptually, MCP-Bench can be understood as a pipeline with three main blocks. The first one is task construction from real tool dependencies, the second is an ecosystem of MCP-servers and tools, and the third is the evaluation that combines deterministic checks with rubric-based judging. In more detail, the framework's three main operational blocks:

- *Task Construction*: Tasks are generated to reflect realistic multistep workflows. Each task specifies a goal that requires coordinated tool use, either within a single server or across multiple servers. Tasks are presented in two forms, a detailed specification used only for evaluation reference, and a fuzzy natural-language variant given to the agent, which omits explicit tool names and requires the agent to infer appropriate tools from context.

- *Execution Pipeline*: The framework connects agents to MCP servers and executes tasks through multi-turn tool invocations. Each task proceeds through an iterative cycle where the agent (1) receives a natural language instruction, (2) plans which tools to invoke, (3) executes tool calls through MCP servers, (4) observes structured outputs, and (5) synthesises a final answer. This cycle continues for up to 20 rounds or until the agent signals completion.

- *Evaluation Layer*: Task performance is assessed through a two-tier system. Rule-based metrics provide deterministic checks, aimed at capturing whether agents can correctly interact with MCP interfaces. Complementing this, an LLM-as-a-Judge component evaluates more complex reasoning through structured rubrics. All metrics are introduced in section 2.6.

Agents in the MCP-Bench setup are connected to a set of production-grade MCP servers whose tools are intentionally complementary. This enables natural dependency chains inside a server and more complex workflows across servers. The benchmark spans 28 MCP servers and 250 tools across multiple domains, such as finance, scientific computing and NASA data.

Tasks are generated automatically by first identifying dependency chains based on the tool's input and output and then turning these dependency patterns into natural language task descriptions. This produces tasks that are structurally grounded in what the tools can actually do while still reflecting realistic goals. A key design choice is that each task is also rewritten into a fuzzy variant. The fuzzy description preserves the core objective but removes explicit tool names and step-by-step guidance. This forces the agent to infer relevant tools under less specified instructions. To further stress test tool selection, MCP-Bench applies distraction servers to each task, which increases the available tool set by over 100 tools and creates a noisy retrieval environment where plausible but incorrect tools exist. Tasks cover

both 1- and 2-server settings and can require extended execution and cross-domain orchestration.

The evaluation agents in MCP-Bench use a hybrid approach. First, it applies rule-based metrics to capture tool use robustness. Second, it applies an LLM-as-a-judge framework to assess more complex qualities that are hard to fully capture with deterministic checks alone. The judge is given the fuzzy task (seen by the agent), a concrete task description (not seen by the agent) and a dependency analysis (not shown to the agent), along with an execution trace and a final answer. It then scores performance along three main dimensions, task completion, tool usage and planning effectiveness. The scores are based on sub-dimensions under these metrics. Each sub dimension is scored on a 1-10 scale and then normalised to 0-1 for benchmarking. Because LLM judges can be sensitive to prompt structure, MCP-Bench additionally applies shuffling of the rubrics and score averaging across multiple runs to improve stability and reduce the effects of order.

2.5.7 Test Standards and Reproducible Evaluation

To ensure consistency and comparability in system evaluation, it is common to use established evaluation standards, such as gold and silver standards, which provide reference points for assessing system performance (Oellrich et al., 2015)(Damonte and Cohen, 2018). In this work, we define our own gold and silver standards to ensure consistency with the specific goals of our MCP evaluation. **Gold standards** are manually validated, precise annotations used as the ground truth for the final evaluation. **Silver standards** are labels produced by automated methods or weak supervision. Gold standards are costly to produce and, therefore, typically limited in scale. Silver standards can enable training on a larger scale, but contain noise and systematic errors. Clear standards reduce ambiguity when interpreting system output. They provide a reproducible basis for comparison.

Fuzzing is an automated software testing technique. It generates large numbers of systematically mutated inputs and executes them on a target program to uncover faults and vulnerabilities. The approach relies on clear, reproducible procedures for generating test inputs and evaluating system behaviour (Schloegel et al., 2024). This has made fuzzing one of the most successful strategies in modern software testing. This thesis adopts similar principles. We use reproducible gold-standard answers and deterministic tool outputs. This ensures that the evaluation of MCP-based LLM behaviour is both clear and comparable.

2.6 Evaluation Metrics

Some prior studies on MCP evaluate system outputs using complementary criteria **Correctness** and **Relevance**. This thesis adopts metrics that quantify these criteria for the same reasons as Wang et al. (2025) and Liu et al. (2025b), they are conceptually simple, easily interpretable, and directly align with the goals of tool-augmented reasoning. Also, they align with existing benchmarks, improving comparability and making results interpretable relative to established work.

Rule Based Metrics

The rule-based metrics that measure these criteria include Tool name validity rate, Input schema compliance rate and Execution success rate. Designed to measure tool usage robustness and execution reliability:

Let:

- R denote a rate metric,
- T_{valid} be the number of valid tool invocations,
- T_{total} be the total number of tool invocations,
- T_{schema} be the number of valid tool invocations whose parameters comply with the expected input schema, and
- T_{success} be the number of tool invocations that execute successfully without runtime errors,
- **Tool name validity rate** Measures the proportion of tool invocations that reference valid tools within the available set. Capturing the model's ability to select appropriate tools.

$$R_{\text{valid}} = \frac{T_{\text{valid}}}{T_{\text{total}}}$$

- **Input schema compliance** Evaluates whether each valid tool invocation complies with the expected input schema.

$$R_{\text{schema}} = \frac{T_{\text{schema}}}{T_{\text{valid}}}$$

- **Execution success rate** Calculates the rate of tool invocations successfully executed without runtime errors.

$$R_{\text{success}} = \frac{T_{\text{success}}}{T_{\text{total}}}$$

Together, these rule-based metrics capture protocol-level correctness independently of task semantics. **Tool name validity** and **input schema compliance** measure whether a model can correctly discover and parameterise tools exposed through MCP. **Execution success rate** complements this by capturing whether syntactically valid tool calls also execute reliably in practice.

Judge Based Metrics

To evaluate higher-level strategic behaviour, beyond raw execution correctness, we additionally adopt the LLM-as-a-Judge evaluation framework introduced in recent MCP benchmarking work by Wang et al. (2025) and Liu et al. (2025b). The judge is explicitly instructed to be strict, base scoring on evidence and using a strict definition of what counts as correct execution. The scores are assigned 1-10 based on how much of the task was successfully completed. For example, for any given judge metric, the judge is informed that a score of 7–8 corresponds to about 70–80% of the requirements being perfectly met. Scoring was based on observable evidence from the task description, execution trace, final answer, and available tools.

The judge model is prompted to score agent performance across the three axes **Task Completion**, **Tool Usage** and **Planning Effectiveness**. Each axis is decomposed into subdimensions designed to capture distinct aspects of agent performance:

- **Task Fulfillment** Assesses how completely and accurately the agent satisfied the objectives stated in the task description. Evaluation refers to specific required elements and estimates the proportion of the task that was fully achieved.
- **Information Grounding** Measures to which degree the outputs are grounded in verifiable tool results, rather than unsupported assumptions or hallucinated facts.
- **Tool Appropriateness** Evaluates whether the tools selected for each subtask were matched to the task requirements, including whether alternative tools would have been more suitable.
- **Parameter Accuracy** Examines the correctness and completeness of parameters used in tool calls, noting missing required fields, malformed arguments, or values inconsistent with the task specification.
- **Dependency Awareness** Assess how well the agent recognises and respects task dependencies. Referencing the dependency analysis section.
- **Parallelism and Efficiency** Evaluates whether the agent executed steps efficiently, leveraging opportunities for parallelism and avoiding unnecessary actions. Also, drawing from the dependency analysis to determine where parallel execution was logically possible.
- **Expected-vs-Generated Correctness** Compares the agent's final solution against the expected answer, focusing strictly on factual or structural correctness. Stylistic variation, such as differences in phrasing or length, is not penalised.

The purpose of the judge-based metrics is to capture end-to-end quality. **Task Fulfillment**, **Information Grounding** and **Expected-vs-Generated Correctness** measure whether tool use actually produces correct answers that are supported by tool outputs. **Tool Appropriateness** and **Parameter Accuracy** check whether the agent chose suitable tools and used correct, complete parameters, not just valid calls. **Dependency Awareness** and **Parallelism and Efficiency** capture how well the agent handles queries that require multiple steps and multiple servers, including avoiding redundant steps that increase rounds, tokens, and runtime.

Aggregate Metrics

In addition to the individual metrics above, we report aggregate scores in overview tables to make cross-model comparisons easier. All aggregate scores are computed as simple means of their underlying components. We intentionally keep these aggregates consistent with MCP-Bench and do not include the additional correctness metric in the Task Completion aggregate to preserve comparability.

- **Schema understanding score** is the mean of the three rule-based protocol metrics: ToolNameValidity, SchemaCompliance and ExecutionSuccess.
- **Task completion score** is the mean of the two task-completion sub-dimensions: TaskFulfillment and InformationGrounding.
- **Tool usage score** is the mean of the two tool-usage sub-dimensions: ToolAppropriateness and ParameterAccuracy
- **Planning effectiveness score** is the mean of the two planning sub-dimensions: DependencyAwareness and ParallelismEfficiency

Each sub-dimension is scored on a 1-10 scale. Scores are averaged per axis and normalised to a [0,1] range for benchmarking. To reduce sensitivity to the ordering of evaluation criteria, we apply prompt shuffling, generating five independent evaluations with permuted orderings and averaging their results. Multiple passes and randomised scoring significantly reduce evaluation variance and improve reliability, as evidenced in prior work mentioned above Wang et al. (2025).

Chapter 3
Methodology

3.1 Process

This chapter describes our design, implementation, and evaluation approach of an MCP-based enterprise question-answering system.

The approach began by identifying E.ON's enterprise requirements through analysis of the operational data landscape and evaluation of language models with respect to capabilities, costs, and deployment constraints. These requirements were then realised in a functional prototype in which modular MCP servers expose enterprise data through standardised tool interfaces. Finally, the system performance is evaluated using the MCP-Bench framework across 1- and 2-server configurations, combining rule-based metrics, LLM-judge scoring, and operational analysis to identify performance patterns and deployment implications.

3.1.1 Methodological Process

Figure 3.1 provides an overview of the complete methodological process, structured into three major blocks:

Identify Problem and Data Sources: This phase establishes the study foundation by gathering E.ON's enterprise needs and identifying appropriate data sources. Activities include surveying E.ON's operational data landscape and selecting public datasets that mirror private data structures. The phase also defines data domains for MCP server exposure and evaluates language models based on capability, cost, and deployment constraints.

Prototyping and Implementation: Here, requirements are translated into a functional MCP-based system. Modular MCP servers are designed to expose enterprise data through standardised tool interfaces. A QA corpus with gold-standard answers is constructed.

Task variants with fuzzy natural-language prompts are generated. The implementation ensures conversational agents interact with data exclusively through MCP protocols.

Evaluation and Analysis: Measuring system performance based on the MCP-Bench framework. Tasks are executed across 1- and 2-server configurations. Models invoke MCP tools to retrieve information and construct responses. Performance is assessed through rule-based metrics, LLM-judge scores, and operational characteristics. Results identify performance patterns, failure modes, and enterprise deployment implications.

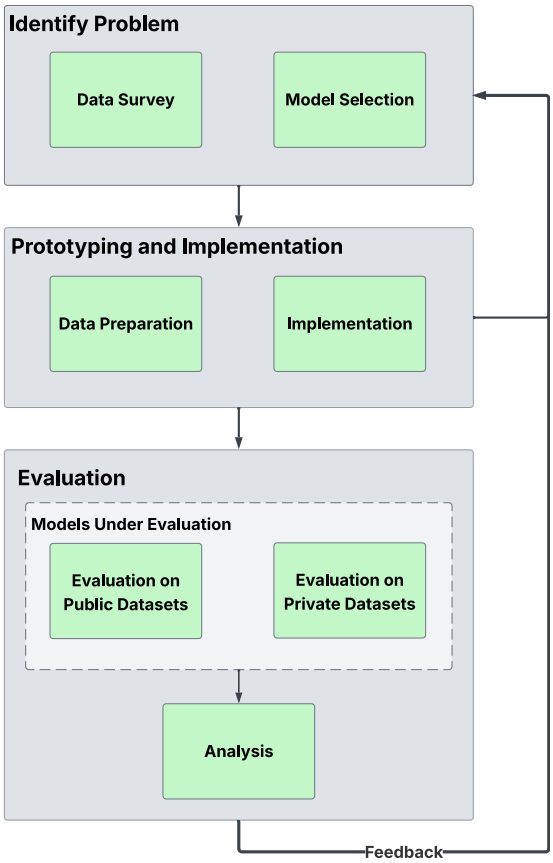

Figure 3.1: Overview of thesis process.

Natural language queries serve as input to conversational agents, which select and invoke MCP tools to retrieve structured information for constructing responses. The same pipeline applies across 1- and 2-server configurations, enabling analysis of model performance under varying degrees of server coordination complexity.

3.2 Data Survey

The first step involved scoping E.ON's data landscape and requirements. This was carried out through briefings with stakeholders across data teams to capture use cases, access needs,

and constraints, as well as a technical review of network operations and system architectures to document what data are collected and how they are managed.

This thesis employs two parallel datasets to study MCP-based conversational agents in a realistic yet reproducible manner. A *Private Dataset* consists of E.ON's internal operational data accessed within a secure environment and cannot be publicly disclosed. To address this constraint and enable reproducibility and comparative analysis, a *Public Dataset* is constructed from UK Power Networks' open data with closely aligned schema characteristics. This design allows correlations between results obtained on the private and public datasets to be systematically examined.

3.2.1 Eliciting Use Cases

We gathered use cases through discussions with domain experts at E.ON, exploratory analysis of existing data platforms, and iterative refinement during early MCP prototyping. The aim was to identify realistic, representative information needs that a conversational agent should support when accessing operational energy data. Through this process, we gained an understanding of the data network infrastructure and how it relates to daily operational work, leading to the three domains described in Section 3.2.2.

Several use cases were explicitly excluded. Write operations and state-changing actions were discarded due to safety and scope constraints. Highly domain-specific analytics and complex cross-system workflows were also not prioritised to maintain comparability and methodological focus. The identified use cases directly informed MCP server separation, tool granularity, and benchmark task design.

3.2.2 Domains

This process resulted in three core use-case categories: metadata discovery (e.g. dataset ownership and descriptions), grid infrastructure queries (e.g. asset attributes, location, and status), and metering-related information retrieval. These use cases reflect common exploratory tasks and can be answered through structured, read-only access to operational datasets. They were also selected to ensure consistent implementation across both the Public and Private Datasets.

The use cases in this thesis require read access to structured operational data and enough schema visibility to support tool-based querying. A key constraint is confidentiality, where E.ON's data cannot be disclosed publicly. Therefore, the study separates it into a Private dataset from E.ON and a public dataset from UK Power Networks open data while keeping the agent workflow comparable.

These examples illustrate the types of questions a conversational agent should be able to answer using the available data:

- **Server - Data Catalog and Metadata**
  Purpose: Exposing descriptions, ownerships, outage reports and equipment metadata.
  Q: who is the owner of {DATASET} and what is the description?
  A: The specified dataset's description is {DESCRIPTION} and its owner is {OWNER}.

- **Server - Geodata and Network Information Data**
  Purpose: Exposing facility locations, status, commission date, types and ids.
  Q: Which substations in {AREA} were installed in 2025, and what is their current status?
  A: Substations in {AREA} installed in 2025 are {STATION_NAME} with status {STATUS} and...

- **Server - Metering and Meter Value Management**
  Purpose: Exposing the process and data flows for metrics like collection errors, missing values and log timestamps.
  Q: How many smart meters have been without measurements for more than 24 hours in the past week?
  A: During the past week, {NUMBER} smart meters experienced measurement gaps exceeding 24 hours.

Across both the Public and Private settings, the underlying datasets are updated over time, but the Public sources typically change sporadically while the Private sources are refreshed more continuously and at higher volume across domains. In the Public metering domain, measurement values follow an append-only pattern, with new rows added at each time step, with timestamps recorded at the date level and down to the second. Confidentiality concerns are strongest for metadata and metering, as these domains can indirectly relate to individuals, while grid infrastructure can still be sensitive if, for example, detailed operational status enables undesired inference about critical assets. To keep the Public and Private settings separated, the study uses the same overall workflow in both cases and limits what each tool returns to only the fields needed for the task. In addition, operational artefacts such as logs and error messages are treated as sensitive, even if they are never shown to the model.

To reduce unnecessary exposure of sensitive information, tool responses are minimal. Each tool specifies exactly which fields are returned, and several tools apply additional constraints, such as limits.

3.2.3 Data Modalities and Structure

All data sources in this study expose structured, tabular records retrieved through query-based APIs. No unstructured text, images, or multimedia content is involved. The evaluation focuses exclusively on operational data stored in relational or tabular formats, where records consist of typed fields with well-defined schemas.

For location information, only simple point-level attributes (e.g., coordinates tied to a substation or asset identifier) are available. The datasets do not include full GIS topology, spatial indices, or network geometry, so agents treat location as an attribute of tabular records rather than a geospatial network model, limiting the spatial reasoning that can be evaluated. This limitation is consistent across both Dremio-backed and Opendatasoft-backed sources and therefore does not affect comparisons.

3.2.4 Data Providers

The initial phase also included a survey of publicly available datasets whose structure resembles the operational data and metadata sources of E.ON. These public datasets were used as concrete examples to characterise the contour of the data and to develop and benchmark methods, thereby supporting generalisability and enabling comparative evaluation.

As outlined in Section 2.3, E.ON's private data platform is accessed via Dremio, which exposes virtualised, strongly typed enterprise schemas through an ANSI-SQL oriented engine with engine-specific extensions, whereas the UK Power Networks Open Data portal runs on Opendatasoft's ODSQL interface, which is lightweight and publication-oriented. The practical consequence is that Dremio supports richer relational operations such as joins across domains, complex predicates, stronger typing and governance, while ODSQL exposes comparatively denormalised tables directed towards filtering and field selection in individually published datasets. In our system, this asymmetry is absorbed at the MCP-server layer. Each backend is wrapped by a dedicated MCP server that issues platform-specific queries and exposes aligned tool semantics. Conceptually similar information is therefore exposed to the agent through aligned schemas, even though the underlying APIs differ.

Access control is primarily enforced through credentials and dataset authorisation rather than through the LLM or the MCP tool layer. The public endpoint can be accessed with an externally obtainable API key, whereas the private endpoint requires access from employment. Furthermore, there are separate approvals for each dataset in Dremio. In practice, this means that possessing an API key is not sufficient for the private setting. Users must also have explicit permissions to the specific datasets they query.

Confidentiality is enforced by ensuring that tokens and API keys are handled server-side and are never exposed to the model. Tool calls do not give the LLM any direct capability to authenticate against the data platforms beyond what the user is permitted to access. In addition, tool outputs are deliberately minimised to only the fields required for each task, and operational records such as logs and error messages are treated as sensitive and are not forwarded to the model.

The platforms also differ in access patterns and result formats. Dremio uses an asynchronous job workflow where queries are submitted and results are later retrieved together with explicit schema metadata, while Opendatasoft exposes synchronous endpoints that return record arrays directly in the response.

3.2.5 Data Retrieval Stack

To retrieve data consistently, a technical stack exposes the datasets through MCP servers. This stack hides the underlying storage systems, letting models interact through standard MCP interfaces instead of direct database access. The data retrieval setup has practical limits. Query timeouts stop long-running operations, specifically, Dremio sets job time limits while Opendatasoft restricts API call frequency. Authentication works differently on each platform. Dremio uses tokens tied to user access rights, Opendatasoft uses API keys per dataset.

Response sizes affect how tools work. Dremio handles large results through job-based retrieval but stays within memory limits, while Opendatasoft caps the number of rows per query. Tools include limit parameters to avoid using too many tokens while still returning

useful data. For time-based metering queries, this meant splitting wide time ranges into focused windows rather than allowing unlimited history access.

The MCP tools are designed as read-only. The tasks are formulated around retrieval, and the server setup does not grant write privileges. However, an important limitation is that the Dremio API accepts arbitrary SQL-statements without server-side restrictions to `SELECT` statements.

3.3 Model Selection

Model selection was guided by four key criteria derived from the enterprise survey and technical requirements:

1. The bilingual operation in Swedish and English reflects E.ON's operational context, in which queries are formulated in Swedish while data schemas and MCP documentation are in English.

2. Conversational agent capabilities with demonstrated tool-use competence, as not all LLMs are trained for agentic behaviour, tool invocation, or structured parameter construction.

3. Architectural and cost diversity to capture deployment trade-offs across high-capacity cloud models, cost-efficient alternatives, and resource-constrained local execution.

4. Reproducibility and comparability with existing MCP research, prioritising models evaluated in peer-reviewed MCP benchmarking work.

