JSPLIT: A Taxonomy-based Solution for Prompt Bloating in Model Context Protocol

Emanuele Antonioni[1], **Stefan Marković**[1], **Anirudha Shankar**[1] **Jaime Bernardo**[1] **Lovro Markovic**[1]
Silvia Pareti[2] **Benedetto Proietti**[1]

[1]Janea Systems
[2]BigFilter.ai
emanuele@janeasystems.com, stefan@janeasystems.com, anirudha@janeasystems.com, jaime@janeasystems.com,
lovro@janeasystems.com, silvia.pareti@bigfilter.ai, benedetto.proietti@janeasystems.com

Abstract

AI systems are continually evolving and advancing, and user expectations are concurrently increasing, with a growing demand for interactions that go beyond simple text-based interaction with Large Language Models (LLMs). Today's applications often require LLMs to interact with external tools, marking a shift toward more complex agentic systems. To support this, standards such as the Model Context Protocol (MCP) have emerged, enabling agents to access tools by including a specification of the capabilities of each tool within the prompt. Although this approach expands what agents can do, it also introduces a growing problem: prompt bloating. As the number of tools increases, the prompts become longer, leading to high prompt token costs, increased latency, and reduced task success resulting from the selection of tools irrelevant to the prompt. To address this issue, we introduce JSPLIT, a taxonomy-driven framework designed to help agents manage prompt size more effectively when using large sets of MCP tools. JSPLIT organizes tools into a hierarchical taxonomy and uses the user's prompt to identify and include only the most relevant tools, based on both the query and the taxonomy structure. In addition to optimizing prompt composition, the taxonomy introduces an additional layer of control and diagnostic transparency, enabling developers to trace tool selection decisions, analyze categorization logic, and systematically debug tool misclassification or over-selection events. This structural visibility allows for fine-grained interpretability of the agent's decision-making process, enhancing reliability in multi-tool environments. In this paper, we describe the design of the taxonomy, the tool selection algorithm, and the dataset used to evaluate JSPLIT. Our results show that JSPLIT significantly reduces prompt size without significantly compromising the agent's ability to respond effectively. As the number of available tools for the agent grows substantially, JSPLIT even improves the tool selection accuracy of the agent, effectively reducing costs while simultaneously improving task success in high-complexity agent environments.

Introduction

During the past year, artificial intelligence has undergone a remarkable shift. What was once defined largely by conversational tools like Large Language Models (LLMs) to respond to user questions in a one-to-one exchange is now evolving into something more complex and dynamic: AI agents. AI agents are fundamentally different from normal conversational AIs (Zhu et al. 2025): they're built to take action on their own using a variety of tools. They can plug into APIs, search databases, fill out spreadsheets, work within CRM systems, move through cloud environments, and much more, without waiting for a human to guide every step. AI agents do not only answer questions, but handle entire autonomous workflows, making decisions and adapting in real time to environmental state changes (Yao et al. 2023).

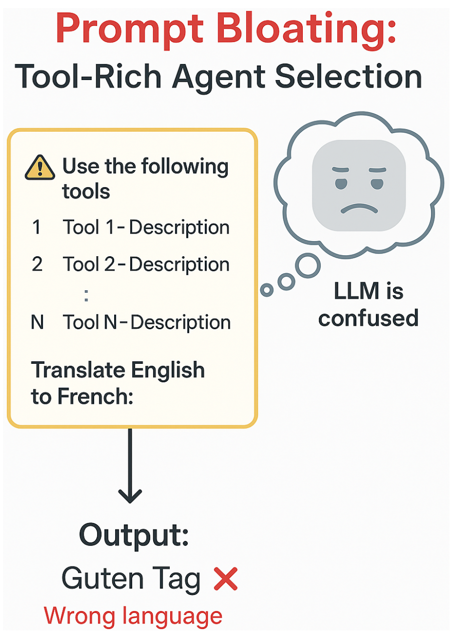

Figure 1: An example of prompt bloating caused by a massive amount of tools descriptions injected into the agent's context

This has been framed by several enterprise investigations. According to the PwC 2025 report (Priest 2025), three out of four business leaders believe that AI agents could bring about a disruptive transformation, stating that they can even overcome the impact of smartphones and the internet. In the same research, more than 80% of companies plan to increase their investments in agent-based systems over the next two years. A similar research carried out by Capgemini in July 2025 (Thibaud et al. 2025) supports this momentum. They found that 14% of organizations around the world have already deployed AI agents in live environments, while another 23% are piloting them.

What is emerging is a new model for how humans and machines work together. In this new paradigm, the AI agent can make autonomous decisions even without the full control of the human operator. These systems can detect changes in context, decide what actions to take, and even collaborate with other agents when necessary.

Following the increasing interest of the entire community around the integration of AI agents, different dedicated tools and technologies arose in the latest year. Among these, the Model Context Protocol (MCP) (Anthropic 2024) has gained significant traction, supported by major partners such as OpenAI, Microsoft, and Google. The Model Context Protocol serves as a universal interface for connecting LLMs with external data sources and tools. It eliminates the need to develop custom integrations for each model-tool combination, instead offering a standardized and secure connection based on JSON-RPC 2.0. The protocol defines a client-server architecture: the AI agent (MCP client) connects to one or more MCP servers that expose functionality via contextual metadata and predefined functions.

This new instrument brings its own set of challenges. One structural problem of this approach is prompt bloating (Figure 1). Prompt bloating (Addad and Kapusta 2024; Long et al. 2025) is a phenomenon that emerges frequently in the practical use of AI, particularly when trying to extend the conversational context or deliver complex instructions through increasingly long textual inputs due to the context injection of tools' descriptions. While this approach makes it possible to maintain a better control on the behavior of the model, it also introduces a number of significant negative effects, both in terms of computational and token costs, and the quality of the responses generated. One of the main disadvantages concerns the increase in computational cost. LLM models typically have a computational complexity quadratic to the length of the prompt. This means that doubling the length of the prompt can quadruple the time and memory required to process it. In addition to the infrastructural impact, prompt bloating also has direct economic consequences when using models accessible through paid APIs. In these cases, the cost of inference is generally calculated based on the total number of tokens processed. As a result, an unnecessarily long prompt not only increases computational requirements but also proportionally affects the actual price of the interaction. Qualitatively, prompt bloating tends to worsen the accuracy and relevance of model responses. With an overly long prompt, the model is exposed to information that is potentially contradictory, obsolete, or simply irrelevant to the current task. This cognitive overload compromises the model's ability to focus on important details, leading to worse performance in terms of general response and tool selection accuracy. Nowadays, users adopt many pragmatic solutions for handling long prompts. Among these, one of the most popular is periodic context summarization (Wang et al. 2024), in which portions of previous text or dialogues are automatically summarized to reduce the number of tokens. Other strategies include the use of sliding windows (Li, Li, and Zhang 2024) to maintain only the most recent and relevant context, and semantic prioritization, which dynamically selects the most relevant information with respect to the current task. These techniques, even if effective, cannot be directly applied to AI agents, because the tools' descriptions are often already short and hard to summarize, and it is impossible to cut out parts of the description using an agnostic approach without seriously compromising the agent's chance of selecting the right tool.

