Intento, logo programo: Uma camada de agentes para traduzir prompts em chamadas para a API de operações do núcleo 5G

Rafael C. Chaves[1,2], Rebeca D. Cabral[1,2], Rafael R. Silva[3], João Paulo Esper[3], Kleber V. Cardoso[3], Leandro C. de Almeida[2], Ruan D. Gomes[2], Fábio L. Verdi[1]

[1]Departamento de Computação (Dcomp) - Universidade Federal de São Carlos (UFSCar)

[2]Unidade Acadêmica de Informática - Instituto Federal da Paraíba (IFPB)

[3]Instituto de Informática (INF) - Universidade Federal de Goiás (UFG)

{cavalvante.chaves, rebeca.cabral}@academico.ifpb.edu.br,

{leandro.almeida, ruan.gomes}@ifpb.edu.br, verdi@ufscar.br,

{silva_rafael, joaopauloesper}@discente.ufg.br, kleber@ufg.br

Resumo. *As redes 5G introduziram mecanismos avançados de programabilidade por meio de APIs padronizadas, com destaque para a Network Exposure Function (NEF), que permite a exposição controlada de capacidades do núcleo da rede a aplicações externas. No entanto, a complexidade semântica e estrutural dessas APIs ainda representa uma barreira significativa para sua adoção e automação. Neste artigo, investigamos o uso de sistemas baseados em Agentes de IA para realizar chamadas à API Traffic Influence da NEF no núcleo 5G, avaliando a capacidade de Large Language Models (LLMs) de traduzir solicitações de alto nível em interações corretas com um componente do sistema 5G. A solução proposta utiliza o Model Context Protocol (MCP) para integrar os modelos a ferramentas que encapsulam a API Traffic Influence da NEF. Avaliamos diferentes LLMs, incluindo Qwen, GPT-OSS-20B e Claude Sonnet 4.5, considerando variações no tamanho dos modelos e diferentes estratégias de engenharia de prompts. As avaliações foram realizadas em um ambiente baseado em free5GC e UERANSIM, contemplando métricas de taxa de acerto, consumo de tokens e tempo de resposta.*

Abstract. *5G networks have introduced advanced programmability mechanisms through standardized APIs, most notably the Network Exposure Function (NEF), which enables the controlled exposure of core network capabilities to external applications. However, the semantic and structural complexity of these APIs still represents a significant barrier to their adoption and automation. In this paper, we investigate the use of AI Agents to perform calls to the NEF Traffic Influence API in the 5G core, evaluating the ability of Large Language Models (LLMs) to translate high-level requests into correct interactions with a 5G system component. The proposed solution leverages the Model Context Protocol (MCP) to integrate the models with tools that encapsulate the NEF Traffic Influence API. We evaluate different LLMs, including Qwen, GPT-OSS-20B, and Claude Sonnet 4.5, considering variations in model size and different prompt engineering strategies. The evaluations were conducted in an environment based on free5GC and UERANSIM, using metrics such as success rate, token consumption, and response time.*

1. Introdução

As redes 5G representam uma evolução significativa na forma como as capacidades de rede são arquitetadas, gerenciadas e expostas a aplicações externas. No núcleo dessas redes, a *Network Exposure Function* (NEF), definida pelo 3GPP, desempenha um papel central ao disponibilizar APIs padronizadas que permitem acesso controlado a informações e funcionalidades do núcleo da rede 5G (5G *Core* - 5GC), como subscrição a eventos, monitoramento de sessões, controle de políticas e otimização de recursos [3GPP 2023].

Apesar desse avanço, a interação direta com as APIs da NEF ainda impõe desafios relevantes aos desenvolvedores. As especificações do 3GPP são extensas, os *endpoints* apresentam forte acoplamento semântico ao domínio de telecomunicações e a correta composição de chamadas exige conhecimento especializado do funcionamento interno da rede 5G. Como consequência, a adoção dessas APIs por desenvolvedores externos ao domínio de redes permanece limitada, motivando a comunidade a buscar abstrações de mais alto nível que simplifiquem o consumo das capacidades de rede [GSMA 2023].

Paralelamente, avanços recentes em *Large Language Models* (LLMs) têm impulsionado o surgimento de sistemas baseados em Agentes de IA, nos quais modelos são capazes de interpretar intenções de alto nível, planejar ações e interagir com sistemas externos por meio de ferramentas e APIs. Trabalhos recentes demonstram o potencial desses agentes para a automação de sistemas complexos e a orquestração de serviços [Zhou et al. 2025, Brodimas et al. 2025, Manias et al. 2024, Khan et al. 2025]. No domínio de redes, estudos vêm explorando o uso de IA para extração de intenções, automação e controle em arquiteturas 5G e além.

Neste artigo, propomos uma solução baseada em Agentes de IA para a realização de chamadas à API *Traffic Influence* da NEF [3GPP 2023], na qual LLMs atuam como agentes responsáveis por *"tentar"* traduzir solicitações de alto nível (*prompts*) em sequências corretas de interações com a rede. A API *Traffic Influence*, definida pelo 3GPP no contexto da NEF, permite que aplicações externas influenciem o tratamento de tráfego no núcleo da rede 5G de forma controlada e programável. Por meio dessa API, aplicações podem solicitar à rede ajustes dinâmicos em políticas de roteamento, priorização e ancoragem de sessões, com o objetivo de otimizar métricas como latência, confiabilidade e utilização de recursos, de acordo com requisitos específicos de serviços ou aplicações.

A arquitetura proposta utiliza o *Model Context Protocol* (MCP) como mecanismo de integração entre os modelos e ferramentas externas, permitindo que o agente descubra e invoque funções associadas à API *Traffic Influence* de forma estruturada. Avaliamos diferentes modelos de LLM representativos de distintas famílias e arquiteturas, incluindo Qwen3:4B, Qwen3:8B, GPT-OSS-20B, Claude Haiku 4.5 e Claude Sonnet 4.5. Investigamos também o impacto do tamanho do modelo sobre o desempenho do agente, variando versões com diferentes números de parâmetros sempre que disponíveis. Analisamos o efeito de diferentes estratégias de engenharia de *prompts* sobre o comportamento dos agentes. Consideramos três categorias de *prompts* — *Informativo*, *Direto* e *Vago* — e avaliamos métricas que refletem tanto a eficácia funcional quanto a viabilidade prática da solução, incluindo a taxa de acerto das chamadas às APIs, o consumo de tokens e o tempo de resposta.