3.3.1 Selection

The initial selection pool was constrained by E.ON's licensed infrastructure and European regulatory requirements. Within these boundaries, we identified models across three deployment tiers. The tiers are premium cloud for foundation models, cost-efficient cloud for smaller foundation models, and local on-device inference for open-weight models. From each tier, we selected the model with the strongest documented performance for MCP-based tool use, prioritising those evaluated in Wang et al. (2025)'s MCP-Bench study to enable direct qualitative comparison. Resulting in three candidates representing distinct points across the performance-cost-infrastructure trade-offs.

All three models have been evaluated on MMLU-Pro Wang et al. (2024b); Artificial Analysis (2025), a respected benchmark from NeurIPS 2024 that has been widely adopted (>1,000 citations) for its increased difficulty and reduced prompt sensitivity. The consistent performance hierarchy provides context for interpreting the results in Chapter 4. The inclusion of Sonnet 4 and GPT-4o-mini in (Wang et al., 2025)'s evaluation enables qualitative comparison between their multi-domain study and our energy-network context, strengthening methodological validity.

3.3.2 Claude Sonnet 4

This model represents the high-capacity cloud tier. Artificial Analysis (2025) consistently positions Sonnet 4 at or near the top across MMLU-Pro, GPQA Diamond, LiveCodeBench, and IFBench. Its context window exceeding 200,000 tokens supports complex multi-step MCP workflows. Wang et al. (2025) demonstrated strong tool-use competence and cross-server orchestration in MCP-Bench, directly aligning with this study's requirements. API costs were approximately 30 SEK per million input tokens and 150 SEK per million output tokens (Anthropic, 2025d), with all inference executed in E.ON's EU-North infrastructure.

3.3.3 GPT-4o-mini

GPT-4o-mini provides a cost-efficient cloud alternative with a distinct architectural lineage. Artificial Analysis shows competitive performance on MMLU-Pro while maintaining costs of approximately 1.50 SEK per million input tokens and 6 SEK per million output tokens (OpenAI, 2025), roughly 20 times reduction compared to Sonnet 4. The model demonstrated reliable protocol adherence and task completion on MCP-Bench, though with observable differences in planning efficiency relative to stronger models.

3.3.4 Phi-4-mini-instruct

Phi-4-mini-instruct enables evaluation of on-device inference under realistic hardware constraints. The Q4_K_M quantisation variant (3 billion parameters, 2.49 GB) was selected as the most downloaded Hugging Face model supporting Swedish that could run on E.ON's standard enterprise laptops rather than on a dedicated AI workstations. The laptop is an HP EliteBook 835 G9 with an AMD Ryzen 5 PRO processor, integrated graphics, 16 GB RAM and 256 GB SSD 13.3 with an approximate cost of 12 000 SEK Elgiganten (2025). Critically, its 131,072-token context window proved essential for our MCP task execution. Alternative models like Llama-3-8B-instruct (maximum 8,192 tokens) were eliminated due to insufficient context for multi-step tool interactions and accumulated server responses. While Artificial Analysis shows Phi models achieve notable MMLU-Pro results despite compact size, absolute scores remain well below foundation models. Its inclusion addresses whether local, cost-free inference can achieve acceptable MCP performance within typical hardware budgets.

3.4 Prototyping

During prototyping, we iterated on tool designs until the MCP servers supported the required queries. This process established how conversational agents would interact with operational data through standardised MCP interfaces.

3.4.1 Tool Exploration

Tool exploration began by mapping representative user queries to underlying data operations. For each data domain identified in the survey (metadata, network information, metering),

we documented common access patterns such as identifier-based lookups, temporal range queries, threshold filtering, and attribute retrieval. These patterns revealed natural groupings where similar queries could be served by a single parameterised tool rather than multiple specialised endpoints. The exploration process followed an iterative cycle. We formulated candidate queries in natural language, traced their execution through the underlying data platforms (Dremio SQL for private data, Opendatasoft REST for public data), and identified which query components could be abstracted as tool parameters. For example, metadata queries consistently required dataset identifiers but varied in which specific metadata field was requested. This pattern motivated creating separate tools per metadata field rather than one generic metadata retrieval tool, as the former yields clearer tool descriptions and more reliable parameter construction by the LLM.

MCP's tool abstraction maps LLM queries to server operations tool discovery. Here, the server exposes available tools with their schemas, tool invocation, where the LLM constructs a structured request containing the tool name and parameters as JSON, and result interpretation, where the server's response is injected into the conversation context as evidence. Our exploration identified that reliable tool selection depends critically on tool naming and description clarity. Tools named after specific operations (e.g. `get_substation_voltage_by_alias`) proved easier for models to select correctly than generic names (e.g. `query_network_data`), even when descriptions were detailed.

3.4.2 Task Creation

Task creation established a question–answer corpus grounded in operational energy data. Each task pairs a natural-language question in Swedish with a gold-standard answer and explicit dependency analysis specifying expected tool chains and outputs. This structure enables verification of both final answer correctness and reasoning alignment with intended workflows. To capture realistic query variation, each logical task was expressed through ten paraphrased fuzzy descriptions, being natural-language variants that preserve intent and tool requirements while varying surface form and complexity. During execution, agents receive only one variant, reducing sensitivity to specific wording while maintaining coverage across formulation styles. All answers were manually authored to avoid model-generated biases. Sample tasks appear in Appendices A and B.

3.5 Implementation

This section describes how the conceptual MCP-based design is realised as a concrete software system. Centring on a set of modular MCP servers that expose read-only access to E.ON's private data and UK Power Networks' public datasets through aligned tool interfaces, allowing models to retrieve operational information without direct database connectivity.

3.5.1 System Architecture

Modular MCP servers were implemented to expose controlled, read-only access to both E.ON's private data and publicly available open-source datasets from UK Power Networks. Each data domain is exposed by a dedicated MCP server. In total, we implemented six

servers, three for UK Power Networks and three for E.ON. For each dataset, we provide a **Metadata server**, a **Network Information System server**, and a **Metering server**. The UK Power Networks and E.ON servers are designed to be analogous, meaning each server pair targets the same data domain and exposes an aligned set of MCP tools with comparable input/output schemas, even though the underlying backends differ (ODSQL/Opendatasoft for UK Power Networks versus Dremio SQL API for E.ON). This mirroring is intentional so that behavioural conclusions observed on the reportable public setup can be validated against the private setup under comparable server composition.

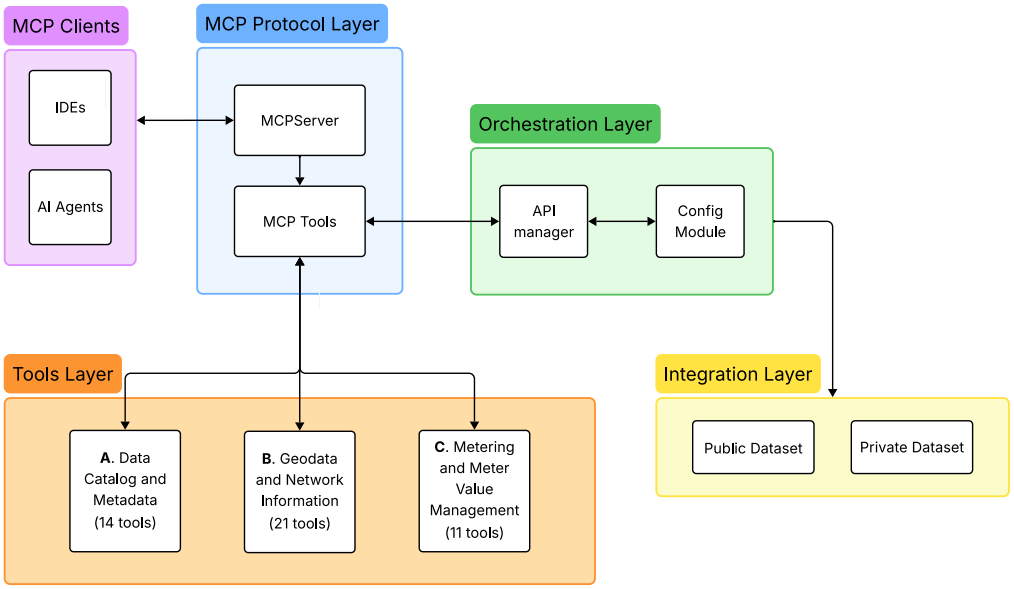

Figure 3.2: System Architecture.

Figure 3.2 illustrates the overall system architecture, which comprises five distinct layers serving complementary roles in the system:

MCP Protocol Layer: Handles all standardised communication with MCP clients, which includes Integrated Development Environments (IDEs) such as VS Code and AI agents like Claude or ChatGPT. Managing message routing, tool discovery, and transport abstraction through the SDKs MCPServer framework.

MCP Client: When the model decides to invoke a tool, it returns a structured request containing the tool name and parameters through this component.

Orchestration Layer: Transforms protocol-level requests into domain-specific operations, intercepting and routing requests to appropriate MCP servers while providing input validation, error handling, and response formatting.

Tool Layer: Encapsulates domain-specific functionality across three energy-network domains (metadata, network information, and metering), with MCPServer exposing tools while maintaining complete independence from protocol concerns.

Integration Layer: Manages all interactions with the underlying data platforms, where the API manager constructs platform-specific queries and executes data retrieval operations.

As described in section 2.4.3, retrieved data flows back through these layers in reverse, formatted as structured MCP tool responses, passed through the Protocol Layer to the Client, and injected into the conversation context as tool response messages. The model then reasons over this result and continues generating its final answer.

3.5.2 Runtime Environments

The model runtime environment depends on the chosen model and infrastructure provider. Claude Sonnet 4 was accessed through Databricks Serving Endpoint on 2025-11-29, which provided access to Databricks' Foundation Model APIs. GPT-4o-mini was accessed through AI-Gateway on 2025-12-09, which routes requests to the Azure OpenAI Endpoint. The local Phi-4-mini-instruct model (described in Section 3.3) was served via LM Studio version 0.3.32 (Build 2) for on-device inference. Task-level evaluation was performed by OpenAI's o4-mini model, accessed via the same AI-Gateway infrastructure as GPT-4o-mini, following the LLM-as-a-judge configuration established in MCP-Bench. E.ON's operational data was accessed through Dremio version 26.x, described in the API Reference section (Dremio, 2025a). The UK Power Networks public data was accessed via the Opendatasoft API version 2.1 UK Power Networks Open Data (2025). Despite these different infrastructure paths, all models interact with MCP servers using the same protocol abstraction, which isolates model-specific execution details from the rest of the system.

For cloud models, the runtime manages conversation history, API authentication, rate limiting, and response streaming. Each infrastructure provider implements their own conversation management and security controls, but both expose tool-calling capabilities through structured function definitions. Our system translates MCP tool schemas into the provider-specific format required by each API. For the local model, LM Studio provides the inference runtime and conversation management on commodity hardware. The same MCP tool schemas are exposed, but the model's ability to reliably parse and generate tool calls depends on its training and capacity. Our implementation ensures that tool definitions, execution logic, and result formatting remain consistent across all three models, meaning any performance differences reflect model capabilities rather than environmental variations.

The core architectural contribution of this work lies in the design and implementation of the MCP servers and their orchestration layer. The MCP, model runtimes, and base evaluation framework are existing components. Our work demonstrates how these components can be integrated into a coherent system for enterprise data access while maintaining methodological precision and cross-model comparability.

3.5.3 Execution

The baseline process begins when an MCP server receives a tool invocation containing the tool name and validated parameters. The server constructs a platform-specific query, either as SQL for Dremio or as REST query parameters for Opendatasoft, and executes it against the underlying data platform. Raw results are received as JSON responses in the platform's native structure. These responses are then serialised to string representations and packaged as MCP tool result messages, which are injected into the conversation context where the LLM can interpret them as evidence for constructing its final answer.

The MCP servers act as a strict interface layer between the LLM host/client and the underlying data platforms. We implement each server using the MCP Python SDK(section 2.4.7), standardising MCP semantics and keeping tool behaviour predictable within the MCP framework. Li et al. (2025) also used the official Python SDK for their EnergyPlus-based MCP server because it standardises communication between the host application and MCP servers, reduces boilerplate, and ensures strict compatibility with the MCP specification. This work adopts it for the same reasons, supporting a scalable, modular architecture where new data sources can be added through additional tool definitions.

3.5.4 Preprocessing and Normalisation

The preprocessing stage was intentionally kept minimal to evaluate model behaviour under conditions that closely resemble real MCP usage. By exposing server outputs with their original field names, value formats, and schema structure, the evaluation captures how well models can interpret and use standardised tool responses without relying on additional task-specific data cleaning or restructuring. In practice, server responses are made available to the model as structured JSON through MCP tool calls. This approach reflects a common design in tool-augmented LLM systems, where tools return structured JSON outputs that are passed to the model for interpretation. Kate et al. (2025) demonstrates that LLMs struggle with field extraction from nested structures and irrelevant information filtering in verbose tool outputs, particularly when critical values are embedded within complex JSON hierarchies. The primary benefit of our approach is transparency and reproducibility, since the model receives the same structured content that an MCP server exposes. The main tradeoff is that raw outputs may contain irrelevant fields or verbose structures, which can increase token usage and place greater demands on the model's ability to extract relevant information. In our implementation, each MCP tool returns only the relevant fields, which negates this potential liability.

Across both sources, all records retrieved via MCP servers are serialised before being injected into the LLM context as a string representation of a nested dictionary with a flat, tabular structure. Each table row becomes a dictionary of column–value pairs and collections of rows are serialised as nested key–value structures. This design follows MCP's JSON-RPC foundation, in which tools and resources are exposed as structured JSON objects with schema-defined fields, transported as text over stdio or HTTP/SSE (Ray, 2025)(Wang et al., 2025). By presenting data as stringified nested dictionaries that preserve the tabular schema, the agents reason over machine-interpretable payloads aligned with MCP's tool and resource abstractions, rather than over ad-hoc textual descriptions of tables.

An alternative preprocessing strategy would involve adding intermediate summaries before passing data to the model. This approach could improve efficiency and reduce context load for large responses, but it also introduces additional design choices that may hide or obscure protocol-level behaviour. For this reason, the evaluation prioritises raw, schema-preserving outputs over additional transformation. We intentionally preserve the original response structure from each platform rather than imposing artificial normalisation. Dremio responses retain their `{rowCount, schema, rows}` envelope, while Opendatasoft responses remain as direct record arrays. This asymmetry is absorbed at the interpretation layer, where the LLM must parse whichever structure the tool returns. In practice, both structures represent tabular data with field-value mappings, and models successfully extract information

from both formats without requiring unified schemas. This approach aligns with MCP's design philosophy, where servers expose heterogeneous resources, and the client layer is responsible for interpretation.

3.5.5 API Manager

To reduce duplication and maintain consistency across both datasets, the servers for public and private data share an API manager that centralises responsibilities such as authentication, request construction (ODSQL vs. Dremio SQL), limits, retries, and structured error handling. The manager functions use asynchronous I/O because tool calls mostly involve waiting for responses from external APIs. While concurrency is not a primary performance bottleneck in our benchmark runs, which are largely sequential, this design follows common MCP server patterns from Anthropic (2025b) where tools wrap external HTTP calls and keep the implementation scalable if tool calls are later parallelised. According to the MCP specification, clients can send multiple requests simultaneously while still receiving predictable responses, which fits well with a non-blocking server design (Anthropic, 2025a). The MCP-Bench results support this, as the tasks often require using tools multiple times, which reflects how MCPs are used in practice. Our implementation is designed to be simple enough to be easily replicated and to work with the modular structure of the system, as described by (Li et al., 2025).

3.5.6 Scope

As stated in section 2.4.4, MCP defines three server-side primitives: *tools*, *resources*, and *prompts*. In our implementation, we expose only tools. This is a deliberate scope choice driven by the requirements identified in the earlier data survey. The target use case is to retrieve structured data from operational datasets via controlled API calls, which is fully supported by tool invocations that return schema-defined JSON. Resources are primarily intended for providing contextual data objects, and prompts act as reusable interaction templates. Neither is required to execute or evaluate our task set. This keeps servers focused on deterministic I/O and simplifies reproducibility.

3.5.7 Server Separation

In MCP, each server publishes a set of tools that the agent can discover and call. In our implementation, we therefore treat server boundaries as a deliberate design choice and map each data domain to a dedicated MCP server. This separation keeps tool inventories domain-coherent and mirrors how the underlying datasets are organised, while also reflecting a realistic enterprise setting where different operational domains are typically owned and maintained independently.

This modular server design reflects how E.ON's datasets are organised by operational domain and ownership, allowing each server to be developed, tested, and deployed independently without affecting others. Schema changes or implementation errors remain contained within a single server rather than destabilising the entire system. While proper test coverage

reduces deployment risks, domain-isolated servers provide additional resilience during iterative development cycles common in enterprise MCP implementations. In practice, this separation simplified cache management and troubleshooting during evaluation. When schema changes occurred in one domain, only the affected server's cache required clearing rather than invalidating the entire system. It also supported parallel implementation across both the private and public datasets without coordination overhead.

3.5.8 Tool Inventory

Each MCP server exposes a domain-specific set of tools that retrieve structured data from the underlying datasets. All tools accept typed parameters, construct appropriate backend queries, and return JSON-formatted responses. Table 3.1 summarises the tool count and primary query patterns for each server.

Server	Domain	Tool Count	Primary Query Pattern
Private			
A. Metadata	Dataset catalog	14	Lookup by identifier/title
B. Network	Grid infrastructure	24	Lookup by site name/alias
C. Metering	Operational data	17	Timestamp/threshold queries
Public			
A. Metadata	Dataset catalog	14	Lookup by identifier/title
B. Network	Grid infrastructure	21	Lookup by site name/alias
C. Metering	Operational data	11	Timestamp/threshold queries

Table 3.1: Tool inventory across MCP servers.

MCPServer mandates structured docstrings for all tools, detailing parameter types, return fields, and data constraints (Gullí, 2025). This stipulation aligns with established conventions in Python API development, where structured documentation facilitates automated schema generation and validation (Li et al., 2025). Beyond their documentation function, the docstring descriptions are crucial, as the framework utilises them to parse and generate the JSON schemas that are exposed. Without properly structured docstrings or a description argument, models must infer parameter requirements from function names alone, substantially reducing reliability in tool selection and parameter construction. This contract-driven approach is central to MCPServer's design, where developer-written type hints and descriptions become the interface contract between tools and agents.

Due to confidentiality requirements, the examples below describe tools from the public UK Power Networks dataset only. The private dataset follows analogous structures with comparable tools across the same three domains.

A. Metadata: Follows a key-value retrieval pattern where tools accept a single string identifier and return specific metadata fields. For example, one tool takes a dataset identifier as input and returns the creator organisation name as output. Another accepts a dataset title and returns its corresponding identifier. This pattern reflects Dublin Core metadata standards, where each attribute is accessed independently through dedicated retrieval functions.

B. Network Information: Uses a hierarchical lookup pattern where tools accept location identifiers or aliases and return infrastructure attributes. A substation alias as input yields voltage level, design type, and ownership details as output. Site names map to voltage measurements, resistance values, and operational classification. Temporal queries accept year and month parameters to filter infrastructure by installation date, returning lists of site names with their corresponding attributes. This design reflects the hierarchical nature of electrical grid infrastructure, where different network layers require different identifier types.

C. Metering: Use a temporal query pattern where tools primarily accept timestamp parameters and return time-series measurements. A datetime input yields the most recent storage percentage and measurement timestamp as output. Incident reference numbers map to creation dates, priority levels, power cut types, and customer-facing descriptions. Threshold-based queries accept percentage values and return lists of timestamps where measurements exceed the specified threshold. This pattern supports both point-in-time lookups and range-based filtering common in operational metering scenarios.

These examples illustrate how tool design reflects underlying data access patterns. Metadata tools provide atomic field retrieval, network tools enable hierarchical navigation across infrastructure layers, and metering tools support temporal and threshold-based filtering. All tools enforce read-only access and return only explicitly requested fields, limiting query scope to 5-10 records where appropriate to prevent excessive token consumption while maintaining representative result sets.

3.6 Evaluation

The evaluation uses two parallel task sets executed against a mirrored MCP server architecture. Two datasets, E.ON operational data (private) and UK Power Network Open Data (public), were used during evaluation. Both datasets are exposed through matching configurations in each of the 3 MCP servers. The tool interfaces per domain are aligned for each server, enabling direct performance comparison across the enterprise and public contexts. Chapter 4 reports quantitative results from both datasets, presented separately as "Private" (E.ON) and "Public" (UK Power Networks) in tables and figures. Due to confidentiality requirements, implementation details specific to E.ON's MCP servers cannot be disclosed. These details are fully documented for the public dataset, with example tasks provided in Appendix B and tool inventories detailed in Section 3.5.8. Both datasets follow the identical evaluation methodology described below. The overall evaluation architecture is illustrated in Figure 3.3, which shows how tasks flow through the MCP servers, model runtimes, and evaluation components. The figure's large blocks correspond directly to the core components described in the following subsections: **Task Input**, which specifies the construction of benchmark queries and their variants. **Execution Environment**, which defines the MCP-based tool configurations and runtime conditions under which agents operate, **Evaluation & Scoring**, which details the automated and model-based procedures used to judge correctness and robustness. Together, these components form a structured, end-to-end evaluation process.