To effectively mitigate the problem of prompt bloating in AI agents interfacing with a large number of MCP servers, we introduce JSPLIT, a novel system designed to optimize context selection through intelligent taxonomy-based filtering. The core idea behind JSPLIT is to organize all available MCP servers into a hierarchical taxonomy, where each taxonomy class is associated with a human-readable description that captures its functional scope. This taxonomy is pre-integrated into the system and serves as a structural map to guide contextual pruning. When a user query is received, the large language model embedded in the agent evaluates the query against the set of class descriptions in the taxonomy. Based on semantic relevance, it selects only those taxonomy classes that are pertinent to the query. The system then filters the MCP server pool accordingly, selecting only the servers that belong to the identified classes. These only the selected servers are included in the agent's execution context. The hierarchical taxonomy also enhances debugging, traceability, and observability of the agent's behavior. By maintaining explicit relationships between tool classes, functions, and their corresponding activation triggers, developers can trace how and why a particular tool was selected for execution. This structured representation allows for fine-grained inspection of decision paths, identification of taxonomy-level misclassifications, and analysis of semantic drift during tool selection. Furthermore, it enables the integration of diagnostic instrumentation—such as tool-level logs and class activation metrics—directly within the taxonomy graph, providing a transparent and auditable view of the agent's reasoning process. This additional control layer improves interpretability, facilitates systematic debugging, and supports iterative refinement of both taxonomy design and agent behavior.

Based on this framework, in this paper we introduce the following contributions:

- A set of different taxonomies to divide the MCP servers
- An evaluation dataset of thousands of MCP servers classified according to the taxonomies
- The JSPLIT system used for selecting the servers according to the user's query
- A test dataset that connects queries with the correct server to use for completing the task

Related Work

The growing adoption of AI agents capable of interfacing with external tools and services has introduced pressing challenges around context management. As these agents are tasked with increasingly complex and diverse operations, the volume of supporting information required for effective reasoning and decision-making can quickly lead to severe prompt bloating. In response, recent research has explored strategies for mitigating context overload, particularly in the areas of task selection and scalable prompt construction within modular AI agent systems. A key benchmark in this space is introduced in (Roberts, Han, and Albanie 2025), where the authors evaluate large language models' (LLMs) ability to retrieve and reason over information spread across contexts approaching a million tokens. Their work focuses on needle threading tasks—scenarios in which models must follow long chains of related information buried within vast prompt windows. They demonstrate that retrieval performance tends to degrade as context size increases, underscoring a fundamental limitation that motivates the core problem addressed in this paper. To address the challenge of effective tool selection, (Kachuee et al. 2025) propose enhancing retrieval through LLM-based query generation. Rather than depending exclusively on dense retrievers or embedding similarity, their approach leverages the LLM's own contextual reasoning abilities to generate more targeted queries. They explore zero-shot prompting, supervised fine-tuning, and alignment learning—ultimately showing that alignment learning provides robust performance, particularly in out-of-domain scenarios. Their findings suggest that LLMs can serve not only as language generators but also as intelligent retrievers of their own operational context. Building on this idea, our work introduces a taxonomy-based selection mechanism that further improves retrieval success rates in long prompts while also reducing token usage throughout the agent's operation. Complementary approaches to tool selection have also been explored. (Gao et al. 2024) present Confucius, a tool-learning framework designed to enhance LLMs' ability to interact with complex tools in real-world tasks. Their curriculum-based training procedure—consisting of warm-up, in-category, and cross-category phases—gradually teaches tool use while refining the training data via introspective feedback (ISIF), which helps the model focus on nuanced tool behaviors. Another related line of work is ToolkenGPT, introduced by (Hao et al. 2023), which takes a different approach by learning new embeddings to support tool selection. Their framework augments frozen language models with a wide array of external tools by learning tool embeddings that integrate into the generation process. ToolkenGPT employs a two-stage training strategy that aligns tool usage with the language modeling objective, enabling parameter-efficient inference without modifying the base model's weights. Unlike these approaches, which require model fine-tuning or embedding updates, our method (JSPLIT) operates post-training without altering the LLM's parameters. This makes our approach more modular and broadly applicable, especially in settings where model weights are inaccessible or frozen.

Method

JSPLIT is an intelligent agent framework designed to run AI agents by combining large language model (LLM) reasoning with external tool execution through a structured and explainable control pipeline. Users provide textual queries or task descriptions, which JSPLIT resolves by orchestrating interactions between an LLM and a suite of tool servers that connect the system to the external world. At the core of JSPLIT is the Taxonomy-MCPResolver, a module responsible for intelligently selecting the most relevant tool servers based on a semantic classification of the user's query. This selective routing enables efficient and context-aware execution of complex tasks by ensuring that only the appropriate tools are invoked during the resolution process. In real-world applications, JSPLIT is expected to be integrated into general-purpose AI orchestration frameworks that support tool-augmented reasoning. These frameworks facilitate iterative interactions between LLMs and external tools, allowing complex tasks to be decomposed, delegated, and refined until a satisfactory result is achieved.

System Overview

JSPLIT is a modular framework for AI agents that combines LLM reasoning with access to large pools of external services, known as Model Context Protocol (MCP) servers. It enables efficient and scalable tool use through intelligent server selection and iterative query resolution. When a user query enters the system, the Taxonomy-MCPResolver determines which MCP servers are relevant. It uses a hierarchical taxonomy that classifies servers by functionality and matches the query to human-readable class descriptions. Only the servers mapped to the most semantically appropriate categories are retained for further processing. The selected MCP servers and the query are then passed to the LLM, which attempts to resolve the task through an iterative process called the call loop. At each step, the LLM decides whether to directly answer the query based on its current context or to invoke tools on the selected MCP servers. If tools are invoked, their outputs are appended to the prompt, updating the context for the next iteration. The loop continues until the LLM produces an answer or a maximum number of iterations is reached. The final output includes the LLM's answer (if one was generated), the list of MCP servers used, and token usage statistics. JSPLIT also supports a baseline mode in which the Taxonomy-MCPResolver is replaced with a Passthrough-MCPResolver. In this configuration, no filtering is applied, and all MCP servers are made available to the LLM. This allows performance comparisons between intelligent and unfiltered tool selection strategies. In the next section we will explore more in detail the core component of the JSPLIT system: the Taxonomy-MCPResolver

Taxonomy Resolver

This component performs intelligent selection of Model Context Protocol (MCP) servers by combining a hierarchical taxonomy of the servers with the large language model classification. The resolver is instantiated with a configuration file defining available MCP servers, a JSON-formatted taxonomy file, and a client interface to a large language model (LLM). The core resolution logic follows a two-phase sequence: taxonomy classification and MCP selection.

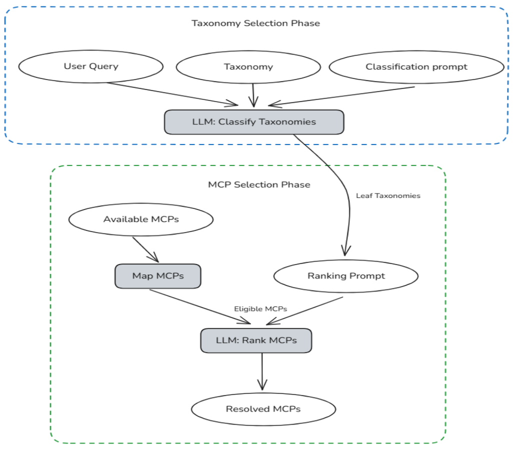

Figure 2: The overall logic of the Taxonomy-MCPResolver component

In the taxonomy classification phase, the system first preprocesses the taxonomy by filtering it to retain only categories that have at least one associated MCP in the current agent's state, then formats the remaining structure into a hierarchical string structured as a tree. It constructs a classification prompt by inserting this formatted taxonomy into a predefined template that instructs the LLM to choose the most specific leaf-level category for the input query. The query and prompt are submitted to the LLM, whose output is parsed to extract a valid taxonomy identifier corresponding to the selected category.

