An MCP-based Solution for Managing Slices in Private 5G Networks

Victor Verissimo[1], Marcus Ferreira[1], Livia Buriti[1], Thiago Moraes[3], Ruan Gomes[2] and João Brunet[1]

[1] UFCG, Departamento de Sistemas e Computação, Campina Grande – Paraíba – Brasil

[2]IFPB, João Pessoa – Paraíba – Brasil

[3]IBM – Rio de Janeiro – Brasil

{victor.verissimo, marcus.paulo.santos.ferreira}@ccc.ufcg.edu.br

livia.buriti@ccc.ufcg.edu.br,joao.arthur@computacao.ufcg.edu.br

ruan.gomes@ifpb.edu.br, tgmoraes@br.ibm.com

Abstract. *The management of private 5G networks remains largely manual and depends on domain expertise. AI-driven agents can reduce this complexity by translating natural-language intents into concrete, executable actions, while the Model Context Protocol (MCP) offers a standardized way to expose network functions as formally defined tools. This paper examines an MCP-enabled AI agent for slice and subscriber management on open-source 5G Core (5GC) platforms. Its main contributions are an MCP server developed from scratch for 5G management, designed to integrate multiple 5GC platforms, and an empirical study of open-source Large Language Models (LLMs) in both CPU-only and GPU-accelerated settings under different prompt-engineering strategies. In CPU-only evaluations, the Granite 4.0 3B-H model provides the highest tool-selection accuracy, whereas the Qwen 2.5 7B model presents a lower latency and higher generation throughput. GPU acceleration improves performance, reducing latency by up to 130×. In general, the findings demonstrate that MCP-enabled AI agents offer a practical solution for managing private 5G networks.*

1. Introduction

5G networks were designed to provide reliable connectivity for services that cover a wide range of requirements in throughput, latency, and device-density. Compared to previous generations, 5G offers a more flexible architecture to meet the increasing demands for connectivity and new applications with stringent performance constraints. To accommodate this service diversity, 5G employs deployment strategies that dynamically adjust to application demands [Eswaran and Honnavalli 2022] and integrates technologies such as network function (NF) virtualization and network slicing, which support rapid instantiation of NFs and the provisioning of logically isolated slices tailored to specific Quality of Service (QoS) needs [Mangipudi and Eswaran 2025].

The 5G network is broadly organized into Radio Access Network (RAN) and 5GC. The RAN is responsible for providing the radio interface that links User Equipments (UEs) with the 5GC. Within the 5GC, the User Plane Function (UPF) is the

NF that connects to the RAN and handles user data traffic, allowing access to external networks. The remaining NFs implement the control and management planes of the 5G system [Peterson and Sunay 2020]. The 5GC follows a Service-Based Architecture (SBA), in which NFs interact through standardized REST Application Programming Interfaces (APIs), yielding an operational model aligned with modern web-based systems [ETSI 2026]. In this architecture, the network configuration becomes a key operational function. The appropriate slice configuration enables more effective QoS monitoring and assurance, which is critical because different services require different parameter settings. Network slicing further allows a shared RAN to assign differentiated resources to various UEs based on service requirements. Consequently, accurate slice-based network configuration is fundamental to exploiting the capabilities of 5G.

Configuring network parameters and provisioning slices is inherently complex. The main difficulty lies in inferring, from each user request, the optimal configuration that ensures QoS given the specific context and operating conditions of the UE. 3GPP specifies three main slice types for typical scenarios: eMBB (enhanced Mobile Broadband), URLLC (Ultra-Reliable Low-Latency Communications), and mMTC (massive Machine-Type Communications) [Peterson and Sunay 2020]. Although these classes offer a useful starting point, they are generic and do not eliminate the need to adjust parameters for each individual service or UE, especially when stringent QoS guaranties are required.

In this context, AI agents represent a promising paradigm for handling intricate operations, including tasks such as 5G network configuration and slice provisioning. These agents are software entities that rely on language models as their main reasoning engine to autonomously solve complex problems by choosing and performing actions [Sapkota et al. 2026]. Their operation follows an iterative cycle in which they interpret goals, plan actions, analyze contextual information, and adapt their strategy based on intermediate results. In addition, they can incorporate external tools to obtain domain-specific knowledge and execute concrete operations, decomposing high-level goals into smaller and tractable subtasks.

The Model Context Protocol (MCP) is an open standard that defines a communication protocol for dynamic discovery between AI models and external resources, such as tools, that can be used to solve specific tasks. Using the MCP standard, external APIs can have their capabilities described in a format that allows AI models to understand, select, and invoke the resources they provide [The Linux Foundation and CAMARA Project 2025]. As communication with 5GC network functions occurs via standardized APIs, the MCP architecture can be used to allow an MCP server to expose network capabilities to AI agents.