A pri ncipal contribuição deste trabalho consiste na observação do que é alcançável, possível e razoável por meio de Agentes de IA para APIs do 5GC. Apesar dos avanços recentes em LLMs, os resultados evidenciam que obter uma taxa de acerto satisfatória a partir de *prompts* ainda é um desafio significativo. A API NEF é intrinsecamente

complexa, com uma grande quantidade de parâmetros, fortes dependências entre campos e semântica acoplada às especificações do 3GPP, o que torna difícil para os modelos gerar chamadas corretas e completas de forma consistente.

Nesse sentido, este trabalho assume um caráter fundamental, ao expor de forma crítica as dificuldades enfrentadas por agentes baseados em LLMs e ao estabelecer uma linha de base necessária para pesquisas futuras, indicando que a adoção efetiva de APIs baseadas em Agentes de IA em redes 5G demandará avanços substanciais em modelagem, abstração, validação e engenharia de *prompts*.

As principais contribuições deste trabalho são:

- A proposição de uma arquitetura baseada em Agentes de IA para interação direta com a API *Traffic Influence* da NEF;
- Uma avaliação comparativa entre diferentes LLMs no contexto de chamadas a APIs de rede;
- Uma análise sistemática do impacto do tipo de *prompt* e do tamanho do modelo sobre a acurácia, o custo e a latência.
- A disponibilização em código aberto dos artefatos gerados para reprodutibilidade, adição de funcionalidades e novas APIs da NEF[1].

Este artigo está organizado da seguinte forma: a Seção 2 apresenta os conceitos fundamentais necessários para a compreensão deste artigo. A Seção 3 posiciona este trabalho em relação à literatura relacionada. Na Seção 4, apresentamos a metodologia experimental e na Seção 5 discutimos os resultados. Finalmente, a Seção 6 conclui o artigo e destaca alguns trabalhos futuros.

2. Conceitos Fundamentais

2.1. Agentes de IA

Agentes de IA são sistemas de software autônomos ou semi-autônomos capazes de raciocinar e executar ações visando atingir objetivos específicos em um contexto bem definido, com base em modelos de IA. Agentes de IA geralmente atuam de forma isolada, mas avanços recentes têm explorado o conceito de IA Agêntica, que se baseia na colaboração e coordenação de múltiplos agentes de IA, visando decompor atividades complexas em um conjunto de tarefas específicas a serem executadas por diferentes agentes de IA especializados [Sapkota et al. 2026].

Recentemente, LLMs têm impulsionado o desenvolvimento de agentes de IA, permitindo a interpretação de instruções em linguagem natural, geração de planos de ação e a decisão sobre quais ferramentas ou APIs devem ser utilizadas para uma determinada tarefa [Sapkota et al. 2026]. Em um Agente de IA, a interação entre os modelos de IA subjacentes (LLMs) e suas capacidades (ferramentas que ele é capaz de usar) é baseada em um padrão denominado ReAct (*Reasoning and Acting*), o qual implementa um ciclo iterativo que alterna entre raciocínio interno e ação externa. Nesse contexto, os LLMs podem processar mensagens de entrada e identificar funções a serem executadas por meio do uso de ferramentas externas. Uma vez que essas funções são executadas, a resposta pode ser reintegrada, permitindo que o agente determine os próximos passos, até que a tarefa seja concluída [Brodimas et al. 2025].

[1]https://github.com/rafaelcchaves/nef-api-agent.

Model Context Protocol (MCP)

O MCP é um padrão que define um protocolo de comunicação para a descoberta dinâmica entre modelos de IA e ferramentas ou recursos que podem ser utilizados para resolver as tarefas solicitadas por meio dos modelos de IA. Usando o padrão MCP, as APIs externas podem ter suas capacidades descritas em um formato adequado para que os modelos de IA consigam entender, selecionar e invocar os recursos disponibilizados por elas [The Linux Foundation and CAMARA Project 2025].

O servidor MCP hospeda e executa os serviços externos que um modelo de IA pode invocar e oferece três funcionalidades principais: ferramentas, recursos e *prompts*. As ferramentas permitem a invocação de serviços externos ou APIs, os recursos fornecem dados ao modelo e os *prompts* são *templates* reutilizáveis que guiam a definição de fluxos de trabalho orientados por IA [Hou et al. 2025]. Uma vez que a comunicação com as funções de rede do 5GC ocorre por meio de APIs padronizadas, como discutido em mais detalhes na Seção 2.2, é possível utilizar a arquitetura MCP para permitir que o servidor MCP exponha as capacidades da rede para os agentes de IA.

2.2. Gerenciamento de Redes 5G por Meio de *Network* APIs

O núcleo da rede 5G (5GC) é composto por um conjunto de funções de rede responsáveis por prover serviços como o controle de acesso dos UEs à rede, o gerenciamento de sessões e a aplicação de políticas de controle e encaminhamento de tráfego de dados. O 5GC adota uma arquitetura baseada em serviços, na qual as funções de rede são implementadas como funções de rede virtualizadas (*Virtual Network Functions* — VNFs). A comunicação entre essas funções ocorre por meio de interfaces padronizadas (*Service-Based Interfaces* — SBIs) que utilizam REST e APIs definidas pelo 3GPP [Peterson and Sunay 2020].

Em geral, a comunicação entre as funções de rede ocorre por meio de interfaces internas que não são expostas para aplicações externas. Por outro lado, a função NEF expõe um conjunto de serviços do 5GC a aplicações externas por meio de APIs padronizadas pelo 3GPP, permitindo que aplicações autorizadas realizem operações de monitoramento, controle e gerenciamento da rede 5G, como criar fatias de rede [Ksentini and Frangoudis 2020] e influenciar as decisões de roteamento para o tráfego do plano de usuário. O 3GPP utiliza o termo *Application Function* (AF) para se referir a uma aplicação externa que interage com o 5GC por meio da NEF [3GPP 2023].