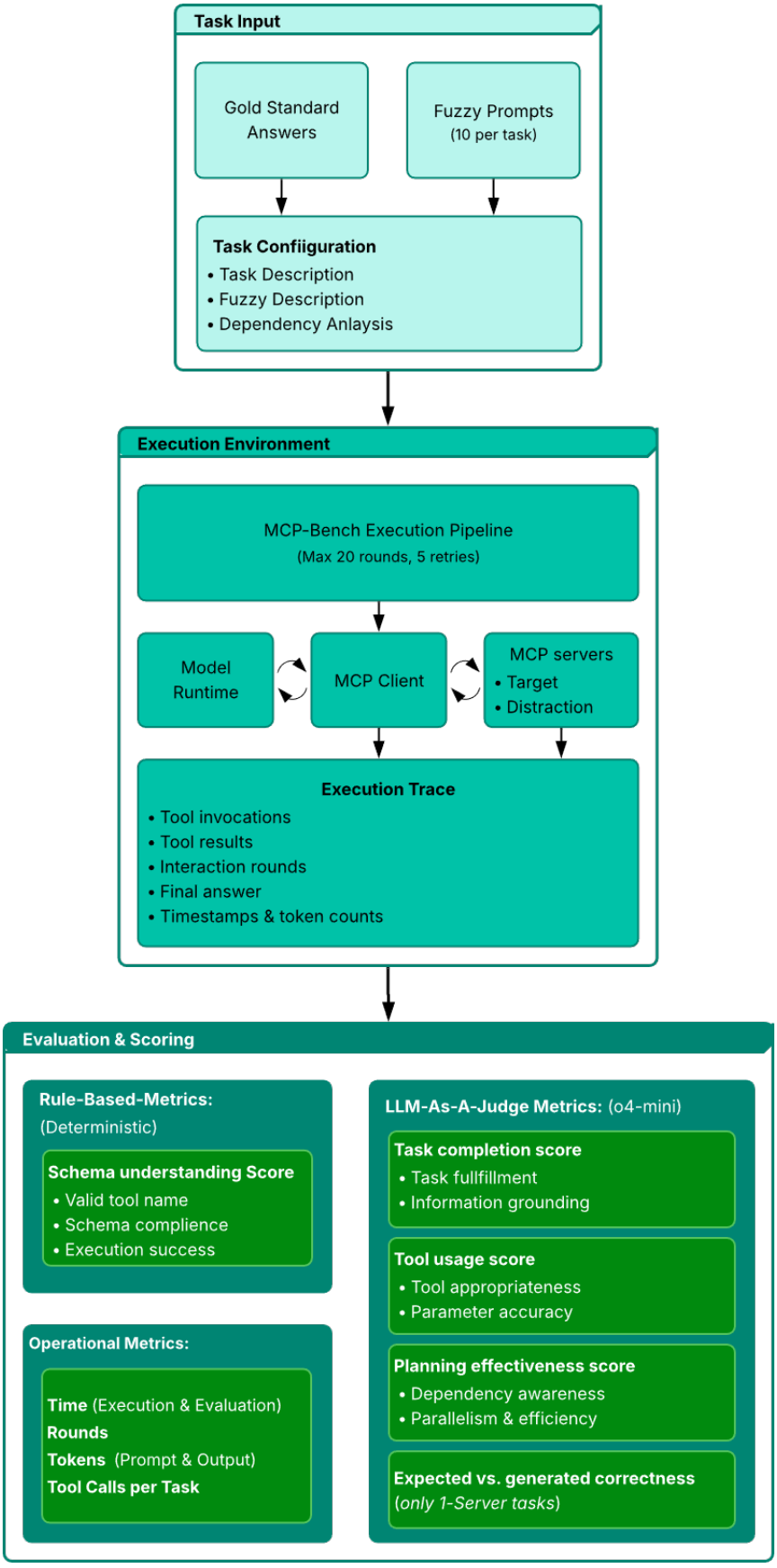

Figure 3.3: System Evaluation.

3.6.1 MCP Benchmark Framework Selection

We adopt MCP-Bench by Wang et al. (2025) as our primary evaluation framework because its architectural components directly address our research objectives. MCP-Bench extends MCPEval by Liu et al. (2025b), a peer-reviewed benchmark published in EMNLP 2025, by adding 2-Server orchestration and fuzzy task descriptions while retaining protocol-aware evaluation foundations. MCP-Bench's design supports our evaluation needs through three key mechanisms. Protocol-level metrics directly assess RQ1 by measuring how reliably models interact with MCP servers under standardised invocation protocols. LLM-as-a-Judge rubrics evaluate higher-order capabilities, including task completion, information grounding, and planning effectiveness, which enable systematic comparison across models (RQ2, RQ3). The multi-dimensional scoring captures both correctness and strategic reasoning quality. Fuzzy task descriptions test whether models can infer appropriate tools from underspecified instructions, reflecting realistic user queries where explicit tool names are absent.

While MCPUniverse by Luo et al. (2025) emphasises execution-based evaluation and raises concerns about judge bias in real-time scenarios, we adopt MCP-Bench's hybrid approach (rule-based + judge-based) and extend it with the explicit expected-answer field for 1-Server tasks. This addition directly addresses factual correctness (RQ1, RQ3) by enabling deterministic verification against gold-standard answers, complementing rubric-based assessment. Our evaluation uses the same base models (Claude Sonnet 4, GPT-4o-mini) and judge model (o4-mini) as Wang et al. (2025), enabling qualitative comparison of model behaviour patterns while accounting for domain-specific differences in our energy-network task distribution. Since Phi-4-mini-instruct was not evaluated in Wang et al. (2025), Liu et al. (2025b), or Luo et al. (2025), direct benchmark comparison is not possible for this model. However, its inclusion addresses RQ2's focus on model suitability across deployment contexts, providing insight into whether resource-constrained local models can handle MCP workflows.

3.6.2 MCP-Bench Extensions

Our evaluation retains MCP-Bench's core architecture, its execution loop, judge-based rubric scoring, and rule-based protocol checks, while extending it in three key ways for our domain-specific requirements. First, we construct energy-network tasks grounded in operational data from our MCP servers covering metadata, grid infrastructure, and metering domains. Second, for 1-Server tasks, we introduce explicit expected-answer fields that enable direct factual correctness verification against gold-standard values, strengthening evaluation beyond rubric-based assessment alone. Third, we enforce determinism constraints on tool outputs by fixing query parameters and validating reproducibility, ensuring that tasks remain stable even as underlying databases grow. These extensions preserve comparability with Wang et al. (2025) while addressing the specific demands of evaluating factual reliability in enterprise data access scenarios. The following subsections detail how we adapted MCP-Bench's task construction methodology, configured execution parameters, and positioned our work relative to related benchmarks.

3.6.3 Task Input

The QA corpus is restricted to the three domains implemented as servers in Section 3.5.8. These three domains correspond to a metadata server, a grid infrastructure, an asset information server and lastly a metering server. These domains reflect common enterprise data categories where conversational access provides value. Tasks were constructed using a hybrid gold/silver standard approach. Gold-standard answers provide precise, manually validated ground truth for final evaluation. Silver-standard intermediate annotations, such as tool selection patterns, were generated systematically and then validated selectively. This combination yields diverse task coverage while maintaining correctness guarantees where they matter most. The corpus comprises two parallel task sets, each containing three servers. Task counts before fuzzy expansion: private (130, 260, 160 tasks for Servers A, B, C), public (140, 230, 110 tasks). Each logical task generates ten *fuzzy descriptions* via OpenAI's GPT-5, chosen for strong multilingual compliance to complex paraphrasing constraints. Because agents see only one variant per run, effective logical task counts are one-tenth of raw totals: private (13, 26, 16), public (14, 23, 11).

Fuzzy descriptions are natural-language task formulations that omit explicit tool names and step-by-step guidance while preserving core objectives. This design, adapted from MCP-Bench, forces agents to infer appropriate tools from underspecified instructions rather than following direct procedural prompts. Multiple tasks require identical tools with different parameters, reflecting realistic patterns where single tools serve multiple query types.

Corpus design guidelines prioritised three criteria. Tasks were selected only if they yielded stable, non-empty results, ensuring reproducibility across runs and datasets. Selected tasks collectively exercised all server tools and primary parameter patterns such as identifier lookups, temporal ranges, and threshold filters. Task formulations mirrored actual user information needs identified during the data survey, validated through stakeholder review for alignment with day-to-day operational and analytical workflows.

Task Selection and Coverage

Task candidates emerged from systematic tool enumeration per server, sampling representative parameter patterns. Retained tasks yield non-empty, stable outputs and collectively span all servers and query types. Prompts follow MCP-Bench's fuzzy-description pattern, detailed in Appendix A, extended with correct answers and correct tools in dependency analysis, plus ten length-varied questions per task. Each task generated three ≈20-word prompts, four ≈40-word prompts, three ≈60-word prompts. Length variation affects phrasing complexity, not core content, providing coverage across query formulation styles without altering solvability. All prompts use Swedish to reflect the deployment context within a Swedish utility. Since underlying schemas are language-independent, Swedish primarily affects model reasoning rather than tool call structure. All generated questions underwent manual review. Acceptance criteria required natural phrasing, no leaked tool names or answers, preserved intent with identical tool workflow, and adherence to target length. Prompts failing any criterion were discarded and regenerated, ensuring the final corpus reads as authentic user requests rather than artificial test cases.

3.6.4 Execution Environment

1-Server tasks are constructed to be solvable using tools from exactly one target server. During evaluation, the agent is still exposed to the other servers as distractions, but the correct solution requires selecting the right tool and parameters within the target server's domain. For this setting, we replace MCP-Bench's task description, which otherwise would function as a reference to the judge, with explicit question-answer tasks. This enables direct expected-vs-generated correctness checks in addition to judge-based scoring, and strengthens factual reliability by verifying that the final answer matches a gold-standard value derived from the underlying data. This ensures that retrieved evidence reflects correct raw data rather than hallucinated information. Because these checks already provide a strong signal for correctness, the broader task description used in MCP-Bench was unnecessary in this mode.

2-Server tasks are designed to require cross-domain coordination, meaning that two tools are called from two different servers. This setting primarily tests planning and orchestration across server boundaries. To remain aligned with MCP-Bench's intent for 2-Server evaluation, we retain their original task descriptions as the judge's reference for success. In total, twelve 2-Server tasks were generated(6 private, 6 public), also using GPT-5, with similar task synthesis used in Wang et al. (2025), seen in Appendix A.

For 2-Server tasks, we retain MCP-Bench's original task description format as the correctness reference rather than defining explicit expected answers. This design choice reflects the increased complexity of coordinating outputs across multiple servers and the challenges of maintaining deterministic constraints across all involved tools. The detailed methodological justification for this asymmetric treatment is provided in section 5.4.3.

Distraction Servers

For 1-Server testing, each task targets exactly one MCP server. 2-Server setup targets two tasks across two servers. Both are executed in an environment that also exposes *distraction servers*. Consequently, 1-Server testing is exposed to two distraction servers while 2-Server testing is exposed to one distraction server. For example, when evaluating 1-Server tasks on the public version of Server A, the agent also has access to the public versions of Servers B and C. The correct tool for the task is always hosted on the target server/s. Tools from distraction servers are never required to solve the task. This design probes the agent's ability to select relevant servers and tools in the presence of plausible but irrelevant alternatives. However, for tests on Phi-4-mini used in a local environment, no distraction servers were exposed due to token usage restrictions. This simplified configuration is documented in table 3.2.

3.6.5 Evaluation & Scoring

We evaluate the three agent models Claude Sonnet 4, GPT-4o-mini, and Phi-4-mini. These models represent different points in the capability-cost trade-off space, with selection criteria detailed in Section 3.3. This choice aligns with Wang et al. (2025), who evaluated the two cloud models in their original MCP-Bench study, enabling direct qualitative comparison of behaviour patterns across our energy-network domain and their multi-domain benchmark.

Task evaluation was performed by o4-mini following the LLM-as-a-judge configuration established in MCP-Bench. They demonstrated that this judge model achieves reliable rubric-

based scoring when combined with prompt shuffling and score averaging. We adopt the same stability measures by having five randomised evaluations per judged metric with shuffled rubric orderings, reducing sensitivity to prompt structure. Following Zheng et al. (2023), who demonstrated that carefully prompted LLM judges achieve over 80% agreement with human preferences. Operational characteristics complement quality metrics by revealing cost-performance trade-offs. Aligning with Mohammadi et al. (2025), who advocate multi-objective agent evaluation accounting for task completion, latency, cost, and robustness.

Additionally, since each logical task has ten fuzzy descriptions, we run the entire benchmark twice and compute task-weighted average scores across runs. As a result, each reported score aggregates over multiple surface forms of the same underlying task and over multiple independent judge calls, substantially reducing variance. This configuration ensures methodological consistency with MCP-Bench while providing stable and reproducible metrics.

We retain MCP-Bench's original execution configuration to ensure comparability. This includes a pipeline timeout of 1.4 hours (5000 seconds) per task, 60-second timeouts for individual MCP connections, up to five retry attempts per failed call with 5-second delays, and a maximum of 20 execution rounds before termination.

We also extend MCP-Bench's caching system by introducing persistent task-level caching, which stores complete task execution results to avoid recomputing already completed tasks across benchmark runs. If a task fails due to external factors such as updated database entries or infrastructure errors, we manually delete both the task cache entry and any associated tool call caches, then re-run that task from scratch. This ensures that all reported results correspond to complete, end-to-end executions while keeping the experimental workload manageable, particularly important given our repeated benchmark runs across multiple fuzzy descriptions per task.

Configuration	Models	Task Count	Distraction Servers	Benchmark Runs
1-Server	Claude Sonnet 4, GPT-4o-mini	103 tasks	2 enabled	2 runs
	Phi-4-mini	103 tasks	0* enabled	1 run
2-Server	Claude Sonnet 4, GPT-4o-mini	12 tasks	1 enabled	1 run
	Phi-4-mini	12 tasks	0* enabled	1 run

*Token constraints prevented distraction server use.

Table 3.2: Experimental Configurations Summary

Table 3.2 summarises the key execution parameters for each model-configuration combination evaluated in this study. All configurations share the same execution constraints. 1-Server tasks include explicit expected-answer validation, while 2-Server tasks rely on judge-based task descriptions as the correctness reference.

3.7 Analysis

The analysis focuses on the empirical results obtained from the experimental evaluation described in the preceding sections. The primary focus is on comparing model performance in terms of response quality, factual accuracy, and cost when interacting with data through

the Model Context Protocol. Also, selected aspects of tool usage behaviour and architectural implications are examined to inform the overall assessment of MCP as a scalable enterprise solution.

3.7.1 Result Validation

Distraction servers were excluded for Phi-4-mini. While this simplified configuration technically allows the local model better conditions than the cloud models faced, early testing showed it still yielded substantially worse scoring. This baseline is primarily intended to assess whether MCP-style workflows are feasible for users running models locally on typical enterprise laptops. The exact local hardware configuration used for this evaluation is specified in Section 3.3.4 and represents a typical business laptop. Token usage and execution time distributions reveal resource demand patterns. We analyse these separately for 1- and 2-Server configurations to identify where coordination costs emerge, assessing operational predictability critical for enterprise deployment planning.

To validate that automatic scores reflect correct executions and stable data, we manually inspected a sample of task logs and outputs as a sanity check to detect clear errors or environmental changes. For example, if grounding and task completion scores are high but Expected vs Generated Correctness drops close to zero for a task, this typically indicates that the underlying database has changed. In such cases, we re-run only the affected task after clearing its cache. All reported metric values are therefore purely model- and judge-based, with manual intervention used solely to detect and repair failures.

3.7.2 Benchmarking and Diagnostics

We positioned results relative to MCP-Bench's reported scores Wang et al. (2025), identifying where our domain-specific implementation performed similarly or differently. Because both studies use the same cloud models, judge model (o4-mini), and fuzzy-prompt methodology, direct comparison is methodologically justified. We identify failure modes by examining tasks where Expected-vs-Generated scores diverge from Information Grounding and Task Fulfilment, distinguishing retrieval failures from reasoning errors and hallucinations Ye et al. (2025), directly informing RQ1's assessment of metric suitability. Additionally, the average cost per task is calculated for each foundation model, providing a practical perspective on RQ3.

Modularity analysis examines how performance scales as server count increases (RQ4). By comparing 1-Server performance against 2-Server performance, we assess whether the modular architecture introduces systematic overhead or whether observed differences stem primarily from task complexity. We document operational patterns where tool selection errors occur across server boundaries, informing design recommendations for modular MCP deployments. This analysis is primarily descriptive without statistical significance testing due to limited unique tasks and inherent LLM-judge variance (Li et al., 2024). Patterns should be interpreted as indicative, particularly for Phi-4-mini-instruct because of its differing setup and 2-Server results, which have fewer tasks. We focus on identifying consistent qualitative differences rather than claiming statistical significance for minor score variations.

Chapter 4
Results

This chapter presents findings from evaluating three language models across MCP-based tool-use tasks. Results are organized into two main sections. For each configuration, we report rule-based metrics, LLM-judge scores, and operational characteristics.

All results in this chapter are based on two mirrored MCP task sets. One public task set was built on UK Power Networks data, and a private task set was built on E.ON data. Each dataset is exposed through three corresponding MCP servers representing the same domains. These are Metadata, Network information and Metering data.

In total, the evaluation includes 103 1-server tasks (55 private, 48 public) and 12 2-server tasks (6 private, 6 public). 1-server tasks target exactly one server while two additional servers are available as distractions. 2-server tasks require tool use across two servers while one server remains as a distraction. Phi-4-mini results are marked with an asterisk, indicating that the model was evaluated without distraction servers due to token constraints. It is still shown alongside the other models in the same tables and figures for easier comparison.

4.1 1-Server Task Performance

All tables and figures in this section show results over 206 tasks for Sonnet 4 and GPT-4o-mini and 103 tasks for Phi-4-mini. This is because there are 103 1-server tasks over two runs for Sonnet 4 GPT-4o-mini, while Phi-4-mini was only run once.

This section isolates performance on 1-server tasks, where agents must select and execute tools from one target server while exposed to two distraction servers. Tables 4.1 and 4.2 present aggregate 1-server performance, while Tables 4.3 and 4.4 break down results by Private and Public datasets. Lastly, Tables 4.5 and 4.6 show performance per server on Private and Public datasets, respectively.

Tables 4.1 and 4.2 show that both cloud models achieve near-perfect protocol compliance, with schema understanding exceeding 99%. 1-server results show the same performance hierarchy as the combined metrics, but with generally higher scores. The cloud models' task

completion scores rise. While the local Phi-4-mini improves modestly, it remains substantially below the cloud models. A key addition in Table 4.2 is the "Expected vs Generated Correctness" metric, which directly compares agent outputs against gold-standard answers. Regarding cost, Claude averages approximately 1.5 SEK per task and GPT-4o-mini 0.09 SEK, derived by multiplying token usage from the tables with each model's cost per million tokens.

Metric	Claude Sonnet 4	GPT-4o-mini	Phi-4-mini*
Rule-based			
Schema understanding score	**0.997**	0.992	0.776
LLM Judge			
Task completion score	**0.958**	0.940	0.355
Tool usage score	**0.871**	0.820	0.299
Planning effectiveness score	**0.804**	0.743	0.282

Table 4.1: 1-server configuration for combined datasets (private + public) - Overview performance, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

Metric	Claude Sonnet 4	GPT-4o-mini	Phi-4-mini*
Rule-based			
Valid tool name	0.999	0.991	0.944
Input schema compliance	1.000	1.000	0.691
Execution success	0.993	0.985	0.692
LLM Judge			
Task fulfillment	0.954	0.907	0.322
Information grounding	0.961	0.973	0.388
Expected vs generated correctness	0.968	0.907	0.327
Tool appropriateness	0.786	0.712	0.267
Parameter accuracy	0.957	0.928	0.331
Dependency awareness	0.927	0.867	0.323
Parallelism & efficiency	0.681	0.620	0.240
Operational			
Avg. tool calls per task	8.6	14.6	**1.9**
Avg. agent execution time (s)	52.0	**34.5**	309.7
Avg. prompt tokens	42767	54317	**7456**
Avg. output tokens	1427	1492	**613**

Table 4.2: 1-server configuration for combined datasets (private + public) - Full metrics, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

4.1.1 1-Server Dataset Differences

Tables 4.3 and 4.4 reveal substantial performance differences between private and public datasets in 1-server configurations. Private data shows notably higher efficiency across all

models, requiring fewer tool calls while achieving stronger correctness scores. Both cloud models demonstrate consistently superior performance on private data, particularly in schema understanding and task completion metrics.

