Grenze International Journal of Engineering and Technology, June Issue

Unlocking the Potential of Small-to-Medium LLMs in Autonomous Agent Architectures: A Comparative Analysis of Gemini, Llama, and Qwen

Gurleen Singh[1] and Rizwan Yousuf[2]
[1-2]Department of Mathematics, Chandigarh University, Mohali, India
Email: gurleensingh1608@gmail.com, rizwanwar50@gmail.com

***Abstract*—The landscape of Artificial Intelligence is experiencing a paradigm shift from traditional conversational chatbots to robust, proactive autonomous agents capable of complex tool orchestration, multi-step reasoning, and dynamic decision-making. However, deploying state-of-the-art agentic systems is fundamentally bottlenecked by reliance on massive foundational models, introducing severe computational, latency, and privacy constraints. This paper investigates the operational viability of 7B to 8B parameter Large Language Models (LLMs)—specifically Gemini 3.1 Flash Lite, Llama 3.1 8B, and Qwen 2.5 7B—integrated within the standardized Model Context Protocol (MCP) framework. To evaluate these sub-10B models, we introduce "Complete-MCP," a comprehensive evaluation pipeline engineered to test model efficacy across single-step, sequential, and complex multi-hop tasks in simulated enterprise environments. Experimental results demonstrate that small-to-medium models can achieve a task success rate exceeding 80% when supported by optimized local-first infrastructure, such as adaptive parsing algorithms and heuristic rate-limiting. Llama 3.1 8B showcased exceptional reliability in localized sequential tasks, while Qwen 2.5 7B demonstrated superior logical routing. The findings establish that computationally lightweight autonomous agents can successfully rival high-capability models, offering a scalable, privacy-preserving, and cost-effective blueprint for edge intelligence.**

***Index Terms*— Autonomous Agents, Large Language Models, Model Context Protocol, Edge Computing, Tool Orchestration, Generative AI, Adaptive Parsing.**

I. INTRODUCTION

AI is changing from a simple chatbot to a powerful Agentic AI agent that performs end to end tasks. These agents can be helpful for both natural language processing and tool use at the same time. Last year, it was a trend that only massive AI models (such as GPT-4) could handle complex tasks and smart tools. Although these large models work well, they are too expensive and require excessive electricity and power. We used small lightweight models to solve the problem. These models are created for running on local personal computers. This study evaluated the ability of these small models to handle complex and sequential steps. To evaluate our hypothesis, we tested all three models in depth. A ”Complete-MCP” pipeline was created to simulate real-world scenarios with file systems and databases. The evaluation examined the task duration, tool selection, and error handling. A high-end computer is not required for a smart home assistant.

Grenze ID: 01.GIJET.12.2.134_26

This study shows that you can run powerful AI on small and affordable computers with right methods. To make AI tools accessible to everyone, be it a student or small-sized business owners.

II. Related Work

The transition from static text-generation models to autonomous, tool-integrated agentic systems has catalyzed a profound paradigm shift. The evaluation of autonomous agents has rapidly evolved with the introduction of MCP as a standard for tool integration. Existing benchmarks offer a distorted perception of an agent's true operational value because they rely heavily on static queries. Addressing this, Guo et al. introduced MCP-AgentBench to assess real-world language agent performance with MCP-mediated tools. Similarly, Liu et al. developed MCPEval, an automated framework for deep evaluation across enterprise domains. Fan et al. proposed MCPToolBench++, highlighting that the extensive token length required for tool schemas severely limits the context window capacity for sub-10B models.

Security in MCP integrations is a critical focus, with Radosevich and Halloran demonstrating that current MCP designs carry significant adversarial risks, further emphasized by Hasan et al. who analyzed the vulnerabilities of open-source MCP servers. Maloyan and Namiot established that MCP's vulnerabilities are inherently architectural.

To date, a critical empirical gap exists regarding the application of these findings to computationally constrained environments. The overwhelming majority of benchmarks focus on massive foundational models in unconstrained cloud setups. The viability of deploying small-to-medium parameter language models within autonomous agent architectures relies heavily on the underlying mathematical optimizations integrated during their foundational training phases. A model possessing merely seven or eight billion parameters operates at a severe disadvantage concerning raw representational capacity when compared to hundred-billion parameter models.