Especificamente, a API *Traffic Influence* da NEF permite realizar o ajuste dinâmico em políticas de roteamento de tráfego visando otimizar o desempenho de serviços ou aplicações. Essa API define dois recursos: *Traffic Influence Subscription* e *Individual Traffic Influence Subscription*. O primeiro possui os métodos GET e POST, que permitem ler todas as *subscriptions* existentes e criar uma nova *subscription* para uma AF, respectivamente. Uma vez que uma nova *subscription* é criada, uma URL específica é associada a ela e retornada pela NEF. O segundo recurso permite manipular *subscriptions* específicas, por meio dos métodos GET, PUT, PATCH e DELETE, que permitem ler os dados da *subscprition*, atualizar todos os dados da *subscription*, atualizar parte dos dados da *subscription* ou remover a *subscription*, respectivamente. Os detalhes a respeito da API podem ser encontrados em [3GPP 2023].

3. Trabalhos Relacionados

Nesta seção, são apresentados os principais trabalhos relacionados ao uso de sistemas baseados em agentes de IA para a realização de chamadas às APIs da NEF no 5GC.

Para esse fim, são definidos seis critérios utilizados na classificação e análise comparativa de cada trabalho, apresentadas na Tabela 1. Todos os critérios são autoexplicativos, exceto *Execução em Multi-etapas* e *Diversidade de Prompt*. O primeiro critério refere-se à capacidade do sistema de orquestrar e executar sequências dependentes de ações ou chamadas de API, nas quais etapas subsequentes dependem dos resultados das anteriores. O segundo está relacionado à avaliação do sistema sob diferentes níveis de informação nos *prompts*, tais como informativo, direto e vago.

Tabela 1. Comparação das principais características dos trabalhos relacionados: ○ não atendido, ◐ parcialmente atendido e ● completamente atendido.

Trabalho	Agente de IA	MCP	Execução Multi-etapas	NEF	*Traffic Influence*	Avaliação Experimental	Diversidade de *Prompt*
Manias et al. 2024	◐	○	○	○	○	◐	●
Khan et al. 2025	◐	○	○	●	◐	●	○
Brodimas et al. 2024	◐	○	○	○	○	◐	○
Brodimas et al. 2025	●	●	●	○	○	●	◐
Habib et al. 2025	◐	○	●	○	○	●	●
Bimo et al. 2025	●	○	●	○	○	●	○
Este trabalho	●	●	●	●	●	●	●

Utilizando um modelo de LLM com engenharia de *prompts*, em [Manias et al. 2024] é descrito um classificador de intenções que identifica seis tipos de requisições de gerenciamento do 5GC conforme a especificação [3GPP 2024], a saber: *Implantação*, *Modificação*, *Garantia de Desempenho*, *Relatório de Intenção*, *Verificação de Viabilidade* e *Notificação Regular*. Porém, a avaliação apresenta apenas seis exemplos qualitativos, sem métricas quantitativas (e.g., acurácia, precisão, recall), posicionando esse componente de extração de intenções como um primeiro passo em direção a um sistema agêntico autônomo completo que ainda precisa ser desenvolvido.

O trabalho apresentado em [Khan et al. 2025] realiza o *fine-tuning* de um modelo de LLM de dois bilhões de parâmetros com um conjunto de dados sintéticos limitado (i.e., apenas 765 registros derivados de 7 especificações da API NEF) para alcançar valores próximos a 100% de acurácia na geração de chamadas de API. Entretanto, os autores reconhecem os riscos de *overfitting* e, notavelmente, falham em demonstrar uma integração prática com APIs, devido a desafios de implementação e restrições arquiteturais.

[Brodimas et al. 2024] apresenta um arcabouço de gerenciamento de rede baseado em intenções que utiliza um modelo de LLM com *few-shot learning* para traduzir descrições de alto nível dos usuários em modelos de arquivos de configuração consumíveis pela infraestrutura de rede. Esses arquivos são convertidos em chamadas de APIs de ordem de serviço padronizadas pelo TM Forum[2] para o OSS OpenSlice[3]. Os autores demonstram uma prova de conceito para três tipos de fatiamento em redes 5G, mas fornecem apenas validação qualitativa.

Estendendo o trabalho anterior, em [Brodimas et al. 2025] os autores apresentam um arcabouço de orquestração de intenções baseado em agentes de IA que decompõe autonomamente solicitações em linguagem natural em ações multi-etapas de gerenciamento de infraestrutura e serviços, utilizando agentes baseados em LLM e ferramentas via MCP. Embora tenha sido validado experimentalmente e seja eficaz para implantações *cloud-native*, o arcabouço não integra interfaces padronizadas de controle das redes 5G, o

[2]https://www.tmforum.org/.

[3]https://osl.etsi.org/.

que limita seu impacto atual em nível de rede. Além disso, APIs de influência de tráfego da NEF também não foram consideradas no trabalho apresentado.

Outros dois trabalhos apresentam arcabouços voltados ao gerenciamento da RAN orientada por intenções, ambos explorando o uso de modelos de linguagem de grande porte. Em [Habib et al. 2025], um arcabouço multi-etapas assistido por LLM que combina interpretação de intenções, validação preditiva e orquestração de aplicações baseadas em *decision transformers* alcança ganhos de desempenho em cenários de simulação. No entanto, sua atuação permanece restrita ao controle em nível de RAN e não incorpora interfaces padronizadas do 5GC, o que limita sua generalidade arquitetural. Já em [Bimo et al. 2025], é apresentado um arcabouço baseado em LLM agêntica capaz de traduzir intenções de alto nível em ações de configuração da RAN, obtendo melhorias na eficiência energética também em simulação, mas mantendo-se igualmente confinado a ajustes na RAN, sem integração com interfaces de controle do 5GC.

4. Metodologia do Experimento

Nesta seção, é descrita a metodologia seguida nos experimentos realizados para avaliar a eficácia e a viabilidade do uso de Agentes de IA como uma camada intermediária na execução de operações no 5GC. Inicialmente, a arquitetura do sistema é apresentada, bem como a integração entre seus componentes. Em seguida, é descrita a metodologia aplicada, incluindo os cenários explorados, as variáveis de controle, as métricas coletadas e o protocolo de execução adotado para as iterações do experimento.