Configuring slices and provisioning subscribers in private 5G networks currently requires direct interaction with low-level APIs and deep knowledge of 3GPP parameters, barriers that limit network agility and increase operational costs. This paper addresses this problem by designing and implementing an MCP server that exposes 5GC management operations as standardized tools, enabling LLM-based agents to perform these tasks from natural-language requests, abstracting the underlying 3GPP parameters from the operator. We conducted an empirical evaluation comparing open-source LLMs under different prompt strategies and hardware configurations, including CPU-only scenarios that reflect realistic deployment conditions for private 5G networks where dedicated GPU resources

may not be available [Eswaran and Honnavalli 2022]. More specifically, we explore the APIs provided by an open-source 5GC(free5GC[1]) to configure slice creation, subscriber provisioning, and policy updates. The main contributions of this work are:

- Design and implementation from scratch of an MCP server that exposes 5GC management functions as tools, allowing AI agents to configure network slices and subscribers through natural language via the free5GC REST API;
- Integration with IBM ContextForge[2], an MCP gateway offering a unified interface for centralized MCP server and tool management and monitoring;
- Adoption of the Adapter Design Pattern to decouple the tool interface from 5GC, facilitating future integration with alternative cores;
- An empirical evaluation of four open-source LLMs focusing on tool selection accuracy, inference latency, and generation throughput, conducted both in a CPU-only and a GPU-accelerated scenarios, to reflect distinct deployment conditions;
- An analysis of the influence of prompt engineering strategies (zero-shot, one-shot, and few-shot) on tool selection accuracy.

The results show a trade-off between tool-selection accuracy and inference latency. In the CPU-only, one-shot setting, Granite 4.0 3B-H reached the highest accuracy (98.33%), with Llama 3.1 8B next (91.67%). Qwen 2.5 7B showed lower accuracy (73.33%) but a reduced latency (9.49 s) and higher throughput (3.20 token/s). Hardware acceleration also substantially improved performance: Granite 4.0 3B-H decreased its latency from 79.40 s to 0.61 s and increased its throughput from 0.40 to 44.10 tokens/s.

The remainder of this paper is structured as follows. Section 2.1 introduces key concepts of 5GC architecture, LLM agents, and MCP. Section 4 reviews related work. Section 5 presents the proposed solution and the evaluation methodology. Section 6 describes the experimental results, and Section 7 concludes the paper.

2. Background Information

2.1. 5GC Architecture

5G networks follows 3GPP specifications to ensure global interoperability and robust infrastructure. A key evolution in 5G is the adoption of a SBA, which defines NFs along with their responsibilities [ETSI 2026]. These functions can communicate with each other via RESTful APIs and can orchestrate tasks ranging from initial user registration to dynamic allocation of radio resources for network subscribers. The NFs most relevant to this work are the Unified Data Management (UDM), which manages user identities and subscription data; the Unified Data Repository (UDR), which provides centralized storage for NF data; and the Policy Control Function (PCF), which defines QoS parameters and service-specific rules based on the slice configuration.

In practice, these NFs are managed in the life cycle by the 5GC platform itself (e.g., Open5GS or free5GC). Therefore, application developers and AI agents do not need to invoke each NF endpoint directly. Instead, they interact with management APIs that use high-level abstractions exposed by the core, which internally route requests to the appropriate NF services. This architecture reduces complexity and allows slice-management workflows to be consistent across different core implementations.

[1] `https://free5gc.org`

[2] `https://github.com/IBM/mcp-context-forge`

2.1.1. 5G Network Slicing

In 5G, network slicing is the capability to instantiate multiple logical end-to-end networks over a shared physical infrastructure so that it can support multiple use cases and diverse user groups simultaneously. When creating a network slice, the policy rules are defined according to the specific service requirements and the subscriber profile [Phyu et al. 2023]. This approach improves efficiency according to the intended use of the network and is also a key mechanism for managing QoS [Wijethilaka and Liyanage 2021]. Slice creation follows patterns defined by 3GPP [ETSI 2026]. Each slice is identified by a Slice/Service Type (SST), which indicates the expected service behavior) and optionally by a Slice Differentiator (SD), which distinguishes slices that share the same SST [Cheng et al. 2024]. Although multiple SST values are defined in 3GPP specifications, the most commonly used categories are summarized in Table 1.

Table 1. Main 5G slice types defined by 3GPP [ETSI 2026].

Type	Target	Use Cases
eMBB	High throughput	4K/HD streaming, cloud access, AR/VR
URLLC	Ultra-low latency	Industrial automation, remote control
mMTC	High device density	Large-scale IoT, smart city sensors

2.2. LLM Agents

LLM agents are systems that combine generative language models with external tools and an iterative control loop, enabling them to solve multi-step tasks. Beyond simply producing text, such agents can infer user intent, devise action plans, determine when to invoke tools, and incorporate tool outputs into ongoing reasoning [Luo et al. 2025]. This is supported by components such as memory buffers, planning modules, and tool-invocation APIs, which link passive response generation to active task execution [Sapkota et al. 2026]. By mapping high-level user goals to concrete procedures, these agents are particularly useful in network management, for example, in Intent-Based Networking (IBN), in which an agent can convert natural-language intents into network configuration commands [Boateng et al. 2026].