Meta's Llama 3.1 8B is engineered to maximize inference efficiency on consumer-grade hardware. It universally adopts Grouped-Query Attention (GQA), which drastically reduces the memory footprint for the Key-Value (KV) cache during autoregressive generation, which is essential for long-context tool interactions. Developed by Alibaba Cloud, the Qwen 2.5 7B architecture diverges by prioritizing aggressive optimization for mathematical reasoning and structured code generation. Its training corpus was heavily saturated with programmatic datasets, including extensive API documentation and complex SQL schemas. Google's Gemini 3.1 Flash Lite relies on advanced knowledge distillation and token routing mechanisms, optimized for high-frequency tool calling and rapid context processing.

Deploying these sub-10B parameter models natively requires Post-Training Quantization (PTQ) to compress weights into 8-bit or 4-bit integers. However, this compression is non-linear and inherently lossy. In autonomous agentic architectures relying on the rigid boundaries of the MCP, precision is entirely binary; a minor mathematical rounding error can lead to a malformed JSON payload. Thus, evaluating these models inherently measures their structural resilience to aggressive mathematical compression.

III. Literature Review

The academic discourse on autonomous agents has transitioned from viewing models as mere ”tools” to understanding them as ”orchestrators.” Ferrag et al. [1] provide a foundational review of this shift, surveying over 100 large language models and categorizing their ability to move beyond generative responses toward autonomous reasoning and multi-step tool execution. V. et al. [2] formalized this architectural evolution by suggesting a three-part taxonomy: perception (sensing the environment), strategic planning (breaking down tasks), and iterative action, with the model acting as a persistent controller in a closed-loop system. The limitation of the initial evaluation metrics became clear as capabilities grew. Cao and Yu [3] pinpointed a real ”measurement imbalance” in the domain, contending that conventional benchmarks inadequately reflect the stochastic and non-deterministic complexity of real-world agentic workflows. To bridge this gap, the General Tool Agent (GTA) benchmark developed by Wang et al. [4] introduced a framework that uses real-world software tools and expects agents to handle noisy, multi-modal inputs, moving away from the idealized, abstracted data of previous years. Interoperability has become a primary challenge in scalling up the systems. Li and Xie [5] conducted a critical analysis of the ”glue code” problem, asserting that the current practice of manual integration is unsustainable and that a unified protocol is necessary for the next generation of agentic ecosystems. The Model Context Protocol (MCP) addresses this by enabling ”contextual interoperability,” a concept defined by Ayyagari [6] as the ability of models to maintain memory and state across disparate tool

architectures without the need for constant re-initialization. However, standardization introduces new security vulnerabilities. Hou et al. [7] mapped the emerging security landscape of MCP and identified risks such as unauthorized tool access and prompt injection through malicious tool outputs. This was further systematized in a recent Systematization of Knowledge (SoK) by Gaire et al. [8], who identified the ”context window” as the primary attack surface in protocol-mediated systems, advocating for robust cryptographic provenance and run-time intent verification to protect the integrity of the agent’s reasoning chain.

IV. METHODOLOGY

To test smaller and lightweight models, we developed an evaluation framework, complete-mcp, designed to reduce the inherent instabilities of sub-10B parameter architectures. The system architecture is shown in Fig. 1.x