4.1. Arquitetura do Sistema

O sistema proposto foi desenvolvido a partir da integração de três componentes principais, cada um responsável por uma etapa específica do processamento. O Agente de IA atua na interface com o usuário, recebendo os pedidos por meio de *prompts*, identificando a intenção, raciocinando e planejando a execução. O Cliente MCP funciona como intermediário entre o Agente e o Servidor MCP, sendo responsável por gerenciar a importação e a disponibilização das ferramentas que o Agente utiliza. A comunicação entre o Cliente e Servidor MCP foi implementada utilizando o protocolo *Server-Sent Events* (SSE). Por fim, o Servidor MCP traduz as requisições de execução de ferramentas em chamadas HTTP REST para a API *Traffic Influence*.

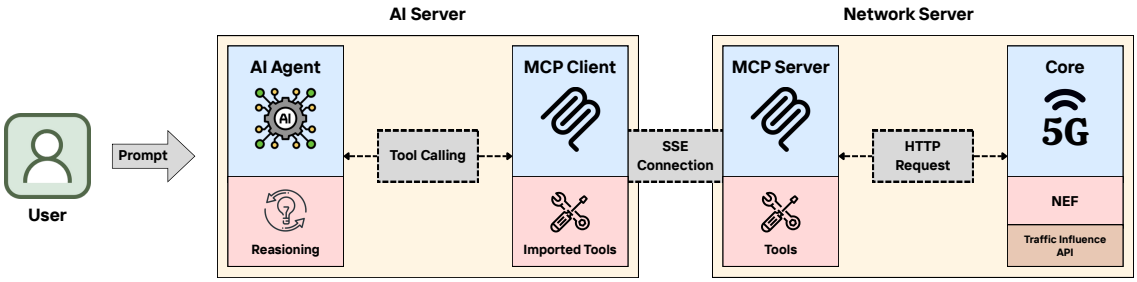

Figura 1. Fluxo de dados entre Agente, Cliente MCP, Servidor MCP e NEF.

A Figura 1 ilustra como ocorre o fluxo de dados no sistema implementado. Ao receber um *prompt*, o Agente de IA processa a solicitação e identifica quais ferramentas MCP devem ser chamadas. O Cliente MCP mantém uma lista dessas ferramentas e encaminha as requisições ao Servidor MCP que então executa as chamadas para a API *Traffic Influence*. Essa separação de responsabilidades permite que o Agente se concentre

no raciocínio de alto nível, enquanto o Cliente e o Servidor MCP cuidam dos detalhes técnicos da materialização dessa comunicação.

Em termos de implementação, o Agente foi construído utilizando o *Framework* LlamaIndex, mais especificamente como um *FunctionAgent*. Trata-se de uma arquitetura que utiliza LLMs com capacidade nativa de chamada de ferramentas para atividades que exigem raciocínio e execução de funções. O Cliente MCP foi implementado utilizando o próprio LlamaIndex, permitindo que as ferramentas MCP sejam acessadas de forma facilitada. Esse cliente atua como ponte, recebendo as chamadas do Agente e as encaminhando ao Servidor MCP. O Servidor MCP foi criado com o pacote Python FastMCP e mapeia cada *endpoint* da API *Traffic Influence* para uma ferramenta. Com isso, o Agente consegue realizar chamadas HTTP para a API de forma transparente e desacoplada.

O sistema descrito foi construído para atuar sobre o núcleo da rede 5G, utilizando o projeto de código aberto free5GC, na versão free5GC compose v4.1.0[4]. O free5GC emprega a tecnologia de contêineres para orquestrar as NFs de forma modular e isolada. No que tange ao plano de dados, o sistema integra o módulo de núcleo do Linux chamado gtp5g (v0.9.15) ao hospedeiro, permitindo que a UPF realize o encapsulamento e o roteamento de pacotes via protocolo GTP de maneira nativa.

Para viabilizar a seleção de rotas alternativas, foi adotada a configuração *Uplink Classifier* (ULCL) que instancia o núcleo da rede com duas UPFs distintas. A infraestrutura de acesso e os dispositivos de usuário são simulados através do UERANSIM, que estabelece conexões de controle com a *Access and Mobility Management Function* (AMF) e planos de dados com a UPF. A partir dessa integração, as políticas de tráfego processadas podem ser aplicadas dinamicamente. Dessa forma, as decisões do Agente refletem imediatamente no fluxo de dados entre o UE e a rede de dados externa, provendo um ambiente controlado para a validação prática da arquitetura proposta.

4.2. Experimentos

Para avaliar a capacidade do sistema em traduzir intenções de usuários em chamadas de API funcionais, foi proposto um experimento de redirecionamento de tráfego, no qual os UEs passam a utilizar uma rota alternativa (I-UPF), em vez da rota padrão (PSA-UPF). Essa configuração pode ser solicitada por aplicações externas através da API *Traffic Influence*. A Figura 2 apresenta os componentes relacionados ao re-roteamento de tráfego na arquitetura utilizada.

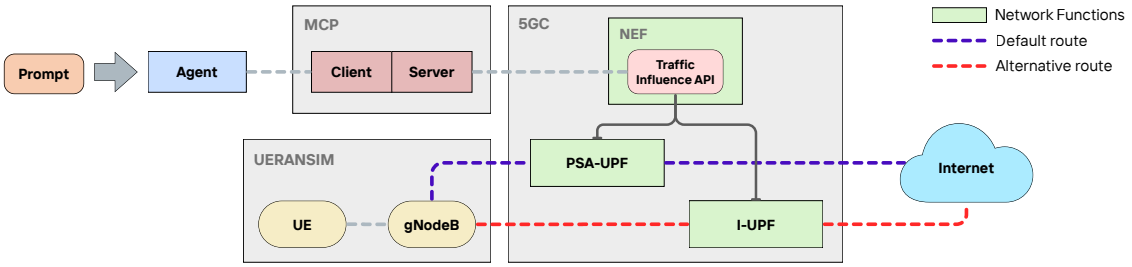

Figura 2. Arquitetura do sistema expondo rota alternativa, acionada quando uma *subscription* é registrada na NEF definindo uma regra de roteamento.

Para verificar o desempenho em múltiplas configurações, testamos o sistema variando os modelos de linguagem utilizados, o tipo de *prompt* empregado e a adição de

[4]https://github.com/free5gc/free5gc-compose.git.

informações da API ao contexto do Agente. Os modelos de linguagem avaliados variam de acordo com o número de parâmetros treinados, a disponibilidade de acesso e a arquitetura. Nas opções de código aberto, testamos o Qwen3:4B e o Qwen3:8B da Alibaba Group. Os modelos Qwen3 são reconhecidos pela capacidade de raciocínio e chamadas de ferramentas, mesmo nas versões menores, tornando-as boas escolhas para ambientes com limitação de hardware. A outra opção aberta é o GPT-OSS-20B da OpenAI, o qual utiliza a arquitetura *Mixture of Experts* (MoE).