The effectiveness of this process depends on three main factors: (i) tool selection accuracy, i.e., correctly identifying the operation required for each task; (ii) parameter grounding, which ensures the consistent and correct generation of values for fields such as identifiers and QoS attributes; and (iii) workflow performance, which demands low latency and stable execution of the overall process [Brodimas et al. 2025]. Consequently, our evaluation considers both the final textual response and the sequence of tool calls performed by the agent. Analyzing this sequence provides information on the agent's internal decision-making and procedural robustness.

2.2.1. Model Context Protocol (MCP)

MCP is an open-source standard that specifies how AI agents interface with external tools, systems, and APIs[3].It enables LLMs to discover available tools, under-

[3]`https://modelcontextprotocol.io`

stand their input/output schemas, and invoke external services in a structured, auditable manner. This supports practical user-centric workflows such as calendar management, local document analysis, and software configuration. Agents enabled by MCP have also been used to orchestrate complex network resources, including autonomous cloud provisioning infrastructure and the deployment of sophisticated network applications [Brodimas et al. 2025]. Within 5G networks, the CAMARA initiative is exploring MCP adoption by exposing network APIs as MCP-compatible tools [The Linux Foundation and CAMARA Project 2025]. In the scenario studied in this paper, MCP enables LLM agents to execute network management tasks, such as slice creation and subscriber provisioning, via 5GC platform APIs used as tools. The MCP ecosystem is composed of two main components:

- **MCP Server**: a program that exposes domain functionalities through the MCP protocol. These capabilities are published as standardized interfaces called tools;
- **MCP Client**: the host application in which the LLM runs, which connects to one or more MCP servers and manages permissions, context delivery, and the exchange of tool output with the model.

Using MCP the AI agent can dynamically choose the most suitable tools for any given user request, informed by the set of exposed capabilities and the specific parameters necessary for correct and effective execution.

3. Related Work

The deployment of MCP-based mechanisms to implement IBN is an emerging approach with significant potential. Although empirical studies directly addressing this combination are still limited, current research already provides valuable insights into assessing and optimizing LLMs for 5G networks.

[Hong et al. 2025] survey various studies on LLM-based network management and observe that many studies assess only individual functions (e.g., intent extraction), while neglecting metrics such as end-to-end service-activation success, scalability, resilience to adversarial prompts, security implications, and long-term performance drift. Across the surveyed works, fine-tuning, prompt engineering and RAG emerge as primary adaptation strategies, whereas MCP implementations are not explicitly addressed.

Similarly, Usman et al. [Usman et al. 2025] present a comprehensive survey of the integration of AI, ML, and LLMs into mobile networks. They emphasize the advantages of these technologies for process automation, resource management, and network slicing, while also identifying open challenges such as scalability and reliability. However, the study remains conceptual, providing neither experimental validation nor a concrete implementation of the proposed approaches. Fine-tuning, prompt engineering, and RAG are also highlighted as the main LLM adaptation strategies in telecommunications.

The work in [Chakraborty et al. 2024] presents an AI-driven engine that converts 5G tariff plans into full network configurations, reducing OSS/BSS provisioning times from months to just days. Its core contribution is a four-stage neural filtering pipeline designed to detect and discard syntactically or semantically invalid inputs. The authors assess three model families: a Named Entity Recognition (NER) model, a fine-tuned GPT-3.5-Turbo-16K, and a fine-tuned Llama-2-7B. F1-scores for each extraction task

were examined, achieving a value of 1 with the four-layer filter. Although their approach leverages tariff plans with NER and LLMs for automated network configuration, it neither specifies nor evaluates any MCP-based methodology. In contrast, our work focuses on a distinct problem domain: managing 5GC platforms using an MCP-based approach.

In [Wang et al. 2025], the authors propose an LLM-enhanced failure localization framework deployed in Alibaba Cloud's NOC. The core contribution is a two-stage reasoning pipeline: first, a fine-tuned Qwen2.5-7B model aggregates and summarizes alerts from 11 heterogeneous monitoring systems; second, an additional reasoning module fuses device information, network topology, and temporal correlations to infer the root cause of failures. Reported results include accuracies of up to 99.3%, an overall time-to-root-cause reduction of 20.5%, and an end-to-end latency of about 30 s. Although this work also uses LLMs as the basis for cloud-infrastructure failure location, its operational context differs from that of our study and does not utilize MCP.

The work in [Bandara et al. 2025] introduces a Slice-MCP framework based on a five-layer architecture that separates functionality into a 5G-core service layer, a slice-orchestration layer, an MCP layer, an LLM layer, and an LLM-agent layer. By fine-tuning Llama-4 on a custom intent-to-function dataset, their method enables accurate mapping of natural language intents to MCP function calls. However, the paper does not provide quantitative metrics for accuracy, latency, or resource consumption, although the authors present it as the first MCP-based 5G slice orchestrator reported in the literature. Although this work is conceptually related to ours, our paper advances it by delivering an empirical evaluation of an MCP-based solution, comparing multiple LLM models and reporting quantitative results for accuracy, latency, and generation throughput in both CPU-only and GPU-accelerated settings.