	Claude Sonnet 4		GPT-4o-mini		Phi-4-mini*	
Metric	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**
Rule-based						
Schema understanding score	0.996	**0.999**	0.990	**0.994**	**0.813**	0.733
LLM Judge						
Task completion score	**0.972**	0.942	**0.946**	0.934	**0.362**	0.347
Tool usage score	**0.913**	0.823	**0.828**	0.811	**0.317**	0.278
Planning effectiveness score	**0.851**	0.749	**0.762**	0.722	**0.291**	0.270

Table 4.3: 1-server configuration for different datasets - Overview performance, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

	Claude Sonnet 4		GPT-4o-mini		Phi-4-mini*	
Metric	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**
Rule-based						
Valid tool name	0.998	1.000	0.990	0.992	0.963	0.921
Input schema compliance	1.000	1.000	1.000	1.000	0.732	0.644
Execution success	0.989	0.997	0.980	0.991	0.742	0.635
LLM Judge						
Task fulfillment	0.986	0.918	0.922	0.890	0.334	0.307
Information grounding	0.957	0.965	0.969	0.978	0.389	0.386
Expected vs generated correctness	0.995	0.936	0.923	0.889	0.341	0.310
Tool appropriateness	0.852	0.710	0.745	0.674	0.283	0.248
Parameter accuracy	0.975	0.936	0.911	0.949	0.352	0.307
Dependency awareness	0.951	0.898	0.859	0.875	0.337	0.307
Parallelism & efficiency	0.751	0.600	0.664	0.569	0.246	0.234
Operational						
Avg. tool calls per task	**4.8**	12.9	**14.1**	15.2	**1.2**	2.6
Avg. agent execution time (s)	**45.1**	60.0	40.4	**27.8**	**299.2**	321.8
Avg. prompt tokens	**37802**	48457	58543	**49474**	**7278**	7659
Avg. output tokens	**1076**	1829	**1453**	1537	**570**	661

Table 4.4: 1-server configuration for different datasets - Full metrics, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

4.1.2 Per Server Performance Breakdown

Tables 4.5 and 4.6 summarise per-server performance for the 1-server setting across private and public datasets. Overall, the cloud models show consistently high schema understanding across servers, while most variation appears in judge-based scores. In contrast, the local model

is more sensitive to server configuration, with larger variation in both quality and operational metrics. Detailed tables are provided in Appendix C.

Private Dataset

	Claude Sonnet 4			GPT-4o-mini			Phi-4-mini*		
Metric	**Server A**	**Server B**	**Server C**	**Server A**	**Server B**	**Server C**	**Server A**	**Server B**	**Server C**
Rule-based									
Schema understanding score	**0.999**	0.992	**0.999**	**0.997**	0.993	0.980	0.778	0.744	**0.953**
LLM Judge									
Task completion score	0.948	0.974	**0.987**	0.915	0.932	**0.991**	0.207	0.338	**0.526**
Tool usage score	0.862	0.905	**0.968**	**0.863**	0.794	0.854	0.189	0.299	**0.451**
Planning effectiveness score	0.805	0.833	**0.918**	**0.809**	0.747	0.747	0.175	0.284	**0.399**
Operational									
Avg. tool calls per task	4.9	6.5	**2.0**	**5.0**	20.2	10.7	**0.4**	1.2	1.9
Avg. agent execution time (s)	40.6	50.6	**39.8**	**20.3**	52.9	36.4	**145.1**	392.8	272.3
Avg. prompt tokens	40398	41821	**29160**	**38058**	71219	54589	**3720**	9143	7138
Avg. output tokens	1122	1244	**765**	**902**	1789	1354	**418**	611	628

Table 4.5: 1-server configuration per server (A–C) for *private* dataset - Overview performance, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

Public Dataset

	Claude Sonnet 4			GPT-4o-mini			Phi-4-mini		
Metric	**Server A**	**Server B**	**Server C**	**Server A**	**Server B**	**Server C**	**Server A**	**Server B**	**Server C**
Rule-based									
Schema understanding score	**1.000**	0.998	**1.000**	**0.999**	0.991	0.997	**0.857**	0.694	0.658
LLM Judge									
Task completion score	0.919	**0.953**	0.946	**0.946**	0.930	0.926	**0.371**	0.368	0.271
Tool usage score	0.820	0.778	**0.921**	0.822	0.795	**0.832**	**0.334**	0.291	0.176
Planning effectiveness score	0.774	0.701	**0.819**	**0.762**	0.700	0.717	**0.301**	0.290	0.190
Operational									
Avg. tool calls per task	13.9	14.0	**9.3**	**10.6**	13.1	25.5	4.4	**1.8**	1.9
Avg. agent execution time (s)	57.1	66.7	**49.4**	**23.1**	25.0	39.5	**246.4**	388.0	279.4
Avg. prompt tokens	51007	50768	**40381**	**34514**	36914	94776	**6189**	9188	6332
Avg. output tokens	1885	1896	**1618**	**1239**	1343	2320	663	**634**	715

Table 4.6: 1-server configuration per server (A–C) for *public* dataset - Overview performance, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

4.1.3 1-Server Operational patterns

Interaction round distributions in Figure 4.2 show that most 1-server tasks complete within 1-3 rounds for Claude Sonnet 4 and 2-5 rounds for GPT-4o-mini. Phi-4-mini's distribution remains concentrated at 0-1 rounds, with minimal successful multi-round reasoning. Agent execution times in Figure 4.1 show that GPT-4o-mini maintains its speed advantage in 1-server mode.

Figure 4.1: 1-server configuration for different datasets, agent execution times per task for each model.

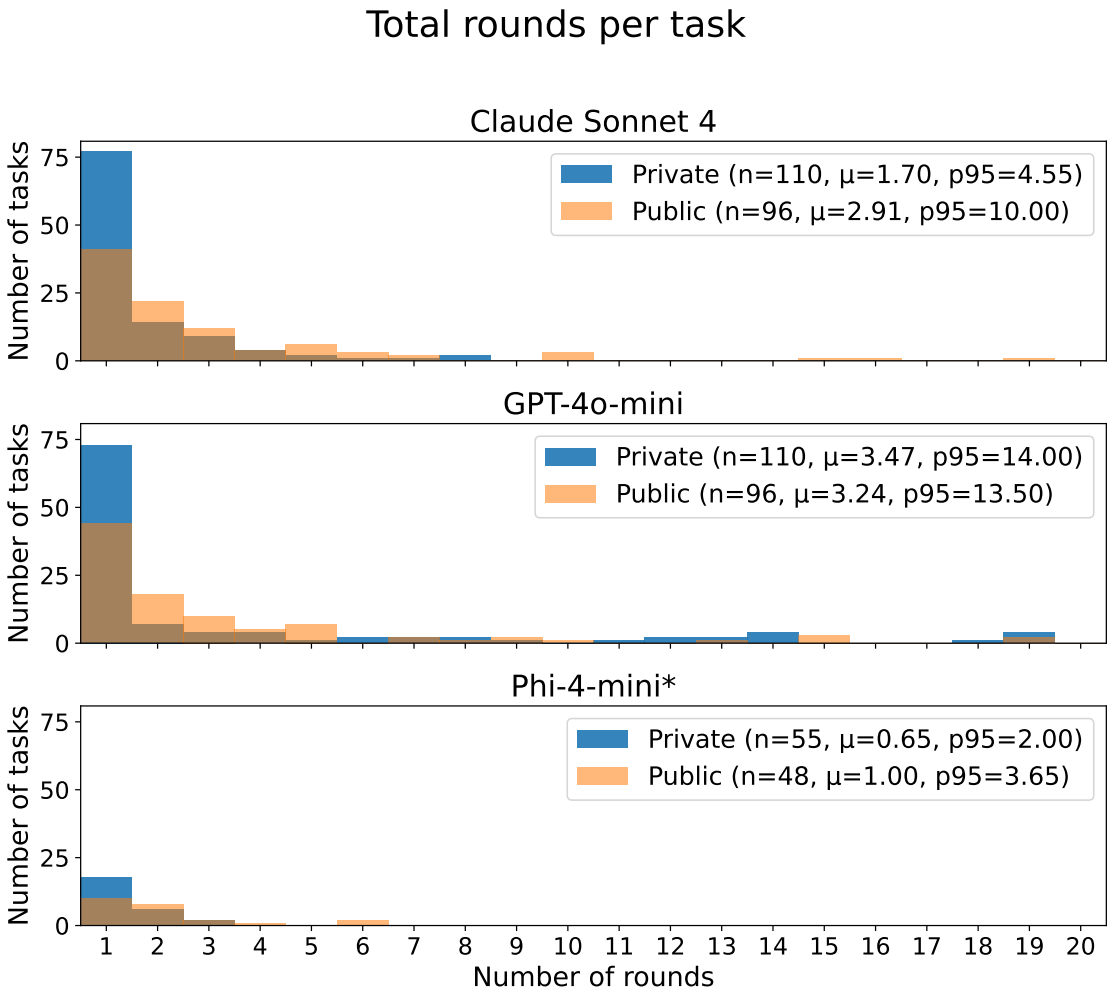

Figure 4.2: 1-server configuration for different datasets, number of agent-tool interaction rounds per task for each model. Illustrating how many iterative reasoning steps the agent required to reach a final answer.

4.1.4 1-Server Failure Samples

To make failure modes more concrete, we include a few representative execution traces below. Each sample highlights a distinct breakdown pattern that the metrics alone can not capture.

Claude Sonnet 4 In a metadata task targeting the identifier ukpn-iis, the agent produced the exact expected publisher string (UK Power Networks, Company number 3870728). However, it did not directly verify that this publisher applies to ukpn-iis through a completed tool-supported dependency chain. Instead, the agent performed extensive title searches and retrieved publishers for related identifiers, then derived the publisher for ukpn-iis without a final confirming tool call. This yields a characteristic "right answer for the wrong reasons" pattern where expected-vs-generated correctness is high, while task fulfilment remains substantially lower because the requirement includes confirming what actually applies to the target identifier rather than inferring it.

GPT-4o-mini A representative near-miss occurs in a network information task that asks

for stations with substation design “GMT” and the number of customers connected to each station. The agent selected the correct tool, executed a single successful call, presented all stations, and all count pairs were directly grounded in the tool output. However, the final report was in English instead of Swedish, and the produced station list only partially overlapped with the expected set. This illustrates a fault where protocol correctness and evidence grounding can still concur with reduced expected vs-generated correctness.

Phi-4-mini For a metering task, the user requested the exact mainmessage for incident reference "INCD-16265-I". Phi-4-mini did not attempt any retrieval step and produced a generic template response instead of querying the incident message tool. The agent never enters a tool-use loop, leading to an ungrounded answer with no verifiable overlap with the expected content. This trace exemplifies how the local baseline often fails by cancellation without executing the first required dependency

4.2 2-Server Task Performance

All tables and figures in this section show results over 12 tasks over one run for the three models, respectively. 2-server tasks require agents to coordinate tools across two servers while one server remains available for distraction. This configuration tests cross-server planning and longer tool chains. Tables 4.7 and 4.8 present 2-server aggregate results, while Figures 4.3-4.6 show operational distributions. Note that the 2-server evaluation included only 12 tasks (6 private, 6 public) compared to the larger 1-server corpus.

2-server performance declines sharply in all models compared to 1-server results. Both cloud models’ completion scores fall. Tool usage and planning effectiveness scores decline even more. Despite these drops, schema understanding remains high for both cloud models. Phi-4-mini’s 2-server performance is severely degraded. With mean rounds below one and low tool calls, the model rarely constructs any multi-step chains.

Looking at table 4.8, GPT-4o-mini achieves higher information grounding and slightly better parameter accuracy, suggesting more successful evidence retrieval once tools are executed. However, Claude Sonnet 4 scores higher on tool appropriateness, dependency awareness, and parallelism & efficiency, indicating better structural alignment with the intended workflow despite lower evidence grounding. Regarding cost, Claude Sonnet 4 averages approximately 7.33 SEK per task and GPT-4o-mini 0.45 SEK, derived by multiplying token usage from the tables with each model’s cost per million tokens.

Metric	Claude Sonnet 4	GPT-4o-mini	Phi-4-mini*
Rule-based			
Schema understanding score	0.988	**0.990**	0.628
LLM Judge			
Task completion score	0.519	**0.538**	0.138
Tool usage score	**0.447**	0.435	0.162
Planning effectiveness score	**0.278**	0.211	0.104

Table 4.7: 2-server configuration for combined datasets (private + public) - Overview performance, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

Metric	Claude Sonnet 4	GPT-4o-mini	Phi-4-mini*
Rule-based			
Valid tool name	0.998	0.992	0.940
Input schema compliance	0.999	0.998	0.496
Execution success	0.968	0.979	0.449
LLM Judge			
Task fulfillment	0.352	0.288	0.110
Information grounding	0.687	0.787	0.165
Tool appropriateness	0.428	0.392	0.133
Parameter accuracy	0.465	0.478	0.190
Dependency awareness	0.325	0.263	0.105
Parallelism & efficiency	0.230	0.158	0.103
Operational			
Avg. tool calls per task	60.9	101.1	**2.2**
Avg. agent execution time (s)	187.6	**169.8**	942.9
Avg. prompt tokens	207795	271413	**17106**
Avg. output tokens	7340	7311	**948**

Table 4.8: 2-server configuration for combined datasets (private + public) - Full metrics, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

4.2.1 2-Server Dataset Differences

Tables 4.9 and 4.10 show that 2-server dataset differences are less pronounced than in 1-server mode. Schema understanding drops more significantly on private data for both cloud models, suggesting increased protocol complexity in cross-server coordination. Despite these difficulties, task completion scores converge across datasets, with tool call counts remaining similarly high regardless of data source. Metrics on the public dataset achieve higher scores across rule- and judge-based metrics. The same patterns can be observed on the operational metrics in figure 4.10, where the lowest values are presented for foundation models. The reverse can be seen for the local model.

	Claude Sonnet 4		GPT-4o-mini		Phi-4-mini*	
Metric	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**
Rule-based						
Schema understanding score	0.979	**0.998**	0.982	**0.998**	0.533	**0.723**
LLM Judge						
Task completion score	0.515	**0.523**	**0.553**	0.522	**0.150**	0.125
Tool usage score	0.423	**0.470**	0.395	**0.475**	0.137	**0.187**
Planning effectiveness score	**0.282**	0.273	0.168	**0.253**	0.093	**0.115**

Table 4.9: 2-server configuration for different datasets - Overview performance, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

	Claude Sonnet 4		GPT-4o-mini		Phi-4-mini*	
Metric	**Private**	**Public**	**Private**	**Public**	**Private**	**Public**
Rule-based						
Valid tool name	0.996	1.000	0.984	0.999	0.917	0.963
Input schema compliance	1.000	0.997	0.997	1.000	0.400	0.592
Execution success	0.939	0.997	0.964	0.994	0.283	0.615
LLM Judge						
Task fulfillment	0.333	0.370	0.257	0.320	0.100	0.120
Information grounding	0.697	0.677	0.850	0.723	0.200	0.130
Tool appropriateness	0.410	0.447	0.373	0.410	0.133	0.133
Parameter accuracy	0.437	0.493	0.417	0.540	0.140	0.240
Dependency awareness	0.313	0.337	0.203	0.323	0.093	0.117
Parallelism & efficiency	0.250	0.210	0.133	0.183	0.093	0.113
Operational						
Avg. tool calls per task	**58.5**	63.3	102.7	**99.5**	**1.8**	2.5
Avg. agent execution time (s)	240.0	**135.2**	218.2	**121.3**	**789.4**	1096.5
Avg. prompt tokens	246742	**168848**	301635	**241192**	**16141**	18071
Avg. output tokens	7589	**7091**	7651	**6972**	**919**	977

Table 4.10: 2-server configuration for different datasets - Full metrics, Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

4.2.2 2-Server Operational Patterns

Figures 4.3 through 4.6 display operational metrics as bar charts due to the small number of 2-server tasks (12 per model). These charts reveal the substantial computational demands of cross-server coordination. Across the figures below, GPT-4o-mini generally uses more interaction rounds and tool calls than Claude Sonnet 4, corresponding to higher prompt-token usage, while output-token usage remains similar between the two cloud models.

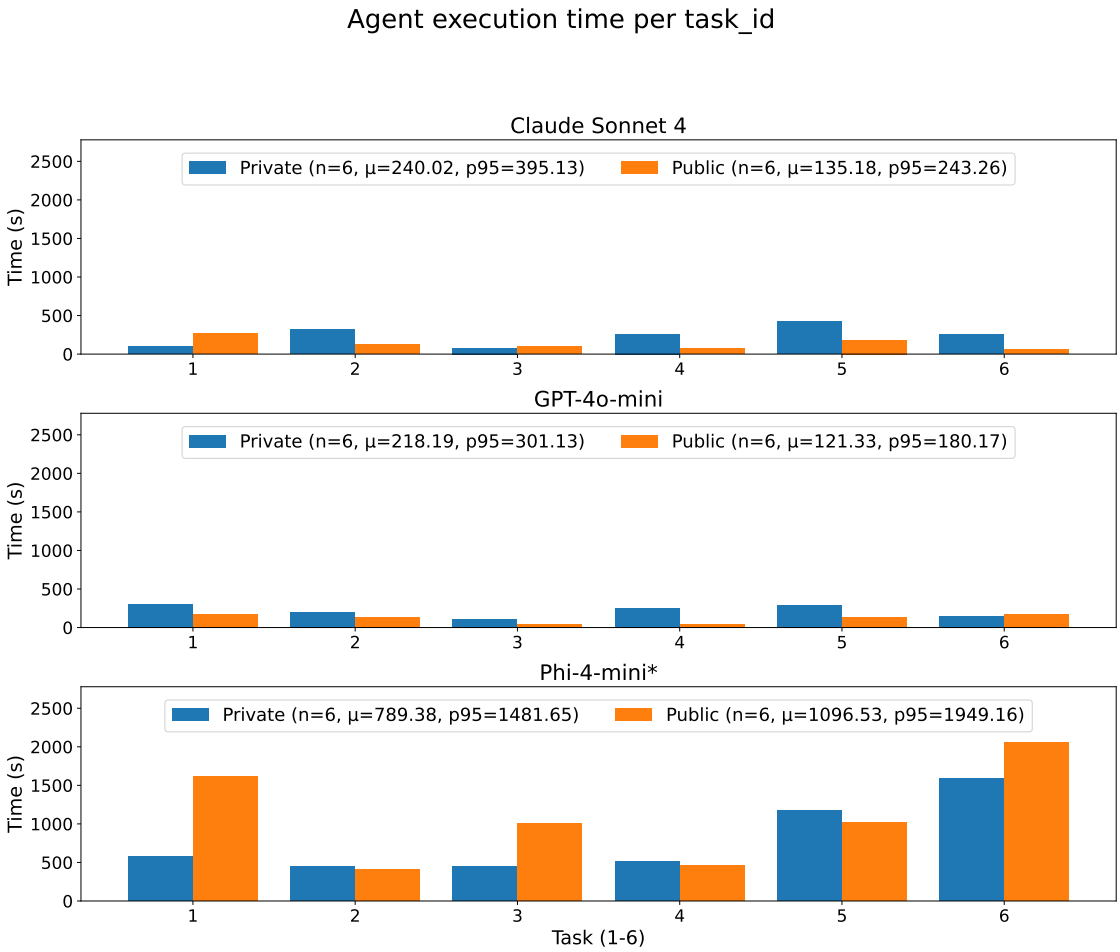

Figure 4.3: 2-server configuration for different datasets, agent execution times per task for each model.