In the MCP mapping and LLM-based ranking phase, the resolver retrieves MCPs directly mapped to the identified taxonomy category via dictionary lookup; if only one MCP matches, the resolver selects it directly. If multiple MCPs are eligible, the resolver generates a ranked-list prompt that describes each candidate with a truncated summary and asks the LLM to rank the options, returning a comma-separated list of indices for the top-k MCPs. The system validates the returned indices, uses them to index into the candidate list, and assembles the selected MCPs into the final result. Figure 2 illustrates the overall schema of the Taxonomy Resolver

Datasets and Taxonomies

In this section, we introduce the core datasets and taxonomies that represent the backbone of the JSPLIT system. These foundational resources enable the platform to classify tools, resolve queries, and guide language model reasoning in a structured and interpretable way. The datasets provide labeled examples of both tools and queries, while the taxonomies offer a hierarchical framework for organizing and navigating the tool landscape. Together, they support the development, evaluation, and continuous improvement of JSPLIT's resolution and orchestration capabilities.

Taxonomies

To support structured tool reasoning and precise query resolution, the JSPLIT system relies on a functional taxonomy—a hierarchical classification system that organizes MCP (Model Context Protocol) servers based on what they do. This taxonomy enables the language model to reason over tool capabilities, supports accurate prompt generation, and guides the system in selecting the most relevant services for a given user query. Over the course of JSPLIT's development, two versions of the taxonomy have been created: Taxonomy v1 and a more advanced Taxonomy v2.

Taxonomy v1 was the first structured attempt to categorize the growing ecosystem of MCP servers. It was designed with simplicity and flexibility in mind, using a hierarchical structure based on primary functionality, supported by optional secondary tags for data type (e.g., text, image, structured) and provider (e.g., OpenAI, Microsoft). Each server was assigned a single primary category from a three-level hierarchy, with additional tags allowed for multi-functional tools. The taxonomy emphasized eight broad functional areas, including search, knowledge management, data processing, simulation, and communication. It supported multi-tool tagging, allowing servers to reflect multiple capabilities without violating the core hierarchy. Taxonomy-v1 served as a preliminary foundation, enabling early experimentation with tool classification and routing, and ultimately guiding the development of the more comprehensive and structured Taxonomy-v2.

Taxonomy v2 builds directly over v1 but introduces a deeper and more structured framework. It expands the number of top-level categories to eleven, covering new domains such as developer tools, specialized industries (e.g., finance, entertainment), and multi-domain orchestration. Each subcategory is now paired with a clear definition, ensuring consistency in both manual and automated classification. Taxonomy v2 introduces fallback subcategories for tools that don't neatly fit into existing slots, providing a formal mechanism for handling outliers. It also improves clarity in naming and scope, and it integrates hybrid and cross-functional tools more explicitly.

Datasets

The JSPLIT system is developed and evaluated using two complementary datasets that reflect different aspects of its functionality: the MCP Server Dataset and the MCP Query Dataset. Each serves a distinct role—one capturing the structure and capabilities of the tool landscape, and the other modeling the types of tasks users might pose to the system.

The MCP Server Dataset is a comprehensive catalog of approximately 2,000 Model Context Protocol (MCP) servers, retrieved from Smithery, a widely used MCP registry. Each MCP represents an external tool or service that can be invoked during the JSPLIT query resolution process. Every server entry includes a name, a human-readable description of its functionality, and a machine-readable specification of the tools it provides, all formatted according to the JSON schema defined in the MCP protocol. To support structured tool selection, we categorized all MCP servers under a multi-level taxonomy that reflects their functional domains and capabilities. This taxonomy underpins the intelligent routing performed by JSPLIT's Taxonomy-MCPResolver, enabling the system to reason about which tools are most appropriate for a given task. Classification of the dataset was conducted in two stages: an initial subset was manually annotated by domain experts to ensure high-quality labels, and these examples were then used to guide automated expansion using an external language model (Claude Sonnet 3.7). The LLM generalized from the expert-labeled entries to assign taxonomy categories to the remaining servers, resulting in a fully labeled and operational dataset.

The MCP Query Dataset contains approximately 200 entries, each representing a realistic user query paired with a ground-truth annotation. For each query, the dataset specifies the correct MCP server that should be used to fulfill the request, along with the appropriate taxonomy category. This dual annotation provides a structured foundation for evaluating both tool selection accuracy and end-to-end task resolution within the JSPLIT pipeline. The dataset is constructed to rigorously test the full capabilities of JSPLIT, including query interpretation, taxonomy-based reasoning, and tool invocation. By defining both the intended outcome and the taxonomy context, it supports detailed evaluation of intermediate steps as well as overall task success. Roughly 50% of the entries were initially generated using a language model and then manually refined to ensure they contained all necessary information for tool invocation. These queries were carefully adjusted for clarity and confirmed to succeed when evaluated in isolation (i.e., without irrelevant tools present). The remaining 50% were automatically generated by selecting MCP tools and producing corresponding queries with the goal of maximizing taxonomy coverage. Each automatically generated query was validated end-to-end through the JSPLIT system, and only retained if the correct tool was successfully invoked in a clean execution environment.

Experimental setup

To evaluate JSPLIT's performance, a structured experimental setup was designed to simulate an AI orchestration pipeline and test the system under various configurations. The experiments aimed to measure both the correct selection and invocation of MCP servers, and the LLM-related costs, such as token usage.

Each experiment followed a "needle in a haystack" design: for every user query in the MCP Query Dataset, the correct target MCP server (the "needle") was embedded among a set of irrelevant, randomly sampled "noise" MCP servers (the "haystack"). These noise MCPs were drawn from a larger pool of approximately 2,000 MCP servers, excluding any from the same leaf-level taxonomy category as the target. To assess scalability and robustness, the number of noise MCPs was varied incrementally, from 1 up to 1,000, allowing the system's performance to be evaluated under increasing levels of task complexity and tool-space clutter.

For each query: (1) a list of available MCPs was constructed by mixing the target server with a randomly selected set of noise MCPs. (2) The user query and this list were passed into the JSPLIT system.(3) JSPLIT executed its full pipeline, including the Taxonomy-MCPResolver (if enabled), LLM reasoning, and tool invocation. (4) The output was recorded, and LLM token usage and estimated costs were logged. (5) Accuracy was computed by checking whether the target MCP was correctly invoked.