Table 2 summarizes the key dimensions of the reviewed works compared to this paper. As highlighted, none of the related works combines MCP adoption, empirical evaluation, multi-LLM comparison, CPU/GPU hardware analysis, and validation against an open-source 5GC platform simultaneously. This combination constitutes the main distinguishing contribution of the present work.

Table 2. Comparison of related work across key dimensions.

Work	MCP	Empirical	Multi-LLM	CPU/GPU	OSS 5GC
[Hong et al. 2025]	✗	✗	✗	✗	✗
[Usman et al. 2025]	✗	✗	✗	✗	✗
[Chakraborty et al. 2024]	✗	✓	✓	✗	✗
[Wang et al. 2025]	✗	✓	✗	✗	✗
[Bandara et al. 2025]	✓	✗	✗	✗	✓
This work	✓	✓	✓	✓	✓

4. Methodology

We guided our work by the following research question: *Can an MCP-based AI agent reliably translate high-level natural-language intents into correct and practical 5G management actions?* To answer this question, we conducted an experiment that evaluates MCP-based agents in realistic 5G management tasks, comparing LLMs, prompt-engineering strategies, and hardware settings using accuracy, latency, and throughput metrics. The main details of the methodology are presented in the following subsections.

4.1. Experimental environment and 5G architecture

Our solution was built around open-source 5GC, such as Open5GS and free5GC. In this paper, only free5GC was used in the evaluation, but the architecture of the solution allows easy integration of other 5GC plataforms. The server contains multiple tools developed with the intent of creating, deleting, and retrieving subscriptions and profiles. Since these open-source 5GC do not provide essentially dynamic slicing, this study uses profiles as a functional proxy for network slices. This abstraction allows to manage the network configurations and QoS parameters, creating a viable environment for the experiments.

To facilitate experimentation within MCP environment, we employed the ContextForge system [4], which is an MCP gateway developed by IBM, designed to help manage and monitor MCP servers. It acts as a streamlined intermediary between LLMs and MCP tools, centralizing essential tasks such as tool registration, security, and logging in a single interface. By simplifying the orchestration of MCP-based applications, it provides the traceability and reproducibility required for experimental settings. Its support for rapid model switching and isolated execution made it a practical choice for debugging and for ensuring precise, controlled testing.

4.2. Models and Hardware Scenarios

The selection criteria prioritized open-source models with native tool-calling support and varying architectural characteristics, enabling a broad comparison across model families. All models were served locally through Ollama[5], without external API dependencies, to ensure reproducibility and avoid rate-limit constraints. Two hardware scenarios were evaluated. In the **CPU-only** scenario, a computer with a Intel Core i7, and 16 GB of RAM was used, while in the **GPU-accelerated** scenario a GPU NVIDIA RTX A2000 model with 20 GB of VRAM was used.

Five models were evaluated, but only four of them were tested in the CPU-only scenario. The models considered in the CPU scenario are detailed as follows.

- **Qwen 2.5 7B Instruct (Q3_K_M)** [Qwen et al. 2025] is a transformer developed by Alibaba Cloud, with 7 billion parameters and support for context windows up to 128K tokens. It features grouped-query attention (GQA), SwiGLU activations, and RMSNorm, and was trained on 18 trillion tokens with an emphasis on instruction following and structured output generation. It was evaluated in Q3_K_M quantization (3-bit), occupying approximately 3.8 GB of RAM, making it the most memory-efficient model in the CPU scenario;
- **Llama 3.1 8B Instruct (Q4_K_M)** [Grattafiori et al. 2024] is a transformer developed by Meta, released in July 2024, with 8 billion parameters and a 128K token context window. It uses GQA for improved inference efficiency and was fine-tuned with supervised fine-tuning and reinforcement learning from human feedback for instruction following and tool use. It was evaluated in Q4_K_M quantization (4-bit), occupying approximately 4.9 GB of RAM;
- **Granite 4.0 3B-H** [Saon and others 2025] is a hybrid model developed by IBM that interleaves standard transformer self-attention blocks with Mamba-2 state-space layers in a 9:1 ratio. With 3 billion parameters, it is designed for low-latency

[4] `https://ibm.github.io/mcp-context-forge/`

[5] `https://github.com/ollama/ollama`

applications and was optimized for agentic tasks, including function calling and instruction following. Compared to conventional transformer, the hybrid architecture reduces memory requirements by up to 70% for long-context inference.

- **Mistral 7B Instruct v0.3 (Q4_K_M)** [Jiang et al. 2023] is a 7-billion parameter transformer developed by Mistral AI, released in May 2024. Version 0.3 introduced native function calling support, representing one of the first openly available small models to implement this capability. It uses GQA and sliding window attention for efficient inference. It was evaluated in Q4_K_M quantization, which required approximately 4.1 GB of RAM.