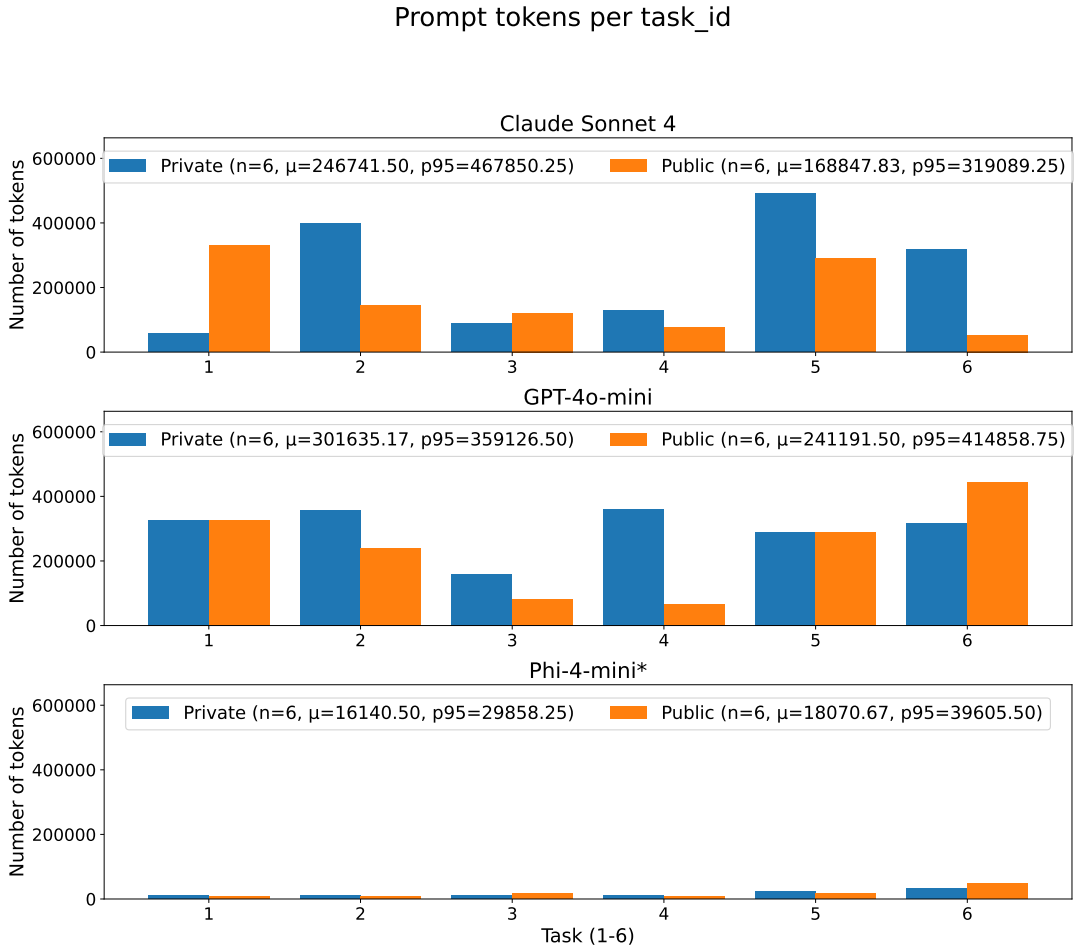

Figure 4.4: 2-server configuration for different datasets, prompt token usage per task for each model.

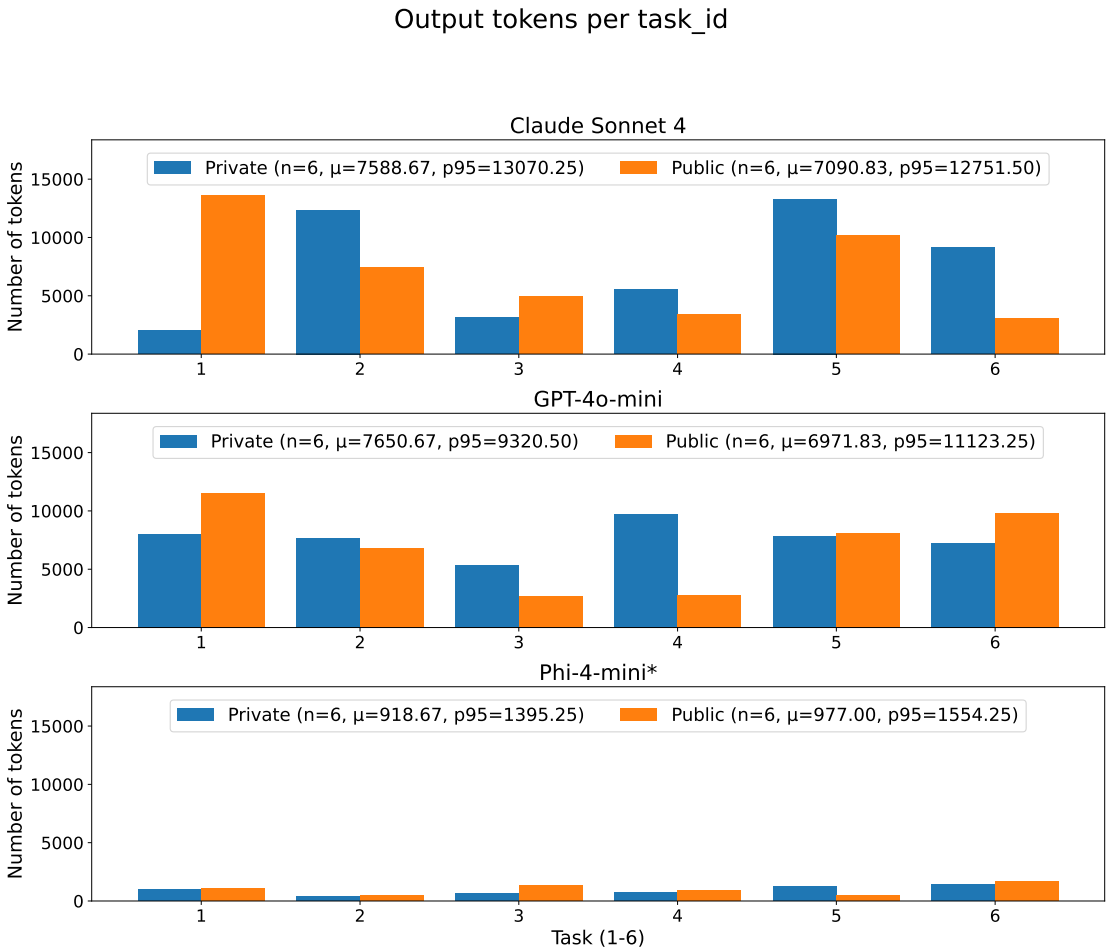

Figure 4.5: 2-server configuration for different datasets, output token usage per task.

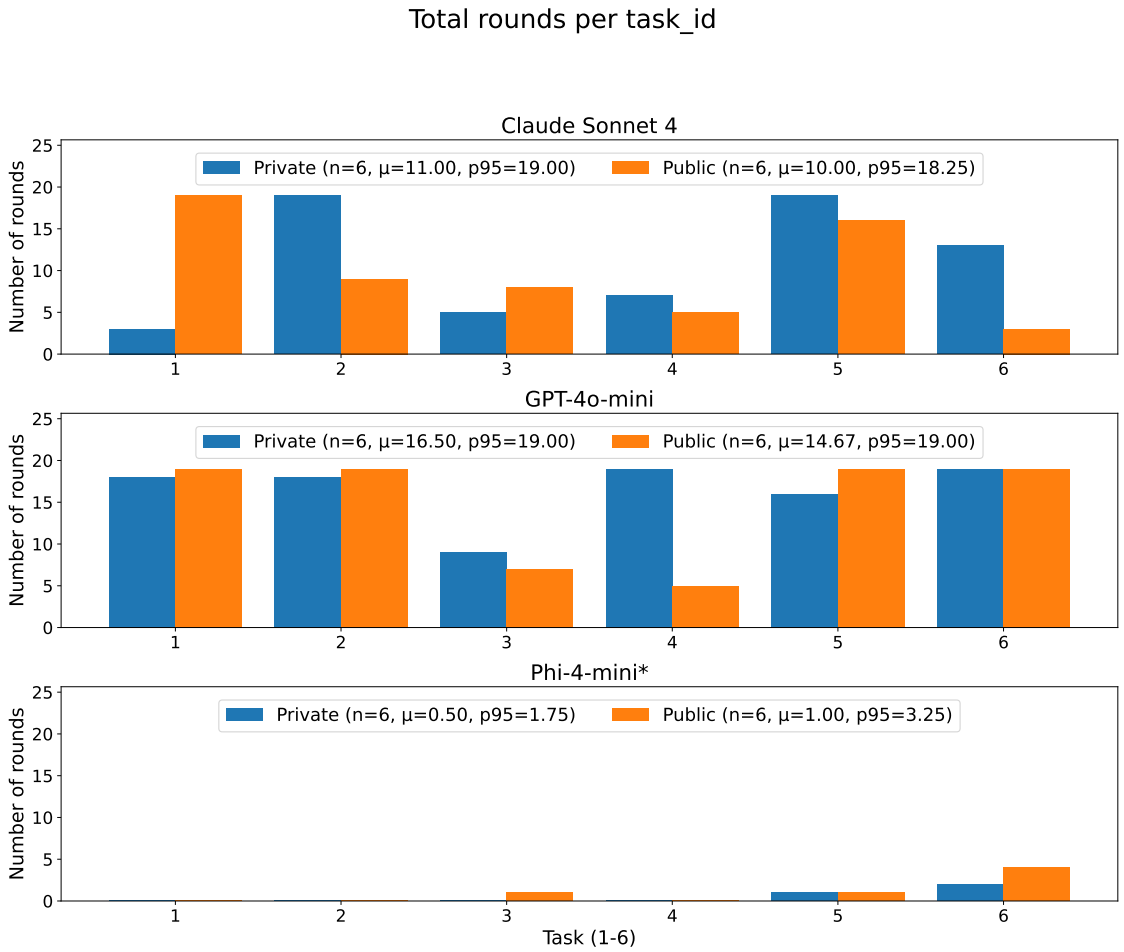

Figure 4.6: 2-server configuration for different datasets, number of agent-tool interaction rounds per task for each model. Bar charts show the number of iterative reasoning steps the agent required to reach a final answer.

4.2.3 2-Server Failure Sample

To illustrate coordination failures in 2-server settings, we include an execution trace below. This example focuses on how dependency handling and stopping behaviour can break down once tool use must be orchestrated across domains.

GPT-4o-mini A representative 2-server failure for GPT-4o-mini occurs in a task relating to the Metadata and Network Information servers. The intended workflow requires retrieving dataset metadata fields (publication status, issue year, last modified, subject/threshold), extracting early site names before a given year, fetching per-site attributes (voltage level, type, licence area), and performing conditional aggregation (count above the stated threshold) and a final publisher-alignment check. In this run, the agent only reported a small subset of the required outputs, such as publication status, publisher, issue year, and last modified, but failed to produce the identifier, the five site entries, and the correct threshold. Notably, most tool calls were syntactically valid and executed successfully, yet the chain was structurally misaligned. The agent never extracted the threshold from the metadata subject field

and repeatedly invoked network information lookups on placeholder site names, resulting in hundreds of redundant calls concentrated on one server. Illustrating why 2-server drops are driven by dependency handling and stopping criteria rather than basic tool invocation.

Chapter 5

Discussion

5.1 Performance on 1-Server

This section interprets the 1-server experiments. It first presents key quantitative results and summary statistics, next discusses model-specific trade-offs in planning and execution, and concludes with operational implications.

The 1-server results isolate the agentic behaviour of each model under a strictly local context, without 2-server planning or inter-server tool chaining. This setup lets us see how well each model selects and executes tools when two irrelevant servers are also available, and when the answer is known and fully verifiable through deterministic tool outputs.

5.1.1 Overall Performance

Cloud models reliably handle single-domain MCP workflows with almost perfect protocol compliance. The critical distinction lies in efficiency, where Claude prioritises planning quality with fewer steps, GPT compensates through speed and exploration, and small local models fundamentally cannot maintain structured reasoning across tool chains. Our results mirror MCP-Bench's finding that strong cloud models retain high protocol/schema compliance while higher-level planning and dependency-awareness remain the primary axes of difference between models.

Given the more than 99% schema-understanding scores in Tables 4.1 and 4.2, protocol errors are rare for the foundation models. The remaining failures mainly come from planning and stopping decisions. The real distinction lies in how models arrive at correct answers. This is interesting for enterprise deployment because efficiency directly translates into operational costs, while failure modes determine which error types organisations must design around.

Cloud Model Trade-offs

Claude's approach prioritises planning efficiency through fewer tool calls, earlier convergence on correct parameters, and structured dependency tracking. This displays as stronger task completion through better orchestration rather than exhaustive exploration. The efficiency advantage compounds in production contexts through reduced token consumption per task, more predictable execution patterns, and fewer opportunities for edge-case failures. GPT compensates for less structured planning through speed and persistence. It reaches similar factual correctness by trying more parameter variations and issuing redundant lookups, effectively exploring the solution space more broadly, where Claude infers solutions more directly. This works well when task objectives are clear and tool schemas are stable, but it introduces unpredictability in token costs and execution depth.

The divergent failure modes reveal the practical implications of this trade-off. Claude tends to fail through conservative stopping, terminating before gathering all evidence or declining to attempt uncertain chains. GPT more often fails through parameter construction errors, selecting correct tools, but misinterpreting temporal phrasings or threshold specifications. Neither failure type typically prevents task completion, but they determine where human oversight becomes necessary.

GPT-4o-mini's speed advantage reflects API response latency rather than reasoning efficiency. Organisations prioritising responsiveness may accept higher token consumption for faster responses, though this advantage disappears under backend delays like asynchronous job polling.

Local Model Limitations

Phi-4-mini's breakdown pattern establishes a capability threshold below which MCP becomes unreliable. The dominant failure mode is schema inference collapse, where the model selects valid tool names but constructs malformed parameter objects due to insufficient reasoning capacity. This does not appear to be a tuning problem but rather reflects a fundamental inability to maintain structured representations across multi-turn exchanges. The secondary pattern of premature termination after single attempts suggests the model cannot sustain tool-use loops even when initial responses succeed. High execution times despite minimal tool activity indicate slow, unproductive reasoning rather than effective planning on constrained hardware.

Looking at figures 4.1 and 4.2, Phi-4-mini's extreme execution times despite minimal rounds indicate the model spends time in failed inference attempts rather than productive tool use. Parameter construction failures occur during generation, causing repeated restarts rather than iterative solutions.

These results establish that substantial capability gaps exist even without cross-server coordination complexity. Providing concrete answers to RQ2 and RQ3 by characterising which models succeed under 1-server conditions and by quantifying their operational trade-offs. Phi-4-mini provides a practical lower bound on MCP-based tool use under local resource constraints. While useful for on-device, cost-free inference, it is not yet capable of handling even moderately structured MCP tasks, reinforcing why cloud-hosted models remain necessary for most real MCP implementations.

5.1.2 1-Server Dataset Differences

Building on the dataset-level differences reported in Tables 4.3 and 4.4, we discuss potential reasons for the observed efficiency gap. Several factors likely contribute to this pattern. Private data reflects E.ON's operational vocabulary and internal naming conventions, which may align more naturally with how tasks were constructed during corpus development. Public data requires navigating Opendatasoft's query conventions and less standardised field naming. The pattern holds across both cloud models, suggesting genuine dataset differences rather than model-specific artefacts. This has practical implications for MCP deployments, where tool design should prioritise alignment with natural query patterns in the target domain.

Whether this efficiency gap reflects genuinely simpler schemas, better tool-task alignment during benchmark construction, or model training distributions remains unclear. However, the consistency across both cloud models strengthens the interpretation that structural dataset characteristics drive the observed differences. Consistent with MCP-Bench single-server experiments, schema compliance stayed high across datasets, but grounding and task-completion quality still varied with dataset complexity.

The divergence between datasets across cloud models seen in figure 4.2 reveals different reasoning strategies under schema variation. Claude's structured planning expects consistent interfaces and requires additional validation rounds with less normalised schemas. GPT's exploratory approach proves less sensitive because it does not rely on inferring optimal paths upfront. The tool calls escalation on public data, combined with stable round counts for GPT, suggests efficiency gaps stem from backend architecture rather than vocabulary alone. Dremio's strongly typed schemas enable precise queries, while Opendatasoft's field-level filtering requires more exploratory calls. Implying model performance varies by backend consistency, not just domain complexity.

5.1.3 Server-Specific Patterns

Tables 4.5 and 4.6 indicate different complexity characteristics across data categories.

Server C, which handles metering, shows the best results in most areas. This is likely because temporal queries are straightforward, using timestamps with clear parameters and limits. Metadata, managed by Server A, has slightly lower planning scores. This might be due to the complex relationships in Dublin Core, which require careful tracking of dependencies. Server B, which handles network information, is the most challenging, especially for GPT. This suggests that the hierarchical lookup patterns across different infrastructure layers make it difficult to choose parameters. Phi-4-mini is very sensitive to domain characteristics; its schema understanding varies by more than 0.2 across servers, while Claude's varies by less than 0.01. This indicates that the local model struggles to maintain consistent reasoning when the domain's complexity changes. The hierarchical identifier types in network infrastructure cause significant failures.

This informs deployment priorities and expectations. Temporal query patterns are most robust across model capabilities, whereas hierarchical domain navigation benefits substantially from foundation-scale reasoning. Organisations should expect domain-specific performance variation and may need different models for different data categories, depending on complexity characteristics and criticality requirements.

5.2 Performance on 2-Server

This section analyses the 2-server experiments. It begins with cross-server performance and cost summaries, then categorises coordination and dependency failures with representative examples, examines differences across datasets and planning strategies, and ends with implications for orchestration and resource management.

2-server tasks require agents to coordinate tools across two servers while one server remains available as a distraction. This configuration tests cross-server planning and longer toolchains through workflows spanning domain boundaries.

5.2.1 Overall Performance

As shown in 4.7 and 4.8, cross-server orchestration remains feasible for cloud models but causes sharp quality degradation and large cost increases, despite maintained protocol compliance. Compared to operational metrics in 4.2, both average tool calls per task and agent execution time increase drastically for the cloud models. The bottleneck lies in strategic coordination rather than protocol mechanics. The small local model becomes completely unreliable, displayed not only by poor judge metrics but also by high agent execution time. Overall, the 2-server setting shows that protocol competence is not sufficient. The bottleneck shifts to coordination, dependency handling across servers, and stopping behaviour. MCP-Bench likewise reports a clear performance drop when tasks span multiple servers, supporting our conclusion that cross-server orchestration increases planning difficulty and error rates.

Cloud Model Strategic Differences

Schema understanding remains essentially tied between Claude Sonnet 4 and GPT-4o-mini, but their styles differ. GPT achieves slightly higher task completion scores, suggesting it more often reaches acceptable end states. It also leads to execution success, parameter accuracy, and information grounding, where it retrieves and cites evidence more reliably once tool chains execute. Claude scores higher on tool usage quality and planning-related dimensions. It leads to task fulfilment, tool appropriateness, dependency awareness, and parallelism & efficiency as shown in table 4.8. This indicates more structured tool chains that better align with task intent despite similar completion rates. Together, these results show a trade-off where GPT tends to do better at assembling and grounding evidence, while Claude tends to maintain better overall workflow structure and alignment with the intended solution path.

Operational Cost Drivers

As suggested by 4.3, 4.4 and 4.6, the multiserver cost difference is mainly driven by longer interaction rather than longer final answers. GPT's slightly lower agent execution time partially offsets its higher token usage, but the overall operational cost is still driven by the longer interaction traces implied by more rounds and tool calls. These distinctions extend RQ3 by showing that cost-performance trade-offs shift when coordination complexity increases. Models can remain strong on protocol competence while diverging in how efficiently they search, validate dependencies, and stop.

The prompt token escalation shown in figure 4.4 stems from context accumulation rather than task complexity. Each tool output from the first server remains in history when querying the second, and every round incorporates all prior exchanges. Output tokens in figure 4.5 appear more stable, confirming final answers are comparable regardless of orchestration complexity. GPT's higher token usage reflects additional rounds, yet execution times converge. API overhead dominates latency in complex orchestration, reversing the 1-server dynamic where speed came from rapid calls. Issuing more calls simply accumulates waiting time without offsetting benefits.

Local Model Differences

Phi-4-mini underperforms severely in 2-server scenarios. While it often selects valid tool names, it struggles with schema compliance and execution success, preventing continuous multi-step chaining across servers. Its low rounds and minimal tool calls indicate early breakdown rather than efficiency. This is consistent with low judge scores across completion, tool usage, and grounding. The high mean agent execution time of nearly 16 minutes in Table 4.8, combined with very low token usage, further suggests slow, unsuccessful reasoning attempts rather than effective coordination. This severe breakdown reinforces RQ2's conclusion that model suitability depends heavily on task complexity, with small local models fundamentally unsuitable for 2-server orchestration.

5.2.2 2-Server Dataset Differences

Tables 4.9 and 4.10 reveal an unexpected turnaround. Where private data showed clear advantages in 1-server mode, the pattern flips for 2-server tasks. Schema understanding drops on private data for both cloud models, suggesting that coordinating across private servers introduces protocol complexities that do not appear in the public setup. This could stem from less uniform interfaces between servers or more complex cross-server parameter dependencies in the private infrastructure.

The execution time reversal seen reveals backend architecture as more critical than schema consistency. Dremio's asynchronous workflow requires separate steps for submission, polling, and retrieval. Cross-server orchestration triggers sequential job submissions with polling intervals between each. Whereas, Opendatasoft's synchronous endpoints enable tighter iteration loops despite higher per-query overhead. Models struggle with the temporal complexity of tracking asynchronous jobs more than the structural complexity of query syntax.

5.3 Implications for Enterprise

The results highlight that MCP can be a feasible standardised layer for enterprise data access, but that the operational risk is driven less by whether models can call tools and more by whether they can orchestrate coherent workflows over the long term. In our evaluation, both cloud models achieved near-perfect rule-based protocol behaviour, while judge-based scores dropped substantially in the 2-server setting, despite still maintaining high schema understanding. Our results reveal that the enterprise bottleneck is not primarily tool invocation, but robust planning, dependency handling, and stopping criteria when tool chains

span multiple servers and rapidly growing contexts.