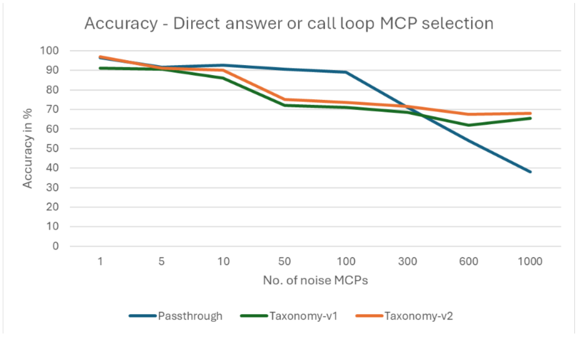

Figure 3: Accuracy of correct tool selection over the number of MCP servers. "Passthrough" represent the vanilla system that embeds all description into the context, "Taxonomy-v1" refers to the use of JSPLIT using the v1 version of the taxonomy, and "Taxonomy-v2" represents JSPLIT using the v2 taxonomy

Over 95% of outputs were tool-call executions. An output was marked correct if any tool from the target MCP server was used. For the minority of direct LLM answers (<5%), a LLM-as-a-judge method was employed: a separate LLM was asked to evaluate the correctness of the answer by comparing it to the known ground truth. This setup also enabled controlled comparison between two configurations of JSPLIT: one using the Taxonomy-MCPResolver for intelligent server filtering, and one using the Passthrough-MCPResolver, which passes all MCP servers to the LLM without filtering. This allowed for a quantitative assessment of the benefits of taxonomy-based routing in terms of both effectiveness and computational efficiency.

Results

In this section, we evaluate the effectiveness of the JSPLIT system in reducing input token cost and improving tool selection accuracy when the number of MCP server connected to the agent arise. Our experiments are designed to assess both the overall performance of taxonomy-based filtering compared to baseline approaches, and the sensitivity of the system to variations in the underlying model used during the filtering phase.

The first experiment evaluates three system configurations: (1) a baseline condition in which all MCP server descriptions are directly injected into the LLM context (Passthrough); (2) the JSPLIT system employing Taxonomy-v1 for tool filtering; and (3) the same system utilizing the more refined Taxonomy-v2. In all configurations, GPT-4.1-mini is used as the language model across all AI interaction steps. As illustrated in Figure 3, JSPLIT demonstrates improved tool selection accuracy over the baseline when the number of MCP servers is low (fewer than five). A slight decrease in performance is observed as the number of servers increases over ten MCP servers, corresponding to a rapid expansion in the number of candidate taxonomy classes injected in the LLM

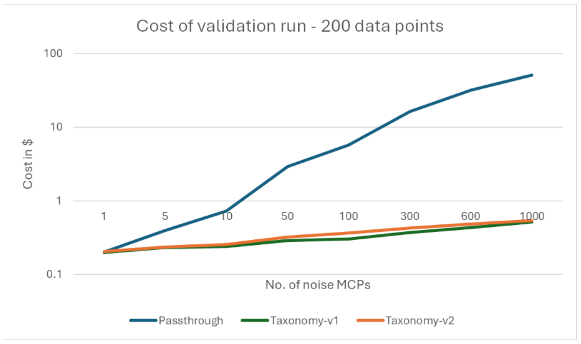

Figure 4: Input token cost for running 200 queries to the LLM over the number of MCP servers. ”Passthrough” represent the vanilla system that embeds all description into the context, ”Taxonomy-v1” refers to the use of JSPLIT using the v1 version of the taxonomy, and ”Taxonomy-v2” represents JSPLIT using the v2 taxonomy

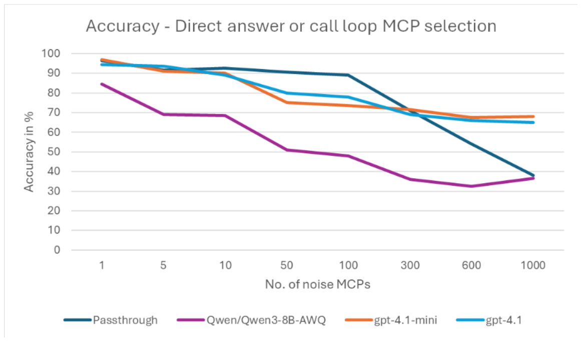

Figure 5: Accuracy of tool selection of different models used for the inner loop of the JSPLIT system.

prompt during the selection phase. However, as the server pool continues to grow—reaching into the hundreds—the performance of JSPLIT stabilizes, whereas the accuracy of the Passthrough approach deteriorates markedly, with the Passthrough approach culminating on less than 40% accuracy, while JSPLIT with Taxonomy-v2 remains stable around 69%. Taxonomy-v2 show comparable performance with Taxonomy-v1 when the noise servers are below five, but it demonstrate better performance from then on. Figure 4 reports the cumulative cost (in USD) of input tokens required to complete a batch of 200 queries as the number of connected MCP servers increases. The results clearly indicate that JSPLIT achieves a substantial reduction in token cost—exceeding two orders of magnitude—compared to the baseline. This demonstrates the effectiveness of JSPLIT in controlling computational and financial overhead while scaling to large tool ecosystems.

The second experiment is an ablation study designed to assess how the choice of language model within JSPLIT’s inner loop component responsible for selecting relevant taxonomy classes and MCP tools—affects overall system performance. Specifically, we compare three variants of JSPLIT, each using a different model for the classification

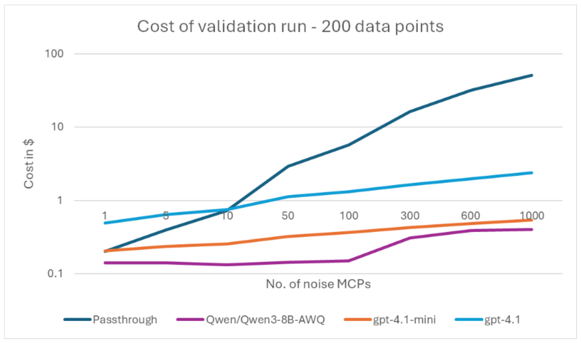

Figure 6: Input token cost for running 200 queries to the LLM over the number of MCP servers using different models for the MCP filtering step of the JSPLIT system

step: a local LLM (Qwen3-8B-AWQ), a small API-based model (GPT-4.1-mini), and a larger API-based model (GPT-4.1). The final interaction step is completed using GPT-4.1-mini in all three environments. Figure 5 presents the results in terms of tool selection accuracy. The findings show that both API-based models achieve comparable levels of performance, whereas the local model results in a substantial drop in accuracy. This suggests that while local models can support inference at lower cost, they may struggle to generalize effectively in complex selection tasks. In Figure 6, we report the input token cost (in USD) associated with each variant. As expected, the local model offers the lowest cost, but its savings are offset by a significant decrease in accuracy. The small API model provides a strong balance, maintaining high accuracy while keeping costs relatively low. The use of the larger API model yields only a marginal improvement in accuracy over the smaller one, but incurs a noticeable increase in token-related expenses. These results highlight a trade-off between computational cost and model performance, suggesting that lightweight API models may offer the best compromise for many practical deployments.