Para as soluções proprietárias, escolhemos o Claude Haiku 4.5 e o Claude Sonnet 4.5, ambos da Anthropic. Por serem modelos de código fechado, utilizamos a API da Anthropic para acessá-los. Os modelos de código aberto foram executados localmente com o Ollama (v0.13.0). Em todos os modelos, mantivemos duas configurações padrão: o modo *thinking* ativado, no qual o modelo segue uma linha de raciocínio intermediária antes de responder, e a janela de contexto limitada a 16 mil tokens, devido a restrições de memória.

Foram avaliadas três variações de *prompts*: Informativo, Direto e Vago. Em todas, foi solicitada a realização de uma requisição para a criação de uma *subscription* no formato adequado para o 5GC. Esta variação teve como objetivo avaliar se o estilo de comunicação possui algum impacto na capacidade de interpretar a intenção do usuário. Os *prompts* foram obtidos a partir da interação com LLMs SOTA, onde foram enviadas configurações pretendidas junto a uma descrição do estilo desejado para cada uma das variações.

A Figura 3 apresenta o *prompt* informativo que contextualiza cada valor, simulando um profissional experiente que justifica cada parâmetro. Na Figura 4, está descrito o *prompt* direto que transmite os dados de forma minimalista e objetiva, sem elaborações, representando um operador técnico. Por fim, a Figura 5 mostra o *prompt* vago que descreve os parâmetros com linguagem imprecisa e genérica, omitindo terminologias de um especialista e simulando um operador técnico sem conhecimento aprofundado a respeito das APIs do 5GC.

```
Acting as the authorized application function af001, I need to register a new Traffic
Influence Subscription to steer User Plane traffic.  This request is for the
'Service1' application targeting the 'internet' Data Network Name (DNN). The network
scope should be limited to the eMBB network slice (SST 1) with the specific
differentiator '010203'.  Because this is a broad policy, set the anyUeInd attribute
to true to ensure it covers every device connected to that slice.  For traffic
steering, define a filter for the outbound traffic originating from the public DNS at
8.8.8.8 and heading to our device subnet at 10.60.0.0/16.  These flows must be
redirected to the 'mec' Data Network Access Identifier (DNAI) to ensure local
processing.
```

Figura 3. *Prompt* Informativo.

```
As application function af001, please create a traffic influence subscription for
'Service1' on the 'internet' DNN using SST 1 and SD 010203.  This policy applies to
all users on this slice.  Route the outbound traffic from the public DNS at 8.8.8.8
to the client network at 10.60.0.0/16 through the 'mec' local gateway.
```

Figura 4. *Prompt* Direto.

Também foram testadas a inserção do *schema* da API *Traffic Influence* como contexto fixo da LLM, simplificado como inserção de contexto. O objetivo foi verificar se

```
Hi, I'm setting things up for af001 regarding our 'Service1' app.  We are using the
'internet' DNN for everyone on our premium high-speed 5G connection (eMBB, 010203).
Specifically, for the outbound flow originating from the public address 8.8.8.8 and
heading to our local office devices at 10.60.0.0/16, please make sure it gets routed
directly through the 'mec' gateway so it doesn't slow down.
```

Figura 5. *Prompt* Vago.

a exposição prévia dos requisitos e da estrutura JSON da API impacta a capacidade do sistema de interpretar as solicitações. Essa variável permitiu avaliar o nível de conhecimento sobre a influência de tráfego em modelos de diferentes escalas, a eficiência do mapeamento de valores do *prompt* à estrutura do *schema*, além do impacto nas métricas de avaliação do sistema.

A avaliação do desempenho do sistema durante os experimentos foi realizada a partir de três métricas principais: acerto, tempo de execução e consumo de tokens. A métrica de acerto verifica, a cada tentativa, se o sistema interpretou corretamente a intenção descrita no *prompt* e conseguiu desviar o fluxo da PSA-UPF para a I-UPF. Essa verificação foi realizada com o aplicativo *traceroute*, a partir do dispositivo do usuário (UE).

O tempo de execução é registrado desde o instante em que a requisição escrita pelo usuário chega ao Agente até o momento em que ele sinaliza o fim da tarefa. Esse intervalo revela a velocidade com que o sistema consegue processar uma solicitação e aplicar as mudanças de rota. Valores elevados de tempo de execução podem comprometer a capacidade de resposta, pois as condições da rede podem se alterar antes que a nova configuração entre em vigor, anulando assim o efeito desejado. Além disso, longos períodos de processamento mantêm os recursos de hardware alocados por mais tempo, dificultando o compartilhamento de recursos computacionais com outras requisições que podem ser enviadas em paralelo.

O consumo de tokens é uma das métricas da indústria para precificar o uso de LLMs via API. O consumo de tokens difere do tempo de execução por ser independente do hardware, refletindo exclusivamente o modelo utilizado e a complexidade da tarefa. Essa propriedade pode ser relevante quando os modelos estão com o modo *thinking* ativado, pois cada token consumido corresponde diretamente ao raciocínio intermediário gerado para resolver o problema, tornando a métrica um indicador objetivo da dificuldade inerente à solicitação. Vale ressaltar que, para fins de simplificação, essa métrica foi contabilizada como o somatório do número de tokens de entrada e de saída.

Para a realização dos experimentos, foi utilizada uma estação de trabalho executando o sistema operacional Linux Ubuntu 24.04, composta por um processador Intel Core i9-14900 (24 núcleos físicos, 32 threads, frequência base de 3,6 GHz), uma GPU NVIDIA Quadro T1000 Mobile com 4 GB de memória VRAM DDR6 e 896 núcleos CUDA. Além disso, a estação conta com 64 GB de memória RAM DDR5, sendo dois módulos de 32 GB operando a 4800 MT/s. Para simular a separação entre o Servidor de Rede e o Servidor de IA, implantamos o MCP Server e o 5GC em uma máquina virtual Linux Ubuntu 22.04, com 2 núcleos de CPU e 4 GB de RAM.