5.3.1 Model Selection by Use Case Requirements

Foundation models such as Claude Sonnet 4 are most justified when tasks are critical, workflows naturally require multiple steps and cross-domain coordination, or tool schemas are complex and error tolerance is low. This reflects RQ2, as it frames model choice around how well each model handles the workflow demands of MCP tool use across different task types. RQ2 does, however, also focus on the generated response of models. In regard to that, expected-vs-generated correctness in 1-server tests and task completion in 2-server tests correspond to how well the model generated a response through MCP. GPT-4o-mini has a slight edge in task completion score in a 2-server setting, while Claude Sonnet 4 scores better in expected-vs-generated correctness. Sonnet 4 consistently achieved the highest overall judge-based scores in 1-server tasks and, in 2-server tasks, showed a stronger tool usage score and planning effectiveness score while using fewer rounds and fewer tool calls than GPT-4o-mini. This indicates more structured orchestration under complexity. This matters operationally because enterprises often care about repeatability and predictable behaviour, especially when decision-making depends on the result.

Lower-cost cloud models, such as GPT-4o-mini, are attractive when tasks are dominated by 1-server retrieval, where the primary challenge is parameter grounding rather than long-term planning. In our 1-server setting, GPT-4o-mini remained close to Sonnet 4 on aggregate quality while being faster on average execution time, at the expense of more rounds and higher prompt-token consumption. As shown in sections 4.1 and 4.2, the cost per task for GPT-4o-mini is approximately 16 times cheaper in both settings. This ties to RQ3 by showing the trade-off between cost and efficiency. GPT-4o-mini is well-suited for enterprise self-service analytics queries where latency matters, the tool's compliance is stable and well-specified, and the organisation can tolerate occasional inefficiency in exchange for a much lower per-token price.

Small local models, such as Phi-4-mini, prove unreliable for moderately structured MCP tool use in our experiments. They consistently underperformed the foundation models in both schema compliance and execution success, and frequently terminated early instead of continuing through multi-step sequences, especially in 2-server scenarios requiring tool chaining. For enterprise adoption, this implies that small local models should be limited to narrowly scoped use cases such as offline environments or simple lookups with very few parameters. If organisations want local deployment for compliance reasons, a reasonable route is to keep the local model for tasks such as summarisation while delegating tool execution and planning to a more capable model. This perspective is particularly relevant in contexts where privacy requirements, vendor lock-in concerns, or regulatory constraints, such as restrictions on running commercial cloud-based models outside Europe, motivate the use of locally deployed or on-premises language models.

5.3.2 Performance and Cost Trade-offs

The key trade-off is predictability versus interaction volume. Claude converges in fewer steps, while GPT more often pays for similar outcomes with additional calls and tokens. Phi-4-mini

mainly reflects local deployment overhead, where wall-clock time becomes the limiting factor rather than tool efficiency.

Consistent with the 2-server operational patterns discussed earlier, cross-server orchestration primarily increases cost by expanding the prompt-side context through long tool dialogues rather than by producing longer final answers. For enterprises, the practical implication is that MCP deployments should be designed to minimise unnecessary tool invocations by strategies such as returning only required fields, splitting large tools into smaller ones, and imposing limits on the maximum number of tool calls and token usage. At approximately 30 SEK per million input tokens and 150 SEK per million output tokens, Sonnet 4 is in the premium tier. GPT-4o-mini at roughly 1.50 SEK input and 6 SEK output offers a compelling middle ground for high-volume routine queries. Phi-4-mini provides zero per-token costs but requires on-premises hardware investment and has proved unreliable for structured MCP tasks.

The cost structure shifts fundamentally between configurations. In 1-server tasks, organisations trade planning efficiency against execution speed. In 2-server tasks, API overhead dominates, and additional rounds yield no speed advantages. This suggests tiered deployment strategies. High-frequency simple queries can use GPT-4o-mini to minimise latency. Complex cross-domain workflows should use Claude Sonnet 4 because planning efficiency directly reduces tokens without sacrificing response time. As MCP-Bench shows, basic execution capabilities have converged for top models while planning effectiveness, not raw token throughput, largely drives practical cost to quality trade-offs in multi-step workflows.

5.3.3 Deployment and Infrastructure Considerations

Cloud models require stable API connectivity and require ongoing operational costs, but eliminate hardware concerns. Local models are cost-free and offer stronger data security guarantees, but demand knowledge in model optimisation and performance tuning. Our evaluation of Phi-4-mini on commodity hardware indicates that models of the same size and similar hardware that are tested on our benchmark cannot reliably handle moderately structured MCP workflows, even in simplified conditions.

For 1-server tasks with stable schemas, both cloud models scale predictably, with GPT-4o-mini offering better cost efficiency at the expense of higher token consumption. For 2-server orchestration, Claude Sonnet 4's more efficient planning becomes increasingly valuable as coordination complexity grows, potentially offsetting its higher per-token cost through reduced interaction depth. Local models remain unsuitable for scaling beyond highly constrained single-tool scenarios under current capability levels.

5.3.4 Modularity Perspective

The modular architecture was easy to extend during development. Adding a new tool required only registering it with the appropriate server's MCPServer instance, defining its input schema, and implementing the query logic, with no changes to other servers or the MCP client. This extensibility would enable organisations like E.ON to incrementally expand MCP coverage to additional data domains by deploying new servers without disrupting existing functionality. Li et al. (2025) similarly notes that their modular EnergyPlus-MCP design "provides an extensible foundation through an initial set of building blocks that can accommodate

additional simulation engines or analysis capabilities without requiring modifications to the core MCP interface implementation", which is essential for enterprise deployments where functionality requirements evolve, and different teams contribute domain-specific capabilities independently.

Architectural Scope of Modular Solution

RQ4 concerns how a modular MCP architecture can be designed. A central design degree of freedom is how tasks, tools, and supporting components are separated into one or many servers. These boundaries shape the agent's execution problem. They determine which tool inventories are visible at a given step, how intermediate results must be carried across server boundaries, and how easily the agent can maintain a correct dependency chain and stopping criteria.

In this thesis, we propose and evaluate one concrete architecture with three domains corresponding to three servers. This domain-oriented separation is also consistent with the multi-domain server ecosystem by MCP-Bench, where agents interact with multiple specialised servers and must coordinate tool usage both within and across server boundaries. However, because we keep the partitioning fixed, our results should be read as evidence about the design properties of this specific modularisation. In particular, the observed gap between 1-server and 2-server performance can indicate a coordination cost introduced by cross-server dependency tracking and tool discovery in the evaluated architecture. This does not imply that "having more servers is worse" in general. It motivates explicit design trade-offs and future design alternatives that vary the server boundaries.

Server and Interface Design Choices

A modular MCP can be designed in multiple valid ways depending on how tools are grouped into servers and which MCP primitives are exposed. Merging tools into fewer servers reduces cross-server coordination, but increases the size and variety of the visible tool set, which can increase tool-selection ambiguity when queries are fuzzy. Splitting domains into more servers can make each tool inventory more coherent and easier to browse locally, but it increases the number of cross-server joins required for end-to-end workflows. Importantly, server boundaries also change what counts as a "2-server" task. A workflow that is 1-server under one partition may become 2-server under another, directly affecting execution difficulty and measured orchestration overhead.

One aspect we did not evaluate is the agent's ability to call two tools within the same server. Our 2-server tasks require tools from two different servers, so coordination costs are challenged with crossing a server boundary. This test would not affect tool visibility because of the use of distraction servers. Testing multiple tool calls in a 1-server setting would clarify whether the main bottleneck is cross-server orchestration and handling different topics or long tool chains in general.

MCP can also expose resources and prompts in addition to tools. In our implementation we only use tools, but resources and prompts could reduce the need for many trial tool calls and help the agent follow dependency chains in longer workflows. A practical design extension is to add resources and prompts that make it easier to find the right tool. We did not test different server boundaries or different exposure types, so we cannot measure the effect

of these choices, but our 2-server results suggest they could reduce problems caused by too many tools and difficult cross-server coordination.

As modular MCP deployments grow, extensibility naturally leads to an increase in the number of available tools, which introduces new design trade-offs. Increasing the total tool count could overwhelm models in 2-server scenarios. This tension reflects a broader design challenge in MCP architectures. Li et al. (2025) faced similar considerations when implementing their EnergyPlus-MCP server, ultimately organising 35 specialised tools into five distinct functional domains spanning model management, inspection, modification, and simulation execution. Their experience demonstrates that tool categorisation and clear functional boundaries help manage complexity as tool counts grow, particularly when tools must cover comprehensive domain-specific workflows.

A key design tension emerged around tool granularity. Our implementation favoured specificity, defining separate tools for distinct queries. While both cloud models achieved high parameter accuracy scores in 1-server tasks, we cannot attribute this performance to our tool design choice since we did not test alternative configurations. The increased total tool count did introduce challenges in 2-server scenarios where dozens of tools became visible simultaneously, as both cloud models showed reduced performance compared to 1-server settings. However, without comparative evaluation against coarser-grained tool designs, we cannot determine whether this performance gap stems from an increased number of tools, task complexity, or other factors. This represents a limitation of our study and suggests an important direction for future work.

Cross-server test results suggests that modular MCP architectures benefit from discovery mechanisms beyond only tool lists, either grouping by domain, lightweight routing hints based on query classification, or selective server exposure where only servers relevant to a user's role or current task are available to the agent.

5.4 Methodological Reflections and Comparison to MCP-Bench

This study deliberately builds on the MCP-Bench framework to evaluate MCP-based systems in a realistic energy-network context. Methodologically, this means that many design choices from task specification and server configuration to the scoring pipeline are inherited rather than designed from scratch. This subsection reflects on how reusing MCP-Bench improved comparability and robustness and where our setup deviates in ways that affect interpretation and external validity.

5.4.1 Benefits of Building on MCP-Bench

Adopting MCP-Bench as our evaluation foundation provided several methodological advantages. The framework's dual-layer assessment, combining deterministic protocol checks with rubric-based semantic evaluation, enabled separate measurement of protocol adherence and task completion quality. The rule-based metrics provided objective measures of MCP protocol interaction, distinguishing protocol violations from reasoning failures. This granularity proved valuable for diagnosing model behaviour, offering more insight than binary

success/failure approaches.

MCP-Bench's validated LLM-as-a-judge methodology removed the need to independently validate a judge framework. The structured rubrics captured dimensions beyond execution success, measuring task completion, tool usage, and planning effectiveness. This allowed identification of inefficient solutions that would otherwise appear successful under purely execution-based evaluation. This design is strong from a scalability perspective. Many tasks across multiple servers and models can be evaluated without the cost of manual annotation. Using an already studied judge, o4-mini, arguably yields more credibility. At the same time, the absence of human ground truth means that we can not quantify judge bias.

The existing infrastructure substantially reduced development overhead. MCP-Bench handled MCP server orchestration, tool call interception, distraction servers, trajectory logging, and judge prompting. This gave space for domain-specific contributions such as our energy-network tasks, expected-answer validation, and deterministic tool outputs. The modular architecture facilitated extensions. Adding expected-vs-generated correctness for 1-server tasks required minimal pipeline changes, enhancing rather than replacing the base framework.

Using the same cloud models and judge as MCP-Bench enabled qualitative comparison. Similar performance hierarchies across benchmarks strengthened confidence that findings reflect genuine model capabilities rather than evaluation artefacts. Shared metrics enabled contextualisation of observed performance differences.

However, building on MCP-Bench does not make our results automatically comparable to Wang et al. (2025). Our task design, domains, and caching policies differ in ways that affect benchmark difficulty and error patterns. MCP-Bench emphasises cross-server orchestration and long-horizon tasks. Our 1-server focus represented a simplification for deterministic outputs and controlled evaluation, limiting direct score comparability with MCP-Bench's aggregated metrics. Our separate reporting of 1-server and 2-server results reflects this architectural difference. Methodologically, MCP-Bench functions as a core framework, but the concrete instantiation remains specific to our enterprise energy-network setting.

5.4.2 1-Server Adaptations

The most consequential deviation from MCP-Bench concerns 1-server tasks. Instead of relying primarily on fuzzy task descriptions, we align the benchmark with our focus on factual reliability by introducing explicit expected answers, expected vs generated. The expected answer is embedded in the dependency analysis and used for direct data comparison, checking that retrieved evidence actually supports the correct raw value.

This design has several methodological advantages. First, it addresses concerns raised in MCPUniverse by Luo et al. (2025) about relying exclusively on judge-based scoring for fact-heavy tasks. By grounding evaluation in explicit evidence, style bias and hallucinations are more easily observed rather than just relying on the grounding score. When the model returns the wrong value but writes a convincing explanation, the judge based expected vs generated metric still flags the failure. In that sense, our 1-server setup tightens the notion of success relative to MCP-Bench's more rubric-driven fuzzy objectives.

Second, embedding the expected answer in the dependency analysis strengthens the link between task specification and database content. Hallucinations are penalised and correct retrieval is rewarded, which is well aligned with our goal of testing ground truth in tool use

rather than reasoning. This also makes the analysis of failure more informative.

5.4.3 2-Server Tasks

For 2-server tasks, we deliberately retained the MCP-Bench's original fuzzy description design and relied on the task description as the judge's reference point. Keeping this setup instead of defining an exact expected answer that spans several servers and tool calls would require considerably more annotation effort. Although it is feasible to attach an expected answer to a 2-server task and compare the final response, we intentionally do not apply strict expected-answer scoring in the 2-server setting. The main reason is sensitivity to continuously updated operational data. 2-server tasks are substantially harder to evaluate using expected-answer scoring because the databases are continuously updated and the final answer depends on the coordinated outputs of multiple tools. While we enforce determinism for certain 1-server tools by constraining the time window and limits of rows of data retrieval, extending the same guarantees across 2-server testing would require synchronising constraints and assumptions across all involved tools. This quickly increases implementation and maintenance complexity. Given this, our evaluation effort focuses on modular 1-server tasks, where deterministic tool behaviour and expected-answer verification can be enforced in a controlled and reproducible manner. Instead, we adopt MCP-Bench's original task generation, which utilises a task description.

Methodologically, this means that our 2-server evaluation is closer to MCP-Bench's original intent. It therefore tests whether agents can coordinate across servers and tools to satisfy a goal. It also preserves a direct comparability for the small set of twelve 2-server tasks we include. At the same time, the small number of tasks and the absence of expected answer scoring limit the strength of any conclusion drawn about complex 2-server behaviour. Meaning that a relaxed correctness criterion is set for 2-server tasks where a response is considered correct to the extent that it satisfies the stated task goal, from the task description, and remains grounded in the tool outputs produced during the run. We ultimately rely on the judge's task fulfilment and grounding rubrics as the correctness reference while separately reporting protocol- and workflow-oriented metrics to capture whether the agent's intermediate steps were sensible, even if multiple valid chains can lead to a plausible final answer. This design choice aligns the 2-server evaluation with Wang et al. (2025)'s original methodology, thereby enabling more directly comparable results.

The asymmetry between 1-server and 2-server setups is therefore both a strength and a limitation. Letting us measure reliability in a setting where the "right answer" is easy to verify, while still using 2-server tasks to probe coordination and orchestration. Since we do not include the expected vs. generated score in the aggregated task completion score, it does not affect it.

5.4.4 Implications for Comparability

Altogether, our choices position this study as a domain-specific extension of MCP-Bench rather than a strict replication. We follow the same architectural principles, use the same base and judge model and adopt a similar group of metrics. Meanwhile, we introduce expected answer scoring, energy-specific schemas, internal and external datasets and an asymmetric treatment of single- and 2-server tasks.

As a result, comparison to MCP-Bench should be interpreted primarily at the level of patterns and relative behaviour. For example, we can meaningfully ask whether Claude Sonnet 4 tends to gain more from a 2-server setting than GPT-4o-mini in both studies or whether both benchmarks identify similar strengths and weaknesses in schema understanding. In contrast, absolute score differences are harder to interpret because they reflect not only on model behaviour but also differences in task mix, expected answer strictness and domain difficulty.

Our contribution lies less in proposing an entirely new benchmark and more in showing how an existing MCP benchmark can be adapted to a concrete industrial domain and research questions while still supporting meaningful comparison.

5.5 Limitations and Threat to Validity

This section summarises key limitations and threats to validity in our evaluation, structured around if we are measuring what we intend to measure, whether results are generalisable and whether results are reproducible.

5.5.1 Construct Validity

Although the rule-based metrics are strong indicators of protocol adherence, they do not guarantee that the right information was retrieved or that the final answer is correct. On the other hand, the LLM-as-a-judge metrics capture higher-level notions, but they are inherently subjective and can be sensitive to rubric phrasing and model bias. Our addition of expected-vs-generated-correctness in the single-server setting improves factual measurement with the gold standard. This construct is not done for multi-server tasks, creating an asymmetry in how correctness is graded across settings.

Only tools that could be mapped fairly across both the public and private settings were implemented, which restricts task diversity and may underrepresent categories common in real enterprise usage. Additionally, the distraction server design probes server selection under noise, but it remains a simplified proxy for real enterprise tool ecosystems where tools may evolve rapidly and include permissions or rate limits that influence behaviour.

5.5.2 Internal Validity

LLM outputs are stochastic. While we applied judge stability with multiple judge passes and repeated runs over different fuzzy phrasings, residual variance remains, particularly in multi-server tasks where long chains add to early divergences. This makes it harder to confidently claim small differences in performance between the two cloud models. Broad patterns are more reliable.

Even with efforts to ensure deterministic tool outputs, underlying datasets and platform behaviours can change over time. Cache management and selective reruns help, but they introduce a risk of subtle inconsistencies across runs if the environment changes between executions. This is especially relevant for enterprise deployments where databases are added to in real time.

5.5.3 External Validity

Tasks were designed around an energy network context and executed with Swedish fuzzy prompts, which are central to our use case but may not generalise to other enterprise domains or languages. Domain-specific terminology and bilingual requirements can interact with model training distributions, meaning that performance observed here may not transfer to healthcare, finance, or purely English tool ecosystems.

Only three models were evaluated, two cloud-based models and one very small local model. This sampling does not represent the broader design space of LLM families, context lengths, and tool use training approaches. As a result, conclusions about "cloud vs local" should be interpreted as evidence about these specific model choices rather than a universal statement.

The benchmark runs were performed in a controlled setup that does not capture real operational pressures such as concurrent users, unpredictable latency, tool rate limiting, authentication, or spikes in query usage. These factors can affect latency, so real-world performance might not match these benchmark results, even with strong benchmark scores.

5.5.4 Reliability and Reproducability

Gold answers and paraphrased prompts were manually generated and reviewed. This improves factual control but introduces potential human bias in what is deemed correct and how natural the prompts are. The risk is that some prompts may unintentionally signal certain tools or encode patterns that favour specific models.

Our evaluation relies on an LLM judge for qualitative dimensions, with no systematic human grading. Even with stability measures, we cannot quantify judge bias directly. This is a known limitation of LLM-as-a-judge approaches, as LLM judges can exhibit systematic judgment biases and be influenced by surface-level presentation disturbance, and they may also overlook factual errors Chen et al. (2024). This becomes more noticeable when comparing models whose writing styles differ or when correctness depends on subtle factual distinctions that are not strongly represented in the rubric.

5.5.5 Overall Implications of Evaluation Threats

Despite the validity threats discussed above, this study establishes three findings. There is a large qualitative gap between the small local model and the cloud models under our MCP tasks. Furthermore, the consistent pattern is that multi-server orchestration is substantially harder than single-server tool use, even when rule-based protocol metrics remain high. Lastly, the trade-off for enterprises is between structured planning efficiency and fast exploratory execution. More smaller claims, especially about small differences between the two cloud models in specific judge subdimensions, should be interpreted as indicative and should be validated with broader task coverage, additional models, and complementary evaluation methods in future work.

5.6 Social and Ethical Aspects

This thesis evaluates MCP-based LLM access to enterprise operational data in the energy domain. Because enterprise data may include sensitive operational and organisational information, a core ethical requirement is preventing the exposure of data while still enabling realistic evaluation.

5.6.1 Confidentiality and Privacy

Even when the model only performs read queries, privacy risks remain because tool outputs can contain identifiers, locations or operational details that should not be broadly exposed. Cloud-hosted deployments add governance concern, sensitive data may leave the organisation's infrastructure, which can create regulatory risks depending on what is sent and how it is processed.

Practically, tools should return only the fields needed to answer the question. This is especially relevant here because server responses were intentionally passed to the model in a raw JSON to reflect realistic MCP interactions. Since all API requests use a personal token for each user, one will only be able to retrieve data that the user is authorised to. This shifts the responsibility to human confirmation outside of the loop rather than MCP-based checks.

5.6.2 Security Risks in Tool Using Agents

Connecting an LLM to tools consequently raises risks such as unsafe tool invocation or attacks that try to manipulate the agent through instructions embedded in retrieved content. These threats should be considered in any enterprise deployment of MCP-based agents. In practice, this motivates strict separation of credentials from code, robust authentication, and detailed logging of tool invocations. Our implementation choices reflect parts of this direction by keeping configuration and credentials separate and performing sensitive access and validation in a secure environment.