Classification Error Analysis

The following analysis operates at a diagnostic depth enabled exclusively by the hierarchical taxonomy embedded within the JSPLIT framework. The structured organization of tools into semantically coherent classes allows for a systematic tracing of misclassifications across multiple abstraction levels. This hierarchical arrangement provides a unique foundation for interpretability, facilitating precise debugging of classification errors and a deeper understanding of the agent’s decision dynamics. The confusion matrix in Figure 7 provides insight into the classification behavior of the JSPLIT system when using Taxonomy v2 in the presence of 1000 noise MCPs not relevant to the target queries. Each row corresponds to the ground truth top-level taxonomy class, while each column shows the predicted top-level class. Values off the diagonal indicate misclassifications at the top category level, while non-zero diagonal entries reflect correct top-level classification with errors at deeper levels of the taxonomy. The top classes are namely: (1)Search and

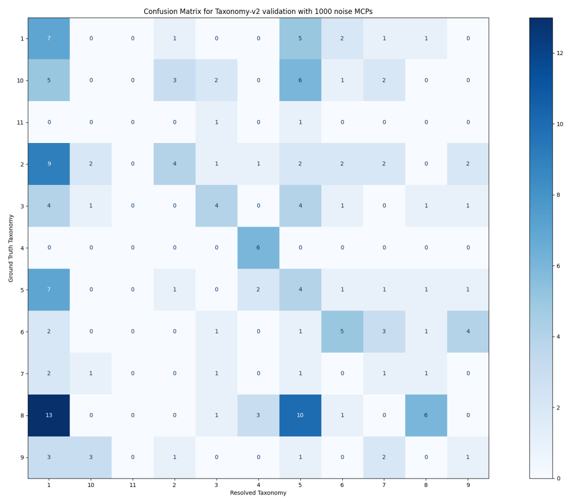

Figure 7: Confusion matrix representing classification errors between top-level classes of the taxonomy-v2

information retrieval, (2)Memory and knowledge management, (3)Simulation and planning, (4)Navigation and mapping, (5)Data extraction and manipulation, (6)System and device control, (7)Communication and interaction, (8)Specialized domains, (9)Developer tools and programming, (10) Multi-domain orchestration, (11) Others.

Several classes exhibit consistent confusion with semantically similar or overlapping categories: Memory and Knowledge Management (2) is frequently confused with Search and Information Retrieval (1) and Multi-Domain Orchestration (10). This likely stems from the overlap between tools that store and retrieve personal knowledge and those used for general search or coordination across domains. Simulation and Planning (3) shows notable misclassification toward Data Extraction and Manipulation (5) and System and Device Control (6), possibly due to the presence of data-driven decision tools and modeling systems that share similar operational features. These patterns highlight the challenge of distinguishing categories that involve compound or overlapping workflows, especially under weak signal conditions or when server descriptions are brief or ambiguous.

The category Search and Information Retrieval (1) appears across multiple rows in the matrix, indicating that it is often selected even when it is not the correct ground truth class. Misclassifications from categories such as Data Extraction and Manipulation (5), Specialized Domains (8), and Memory and Knowledge Management (2) frequently target this class. This suggests that descriptions of tools involving data access, content lookup, or general retrieval behavior may be semantically close to those found in search-related services. This over-selection highlights the need for more discriminative features or clearer category boundaries between search and related data access functionalities. The Specialized Domains (8) category suffers from significant misclassification, particularly into Search and Information Retrieval (1) and System and Device Control (6). This is unsurprising, as tools in specialized domains (e.g., finance, healthcare) often reuse common backend technologies and may not be distinguishable without highly domain-specific context.

Conclusion and Future Work

In this paper, we introduced JSPLIT, a system designed to address the challenge of prompt bloating in AI agent architectures that operate over large pools of external tools. Using a taxonomy-based filtering mechanism, JSPLIT dynamically selects a subset of relevant MCP servers to include in the language model's context, significantly reducing token usage while preserving the agent's operational effectiveness. Our evaluation, based on a large dataset of classified MCP servers and a set of task-linked queries, demonstrates that JSPLIT consistently reduces prompt size without significantly sacrificing task accuracy. In particular, as the number of available servers scales into the hundreds, the system shows clear advantages over baseline approaches that inject the full tool context. Notably, JSPLIT achieves better tool selection accuracy under these high-density conditions, highlighting the importance of structured pruning in this type of agent environment. Beyond efficiency gains, the hierarchical taxonomy introduced by JSPLIT contributes an additional benefit in terms of interpretability and traceability of agent behaviors. By providing explicit mappings between tools, their functional categories, and the contextual triggers leading to their selection, the taxonomy enables systematic introspection into the agent's decision-making process. This structure allows developers and researchers to analyze how specific classes influence selection outcomes, identify points of semantic overlap or ambiguity, and trace the reasoning chain that led to tool activation.

While these results are encouraging, several areas remain open for improvement. One key direction involves refining the descriptions associated with taxonomy categories, which play a central role in semantic matching between user queries and tool functions. Improving these descriptions, both in clarity and coverage, could further enhance the quality of classification and context filtering. Additionally, we plan to explore the development of a real-time classification mechanism for onboarding new MCP servers. Currently, classification relies on a static initial annotation, which limits the system's adaptability in dynamic environments. A live, model-assisted classification pipeline would allow JSPLIT to scale more fluidly and remain aligned with evolving tool ecosystems. In parallel, we have begun work on a more sophisticated Taxonomy-v3, which introduces more independent categories and separates domain as its own classification dimension. While still under development and requiring further experimentation, the design of Taxonomy-v3 shows promise as a more flexible and expressive foundation for future iterations of JSPLIT.

Overall, this work demonstrates that taxonomy-guided context management is a viable and effective strategy for improving the scalability, interpretability, and reliability of AI agents working with large sets of tools and MCP servers. Continued improvements in adaptability and taxonomy structure are likely to further extend the system's capabilities in real-world deployments.

References

Addad, B.; and Kapusta, K. 2024. Homeopathic Poisoning of RAG Systems. In *International Conference on Computer Safety, Reliability, and Security*, 358–364. Springer.

Anthropic. 2024. Anthropic: Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol. Accessed: 2025-07-15.

Gao, S.; Shi, Z.; Zhu, M.; Fang, B.; Xin, X.; Ren, P.; Chen, Z.; Ma, J.; and Ren, Z. 2024. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, 18030–18038.

Hao, S.; Liu, T.; Wang, Z.; and Hu, Z. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36: 45870–45894.

Kachuee, M.; Ahuja, S.; Kumar, V.; Xu, P.; and Liu, X. 2025. Improving Tool Retrieval by Leveraging Large Language Models for Query Generation. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, 29–38.

Li, T.; Li, Z.; and Zhang, Y. 2024. Improving Faithfulness of Large Language Models in Summarization via Sliding Generation and Self-Consistency.

Long, X.; Zhuang, L.; Li, A.; Yao, M.; and Wang, S. 2025. Eperm: An evidence path enhanced reasoning model for knowledge graph question and answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 12282–12290.

Priest, D. 2025. pwc ai agent survey. https://www.pwc.com/us/en/tech-effect/ai-analytics/ai-agent-survey.html. Accessed: 2025-07-15.

Roberts, J.; Han, K.; and Albanie, S. 2025. Needle Threading: Can LLMs Follow Threads Through Near-Million-Scale Haystacks? In *The Thirteenth International Conference on Learning Representations*.

Thibaud; Patsko; Brier; and Goicoechea-Martinez. 2025. Capgemini AI agents report. https://www.capgemini.com/wp-content/uploads/2025/07/Final-Web-Version-Report-AI-Agents.pdf? Accessed: 2025-07-15.