In the GPU-accelerated scenario, two Granite 4.0 models were evaluated: **Granite 4.0 Tiny-H**, a 7-billion parameter hybrid Mixture of Experts (MoE) model with approximately 1 billion active parameters per token, designed for high-throughput deployment with reduced computational cost; and **Granite 4.0 3B-H**, included to assess whether the hybrid Mamba-2 architecture, which showed high accuracy but prohibitive inference latency in the CPU-only scenario, would benefit significantly from GPU acceleration. This hypothesis is supported by the fact that Mamba-2 state-space kernels are optimized for parallel execution on GPU hardware.

4.3. System Architecture

Figure 1 illustrates the end-to-end workflow of the proposed solution. The process begins on the **MCP Client**, where a system prompt defining the agent behavior and 3GPP knowledge base is combined with an Ollama-served LLM, whose identity is configured via environment variables. When a user submits a natural language request, the client forwards it to the LLM, which reasons over the available tools and makes the decision.

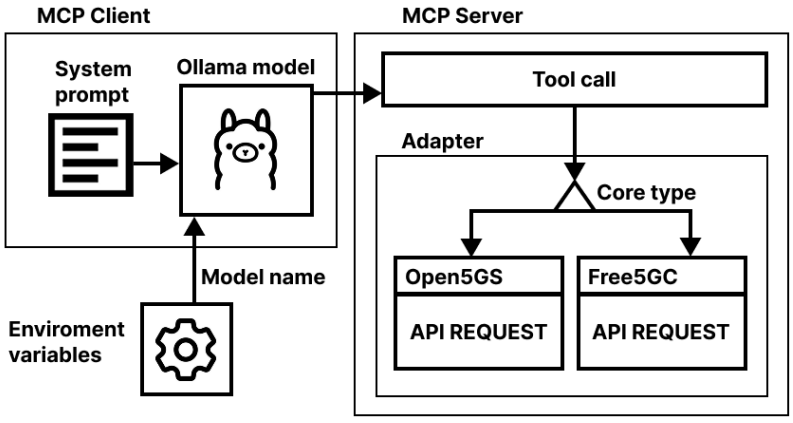

Figure 1. System workflow

The tool call is dispatched to the **MCP Server**, which receives the request and identifies the appropriate tool to invoke. The server then delegates execution to the **Adapter** layer, which decouples the tool interface from the underlying 5GC implementation. Based on the core name provided via environment variables, the Adapter selects the corresponding implementation (e.g. Open5GS or free5GC) and translates the tool call into one or more REST API requests directed at the target core. The result is returned through the same chain to the LLM, which summarizes the outcome and presents it to the user in natural language. Using this strategy, the system can interact with different 5GC platforms, and the LLM model only needs to worry about choosing the right tools.

4.4. Evaluation Framework and Metrics

To evaluate the response of the LLM agent and the use of the tools, a script was developed that submits 12 natural language tasks to record the calls and metrics of the tool produced at each step. The tasks submitted to the agent fully cover the life cycle of 5G network subscriber and slice management: listing, creating, and deleting subscribers and network profiles. Tasks are divided between low-level intent (e.g. *"create a new network profile for AR/VR with SST 1 and SD 010203, 1Gbps uplink and downlink"*), which specifies the expected configuration parameters, and high-level intent (e.g. *"set up a slice suitable for online gaming"*) that require the model to infer 3GPP slice parameters such as SST (*Slice/Service Type*) and SD (*Service Differentiator*) from natural language without explicit parameter specification.

Each task is annotated with a list of expected tools. A task is marked as *correct* if the model invokes at least one of the expected tools before producing a final text response. Tasks that result in a text response without any tool invocation or that invoke only unexpected tools are marked *incorrect*. A warm up task (*"what operations can you perform on the 5G network?"*) expects no tool invocation and is excluded from accuracy computation. The benchmark resets the conversation history between tasks to ensure that prompt token counts are comparable across models and runs, and to prevent dependency between successive tasks. A cleanup routine runs before each benchmark run to guarantee a consistent network initial state. Each unique combination of model, prompt, and hardware configuration was assessed across five independent trials. The following quantitative indicators were recorded:

- **Tool selection accuracy**: proportion of tasks in which the model invoked the correct tool;
- **Inference latency**: time elapsed from prompt submission to tool invocation (`inference_time_sec`);
- **Generation throughput**: completion tokens per second (`tokens_per_second`).

4.5. Prompt Engineering

Three prompt strategies were evaluated to assess their influence on tool selection accuracy:

- **Zero-shot**: instructions and a 3GPP knowledge base (SST/SD mappings) with no usage examples;
- **One-shot**: same as zero-shot, with one interaction example (creation of a network slice);
- **Few-shot**: same as zero-shot, with seven examples covering all available tools.

All prompts include the same behavioral instructions (e.g. "execute only the requested tool, stop after tool invocation") and the same 3GPP knowledge base. The only variable between strategies is the number of in-context examples. The selection of models and prompt strategy were managed using environment variables to ensure reproducibility.