5.6.3 Reliability in LLMs

From a social perspective, one of the main risks is trusting an LLM too much. LLM outputs can sound confident even when they are wrong. The background discussed known issues such as hallucinations and the reproduction of training data biases, which are especially problematic if outputs are used for critical decisions.

For energy-network contexts, incorrect statements about outages, assets, or maintenance status could lead to real operational costs. The system should therefore be treated as decision support, and users should require evidence-grounded answers. When possible, checking tool invocations and their outputs, so claims can be verified.

Also, MCP-Bench's large-scale runs highlight persistent weaknesses in long-horizon planning and dependency tracking even for state-of-the-art models, reinforcing our recommendation that enterprise deployments require strict evidence grounding and verification.

5.6.4 Environmental Considerations

Finally, there is an environmental perspective. Larger cloud models and long multi-step tool dialogues consume more energy than lightweight solutions. While this thesis does not quantify environmental impact directly, the measured differences in token usage, rounds, and execution time imply that model and workflow choices can significantly affect compute cost. For an energy organisation, this is a particularly relevant trade-off, since efficiency is not only a financial question, but also part of responsible system design.

Chapter 6
Conclusion

This thesis investigated MCP as a layer for connecting large language models to enterprise operational data in energy network contexts. Using E.ON's internal data and publicly available UK Power Networks datasets, we evaluated three language models across 1-server and 2-server tasks.

6.1 Main Findings

Addressing **RQ1**, MCP-generated responses require a multidimensional evaluation that combines protocol-level correctness with evidence-grounded task completion. We found that the most suitable approach integrates rule-based metrics for protocol adherence, LLM-as-a-judge metrics for strategic behaviour, and expected-vs-generated correctness for factual accuracy. Rule-based metrics measure basic competence and identify failures in parameter construction. Judge metrics capture planning quality that rule-based checks miss. Expected-vs-generated correctness provides direct accuracy signals for 1-server tasks with deterministic outputs. Metric choice depends on task characteristics, where 1-server tasks benefit from expected-answer validation, while 2-server coordination relies on judge-based assessment.

To answer **RQ2**, it is important to acknowledge that the 1-server setting had expected vs generated correctness, whereas the 2-server setting did not. Because expected-vs-generated correctness deterministically reflects a model's capability of generating accurate responses through MCP, it is the most sufficient metric to answer **RQ2**. Claude Sonnet 4 shows the best result with regard to that metric, with GPT-4o-mini close behind. A corresponding metric in the 2-server setting was task completion, where GPT-4o-mini showed a slightly better score (4.7). This concludes that both have similar capabilities in generating responses with MCP. Phi-4-mini falls short by a considerable margin, making it unsuitable for generating responses through MCP.

For **RQ3**, performance varies dramatically across task complexity and model capability. Cloud-hosted models showed strong protocol adherence and task completion in 1-server con-

figurations, achieving near-perfect schema understanding and reliable retrieval. 2-server coordination proved substantially harder, with task completion declining significantly despite maintained protocol compliance. This decline seems to stem from planning and dependency management difficulties rather than protocol mechanics, with token consumption increasing dramatically.

The models demonstrate distinct performance-to-efficiency trade-offs. Claude Sonnet 4 achieved the highest quality scores while using fewer tool calls and interaction rounds, showing that planning efficiency correlates with model capability. GPT-4o-mini achieved competitive task completion in 1-server scenarios at much lower cost, though requiring more invocations and consuming more tokens, compensated by faster execution times. Overall, GPT-4o-mini reaches similar results as Claude Sonnet 4 but at one-sixteenth of the price, concluding that it is the best option in terms of cost-efficiency. Phi-4-mini revealed a capability threshold where the MCP tool-use becomes unreliable. Despite simplified conditions without distraction servers, it struggled with schema compliance and parameter construction, indicating systematic difficulties with structured reasoning that small models cannot yet overcome. Cost differences compound in 2-server scenarios where inefficient planning accumulates over many rounds.

Regarding **RQ4**, modularity works through domain-based server separation, where each server encapsulates a distinct data category with its own tools and schemas. We divided operational data into three independent servers, each exposing domain-specific functionality through standardised interfaces. However, modularity introduces coordination costs, where 2-server orchestration demands substantial reasoning capacity, with performance degrading and token consumption increasing significantly in multi-domain workflows. This creates tension between architectural flexibility and operational cost, currently favouring more capable models for complex coordination.

Performance varied substantially between datasets. Private data yielded higher efficiency across both cloud models, requiring fewer tool calls while achieving stronger correctness scores. This pattern stems from backend architecture rather than domain complexity alone. Dremio's strongly typed schemas enabled precise queries, while Opendatasoft's field-level filtering required more exploratory calls. Tool design proved critical, with our domain-based server separation supporting maintainability but introducing coordination costs when workflows crossed server boundaries. The evaluation methodology itself advanced MCP-Bench by adding expected-vs-generated correctness for 1-server tasks, enabling direct factual verification against gold-standard answers while preserving fuzzy task descriptions for 2-server orchestration assessment.

6.1.1 Limitations

Key limitations include restricted model coverage, task diversity limited to mappable schemas across both datasets, and controlled evaluation environments lacking real operational variability. The 2-server evaluation provided only indicative insights due to limited task counts. QA corpus construction introduced potential human bias, and exclusive reliance on LLM-judge scoring without systematic human grading limits bias quantification. The most reliable conclusions are the large qualitative gap between the small local model and cloud models, the consistent pattern that 2-server orchestration is substantially harder than 1-server tool use even when protocol metrics remain high, and the trade-off between structured planning ef-

ficiency and fast exploratory execution.

MCP provides a viable, standardised interface for connecting LLMs to enterprise data. Cloud models achieve reliable performance on 1-server tasks and acceptable performance on moderately complex 2-server workflows. The protocol's modularity enables organisational scaling and flexible evolution, though coordination costs currently favour more capable models for cross-domain tasks. As model capabilities improve and tool design practices mature, MCP-based architectures are positioned to become a practical foundation for AI-augmented enterprise data access.

6.2 Future Work

Several research directions require investigation. Key priorities emerge from observed limitations and unanswered design questions:

Broader Model Coverage: Future work should expand model coverage to include additional families and sizes, particularly stronger local models and small cloud models, to better map capability to cost and latency trade-offs.

Tool Granularity and MCP Server Design: Since this study employed a single tool design strategy with many specific tools with narrow scopes, without empirical comparison to alternatives, future work should evaluate different tool granularity approaches. Such as a few flexible tools with many optional parameters versus many specific tools with minimal parameters. Additionally, repeating the experiments without distraction servers, or by varying how many tools or servers are exposed. This would test whether showing fewer tools improves reliability and identify an "optimal" tool availability for agents. This could establish evidence-based guidelines for MCP server design.

Stronger Evaluation and Validation For 2-server Tasks: Evaluation should incorporate more direct, confirmable outputs in 2-server tasks through consistency checks across retrieved evidence and expected answers, complemented with human verification to provide stronger validity for model comparisons. Also, future work should include testing of the agent's ability to use multiple tools within a 1-server setting. This would help distinguish cross-server coordination costs from general tool-chaining complexity.

Tools With Write Access: The tool set should expand beyond read-only retrieval to include controlled write actions under strict safety constraints, enabling study of permission boundaries, confirmation steps, and rollback strategies when tools have side effects.

Evaluation of Alternative Agent Protocols: Overall, future work should separate improvements achievable through better models from those requiring better system design, as the latter often provides the most actionable path for enterprise adoption.

References

Anthropic (2025a). *Model Context Protocol Documentation*. https://modelcontextprotocol.io/docs/getting-started/intro/ Accessed: 2025-12-29.

Anthropic (2025b). *Model Context Protocol Python SDK*. https://github.com/modelcontextprotocol/python-sdk Accessed: 2025-12-29.

Anthropic (2025c). Model context protocol – official github organization. https://github.com/modelcontextprotocol. Accessed: 2025-12-08.

Anthropic (2025d). Pricing. https://www.claude.com/pricing#api. Accessed: 2025-12-11.

Anthropic (2025e). System card: Claude opus 4 & claude sonnet 4. Technical report, Anthropic. https://www-cdn.anthropic.com/6d8a8055020700718b0c49369f60816ba2a7c285.pdf Accessed: 2025-12-20.

Anthropic, M. (2026). Rename FastMCP to MCPServer. https://github.com/modelcontextprotocol/python-sdk/commit/65c614e48e833df4622b07da698ca64935c31f36. Commit hash: 65c614e48e833df4622b07da698ca64935c31f36. Accessed: 2026-02-02.

Artificial Analysis (2025). Artificial analysis intelligence index. https://artificialanalysis.ai/evaluations/artificial-analysis-intelligence-index Accessed: 2025-12-23.

Avila, C., Ilbay, D., and Rivera, D. (2025). Human–AI Teaming in Structural Analysis: A Model Context Protocol Approach for Explainable and Accurate Generative AI. *Buildings*, 15(17). https://www.mdpi.com/2075-5309/15/17/3190 Accessed: 2025-10-15.

Bai (2022). Constitutional ai: Harmlessness from ai feedback. https://arxiv.org/abs/2212.08073 Accessed: 2026-01-08.

Bavaresco, A., Bernardi, R., Bertolazzi, L., Elliott, D., Fernández, R., Gatt, A., Ghaleb, E., Giulianelli, M., Hanna, M., Koller, A., Martins, A., Mondorf, P., Neplenbroek, V., Pezzelle, S., Plank, B., Schlangen, D., Suglia, A., Surikuchi, A. K., Takmaz, E., and Testoni, A. (2025). LLMs instead of human judges? a large scale empirical study across 20 NLP evaluation tasks. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T., editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 238–255, Vienna, Austria. Association for Computational Linguistics. Accessed: 2025-11-06.

Bick, A., Blandin, A., and Deming, D. J. (2024). The rapid adoption of generative ai. Working Paper 32966, National Bureau of Economic Research.

Bommasani, R., Liang, P., and Lee, T. (2023). Holistic evaluation of language models. *Annals of the New York Academy of Sciences*, 1525(1):140–146. Accessed: 2025-10-10.

Bontridder, N. and Poullet, Y. (2021). The role of artificial intelligence in disinformation. *Data & Policy*, 3:e32.

Brodimas, D., Birbas, A., Kapolos, D., and Denazis, S. (2025). Intent-based infrastructure and service orchestration using agentic-ai. *IEEE Open Journal of the Communications Society*, 6:7150–7168.

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P. S., Yang, Q., and Xie, X. (2024). A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3).

Chatterji, A., Cunningham, T., Deming, D. J., Hitzig, Z., Ong, C., Shan, C. Y., and Wadman, K. (2025). How people use chatgpt. Working Paper 34255, National Bureau of Economic Research.

Chen, G. H., Chen, S., Liu, Z., Jiang, F., and Wang, B. (2024). Humans or LLMs as the judge? a study on judgement bias. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N., editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8301–8327, Miami, Florida, USA. Association for Computational Linguistics. Accessed: 2025-11-07.

Chen, J., Wu, H., Pang, J., Wang, Y., Zhang, D., and Sun, C. (2025). Tool learning with language models: a comprehensive survey of methods, pipelines, and benchmarks. *Vicinagearth*, 2(1):16.

Chiang, C.-H. and Lee, H.-y. (2023). A closer look into using large language models for automatic evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8928–8942. Accessed: 2025-11-10.

Currie, G. M. (2023). Academic integrity and artificial intelligence: is chatgpt hype, hero or heresy? *Seminars in Nuclear Medicine*, 53(5):719–730. Preclinical.

Damonte, M. and Cohen, S. B. (2018). Cross-lingual Abstract Meaning Representation parsing. In Walker, M., Ji, H., and Stent, A., editors, *Proceedings of the 2018 Conference of the*

North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 1146–1155, New Orleans, Louisiana. Association for Computational Linguistics.

Dantas, P. V., Cordeiro, L. C., and Junior, W. S. (2025). A review of state-of-the-art techniques for large language model compression. *Complex & Intelligent Systems*, 11(9):1–40.

Doostmohammadi, E., Holmström, O., and Kuhlmann, M. (2024). How reliable are automatic evaluation methods for instruction-tuned LLMs? In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N., editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6321–6336, Miami, Florida, USA. Association for Computational Linguistics. Accessed: 2025-11-04.

Dorner, F. E., Nastl, V., and Hardt, M. (2025). Limits to scalable evaluation at the frontier: Llm as judge won't beat twice the data. In Yue, Y., Garg, A., Peng, N., Sha, F., and Yu, R., editors, *International Conference on Learning Representations*, volume 2025, pages 26467–26491. `https://proceedings.iclr.cc/paper_files/paper/2025/file/4264ee4376776907c0b87ed70b959585-Paper-Conference.pdf` Accessed: 2025-11-04.

Dremio (2025a). *API Reference*, [26.x] edition. `https://docs.dremio.com/current/reference/api/` Accessed: 2025-10-30.

Dremio (2025b). *Architecture*, [26.x] edition. `https://docs.dremio.com/current/what-is-dremio/architecture/` Accessed: 2025-10-20.

Dublin Core (2026). Metadata basics. `https://www.dublincore.org/resources/metadata-basics/`. Accessed: 2026-01-12.

Elgiganten (2025). Hp elitebook 835 g9 ryzen 5 pro 16gb ram 256gb ssd 13.3". `https://www.elgiganten.se/product/datorer-kontor/datorer/laptop/hp-elitebook-835-g9-ryzen-5-pro-16gb-256gb-ssd-133/627085`. Accessed: 2026-01-25.

E.ON (2025). About us. `https://www.eon.com/en/about-us.html`. Accessed: 2025-09-30.

Fan, S., Ding, X., Zhang, L., and Mo, L. (2025). Mcptoolbench++: A large scale ai agent model context protocol mcp tool use benchmark. `https://arxiv.org/abs/2508.07575`.

Gao, M., Hu, X., Yin, X., Ruan, J., Pu, X., and Wan, X. (2025a). Llm-based nlg evaluation: Current status and challenges. *Computational Linguistics*, 51(2). Accessed: 2025-11-06.

Gao, X., Xie, S., Zhai, J., Ma, S., and Shen, C. (2025b). Mcp-radar: A multi-dimensional benchmark for evaluating tool use capabilities in large language models. `https://arxiv.org/abs/2505.16700`.

Gullí, A. (2025). *Model Context Protocol*, pages 147–162. Springer Nature Switzerland, Cham.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. (2021). Measuring massive multitask language understanding. https://arxiv.org/abs/2009.03300 Accessed: 2025-10-10.

Hou, X., Zhao, Y., Wang, S., and Wang, H. (2025). Model context protocol (mcp): Landscape, security threats, and future research directions. https://arxiv.org/abs/2503.23278.

Huwise (2025). Huwise's explore api reference documentation (v2.1). https://help.opendatasoft.com/apis/ods-explore-v2/explore_v2.1.html. Accessed: 2026-02-09.

Kachris, C. (2025). A survey on hardware accelerators for large language models. *Applied Sciences*, 15(2):586.

Kate, K., Rizk, Y., Ghosh, P., Gulati, A., Chakraborti, T., Wright, Z., and Agarwal, M. (2025). How good are llms at processing tool outputs? https://arxiv.org/abs/2510.15955 Accessed: 2025-11-20.

Kucharavy, A. (2024). *Overview of Existing LLM Families*, pages 31–44. Springer Nature Switzerland, Cham.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks. https://arxiv.org/abs/2005.11401.

Li, H., Dong, Q., Chen, J., Su, H., Zhou, Y., Ai, Q., Ye, Z., and Liu, Y. (2024). Llms-as-judges: A comprehensive survey on llm-based evaluation methods. https://arxiv.org/abs/2412.05579 Accessed: 2025-11-05.

Li, H., Xu, Y., and Hong, T. (2025). Energyplus-mcp: A model-context-protocol server for ai-driven building energy modeling. *SoftwareX*, 32:102367.

Li, X. (2025). A review of prominent paradigms for LLM-based agents: Tool use, planning (including RAG), and feedback learning. In Rambow, O., Wanner, L., Apidianaki, M., Al-Khalifa, H., Eugenio, B. D., and Schockaert, S., editors, *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9760–9779, Abu Dhabi, UAE. Association for Computational Linguistics.

Lin, S., Hilton, J., and Evans, O. (2022). Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 3214–3252. Accessed: 2025-10-10.

Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., Huang, M., Dong, Y., and Tang, J. (2025a). Agentbench: Evaluating llms as agents. https://arxiv.org/abs/2308.03688.

Liu, Z., Qiu, J., Wang, S., Zhang, J., Liu, Z., Ram, R., Chen, H., Yao, W., Heinecke, S., Savarese, S., et al. (2025b). Mcpeval: Automatic mcp-based deep evaluation for ai agent

models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 373–402.

Luo, Z., Shen, Z., Yang, W., Zhao, Z., Jwalapuram, P., Saha, A., Sahoo, D., Savarese, S., Xiong, C., and Li, J. (2025). Mcp-universe: Benchmarking large language models with real-world model context protocol servers. https://arxiv.org/abs/2508.14704.

Microsoft (2025). Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras. https://arxiv.org/abs/2503.01743.

Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., and Gao, J. (2025). Large language models: A survey. https://arxiv.org/abs/2402.06196.

Mo, G., Zhong, W., Chen, J., Chen, X., Lu, Y., Lin, H., He, B., Han, X., and Sun, L. (2025). Livemcpbench: Can agents navigate an ocean of mcp tools? https://arxiv.org/abs/2508.01780.

Mohammadi, M., Li, Y., Lo, J., and Yip, W. (2025). Evaluation and benchmarking of llm agents: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 6129–6139.

Oellrich, A., Collier, N., Smedley, D., and Groza, T. (2015). Generation of silver standard concept annotations from biomedical texts with special relevance to phenotypes. *PLOS ONE*, 10(1):1–17.

OpenAI (2024). Gpt-4 technical report. https://arxiv.org/abs/2303.08774 Accessed: 2026-01-08.

OpenAI (2025). Pricing. https://platform.openai.com/docs/pricing. Accessed: 2025-12-11.

Prefect, jlowin (2026). fastmcp. https://github.com/jlowin/fastmcp?tab=readme-ov-file. Accessed: 2026-02-02.

Qin, Y., Hu, S., Lin, Y., Chen, W., Ding, N., Cui, G., Zeng, Z., Zhou, X., Huang, Y., Xiao, C., Han, C., Fung, Y. R., Su, Y., Wang, H., Qian, C., Tian, R., Zhu, K., Liang, S., Shen, X., Xu, B., Zhang, Z., Ye, Y., Li, B., Tang, Z., Yi, J., Zhu, Y., Dai, Z., Yan, L., Cong, X., Lu, Y., Zhao, W., Huang, Y., Yan, J., Han, X., Sun, X., Li, D., Phang, J., Yang, C., Wu, T., Ji, H., Li, G., Liu, Z., and Sun, M. (2024). Tool learning with foundation models. *ACM Comput. Surv.*, 57(4).

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. (2023). ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. https://arxiv.org/abs/2307.16789.

Ray, P. P. (2025). A survey on model context protocol: Architecture, state-of-the-art, challenges and future directions. http://dx.doi.org/10.36227/techrxiv.174495492.22752319/v1.

Schick, T., Dwivedi-Yu, J., Dessí, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Schloegel, M., Bars, N., Schiller, N., Bernhard, L., Scharnowski, T., Crump, A., Ale-Ebrahim, A., Bissantz, N., Muench, M., and Holz, T. (2024). Sok: Prudent evaluation practices for fuzzing. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 1974–1993.

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., et al. (2023). Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023(5):1–95.

Tan, S., Zhuang, S., Montgomery, K., Tang, W. Y., Cuadron, A., Wang, C., Popa, R. A., and Stoica, I. (2025). Judgebench: A benchmark for evaluating llm-based judges. `https://arxiv.org/abs/2410.12784` Accessed: 2025-11-04.

UK Power Networks Open Data (2025). *Explore API*, v2.1 edition. `https://ukpowernetworks.opendatasoft.com/api/explore/v2.1/console` Accessed: 2025-10-17.

UTCP Contributors (2025a). *Universal Tool Calling Protocol (UTCP)*. `https://www.utcp.io/` Accessed: 2025-12-03.

UTCP Contributors (2025b). Universal tool calling protocol (utcp) – github repository. `https://github.com/universal-tool-calling-protocol`. Accessed: 2025-12-08.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. (2024a). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., Li, T., Ku, M., Wang, K., Zhuang, A., Fan, R., Yue, X., and Chen, W. (2024b). Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C., editors, *Advances in Neural Information Processing Systems*, volume 37, pages 95266–95290. Curran Associates, Inc.