Wang, C.; Yang, Y.; Li, R.; Sun, D.; Cai, R.; Zhang, Y.; and Fu, C. 2024. Adapting llms for efficient context processing through soft prompt compression. In *Proceedings of the International Conference on Modeling, Natural Language Processing and Machine Learning*, 91–97.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Zhu, J.; Zhu, M.; Rui, R.; Shan, R.; Zheng, C.; Chen, B.; Xi, Y.; Lin, J.; Liu, W.; Tang, R.; et al. 2025. Evolutionary Perspectives on the Evaluation of LLM-Based AI Agents: A Comprehensive Survey. *arXiv preprint arXiv:2506.11102*.

Appendix A: Taxonomies

Emanuele Antonioni[1], Stefan Marković[1], Anirudha Shankar[1], Jaime Bernardo[1], Lovro Markovic[1], Silvia Pareti[2], and Benedetto Proietti[1]

[1]Janea Systems
[2]BigFilter.ai

November 30, 2025

Taxonomy-v1

Approach

- **Primary Axis:** Functionality (What the server does)
- **Secondary Dimensions:** Data Type, Provider

Handling Multi-Tool Servers

- Allow servers to be tagged with multiple categories.
- This deviates from "Single Inheritance" but enables recognition of ambiguity and allows proposing alternatives or refining requests.

Primary Classification Rules

- **Dominant Function:** Classify by the server's primary or most prominent capability.
- Each MCP server must have a single, primary classification (Level 1, 2, or 3) to preserve the search hierarchy.
- **Secondary Function:** Additional functionality tags are allowed.
- **Example:** A server primarily offering "1.1.1 General Web Search" but also "5.1.1 NLP" is classified under 1.1.1 and tagged with 5.1.1.

1. SEARCH AND INFORMATION RETRIEVAL

Definition: Servers that locate, fetch, and deliver information from various sources.

1.1 Web Search and Discovery

- **1.1.1 General Web Search:** Broad internet search engines (Google, Bing, DuckDuckGo)
- **1.1.2 Academic and Research:** Scholarly databases, research repositories
- **1.1.3 News and Media:** Real-time news, media monitoring
- **1.1.4 Social Media Search:** Platform-specific content discovery
- **1.1.5 Specialized Directories:** Professional networks, business directories

1.2 Database and Repository Access

- **1.2.1 Structured Databases:** SQL databases, data warehouses
- **1.2.2 Document Repositories:** File systems, document management systems
- **1.2.3 Version Control:** Git repositories, code hosting platforms
- **1.2.4 Cloud Storage:** Object storage, file sharing services
- **1.2.5 External API Data Access:** REST API Clients, GraphQL Clients, Service-Specific APIs

2. MEMORY AND KNOWLEDGE MANAGEMENT

Definition: Servers that store, organize, retrieve, and manage information for persistence and reuse.

2.1 Personal Knowledge Systems

- **2.1.1 Note-Taking Platforms:** Notion, Obsidian, Roam Research
- **2.1.2 Personal Wikis:** Individual knowledge bases
- **2.1.3 Bookmark Management:** Link organization and retrieval

2.2 Organizational Knowledge

- **2.2.1 Enterprise Knowledge Bases:** Corporate wikis, documentation systems
- **2.2.2 Collaborative Platforms:** Shared workspaces, team knowledge
- **2.2.3 Documentation Systems:** Technical documentation, help systems

2.3 Memory Persistence

- **2.3.1 Conversation Memory:** Chat history, interaction logs
- **2.3.2 Context Preservation:** Session state, user preferences
- **2.3.3 Learning Systems:** Adaptive knowledge accumulation

2.4 Knowledge Graphs and Ontologies

- **2.4.1 Semantic Networks:** RDF, OWL-based systems
- **2.4.2 Entity Relationship Systems:** Knowledge graph databases
- **2.4.3 Taxonomy Management:** Classification systems, controlled vocabularies

3. SIMULATION AND PLANNING

Definition: Servers that model scenarios, predict outcomes, and generate strategic plans.

3.1 Computational Simulation

- **3.1.1 Mathematical Modeling:** Numerical simulations, equation solving
- **3.1.2 Physical Simulations:** Physics engines, material modeling
- **3.1.3 System Dynamics:** Complex system behavior modeling

3.2 Strategic Planning

- **3.2.1 Project Planning:** Task scheduling, resource allocation
- **3.2.2 Decision Support:** Multi-criteria analysis, option evaluation
- **3.2.3 Scenario Analysis:** What-if modeling, risk assessment

3.3 Predictive Analytics

- **3.3.1 Forecasting:** Time series prediction, trend analysis
- **3.3.2 Machine Learning Models:** Trained model inference
- **3.3.3 Statistical Analysis:** Hypothesis testing, correlation analysis

4. NAVIGATION AND MAPPING

Definition: Servers that provide spatial awareness, location services, and navigation capabilities.

4.1 Geographic Information Systems

- **4.1.1 Mapping Services:** Google Maps, OpenStreetMap integrations
- **4.1.2 Geospatial Analysis:** GIS operations, spatial queries
- **4.1.3 Location Intelligence:** Place recognition, geocoding

4.2 Navigation and Routing

- **4.2.1 Route Planning:** Optimal path calculation
- **4.2.2 Real-time Navigation:** Turn-by-turn directions
- **4.2.3 Traffic and Conditions:** Dynamic routing with live data

4.3 Virtual Space Navigation

- **4.3.1 Digital Environment Mapping:** Virtual world navigation
- **4.3.2 Information Architecture:** Website/app structure navigation
- **4.3.3 Network Topology:** System architecture exploration

5. DATA EXTRACTION AND MANIPULATION

Definition: Servers that process, transform, analyze, and manipulate various data types.

5.1 Text Processing

- **5.1.1 Natural Language Processing:** Parsing, analysis, generation
- **5.1.2 Document Processing:** OCR, format conversion, extraction
- **5.1.3 Content Analysis:** Sentiment, classification, summarization

5.2 Structured Data Operations

- **5.2.1 Data Transformation:** ETL processes, format conversion
- **5.2.2 Analytics and Reporting:** Statistical analysis, visualization
- **5.2.3 Database Operations:** CRUD operations, query execution

5.3 Multimedia Processing

- **5.3.1 Image Processing:** Computer vision, image manipulation, image generation
- **5.3.2 Audio Processing:** Speech recognition, speech to text, audio analysis
- **5.3.3 Video Processing:** Video analysis, content extraction

5.4 Web Data Extraction

- **5.4.1 Web Scraping:** HTML parsing, content extraction
- **5.4.2 API Data Harvesting:** Automated data collection
- **5.4.3 Real-time Monitoring:** Change detection, alert systems

6. REMOTE DEVICE CONTROL

Definition: Servers that interface with and control external devices, systems, or IoT endpoints.

6.1 Smart Home and IoT

- **6.1.1 Home Automation:** Smart devices, environmental control
- **6.1.2 Security Systems:** Cameras, alarms, access control
- **6.1.3 Energy Management:** Smart meters, efficiency optimization

6.2 Industrial and Enterprise Systems

- **6.2.1 Network Infrastructure:** Router, switch, firewall management
- **6.2.2 Server Administration:** Remote system management
- **6.2.3 Manufacturing Control:** Industrial IoT, process control

6.3 Mobile and Wearable Devices

- **6.3.1 Smartphone Integration:** Device control, sensor access
- **6.3.2 Wearable Technology:** Fitness trackers, smartwatches
- **6.3.3 Location-based Services:** GPS, proximity systems

7. COMMUNICATION AND INTERACTION

Definition: Servers that facilitate communication between entities, manage interactions, and handle messaging.