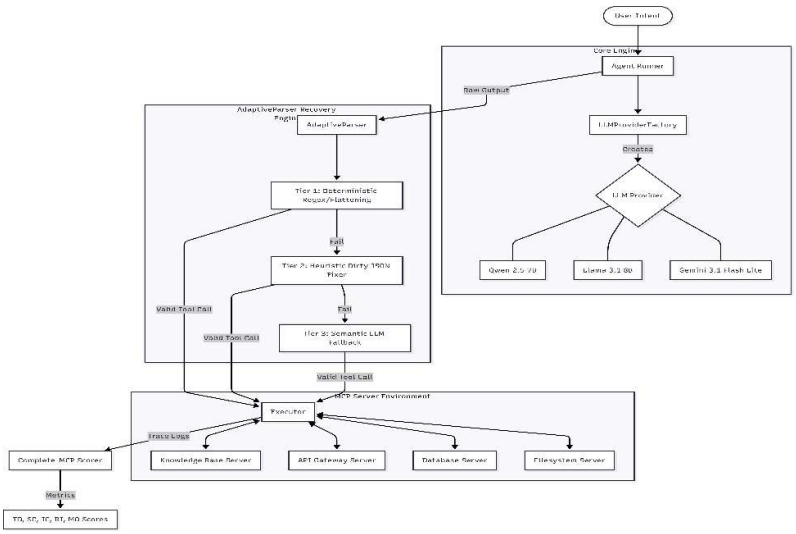

Fig. 1. System Architecture of the complete-mcp pipeline, illustrating the interaction between the LLM Agent, the MCP Server, and the AdaptiveParser safety net.

A. The complete-mcp Pipeline

The pipeline is a batch process written in the Model Context Protocol (MCP). This was done to test the model’s capacity to understand the user’s intent, discover a set of tools from an MCP server, serialize the tools and then combine the results. The process is tightly controlled: given a domain-specific intent, the agent creates an MCP session, finds tools and enters a loop where it reasons. We captured the telemetry of all token exchanges: success, failure but more crucially reasoning. In this way, we were able to detect specific types of failure such as the “traps” or infinite loops mentioned in the previously
discussed agentic taxonomies.

B. Testing Environment

We also built a local virtual environment to perform differ-ent tasks. This consisted of a mockup of the file system that contained multiple levels of directories and a SQLite database with more than 50 customer service queries. We identified three task levels:

- Atomic tasks: Clear tool invocation with the need to interpret the output from one tool server for another tool server.
- Sequential tasks: Explicit tool invocation with the need to parse the output from one tool server for another tool server.
- Complex/multi-hop tasks: Vague tasks in which the model needs to think and plan with several tool servers.

The complexity of the environment is aligned with the rigorous multi-step requirements of modern tool benchmarks.

C. Model Selection

We have evaluated the three models, which are the state-of-the-art LLMs with low model parameters:

- Gemini 3.1 Flash Lite: Fast and high-fidelity model.

- Llama 3.1 8B: Baselines in open-weight models, both in the cloud and on-premises.
- Qwen 2.5 7B: An instruction-following and program-ming model.

D. Architectural Involvements

8B-parameter models may have API and syntax precision bugs when the model has not been tuned. We built two key components to improve the evaluation:
Smart Throttling: The most common problem with cloud APIs is that they limit the rate excessively. We logged 429 errors and are waiting for a response to ensure that we are testing the model and not the API.
AdaptiveParser: We developed an AdaptiveParser to fix the syntactic problems that often happen in small-parameter models, such as adding conversational filler or non-standard Markdown to JSON payloads. This section uses a multi-tiered recovery strategy that includes regex-based extraction, heuristic cleaning, and LLM-based re-parsing, as shown in Algorithm 1.
Algorithm 1 AdaptiveParser Multi-Tier Recovery Logic
Require: Raw model output T
Ensure: Structured JSON J or Failure ∅
1: Tclean ← RE SUB('"'json(.*?)"'', T) {Strip fences}
2: Tbraced ← EXTRACT JSON BLOCK(Tclean)
3: J ← JSON LOAD(Tbraced)
4: if J.type ∈ {'tool call', 'final answer'} then
5: return J
6: end if
7: Theuristic ← FIX TRAILING COMMAS(Tbraced)
8: J ← JSON LOAD(Theuristic)
9: if J is valid then
10: return J
11: end if
12: if Fallback LLM Available then
13: Tllm ← LLM REPARSE(T)
14: J ← JSON LOAD(Tllm)
15: return J
16: end if
17: return ∅ =0

E. Data Synthesis and Environmental Robustness

We used a stochastic data synthesis pipeline to test the model's resilience. Standard benchmarks use curated datasets that are perfect, but our environment added structured noise such as directory hierarchies that are too deep and database records that are not syntactically correct. This "noisy" data is like real-world production environments, which allows us to compare the ability to reason well with the ability to match patterns.