5. Results

This section presents the results obtained in the evaluations of the proposed MCP-based system. The results are organized into three parts: (i) tool selection accuracy and inference performance across models in the CPU-only scenario; (ii) a comparison between CPU-only and GPU-accelerated scenarios using Granite 4.0 models; and (iii) the influence of prompt engineering strategies on tool selection accuracy.

5.1. Tool Selection Accuracy and Inference Performance

Table 3 summarizes tool selection accuracy, mean inference latency, and generation throughput for the four models evaluated in the CPU-only scenario, using the one-shot system prompt over five runs. Mistral 7B failed to invoke any tools in all runs, producing only responses in natural language. This behavior is attributed to an incompatibility between the tool-calling format expected by the model and the format provided by the Ollama inference backend, rather than a fundamental limitation of the model.

Table 3. Results for CPU-only scenario, one-shot prompt, and five runs.

Model	Accuracy (%)	± Std	Latency (s)	Throughput (t/s)
Granite 4.0 3B-H	98.33	3.73	79.40	0.40
Llama 3.1 8B (Q4)	91.67	0.00	82.05	0.34
Qwen 2.5 7B (Q3)	73.33	10.87	9.49	3.20
Mistral 7B (Q4)	0.00	0.00	—	—

Granite 4.0 3B-H achieved the highest tool selection accuracy (98.33%), with low variation between runs demonstrating consistent behavior in function-calling tasks. Llama 3.1 8B showed stable but slightly lower accuracy (91.67%) with zero standard deviation, indicating deterministic tool selection behavior across all runs. Qwen 2.5 7B presented the highest variance (± 10.87%), with occasional failures including malformed tool invocations and responses in unexpected languages (e. g. chinese), behavior consistent with output instability caused by Q3_K_M quantization.

Granite 4.0 3B-H and Llama 3.1 8B exhibited similar latency (80 s per tool call), despite their architectural differences. Granite 4.0 3B-H, with 3 billion parameters, was executed in full precision, while Llama 3.1 8B, with 8 billion parameters, was run using 4-bit quantization. Qwen 2.5 7B achieved significantly lower latency (9.49 s) under Q3_K_M quantization. This discrepancy suggests that the combination of more aggressive quantization (3-bit vs. 4-bit) and architectural optimizations in Qwen 2.5 [Qwen et al. 2025], such as grouped-query attention and efficient tokenization, results in a smaller effective memory footprint that better alleviates the CPU bandwidth bottleneck, at the cost of higher output instability observed in its accuracy variance.

5.2. Comparison between CPU-only and GPU-accelerated Scenarios

Figure 2 shows a comparison between Granite 4.0 3B-H in CPU-only and GPU-accelerated hardware scenarios, and include Granite 4.0 Tiny-H as an additional GPU reference point.

GPU acceleration had a significant impact on inference performance. Granite 4.0 3B-H latency dropped from 79.40 s to 0.61 s when moving from CPU to GPU, a reduction of approximately 130×, while throughput increased from 0.40 to 44.10 token/s.

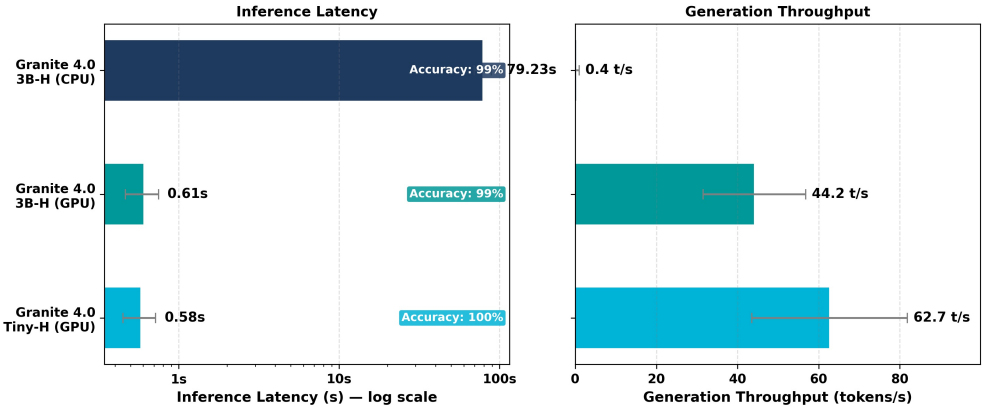

Figure 2. Hardware comparison in the one-shot setting for Granite models.

Importantly, tool selection accuracy also improved slightly, reaching 100% on the GPU compared to 98.33% on CPU, suggesting that higher generation quality under GPU execution reduces borderline failure cases. Granite 4.0 Tiny-H, being the evaluated model with more parameters on this scenario, achieved even higher throughput (62.69 token/s) with comparable accuracy (100%), demonstrating that the smaller MoE variant is a viable alternative for latency-sensitive deployments. These results confirm that the hybrid Mamba-2 architecture of Granite 4.0 [Saon and others 2025] benefits significantly from GPU-level parallelism, where its state-space kernels can be fully exploited [Dao and Gu 2024].