7.1 Messaging and Chat

- **7.1.1 Instant Messaging:** Real-time chat platforms
- **7.1.2 Email Integration:** Email sending, management, processing
- **7.1.3 Social Media Interaction:** Platform posting, engagement

7.2 Collaboration Tools

- **7.2.1 Video Conferencing:** Meeting platforms, screen sharing
- **7.2.2 Shared Workspaces:** Collaborative editing, project management
- **7.2.3 Workflow Automation:** Process orchestration, approval systems

7.3 Notification and Alerting

- **7.3.1 Push Notifications:** Mobile, desktop alerts
- **7.3.2 Monitoring Alerts:** System health, threshold notifications
- **7.3.3 Event Broadcasting:** Webhook delivery, event streaming

8. MULTI-DOMAIN ORCHESTRATION

Definition: Servers that aggregate multiple tools, coordinate complex workflows, or provide meta-functionality across domains.

8.1 Tool Aggregators

- **8.1.1 Multi-Service Hubs:** Zapier-like integrations
- **8.1.2 Workflow Orchestrators:** Complex process automation
- **8.1.3 API Gateways:** Service mesh, API management

8.2 Meta-Programming Interfaces

- **8.2.1 Code Generation:** Automated programming assistance
- **8.2.2 System Integration:** Cross-platform connectivity
- **8.2.3 Configuration Management:** System setup, deployment

8.3 Hybrid Functionality Servers

- **8.3.1 Platform-Specific Suites:** Single-provider multi-tool servers
- **8.3.2 Domain-Crossing Tools:** Servers spanning multiple primary categories
- **8.3.3 Adaptive Interfaces:** Context-sensitive tool selection

Taxonomy-v2

Approach

- **Primary Axis:** Functionality (What the server does)
- **Secondary Dimensions:** Data Type, Provider

Handling Multi-Tool Servers

- Allow servers to be tagged with multiple categories – **PREFERRED**.
- This deviates from "Single Inheritance" but enables recognition of ambiguity and allows proposing alternatives or refining requests.

Primary Classification Rules

- **Dominant Function:** Each MCP server must have a **single, primary classification** based on its most prominent functionality (Level 1, 2, or 3).
- **Secondary Function:** Allow tagging the server with one or more additional functionalities.
- **Example:** A server with primary function "1.1.1 General Web Search" and secondary capability "5.1.1 Natural Language Processing" should be classified under 1.1.1 and tagged with 5.1.1.
- **"Other" Categories:** Use `x.9` or `x.x.9` for broader uncategorized items.

Secondary Dimensions (Tags)

These may be considered Level 4, but are treated as separate filters:

- **Data Type:** Text, Image, Audio, Video, Multimodal, Structured, Unstructured
- **Provider:** Google, Microsoft, OpenAI, Custom, Open Source

1. SEARCH AND INFORMATION RETRIEVAL

Definition: Servers that locate, fetch, and deliver information from external sources including web search, databases, and external APIs.

1.1 Web Search and Discovery

- **1.1.1 General Web Search:** Broad internet search engines providing web results (e.g., Google, Bing, DuckDuckGo, general search APIs)
- **1.1.2 Academic and Research:** Specialized search for scholarly content including research papers, academic databases, scientific repositories, and educational resources
- **1.1.3 News and Media:** Real-time news aggregation, media monitoring, journalism databases, and current events tracking systems
- **1.1.4 Social Media Discovery:** Platform-specific content discovery across social networks
- **1.1.5 Specialized Directories:** Professional networks, business directories, industry-specific databases, and niche community platforms

1.2 Database and Repository Access

- **1.2.1 Structured Databases:** Direct access to SQL databases, data warehouses, and structured data stores
- **1.2.2 Document Repositories:** File systems, document management systems, enterprise content management, and document libraries
- **1.2.3 Version Control:** Git repositories, code hosting platforms, source control systems, and development collaboration tools
- **1.2.4 Cloud Storage:** Object storage services, file sharing platforms, cloud-based storage systems, and distributed file systems
- **1.2.5 External API Data Access:** REST API clients, GraphQL clients, service-specific APIs, and third-party integration endpoints

2. MEMORY AND KNOWLEDGE MANAGEMENT

Definition: Servers that store, organize, retrieve, and manage internal information for persistence, reuse, and knowledge building.

2.1 Personal Knowledge Systems

- **2.1.1 Note-Taking Platforms:** Digital note-taking systems including Notion, Obsidian, Roam Research, and personal note management
- **2.1.2 Bookmark and Reference Management:** Link organization, reference collection, citation management, and personal library systems

2.2 Organizational Knowledge Systems

- **2.2.1 Enterprise Knowledge Bases:** Corporate wikis, institutional documentation systems, and organizational memory platforms
- **2.2.2 Collaborative Knowledge Platforms:** Shared workspaces, team knowledge systems, and collaborative documentation tools
- **2.2.3 Documentation and Help Systems:** Technical documentation, user guides, help desk knowledge bases, and support systems

2.3 Memory Persistence

- **2.3.1 Conversation Memory:** Chat history, interaction logs, session continuity, and dialogue state management
- **2.3.2 Context Preservation:** User preferences, personalization data, and settings management
- **2.3.3 Learning and Adaptation Systems:** Knowledge accumulation, user behavior learning, and adaptive interfaces

2.4 Knowledge Graphs and Semantic Systems

- **2.4.1 Semantic Networks:** RDF systems, OWL-based ontologies, linked data platforms, and semantic web technologies
- **2.4.2 Entity and Relationship Systems:** Knowledge graph databases, entity recognition, relationship mapping, and graph-based knowledge systems

3. SIMULATION AND PLANNING

Definition: Servers that model scenarios, predict outcomes, generate strategic plans, and support decision-making processes.

3.1 Computational Simulation

- **3.1.1 Mathematical Modeling:** Numerical simulations, equation solving, computational mathematics, and algorithmic modeling
- **3.1.2 Physical and Scientific Simulations:** Physics engines, material modeling, scientific simulations, and engineering analysis
- **3.1.3 System Dynamics:** Complex system behavior modeling, network simulations, and dynamic system analysis

3.2 Strategic Planning and Management

- **3.2.1 Project and Task Management:** Task scheduling, resource allocation, project planning tools, and workflow management
- **3.2.2 Decision Support Systems:** Multi-criteria analysis, option evaluation, strategic planning, and business intelligence
- **3.2.3 Scenario and Risk Analysis:** What-if modeling, risk assessment, contingency planning, and strategic scenario evaluation

3.3 Predictive Analytics

- **3.3.1 Forecasting and Trend Analysis:** Time series prediction, market forecasting, trend identification, and predictive modeling
- **3.3.2 Machine Learning Inference:** Trained model deployment, AI-powered predictions, and intelligent decision support
- **3.3.3 Statistical Analysis:** Hypothesis testing, correlation analysis, data mining, and statistical modeling

4. NAVIGATION AND MAPPING

Definition: Servers that provide spatial awareness, location services, navigation capabilities, and geographic information systems.