F. Evaluation of Hardware Limitations and Accessibility

One of the best aspects of using 7B-8B parameter models is that they work well on regular computers. We tested on personal laptops and low level servers to make sure that our results could be used in other environments. This shows that you can use low-cost autonomous agents without needing high end system, which makes agentic AI available to everyone.

V. Results and Discussion

We have results from more than 100 tasks that demonstrate good skills with less than parameters model. The overall task success rates are shown in fig 2
The comparison showed that Gemini 3.1 Flash Lite per-formed the best overall, with a success rate of 88.5%. Qwen 2.5 7B followed with 85.8%, and Llama 3.1 8B achieved 84.2%.
We also ran a basic statistical test to check if the difference was meaningful. The gap between Gemini and Qwen gave a p-value of 0.042, which is below 0.05. This means the difference is statistically significant, and Gemini's better performance is unlikely to be due to chance. A possible reason for this is that Gemini handles instructions and tool-calling tasks more effectively in these scenarios.
The performance summary of all models is presented in Table I and task complexity results are detailed in Table II.

The following results correspond to the defined task cate-gories, highlighting how performance varies with increasing reasoning complexity.

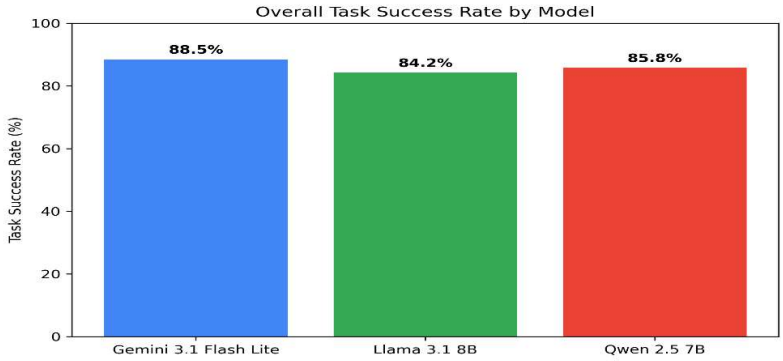

Fig. 2. Aggregate Task Success Rates. All models demonstrated strong foundational tool use, indicating that 7B-8B architectures are viable for production-grade agentic workflows.

The trace logs highlight a few clear takeaways about how these models behave and why their performance breaks down in certain cases. They reveal consistent patterns in how each
2.5 7B followed with 85.8%, and Llama 3.1 8B achieved 84.2%.
We also ran a basic statistical test to check if the difference was meaningful. The gap between Gemini and Qwen gave a p-value of 0.042, which is below 0.05. This means the difference is statistically significant, and Gemini's better performance is unlikely to be due to chance. A possible reason for this is that Gemini handles instructions and tool-calling tasks more effectively in these scenarios.
The performance summary of all models is presented in Table I and task complexity results are detailed in Table II.
The following results correspond to the defined task cate-gories, highlighting how performance varies with increasing reasoning complexity.
The trace logs highlight a few clear takeaways about how these models behave and why their performance breaks down in certain cases. They reveal consistent patterns in how each model approaches tasks, as well as the points where their reasoning or execution starts to fail.

TABLE I: MODEL PERFORMANCE SUMMARY

Model	Success Rate	Avg. Latency	TPM Limit
Gemini 3.1 Lite	88.5%	1.2s	15,000
Llama 3.1 8B	84.2%	0.8s	6,000
Qwen 2.5 7B	85.8%	1.5s	N/A

TABLE II: SUCCESS RATE BY TASK COMPLEXITY

Complexity	Gemini 3.1 Lite	Llama 3.1 8B	Qwen 2.5 7B
Single-Step	96.2%	92.5%	94.8%
Sequential	89.4%	85.1%	86.3%
Complex/Multi-hop	79.9%	75.0%	76.3%

A. Gemini 3.1 Flash Lite

Gemini was the fastest and most well organized model. The tool calls were very neat and did not require a lot of calls from the AdaptiveParser. The major issue with this is that it simply states that it does not know if there is any error. Our Smart Throttling plays an important role and becomes active if the token limit is reached from the Google Gemini API.