5.3. Influence of Prompt Engineering

To evaluate the sensitivity of the models to the system prompt, three prompting strategies were compared: zero-shot (no examples), one-shot (two examples), and few-shot (seven examples covering all available tools). Figure 3 presents the tool selection accuracy for each strategy across the evaluated model/hardware configurations.

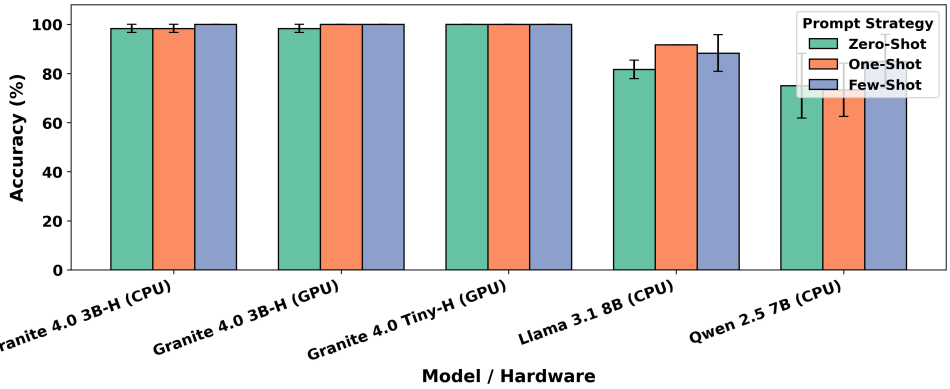

Figure 3. Influence of prompt engineering strategies on tool selection accuracy.

As shown in Figure 3, Granite 4.0 models demonstrated high accuracy in all three prompt strategies (consistently at or above 98%) suggesting robustness to prompt engineering choices, likely due to their targeted training for agentic function calling [Saon and others 2025]. Llama 3.1 8B [Grattafiori et al. 2024] similarly maintained stable accuracy (≈91–100%) across strategies. In contrast, Qwen 2.5 7B [Qwen et al. 2025] showed a higher sensitivity to the prompt strategy: zero-shot

yielded the lowest accuracy (≈82%), while one-shot improved the results to ≈92%. The non-monotonic behavior between one-shot and few-shot suggests that additional examples may introduce noise under aggressive quantization due to the longer context required. Overall, prompt engineering had limited influence on models with strong agentic training, while one-shot prompting offered the best trade-off for general-purpose models.

5.4. QoS-Oriented Qualitative Assessment

In addition to tool-selection accuracy and inference performance, we conducted a qualitative QoS-oriented assessment for high-level intent tasks. The goal was to verify whether the selected parameters were semantically coherent with the intended service profile, not only whether the correct tool was called. We adopted a simple 0–2 adequacy score: (0) semantically incorrect configuration; (1) partially coherent configuration; and (2) semantically coherent configuration according to the requested slice intent.

Using this criterion in the one-shot CPU consolidated runs (Qwen 2.5 7B and Granite 4.0 3B-H), both models were generally consistent for eMBB and gaming-oriented requests, but differences emerged in mMTC provisioning. Qwen repeatedly generated low-throughput mMTC profiles (e.g., 100Kbps–1Mbps), while Granite produced mMTC profiles with downlink bitrate values in the 50–100Mbps range, which are less aligned with typical massive-IoT traffic behavior.

The following User-Agent interaction examples illustrate both cases:

Example A (coherent QoS configuration):

- User request: *"provision a massive IoT slice for smart city sensors"*
- Agent response (tool call):
  `create_network_profile(profile_name="SmartCitySensors", sst=3, sd="000003", uplink="100Kbps", downlink="100Kbps")`
- Interpretation: the selected SST and low-throughput parameters are consistent with mMTC requirements.

Example B (tool correct, QoS parameters suboptimal):

- User request: *"provision a massive IoT slice for smart city sensors"*
- Agent response (tool call):
  `create_network_profile(profile_name="mMTC_SmartCity", sst=3, sd="000003", uplink="1Mbps", downlink="100Mbps")`
- Interpretation: although the selected tool and SST are correct, the downlink target is comparatively high for a typical mMTC scenario, reducing semantic adequacy.

This analysis reinforces that, for slice orchestration, deployment viability in CPU-only environments, and tool-calling accuracy should be complemented by semantic QoS adequacy. In the one-shot CPU dataset, Qwen achieved lower latency and better semantic consistency for mMTC intent, while Granite remained competitive in tool execution but with occasional QoS-parameter mismatch in high-level intent inference.

6. Conclusion and Future Work

This work introduces an MCP-based solution for automated slice and subscriber management in private 5G networks. An MCP server, along with IBM ContextForge integration, exposes REST endpoints as standardized tools that interface with 5G core APIs,

while the Adapter Design Pattern decouples tool definitions from the underlying core, permitting transparent interchange of platforms such as free5GC and Open5GS without altering agent logic. The MCP client integrates a locally hosted LLM (via Ollama) that interprets natural-language intents, selects the appropriate tool, and executes the required configuration. The evaluation of five open-source LLMs under both CPU-only and GPU-accelerated scenarios, in conjunction with zero-, one-, and few-shot prompt engineering strategies, shows that Granite 4.0 3B-H achieves the highest tool-selection accuracy ($\approx$98%) in the one-shot CPU configuration. Furthermore, GPU acceleration decreases inference latency from 79 s to 0.61 s and increases generation throughput from 0.40 to 44.10 tokens/s ($\approx$130$\times$ speed-up). Across all evaluated models, one-shot prompting consistently delivers the best accuracy.