5. Resultados e Discussão

Nesta seção, são apresentados e discutidos os resultados obtidos a partir dos experimentos de interação entre o Agente e o 5GC. A análise concentra-se no impacto das variáveis descritas na Seção 4, visando determinar a viabilidade técnica a partir das métricas obtidas.

Para verificar a consistência do sistema, cada *prompt* foi submetido três vezes para cada modelo, alternando com a inserção ou não do *schema* (contexto).

Tabela 2. Resultados dos modelos na execução de ações sem inserção do *schema* da API *Traffic Influence*.

Modelo	Informativo			Direto			Vago		
	E1	E2	E3	E1	E2	E3	E1	E2	E3
Qwen3 4B	◐	○	◐	◐	◐	◐	○	○	○
Qwen3 8B	◐	◐	◐	○	○	○	○	○	○
GPT OSS 20B	◐	◐	◐	○	○	◐	○	○	◐
Claude Haiku 4.5	◐	◐	●	◐	◐	◐	○	◐	◐
Claude Sonnet 4.5	●	●	●	●	●	●	●	◐	●

Legenda: As três colunas de cada tipo de *prompt* correspondem às execuções **E1**, **E2** e **E3**, respectivamente.

A Tabela 2 indica os acertos para cada uma das tentativas realizadas, mapeando o modelo e o tipo de *prompt* aplicado, sem a inserção de contexto. O símbolo "●" indica o êxito no re-roteamento de tráfego, enquanto que "◐" refere-se ao sucesso apenas na criação de *subscriptions*, porém, com a definição de parâmetros incorretos, mantendo o tráfego inalterado. O símbolo "○" representa a realização de requisições inválidas. A partir dos resultados apresentados, podemos verificar que nenhum modelo aberto foi capaz de cumprir a tarefa solicitada, independentemente do *prompt*. Entre os modelos proprietários, percebe-se que apenas o Claude Sonnet obteve êxito em sua quase totalidade, com uma única falha na execução do *prompt* vago. O Claude Haiku foi capaz de acertar apenas em uma das tentativas executadas com o *prompt* informativo.

Durante os experimentos, percebemos que, embora os modelos sejam capazes de entender a tarefa solicitada e realizar requisições, eles ainda têm dificuldade em determinar a estrutura exata e a nomenclatura correta dos parâmetros associados aos valores descritos nos *prompts*. Consequentemente, as solicitações foram comprometidas por falhas nos parâmetros de regra de roteamento ou pela ausência de campos obrigatórios. Além disso, na Tabela 2, é possível observar um comportamento decrescente na quantidade de tentativas em que o Agente conseguiu criar *subscriptions* em relação ao *prompt* utilizado. Indicando que, quanto mais descritiva for a entrada, maior será a chance do modelo mapear corretamente as informações para a estrutura de parâmetros necessários. Porém, essa relação depende da complexidade da estrutura e do nível de conhecimento do modelo.

As Figuras 6 e 7 apresentam, respectivamente, o tempo de execução e o uso de tokens dos experimentos descritos na Tabela 2. Em relação ao tempo de resposta, é possível notar que os modelos abertos apresentam valores mais elevados. Conjectura-se que tais resultados possuem relação com os seguintes fatores:

1. Infraestrutura de hardware limitada – os modelos abertos são executados em máquinas com baixo poder computacional, enquanto os modelos fechados utilizam recursos de *data centers* de alto desempenho;
2. Complexidade das cadeias de raciocínio – modelos menores precisam gerar longas sequências de raciocínio para entender a tarefa e mapear os parâmetros, o que eleva o consumo de tokens;
3. Instabilidade nas tentativas – esses modelos frequentemente entram em laços de tentativa e erro, às vezes ignorando ou sobrepondo instruções devido à janela de

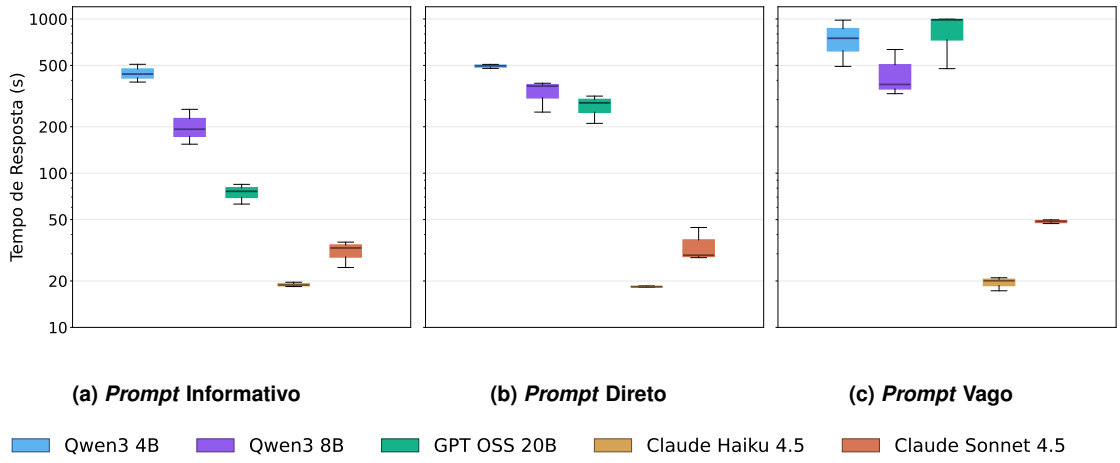

Figura 6. Comparação do tempo de resposta entre modelos com variação de tipos de *prompts* sem inserção do *schema* da API *Traffic Influence*.