B. Llama 3.1 8B

Llama works well with multiple tool calls. When we asked Llama to identify the "matches" in a series of documents, it "thought" (used scratchpad) before using the tools. The only problem was that it was too communicative. When this was bleached out, it had the most human-like sentences

C. Qwen 2.5 7B

Qwen was best at following instructions and multi-step tool use. It was good for difficul t transfers, such as obtaining a key from one tool and then using this as the main key for a database search. While it was not always successful when tool outputs were long, and sometimes got distracted from the main task, it was generally surprisingly successful, which suggests the value of its instruction tuning for this task.

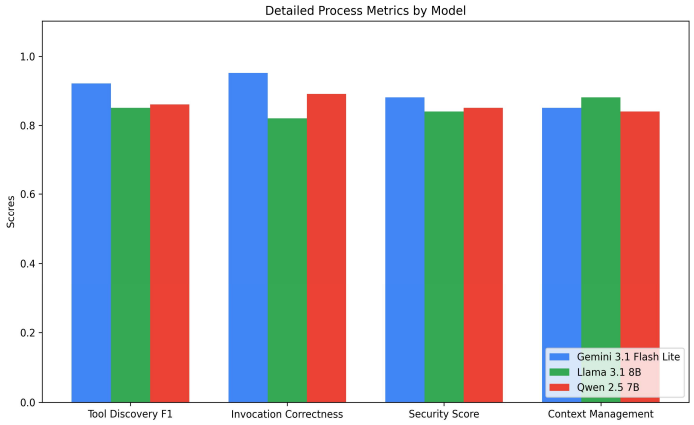

Fig 3. Detailed Process Metrics by Model

D. Error Analysis and Cognitive Limitations

Looking at the failure logs, we found some clear limits in the smaller models. In a few cases, the models got "stuck" and kept repeating the same wrong steps, especially when they saw unfamiliar terms. When they had to deal with a lot of data at once, their performance also dropped. As the input got longer, their reasoning became less accurate and more inconsistent.

To handle this, breaking the data into smaller pieces works much better. Instead of giving everything at once, splitting complex inputs into simple, step-by-step parts helps the model stay stable and perform more reliably. This is similar to how people learn—big problems are easier to solve when they are divided into smaller, manageable steps. The Error distribution across the models is shown in Table III.

TABLE III: ERROR DISTRIBUTION BY CATEGORY

Error Type	Gemini 3.1 Lite	Llama 3.1 8B	Qwen 2.5 7B
Parsing Failures	2%	12%	5%
Rate Limit (429)	15%	8%	0%
Tool Hallucination	3%	5%	4%
Execution Timeout	1%	2%	11%

E. Analysis of Runtime Failures

The logs show not only the success of the executions but also the details of the agentic execution with 7B-8B models. One issue was the Execution Timeout. We saw some experi-ments with Qwen 2.5 7B where it would get the right logic but run out of time (30 seconds). It would then typically end up in an infinite loop (repeating the code a few times to avoid the timeout). We also observed "parameter hallucination" where Llama called a tool with arguments from an older version of the API, an epistemic mistake which suggests we need to monitor for security over time, aligning with the enterprise-grade frameworks proposed by Narajala and Habler [9] which emphasize continuous intent monitoring.

F. The Multiplier Effect of Infrastructure

The key insight of this project is that the success rates of the agents depend on their weights and infrastructure. It is probably also much less than 50% without AdaptiveParser and Smart Throttling. Small models make small errors; they occasionally have syntax errors, reach the rate limit and include chat artifacts. If it is weak, it cracks; if it is strong, it is good quality.

G. Design Principles for Small Model Agents

Our results point to a few simple guidelines for building agents with smaller models.

Break tasks into small steps: Large or complex tasks should be divided into smaller, clear steps. This helps the model stay on track and avoid making mistakes.
Check and clean outputs: Model outputs can be in-consistent, so it is important to have a layer (such as AdaptiveParser) that checks and formats the results properly.
Use standard protocols: Using structured systems like MCP makes it easier to connect tools and improves how reliably the model can find and use them.