4.1 Geographic Information Systems

- **4.1.1 Mapping Services:** Google Maps, OpenStreetMap integrations, cartographic services, and geographic visualization
- **4.1.2 Geospatial Analysis:** GIS operations, spatial queries, geographic data processing, and location intelligence

- **4.1.3 Location and Place Services:** Geocoding, place recognition, address validation, and location-based information

4.2 Physical Navigation and Routing

- **4.2.1 Route Planning:** Optimal path calculation, multi-modal routing, and travel planning systems
- **4.2.2 Real-time Navigation:** Turn-by-turn directions, live traffic integration, and dynamic route guidance
- **4.2.3 Traffic and Environmental Conditions:** Real-time traffic data, weather-aware routing, and condition-based navigation

4.3 Virtual and Digital Navigation

- **4.3.1 Digital Environment Mapping:** Virtual world navigation, game environment mapping, and 3D space orientation
- **4.3.2 Information Architecture Navigation:** Website structure exploration, app navigation assistance, and content hierarchy mapping
- **4.3.3 Network and System Topology:** Infrastructure mapping, system architecture exploration, and network navigation

5. DATA EXTRACTION AND MANIPULATION

Definition: Servers that process, transform, analyze, and manipulate various data types including text, multimedia, and structured data.

5.1 Text and Document Processing

- **5.1.1 Natural Language Processing:** Text parsing, language analysis, generation, translation, and linguistic operations
- **5.1.2 Document Processing:** OCR, format conversion, document extraction, and file format manipulation
- **5.1.3 Content Analysis:** Sentiment analysis, text classification, summarization, and content understanding

5.2 Structured Data Operations

- **5.2.1 Data Transformation:** ETL processes, format conversion, data cleaning, and structural data manipulation
- **5.2.2 Analytics and Reporting:** Statistical analysis, data visualization, business intelligence, and reporting systems
- **5.2.3 Database Operations:** CRUD operations, query execution, database management, and data persistence

5.3 Multimedia Processing

- **5.3.1 Image Processing:** Computer vision, image manipulation, image generation, and visual content analysis
- **5.3.2 Audio Processing:** Speech recognition, audio analysis, sound processing, and voice-to-text conversion
- **5.3.3 Video Processing:** Video analysis, content extraction, video editing, and multimedia content processing

5.4 Web Data Extraction

- **5.4.1 Web Scraping:** HTML parsing, content extraction, web crawling, and automated data harvesting
- **5.4.2 API Data Collection:** Automated data gathering from APIs, bulk data retrieval, and systematic data acquisition
- **5.4.3 Monitoring and Change Detection:** Real-time monitoring, alert systems, and automated change tracking

6. SYSTEM AND DEVICE CONTROL

Definition: Servers that interface with and control external devices, systems, applications, or IoT endpoints.

6.1 Smart Home and IoT

- **6.1.1 Home Automation:** Smart devices, environmental control, lighting systems, and household device management
- **6.1.2 Security and Monitoring:** Cameras, alarms, access control, and home security systems
- **6.1.3 Energy and Utilities Management:** Smart meters, efficiency optimization, utility monitoring, and resource management

6.2 Enterprise and Infrastructure Systems

- **6.2.1 Network Infrastructure:** Router, switch, firewall management, and network device configuration
- **6.2.2 Server and System Administration:** Remote system management, server operations, and infrastructure control
- **6.2.3 Industrial and Manufacturing Control:** Industrial IoT, process control, manufacturing systems, and operational technology

6.3 Computer and Application Control

- **6.3.1 Operating System Control:** Desktop automation, system-level operations, OS-specific integrations (Windows, macOS, Linux)
- **6.3.2 Browser and Application Control:** Web browser automation, application scripting, and software control interfaces
- **6.3.3 Mobile and Wearable Devices:** Smartphone integration, device sensor access, wearable technology, and mobile device management

7. COMMUNICATION AND INTERACTION

Definition: Servers that facilitate communication between entities, manage interactions, handle messaging, and coordinate collaborative activities.

7.1 Messaging and Social Interaction

- **7.1.1 Instant Messaging:** Real-time chat platforms, messaging apps, and communication channels
- **7.1.2 Email Integration:** Email sending, management, processing, and email automation systems
- **7.1.3 Social Media Interaction:** Platform posting, social engagement, community management, and social media automation

7.2 Collaboration and Scheduling

- **7.2.1 Video Conferencing:** Meeting platforms, screen sharing, virtual collaboration, and video communication tools
- **7.2.2 Calendar and Scheduling:** Meeting scheduling, calendar management, appointment booking, and time coordination
- **7.2.3 Shared Workspaces:** Collaborative editing, team coordination, document sharing, and group productivity tools

7.3 Workflow and Process Automation

- **7.3.1 Workflow Automation:** Process orchestration, approval systems, business process automation, and workflow management
- **7.3.2 Event-based Alerts:** Push notifications, system alerts, event broadcasting, and notification management
- **7.3.3 Event and Integration Management:** Webhook delivery, event streaming, system integration, and inter-service communication

8. SPECIALIZED DOMAINS

Definition: Domain-specific servers tailored for particular industries, activities, or specialized use cases.

8.1 Financial Services

- **8.1.1 Personal and Business Finance:** Budget planning, expense tracking, financial health analysis, accounting integration
- **8.1.2 Trading and Investments:** Stock trading, crypto, portfolio management, and market research
- **8.1.3 Payment Processing:** Payment gateways, money transfers, digital wallets, and payment services
- **8.1.4 Market Data and Analytics:** Financial data feeds, economic indicators, and financial research

8.2 Entertainment and Gaming

- **8.2.1 Gaming Platforms:** Game APIs, player statistics, platform integrations
- **8.2.2 Game Development:** Game engines, development frameworks, and tools
- **8.2.3 Media and Entertainment:** Streaming platforms, music, podcasts, and content recommendation

8.9 Other Specialized Domains

- **Definition:** Domains like Shopping, Food, Health, or Travel not covered above.

9. DEVELOPER TOOLS AND PROGRAMMING

Definition: Servers that assist software development, provide programming utilities, code management, and development workflow support.

9.1 Code Development and Management

- **9.1.1 Code Generation and AI Assistance:** Automated programming, code completion
- **9.1.2 Code Analysis and Quality:** Review, testing frameworks, quality tools
- **9.1.3 Development Environment:** IDE integrations, toolchains

- **9.1.4 API Development and Testing:** API design, documentation, and testing
- **9.1.5 Development Utilities:** Build tools, package and dependency management

9.2 Development Operations

- **9.2.1 Deployment and CI/CD:** Integration pipelines, deployment automation
- **9.2.2 Infrastructure as Code:** Provisioning, configuration automation
- **9.2.3 Monitoring and Debugging:** Application monitoring, performance analysis
- **9.2.4 Integration and Connectivity:** Middleware, service mesh, integration platforms

10. MULTI-DOMAIN ORCHESTRATION

Definition: Servers that aggregate multiple tools, coordinate complex workflows across domains, or provide meta-functionality spanning multiple categories.

10.1 Integration and Orchestration

- **10.1.1 Multi-Service Integration:** Zapier-like tools, API gateways, cross-platform workflows
- **10.1.2 Workflow and Process Orchestration:** Enterprise workflow systems, BPM, approval workflows

10.2 Cross-Domain and Platform Tools

- **10.2.1 Platform-Specific Suites:** Vendor ecosystems, integrated toolsets
- **10.2.2 Domain-Crossing Applications:** Servers spanning multiple primary categories

11. Other

Definition: Servers that do not fall into any of the available categories.