Wang, Z., Chang, Q., Patel, H., Biju, S., Wu, C.-E., Liu, Q., Ding, A., Rezazadeh, A., Shah, A., Bao, Y., and Siow, E. (2025). Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers. `https://arxiv.org/abs/2508.20453`.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Wu, Z., Liu, X., Zhang, X., Chen, L., Meng, F., Du, L., Zhao, Y., Zhang, F., Ye, Y., Wang, J., Wang, Z., Ni, J., Yang, Y., Xu, A., and Shieh, M. Q. (2025). Mcpmark: A benchmark for stress-testing realistic and comprehensive mcp use. https://arxiv.org/abs/2509.24002.

Xu, W., Huang, C., Gao, S., and Shang, S. (2025). LLM-Based Agents for Tool Learning: A Survey. *Data Science and Engineering*, pages 1–31.

Yang, Y., Wu, D., and Chen, Y. (2025). Mcpsecbench: A systematic security benchmark and playground for testing model context protocols. https://arxiv.org/abs/2508.13220.

Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. (2024). τ-bench: A benchmark for tool-agent-user interaction in real-world domains. https://arxiv.org/abs/2406.12045.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). React: Synergizing reasoning and acting in language models. Publisher Copyright: © 2023 11th International Conference on Learning Representations, ICLR 2023. All rights reserved.; Conference date: 01-05-2023 Through 05-05-2023.

Ye, J., Li, G., Gao, S., Huang, C., Wu, Y., Li, S., Fan, X., Dou, S., Ji, T., Zhang, Q., Gui, T., and Huang, X. (2025). ToolEyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. In Rambow, O., Wanner, L., Apidianaki, M., Al-Khalifa, H., Eugenio, B. D., and Schockaert, S., editors, *Proceedings of the 31st International Conference on Computational Linguistics*, pages 156–187, Abu Dhabi, UAE. Association for Computational Linguistics.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. (2023). Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623. Accessed: 2025-11-05.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., and Neubig, G. (2024). Webarena: A realistic web environment for building autonomous agents. https://arxiv.org/abs/2307.13854.

Zhu, L., Wang, X., and Wang, X. (2025). Judgelm: Fine-tuned large language models are scalable judges. https://arxiv.org/abs/2310.17631. Accessed: 2025-11-04.

Appendices

Appendix A
Prompts

A.1 Task Prompt - 1-Server

Prompt For Generating 1-Server Tasks

Generera frågor
Skapa 10 frågor baserat på beskrivningarna nedan. 3 av frågorna ska vara runt 20 ord, 3 ska vara runt 60 ord och 4 ska vara runt 40 ord. Skriv även en beroendeanalys enligt beskrivningen under. För varje förväntat svar ska 10 frågor genereras och för varje svar ska en beroendeanalys genereras. Allt ska vara på svenska.
Fuzzy uppgiftsbeskrivning:
KRITISKT: SKAPA EN VERKLIGT NATURLIG FÖRFRÅGAN
ABSOLUT FÖRBJUDET: ALL strukturerad språkform som ser ut som en uppgiftsbeskrivning Fraser som "Jag behöver analysera", "Jag vill jämföra", "Vänligen utvärdera" ALLA specifika server-/plattformnamn (arXiv, PubMed, Yahoo Finance, Google Maps, etc.) ALLA verktygsnamn eller tekniska implementeringsdetaljer Listor, numreringar eller steg-för-steg-instruktioner Formellt uppgiftsspråk ("utföra", "genomföra", "verkställa", "implementera") INGEN uppgiftsbeskrivning ska överstiga 60 ord.
ISTÄLLET, SKAPA EN NATURLIG KONVERSATION: Börja med ett sammanhang eller ett problem användaren står inför Använd konversationella öppningar: "Jag försöker lista ut...", "Har funderat på...", "Har en situation här..." Inkludera osäkerhet: "inte säker på om", "kanske", "möjligen", "kan vara" Lägg till personlig kontext: "för mitt projekt", "min chef frågade", "jag är nyfiken på" Uttryck behovet genom en berättelse eller situation, inte en uppgiftslista
DÖLJ UPPGIFTSSTRUKTUREN HELT: Säg inte: “Ge mig vilka rättigheter som identifieraren grid-and-primary-sites-epr-data har.” Säg istället: “Jag har funderat på att ta reda på behörigheter för data. Det är till för att öka min kunskap. Vad för

behörighet har kännetecknet grid-and-primary-sites-epr-data?"
Säg inte: "Vad har substation med alias 08468 för spänningsnivå?" Säg istället: "Jag ska hålla en presentation nästa vecka om spänningsnivåer på olika elstationer. Vad är den senaste nivån för station 08468?"
Säg inte: "Vad är storage för generation (i procent) för 2019-11-01?" Säg istället: "Den första november för 6 år sedan så minns jag att det gjordes en mätning på genererad lagring. Jag kan inte hitta just de mätvärdena. Kan du hitta det åt mig?"
BEHÅLL VIKTIGA DATA NATURLIGT: Bädda in specifika värden konversationellt: "runt 150 eller så", "någonstans nära Malmö" Använd ungefärliga uttryck när det passar: "ungefär", "cirka", "någonstans mellan" Behåll exakta värden bara när det verkligen behövs (beräkningar, ID:n, osv.)
FÅ DET ATT LÅTA SOM EN RIKTIG PERSON: Använd utfyllnadsord sparsamt: "egentligen", "typ", "du vet" Visa känsla eller angelägenhet när det passar: "verkligen behöver veta", "det här har stört mig" Ställ frågor naturligt: "Vad tycker du?", "Låter det rimligt?", "Övertänker jag detta?"
KRITISKT: Avsluta naturligt med krav på bevis invävda i samtalet: Istället för: "Vänligen ge evidensbaserad analys med konkreta data" Säg: "Jag behöver verkligen faktiska data på det här – kan inte gå till min chef med bara åsikter. Vad du än hittar, se till att det är stödd av riktiga siffror eller tillförlitliga källor, okej?"
ANVÄND ALLTID relativa datum/tidsangivelser (t.ex. "nästa 7 dagar", "senaste 3 månaderna", "kommande vecka" inte "januari 2024" eller "2024-01-15")
Returnera ENDAST den naturliga, konversationella mjuka beskrivningen – få det att låta som en riktig person som ber om hjälp, inte en robot som utför en uppgift.
Beroendeanalys:
A) INNEBOENDE beroenden (verktygens naturliga in-/utdatarelationer): Vilka verktyg som naturligt producerar data som andra använder Standardiserade arbetsflöden (sök → hämta → analysera)
B) SCENARIOBASERADE beroenden (skapa logiska kopplingar för din uppgift), till exempel: Verktyg A:s resultat avgör vilket verktyg som används härnäst Verktyg B:s utdata bestämmer parametrarna för Verktyg C Verktyg D validerar eller motsäger Verktyg E:s slutsatser Parallella verktyg vars resultat måste kombineras Iterativa slingor där resultat utlöser ny analys Dokumentera beroendeanalysen i ett fält kallat "dependency_analysis" som beskriver: Centrala verktygskedjor och dataflöde Viktiga beslutspunkter Parallella vs sekventiella krav Beroenden mellan servrar (för uppgifter som involverar flera servrar)
C) Det exakta förväntade svaret ska vara med. Inget ska ändras i det. D) Den exakta toolen som ska användas måste vara med. Inget ska ändras i det.
Tools och förväntade svar:
Tool: live_faults_description_for_reference Förväntat svar: Beskrivning (incident-categorycustomerfriendlydescription) för INCD-104901-C är: "We're carrying out planned work in your area. For our engineers to carry it out safely they need to turn the power off. We're doing this work as it's essential to provide reliable electricity supplies to your area. We're sorry for any inconvenience caused and thank you for your patience."

A.2 Task Prompt - 2-Server

Prompt For Generating 2-Server Orchestrated Tasks

You are a task designer for testing AI agents with MCP tools.
STEP 1: ANALYZE AND CREATE TOOL DEPENDENCIES
Analyze these available tools and CREATE meaningful dependencies for your task scenario:
[Tool descriptions will be inserted here]
Consider both:
A) INNEBOENDE beroenden (verktygens naturliga in-/utdatarelationer)

- Vilka verktyg som naturligt producerar data som andra använder
- Standardiserade arbetsflöden (sök → hämta → analysera)

B) SCENARIOBASERADE beroenden (skapa logiska kopplingar för din uppgift), till exempel:

- Verktyg A:s resultat avgör VILKET verktyg som används härnäst
- Verktyg B:s utdata bestämmer PARAMETRARNA för Verktyg C
- Verktyg D validerar eller motsäger Verktyg E:s slutsatser
- Parallella verktyg vars resultat måste KOMBINERAS
- Iterativa slingor där resultat utlöser NY analys

Dokumentera din beroendeanalys i ett fält kallat "dependency_analysis" som beskriver:

- Centrala verktygskedjor och dataflöde
- Viktiga beslutspunkter
- Parallella vs sekventiella krav
- Beroenden mellan servrar (för uppgifter som involverar flera servrar)

C) Den exakta toolen som ska användas måste vara med. Inget ska ändras i det.
Record your dependency analysis in a "dependency_analysis" field that describes:

- Key tool chains and data flow
- Critical decision points
- Parallel vs sequential requirements
- Cross-server dependencies (for 2-server tasks)

For 2-server tasks, create CROSS-SERVER dependencies:

- Server A data influences Server B queries
- Cross-validation between different data sources
- One server's limits trigger fallback to another

STEP 2: DESIGN ONE COMPLEX TASK
Based on your dependency analysis, create ONE task that:

- Create MAXIMUM complexity requiring massive tool calls
- Must use tools from all available servers
- Must consider inter-servers dependency if more than 1 server available

You may create the tasks with the following properties if suitable:

- Deep dependency chains where Tool B needs Tool A's output, Tool C needs B's output, etc.
- Multiple decision branches based on intermediate results
- Iterative refinement: initial findings lead to deeper investigation
- Cross-validation: use multiple tools to verify critical findings
- Data transformation: output from one tool needs processing before next tool
- Conditional workflows: if condition X, then workflow Y, else workflow Z

CRITICAL DATA REQUIREMENTS:

1. ALL tasks MUST be self-contained and executable WITHOUT any external dependencies
2. NEVER reference external resources like:
    - URLs (like "https://api.example.com" or any external API)
    - Local files (like "user-management.yaml" or "config.json")
    - Databases or external systems
    - "Our API", "our system", "our database"
3. ALL data must come from either:
    - The provided tools themselves (what they can generate/fetch/calculate)
    - Concrete values you specify in the task (numbers, names, parameters)
4. NEVER use vague references:
    - "user-provided parameters" or "user-specified"

- "fetched from database" or "retrieved from external source"
- "based on user preferences" or "according to input"
- "specified location/value" or "to be determined"

5. ALWAYS provide concrete values:
   - Specific numbers (e.g., "analyze heat exchanger with inlet temp 80°C, outlet 60°C, flow rate 0.5 kg/s")
   - Named entities (e.g., "analyze weather in San Francisco" not "specified city")
   - For locations: Use city names, landmark names, or general areas, NOT specific street addresses
     - GOOD: "San Francisco", "Times Square", "Central Park area", "downtown Seattle"
     - BAD: "123 Main Street", "456 Park Avenue", specific house numbers or street addresses
   - Exact thresholds (e.g., "alert if efficiency drops below 85%" not "desired threshold")
   - ALWAYS USE relative dates/times (e.g., "next 7 days", "past 3 months", "upcoming week" not "January 2024" or "2024-01-15")
6. If the task involves analysis, provide ALL input data in the task description:
   - For calculations: provide all numbers, formulas, and units needed
   - For searches: provide specific search terms and criteria
   - For comparisons: provide specific items with their properties
   - For optimization: provide current values and target metrics

Requirements:

1. MUST require multiple tools in a specific sequence
2. Tool B should need output from Tool A (dependency chain)
3. Include decision points based on intermediate results
4. Be realistic and valuable for business/research purposes
5. Define expected analysis and output format
6. Task must be immediately executable - agent should never need to ask for more information
7. Task should be executable and solvable by using the provided tools. You need to pay attention to the function and the output of the provided tools.

Output Format:
Output ONLY a JSON object (not an array). ALWAYS USE relative dates/times (e.g., "next 7 days", "past 3 months", "upcoming week" not "January 2024" or "2024-01-15"):

```
{
  "task_id": "task_XXX",
  "task_description": "detailed task that leverages
                       the identified tool dependencies",
  "dependency_analysis": "Your analysis from STEP 1 -
                          describe the key dependencies,
                          tool chains, decision points,
                          and data flow patterns that
                          this task requires"
}
```

Focus on creating a task that CANNOT be completed without understanding tool dependencies.

Appendix B

QA Corpus Samples From Public Dataset

B.1 Metadata QA Samples

Sample 1

Question: Jag försöker få koll på vem som faktiskt står som skapare bakom ukpn-licence-boundaries. Är det UK Power Networks?
Expected answer: Skaparen av metadata som tillhör identifieraren ukpn-licence-boundaries är UK Power Networks, Company number 3870728.

B.2 Grid Infrastructure and Asset Information QA Samples

Sample 2

Question: Har en situation här: vi synkar geodata och måste få rätt på 74275. Kan du bekräfta byggnadsadressfältet (om något), samt ge postkod, latitud och longitud exakt? Jag behöver verkliga fältvärden så vi slipper korrigera kartan igen de kommande dagarna.
Expected answer: Platsdata för alias 74275 är: buildingaddress: ATRIUM HEIGHTS 4 LITTLE THAMES WALK LONDON, postcode: SE8 3BZ, latitude: 51.480555, longitude: -0.020352.

B.3 Metering QA Samples

Sample 3

Question: Jag ska knyta ihop en kort notis för vår interna uppföljning, och jag vill bara få svart på vitt när hushållens efterfrågan senast låg över tio procent. Gissar att det rör sig om några täta datapunkter under samma kväll, men kanske inte. Kan du ge mig de senaste tiderna (UTC) och sådant jag kan peka på? Jag behöver faktiska data eller källor för att inte skjuta från höften, okej?
Expected answer: De senaste timestamps där domestic electricity demand överstiger 10% (UTC): 2019-12-31T23:30:00+00:00 — 72.46%, 2019-12-31T23:00:00+00:00 — 74.76%, 2019-12-31T22:30:00+00:00 — 72.64%, 2019-12-31T22:00:00+00:00 — 68.35%, 2019-12-31T21:30:00+00:00 — 67.36%.

B.4 2-Server Question Sample

Sample 4

Question: Jag jobbar med datasetet ukpn-licence-boundaries men är osäker på om det är publicerat eller uppdaterat. Kan du först ta fram utgivningsår och fem tidiga stationer med deras spänningsnivå, typ och licensområde? Sen vill jag veta det angivna tröskelvärdet, antal stationer över det och senaste uppdateringstid. Slutligen, matchar utgivaren stationernas ägare? Jag behöver exakta siffror och källor. Snälla försäkra att allt du hittar finns och stöds av konkret data och verifierbara källor. Jag behöver specifika nummer och bevis, inte generella svar.

Appendix C

Supplementary Evaluation Tables

C.1 Detailed Per-Server Results for 1-Server on Private Datasets

	Claude Sonnet 4			GPT-4o-mini			Phi-4-mini		
Metric	**Server A**	**Server B**	**Server C**	**Server A**	**Server B**	**Server C**	**Server A**	**Server B**	**Server C**
Rule-based									
Valid tool name	1.000	0.995	1.000	0.995	1.000	0.970	1.000	0.942	0.967
Input schema compliance	1.000	1.000	1.000	1.000	1.000	1.000	0.667	0.615	0.975
Execution success	0.997	0.981	0.997	0.995	0.978	0.970	0.667	0.673	0.917
LLM Judge									
Task fulfillment	0.973	0.985	0.998	0.895	0.892	0.993	0.174	0.317	0.491
Information grounding	0.924	0.963	0.976	0.935	0.972	0.990	0.240	0.358	0.561
Expected vs generated correctness	0.985	0.998	0.999	0.895	0.893	0.994	0.171	0.328	0.500
Tool appropriateness	0.780	0.835	0.938	0.806	0.702	0.767	0.177	0.293	0.352
Parameter accuracy	0.945	0.975	0.998	0.919	0.887	0.942	0.202	0.305	0.550
Dependency awareness	0.915	0.941	0.997	0.867	0.843	0.879	0.171	0.310	0.516
Parallelism & efficiency	0.695	0.725	0.839	0.752	0.651	0.615	0.178	0.258	0.281

Table C.1: 1-server (servers A–C) for private dataset - Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

C.2 Detailed Per-Server Results for 1-Server on Public Dataset

	Claude Sonnet 4			GPT-4o-mini			Phi-4-mini		
Metric	**Server A**	**Server B**	**Server C**	**Server A**	**Server B**	**Server C**	**Server A**	**Server B**	**Server C**
Rule-based									
Valid tool name	1.000	1.000	1.000	0.998	0.987	0.996	1.000	0.836	1.000
Input schema compliance	1.000	1.000	1.000	1.000	1.000	1.000	0.857	0.618	0.426
Execution success	1.000	0.994	1.000	0.998	0.986	0.995	0.714	0.629	0.548
LLM Judge									
Task fulfillment	0.888	0.934	0.921	0.914	0.880	0.880	0.340	0.294	0.295
Information grounding	0.951	0.972	0.971	0.979	0.980	0.972	0.403	0.443	0.247
Expected vs generated correctness	0.919	0.949	0.932	0.908	0.875	0.893	0.350	0.289	0.304
Tool appropriateness	0.706	0.636	0.873	0.741	0.609	0.725	0.253	0.289	0.158
Parameter accuracy	0.934	0.921	0.970	0.904	0.981	0.939	0.416	0.294	0.195
Dependency awareness	0.889	0.882	0.944	0.871	0.880	0.871	0.350	0.319	0.225
Parallelism & efficiency	0.659	0.519	0.694	0.653	0.521	0.564	0.251	0.261	0.155

Table C.2: 1-Server (servers A–C) for public dataset - Claude Sonnet 4 vs GPT-4o-mini vs Phi-4-mini

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2026-03-03

EXAMENSARBETE Evaluating LLMs as Agents with the Model Context Protocol in the Energy Sector
STUDENTER Jona Waldfogel, Jonathan Giegold
HANDLEDARE Marcus Klang (LTH)
EXAMINATOR Jacek Malec (LTH)

AI-assistenter hos elnätsföretag: att hitta information i en höstack

POPULÄRVETENSKAPLIG SAMMANFATTNING **Jona Waldfogel, Jonathan Giegold**

Energibolag har mycket data, men den är ofta svår att nå utan teknisk kunskap och flera system. I detta examensarbete utvärderas om språkmodeller kan ge faktabaserade svar genom att kopplas till energidata via Model Context Protocol (MCP).

Stora organisationer samlar data om nätstruktur, mätvärden och driftstatus. Informationen är värdefull men ligger ofta i olika databaser med olika format och kräver teknisk kunskap samt rätt behörigheter, vilket gör att enkla frågor blir tidskrävande att besvara.

Språkmodeller kan fungera som ett vardagligt gränssnitt där användaren ställer frågor i naturligt språk. Problemet är att modellerna saknar tillgång till intern data och kan fabricera felaktiga svar. För att bli användbara i företagsmiljö måste de hämta fakta från datakällor på ett kontrollerat sätt.

Här undersöks MCP, en öppen standard som kopplar språkmodeller till externa verktyg via tydliga gränssnitt. Istället för direkt databasåtkomst anropar modellen avgränsade verktyg på separata MCP-servrar som returnerar strukturerade svar. Vi byggde en prototyp med tre MCP-servrar för energidata och testade privat E.ON-data samt publik elnätsdata. Tre språkmodeller jämfördes, varav två molnbaserade (Claude Sonnet 4 och GPT-4o-mini) och en mindre lokal modell (Phi-4-mini).

För att utvärdera systemet genererades två typer av uppgifter. Först naturliga språkfrågor som kunde besvaras med data från en server, därefter uppgifter som krävde att information kombinerades från två servrar.

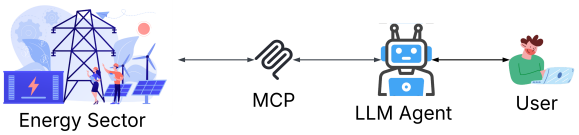

Figure 1: Övergripande arkitektur för implementeringen

Molnmodellerna klarade enkla uppgifter inom en server väl, med nästan perfekt protokollhantering och hög faktakorrekthet. När flera verktyg från olika servrar måste kombineras sjönk kvaliteten kraftigt samtidigt som antalet verktygsanrop och kontextmängden ökade. Claude Sonnet 4 visade bättre planering med färre onödiga iterationer, medan GPT-4o-mini kompenserade genom snabbare exekvering. Den lokala modellen presterade genomgående sämre.

Studien visar att modulär uppdelning av data på separata servrar underlättar underhåll, men att koordinering mellan servrar är utmanande. Slutsatsen är att MCP är en lovande standard för modulära AI-assistenter i företagsmiljö. Enkla frågor kan besvaras redan idag, men komplexa flerstegsfrågor kräver extra styrning och tydliga verktyg för att balansera kvalitet, kostnad och komplexitet.