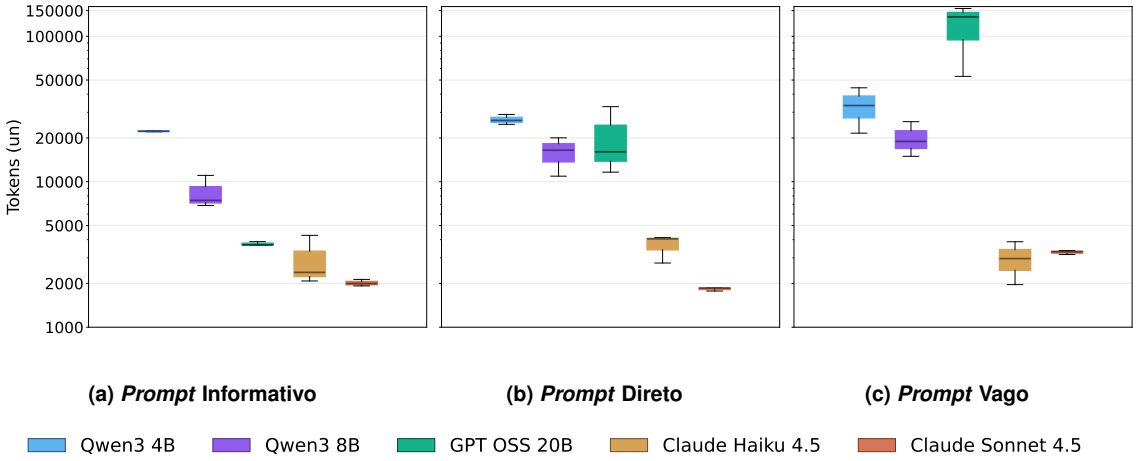

Figura 7. Comparação do uso de tokens entre modelos com variação de tipos de *prompts* sem inserção do *schema* da API *Traffic Influence*.

contexto mais curta, o que impede a execução adequada de solicitações críticas.

Entendemos que estes fatores podem tornar os modelos menores pouco adequados para tarefas que exigem confiabilidade, pois, ao falharem, não há garantia de que as instruções sejam respeitadas ou de que a execução seja interrompida. Os modelos fechados, por outro lado, apresentaram um comportamento mais controlado, executando poucas tentativas e respeitando as instruções mesmo quando erram.

A Tabela 3 apresenta os resultados quando o Agente é executado com a inserção de contexto ativada. Nesse cenário, observa-se que todos os modelos, inclusive os menores, conseguiram executar a tarefa solicitada de maneira completa, com exceção do Qwen3 8B, que falhou em um dos testes com o *prompt* vago. Os resultados obtidos com a inserção de contexto reforçam a tese de que executar o modelo apenas com o conhecimento adquirido durante a fase de treinamento pode não ser suficiente para obter os resultados adequados. Além disso, mesmo com esse auxílio, alguns modelos permanecem suscetíveis a falhas.

Quanto às métricas de desempenho apresentadas nas Figuras 8 e 9, mantém-se o

Tabela 3. Análise dos modelos na execução de ações com inserção do *schema* da API Traffic Influence.

Modelo	Informativo			Direto			Vago		
	E1	E2	E3	E1	E2	E3	E1	E2	E3
Qwen3 4B	●	●	●	●	●	●	●	●	●
Qwen3 8B	●	●	●	●	●	●	●	●	○
GPT OSS 20B	●	●	●	●	●	●	●	●	●
Claude Haiku 4.5	●	●	●	●	●	●	●	●	●
Claude Sonnet 4.5	●	●	●	●	●	●	●	●	●

Legenda: As três colunas de cada tipo de *prompt* correspondem às execuções **E1**, **E2** e **E3**, respectivamente.

comportamento observado nos experimentos sem inserção de contexto, desconsiderando a execução em que o Qwen3 8B falhou. Nota-se nos modelos de código aberto uma correlação inversa interessante: modelos maiores demandaram menos tempo de execução e consumo de tokens. Isso se justifica devido a eficiência de raciocínio. Embora modelos menores possuam uma taxa de geração de tokens superior, modelos maiores convergem para a resposta correta de forma mais direta, evitando a verbosidade e reduzindo tanto o tempo quanto o uso de recursos computacionais. Nos modelos proprietários, o Claude Haiku combinou uma baixa latência de resposta com uma eficiência de raciocínio tão elevada quanto a do Claude Sonnet, resultando em desempenho significativamente superior.

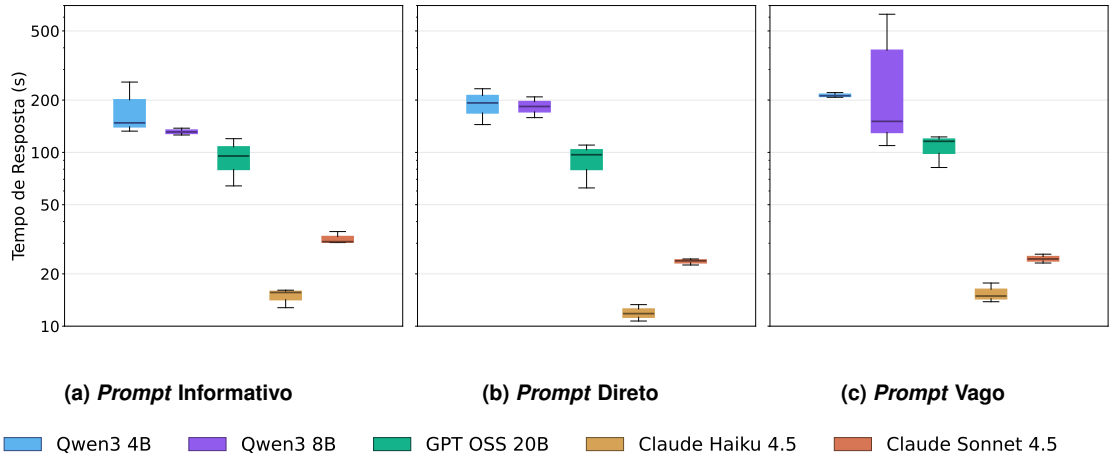

Figura 8. Comparação do tempo de resposta entre modelos com variação de tipos de *prompts* com inserção do *schema* da API *Traffic Influence*.