As future work, we intend to (i) fine-tune LLMs on domain-specific 5G data to enhance QoS-parameter grounding and eliminate residual mismatches, (ii) extend the orchestration layer to invoke native 3GPP network-function APIs in production-grade cores, thereby minimizing reliance on intermediate abstractions, and (iii) explore a multi-agent paradigm where dedicated agents for intent parsing, policy synthesis, execution verification, and real-time monitoring cooperate to improve robustness, scalability, and decision quality in complex slice-management workflows.

Acknowledgments

This work was funded by FIT Instituto de Tecnologia/IBM (EMBRAPII PCEE-2310.0243). The authors also thank CNPq (307108/2025-2).

References

Bandara, E., Bouk, S. H., et al. (2025). Slice-mcp — fine-tuned llama-4 llm and mcp enabled 5g network slice orchestration platform. In *MILCOM 2025 - 2025 IEEE Military Communications Conference (MILCOM)*, pages 1506–1511.

Boateng, G. O., Sami, H., et al. (2026). A survey on large language models for communication, network, and service management: Application insights, challenges, and future directions. 28:527–566.

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

Chakraborty, S., Chitta, N., and Sundaresan, R. (2024). Automation of network configuration generation using large language models. In *2024 20th International Conference on Network and Service Management (CNSM)*, pages 1–7.

Cheng, H., D'Oro, S., et al. (2024). Oranslice: An open source 5g network slicing platform for o-ran. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ACM MobiCom '24, page 2297–2302, New York, NY, USA. Association for Computing Machinery.

Dao, T. and Gu, A. (2024). Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. https://arxiv.org/abs/2405.21060.

Eswaran, S. and Honnavalli, P. (2022). Private 5G networks: a survey on enabling technologies, deployment models, use cases and research directions: Private 5G networks:

a survey on enabling technologies, deployment models, use cases and research directions. *Telecommun. Syst.*, 82(1):3–26.

ETSI (2026). 5G; System architecture for the 5G System (5GS). Technical Specification ETSI TS 123 501, European Telecommunications Standards Institute (ETSI). 3GPP TS 23.501 version 19.6.0 Release 19. Accessed: 2026-01-31.

Grattafiori, A., Dubey, A., et al. (2024). The llama 3 herd of models. https://arxiv.org/abs/2407.21783.

Hong, J., Tu, N. V., and Hong, J. W.-K. (2025). A comprehensive survey on llm-based network management and operations. *International Journal of Network Management*, 35(6):e70029. e70029 NEM-25-0670.

Jiang, A. Q., Sablayrolles, A., et al. (2023). Mistral 7b. https://arxiv.org/abs/2310.06825.

Luo, J., Zhang, W., et al. (2025). Large language model agent: A survey on methodology, applications and challenges. https://arxiv.org/abs/2503.21460.

Mangipudi, G. M. and Eswaran, S. (2025). Network slicing in 5g: A survey on the concepts, use cases, testbeds and open challenges. In *2025 1st International Conference on Emerging Trends in Information Systems and Informatics (ICETISI)*, pages 1–7.

Peterson, L. and Sunay, O. (2020). *5G Mobile Networks: A Systems Approach*. Morgan & Claypool Press, 1 edition.

Phyu, H. P., Naboulsi, D., and Stanica, R. (2023). Machine learning in network slicing—a survey. *IEEE Access*, 11:39123–39153.

Qwen, :, Yang, A., Yang, B., et al. (2025). Qwen2.5 technical report. https://arxiv.org/abs/2412.15115.

Saon, G. and others, A. D. (2025). Granite-speech: open-source speech-aware llms with strong english asr capabilities. *arXiv*.

Sapkota, R., Roumeliotis, K. I., and Karkee, M. (2026). Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *Information Fusion*, 126:103599.

The Linux Foundation and CAMARA Project (2025). In Concert: Bridging AI Systems Network Infrastructure through MCP: How to Build Network-Aware Intelligent Applications. Technical report. 12 pages.

Usman, Y., Oladipupo, H., et al. (2025). Ai, ml, and llm integration in 5g/6g networks: A comprehensive survey of architectures, challenges, and future directions. *IEEE Access*, 13:168914–168950.

Wang, C., Zhang, X., et al. (2025). Towards llm-based failure localization in production-scale networks. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 496–511, New York, NY, USA. Association for Computing Machinery.

Wijethilaka, S. and Liyanage, M. (2021). Survey on network slicing for internet of things realization in 5g networks. *IEEE Communications Surveys & Tutorials*, 23(2):957–994.