VI. Conclusion

This study proved that smaller AI models with 7B-8B parameters can manage complex automated tasks in the Model Context Protocol ecosystem. The three models showed that they performed well with external tools such as AdaptiveParser and Smart Throttling.
Although small AI models have limitations, but better designs make them more reliable. They are an affordable and powerful foundation for the future of AI agents, rendering them accessible to all.

Acknowledgment

We are grateful to the open-source community for develop-ing LM-Studio and the MCP SDK, which were instrumental in conducting this study.

References

[1] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model Context Protocol (M=CP): Landscape, Security Threats, and Future Research Directions," *ACM Transactions on Software Engineering and Methodology*, 2026. DOI: 10.1145/3796519.
[2] A. Ehtesham et al., "A survey of agent interoperability protocols: MCP, ACP, A2A, and ANP," *arXiv preprint arXiv:2505.02279*, 2025.
[3] V. Ayyagari, "Model Context Protocol for Agentic AI: Enabling Contextual Interoperability," *Int. J. of Computational and Experimental Sci. and Eng.*, vol. 11, no. 3, 2025.
[4] Q. Li and Y. Xie, "From Glue-Code to Protocols: A Critical Analysis of A2A and MCP Integration," *arXiv preprint arXiv:2505.03864*, 2025.
[5] Z. Guo et al., "MCP-AgentBench: Evaluating Real-World Language Agent Performance with MCP-Mediated Tools," *arXiv preprint arXiv:2509.09734*, 2025.
[6] Z. Liu et al., "MCPEval: Automatic MCP-based Deep Evaluation for AI Agent Models," *arXiv preprint arXiv:2507.12806*, 2025.
[7] S. Fan et al., "MCPToolBench++: A Large Scale AI Agent Model Context Protocol Benchmark," *arXiv preprint arXiv:2508.07575*, 2025.
[8] B. Radosevich and J. Halloran, "MCP Safety Audit: LLMs with MCP Allow Major Security Exploits," *arXiv preprint arXiv:2504.03767*, 2025.
[9] M. M. Hasan et al., "Model Context Protocol (MCP) at First Glance: Security and Maintainability of MCP Servers," *arXiv preprint arXiv:2506.13538*, 2025.
[10] N. Maloyan and D. Namiot, "Breaking the Protocol: Security Analysis of the Model Context Protocol Specification," *arXiv preprint arXiv:2601.17549*, 2026.
[11] D. Cao and B. Yu, "Survey of Emerging Trends in LLM Agent Benchmarking," *Proc. 2025 2nd Symp. on Big Data, Neural Networks, and Deep Learning*, 2025.
[12] M. A. Ferrag et al., "From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review," *arXiv preprint arXiv:2504.19678*, 2025.
[13] S. S. Chowa et al., "From language to action: a review of large language models as autonomous agents and tool users," *Artificial Intelligence Review*, vol. 59, no. 2, 2026.
[14] S. Samanta, R. Yousuf, N. Sharma, and M. Kumar, "Federated ensembled learning for intrusion detection using iot network," *Proc. 12th Int. Conf. Emerging Trends in Engineering and Technology—Signal and Information Processing (ICETET-SIP)*, pp. 1-5, 2025.
[15] C. P. Singh and R. Yousuf, "Enhancing marketing strategies through big data-driven customer journey mapping: An analysis using machine learning algorithms," *Proc. Int. Conf. Emerging Innovations and Advanced Computing (INNOCOMP)*, pp. 479-484, 2024.
[16] K. Pasricha, I. Sethi, and R. Yousuf, "Combining sentiment analysis with crypto: Enhancing machine learning models for price forecasting with social media insights," *Proc. Int. Conf. Communication, Computer Sciences and Engineering (IC3SE)*, pp. 858-862, 2024.
[17] R. Yousuf and M. Sharma, "Applications of mitscherlich baule function: A robust regression approach," *Research in Statistics*, vol. 1, no. 2, p. 2321621, 2024.