Uma análise comparativa mais detalhada foi realizada com o modelo Claude Sonnet 4.5, único a obter sucesso em ambos os cenários experimentais. Do ponto de vista do tempo de execução, observou-se que para os *prompts* direto e vago, a inserção de contexto é benéfica, enquanto que para os testes com *prompt* informativo não há ganho significativo. Com relação ao uso de tokens, a adição de contexto resulta em um aumento expressivo para os testes com *prompts* informativo e direto, ao passo que para aqueles que utilizam o *prompt* vago, o consumo é muito similar entre os dois cenários. Nesse sentido, pode-se afirmar que a inserção de contexto não traz benefícios para o tempo de resposta quando se utilizam instruções mais explicativas, nas quais o raciocínio é mais fácil. Por outro

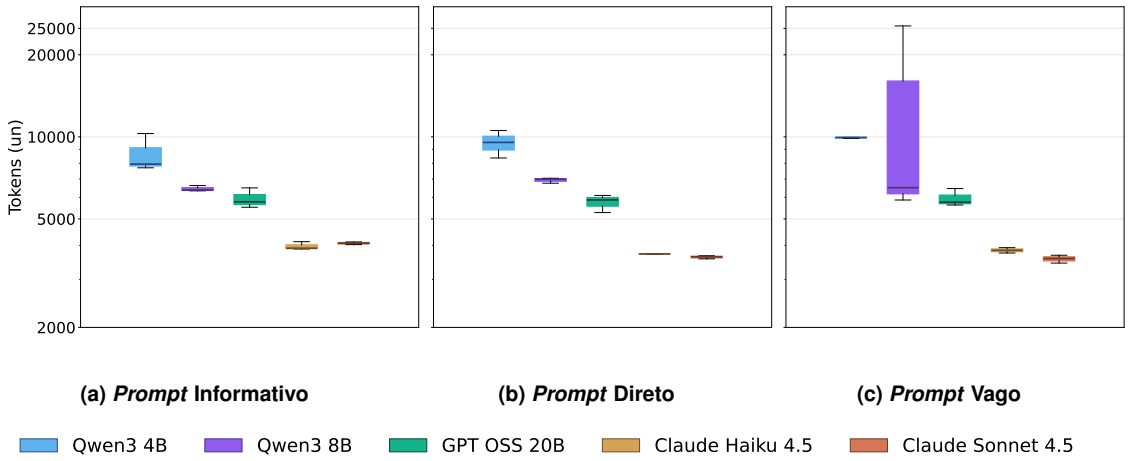

Figura 9. Comparação do uso de tokens entre modelos com variação de tipos de *prompts* com inserção do *schema* da API *Traffic Influence*.

lado, nos demais casos, essa estratégia funciona como um acelerador. Quanto ao consumo de tokens, sua adição sempre implica um gasto adicional. Contudo, para cenários com informações imprecisas, nos quais longas cadeias de raciocínio precisam ser geradas, tal inserção não resulta em aumento expressivo dessa métrica.

6. Conclusões

Este artigo avaliou o uso de Agentes de IA baseados em LLMs para traduzir intenções em linguagem natural em chamadas à API *Traffic Influence* da NEF no núcleo 5G. Os resultados mostram que, apesar de os modelos compreenderem a tarefa em nível conceitual, a complexidade estrutural e semântica da API limita fortemente a geração consistente de requisições corretas. Sem a inserção explícita do *schema*, apenas modelos proprietários de grande porte apresentaram taxas de sucesso relevantes, enquanto modelos de código aberto falharam de forma recorrente.

A inserção do *schema* mostrou-se determinante para aumentar a taxa de acerto entre os diferentes modelos, ainda que à custa de maior consumo de tokens e tempo de resposta. Observou-se que modelos maiores tendem a convergir mais rápido para soluções válidas, reduzindo tentativas redundantes e apresentando maior eficiência de raciocínio, o que evidencia um compromisso entre custo computacional, desempenho e confiabilidade.

Como contribuição, este trabalho estabelece uma linha de base prática para o uso de agentes baseados em LLMs na interação direta com APIs do 5GC, evidenciando limitações, custos e dependências de contexto ainda pouco discutidos na literatura. A arquitetura proposta e os artefatos disponíveis em código aberto fornecem subsídios para pesquisas futuras voltadas à abstração, validação e automação robusta de APIs da NEF.

7. Agradecimentos

Este trabalho foi apoiado pela Ericsson Telecomunicações Ltda., pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), por meio do projeto 2021/00199-8, CPE SMARTNESS e pelo Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB).

Referências

3GPP (2023). Network Exposure Function (NEF) Services API. Technical Report TS 29.522, 3GPP.

3GPP (2024). Management and orchestration; Intent driven management services for mobile networks. Technical Report TS 28.312, 3GPP.

Bimo, F. A., Galdon, M. A. C., Lai, C.-K., Cheng, R.-G., and Chong, E. K. (2025). Intent-Based Network for RAN Management with Large Language Models. *arXiv preprint arXiv:2507.14230*.

Brodimas, D., Birbas, A., Kapolos, D., and Denazis, S. (2025). Intent-Based Infrastructure and Service Orchestration Using Agentic-AI. *IEEE Open Journal of the Communications Society*, 6:7150–7168.

Brodimas, D., Trantzas, K., Agko, B., Tziavas, G. C., Tranoris, C., Denazis, S., and Birbas, A. (2024). Towards Intent-based Network Management for the 6G System adopting Multimodal Generative AI. In *2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 848–853. IEEE.

GSMA (2023). CAMARA Project: Network APIs for Developers. https://www.gsma.com/solutions-and-impact/technologies/networks/camara/.

Habib, M. A., Iturria-Rivera, P. E., Ozcan, Y., Elsayed, M., Bavand, M., Gaigalas, R., and Erol-Kantarci, M. (2025). Harnessing the Power of LLMs, Informers and Decision Transformers for Intent-driven RAN Management in 6G. *IEEE Transactions on Network Science and Engineering*.

Hou, X., Zhao, Y., Wang, S., and Wang, H. (2025). Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. *arXiv preprint arXiv:2503.23278*.

Khan, Z., Hussain, A., Thakur, M., Hellas, A., and Papadimitratos, P. (2025). NEFMind: Parameter-Efficient Fine-Tuning of Open-Source LLMs for Telecom APIs Automation.

Ksentini, A. and Frangoudis, P. A. (2020). Toward slicing-enabled multi-access edge computing in 5g. *IEEE Network*, 34(2):99–105.

Manias, D. M., Chouman, A., and Shami, A. (2024). Towards Intent-Based Network Management: Large Language Models for Intent Extraction in 5G Core Networks. In *2024 20th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 1–6. IEEE.

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

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.

Zhou, H., Hu, C., Yuan, Y., Cui, Y., Jin, Y., Chen, C., Wu, H., Yuan, D., Jiang, L., Wu, D., Liu, X., Zhang, J., Wang, X., and Liu, J. (2025). Large Language Model (LLM) for Telecommunications: A Comprehensive Survey on Principles, Key Techniques, and Opportunities. *IEEE Communications Surveys & Tutorials*, 27(3):1955–2005.