Análise do uso de Modelos de Linguagem de Grande Escala na Geração de Códigos para Simulação de Redes utilizando o Mininet-WiFi

Antonio de Sousa Cruz Neto[1], **Vinícius A. R. Almeida**[2], **Paulo H. L. Rettore**[2,3], **Bruno Santos**[1]

[1]Instituto de Computação
Universidade Federal da Bahia (UFBA), Salvador - BA – Brasil

[2]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG), Belo Horizonte - MG – Brasil

[3]Departamento de Computação Aplicada
Universidade Federal de Lavras (UFLA), Lavras - MG – Brasil.

{antonioscn, bruno.ps}@ufba.br
{vinicius.almeida, rettore}@dcc.ufmg.br

Abstract. *This article investigates the ability of large-scale language models (LLMs) to generate Python scripts for a wired and wireless network emulator, Mininet-WiFi. The study analyzes the factors that influence the effectiveness of these models in producing functional code at three levels of complexity. For each level, simple and detailed prompts were applied in order to examine how the instructions affect the performance of the generated code. The study also compares the best online model and a smaller open-weight local model in reproducing an official Mininet-WiFi example, evaluating their ability to generate similar functional scripts. These results aim to reinforce the potential of LLMs to represent network structures and support the automation of simulations.*

Resumo. *O presente artigo investiga a capacidade de modelos de linguagem de grande escala (LGEs) em gerar scripts em Python para um emulador de redes cabeadas e sem fio, o Mininet-Wifi. O estudo realiza uma análise dos fatores que influenciam a eficácia desses modelos na produção de códigos funcionais em três níveis de complexidade. Para cada nível, foram aplicados prompts simples e detalhados, a fim de analisar como as instruções influenciam no desempenho dos códigos gerados. O estudo ainda compara o melhor modelo online e um modelo local open-weight na reprodução de um exemplo oficial do Mininet-WiFi, avaliando sua capacidade de gerar scripts funcionais semelhantes. Esses resultados buscam reforçar o potencial dos LGEs para representar estruturas de rede e apoiar a automação de simulações.*

1. Introdução

A emulação e simulação de redes de computadores são etapas cruciais para o desenvolvimento e a validação de novas soluções em rede, oferecendo uma alternativa escalável e de baixo custo em comparação aos testes físicos [Burfeid et al. 2025]. Nesse contexto de Redes Definidas por Software (do Inglês *Software Defined Network (SDN)*), o Mininet

e Mininet-WiFi [Fontes and Rothenberg 2019] surgem como plataformas de emulação de referência, permitindo a prototipagem rápida de cenários que integram mobilidade e modelos de propagação realistas.

Apesar dessas vantagens, a geração automática de scripts para emuladores de redes sem fio representa um desafio técnico-científico ainda pouco explorado. Diferentemente de tarefas genéricas de geração de código, a construção de cenários no Mininet-WiFi exige que o modelo produza scripts com representação correta de parâmetros físicos de propagação, dinâmicas de mobilidade e aderência estrita à API do emulador, elementos cuja especificidade técnica vai além da sintaxe e demanda conhecimento aprofundado do domínio [Tiamiyu et al. 2025]. Pesquisadores frequentemente precisam traduzir intenções de rede em scripts Python funcionais e semanticamente coerentes com o ambiente de emulação, e qualquer imprecisão nesses elementos pode comprometer a validade dos experimentos sem emitir erros explícitos [Ifland et al. 2025]. Além disso, a ausência de benchmarks sistemáticos para Linguagem de Grande Escalas (LGEs) aplicados especificamente a emuladores wireless, em contraste com trabalhos voltados ao ns-3 e ao Mininet cabeado, configura uma lacuna metodológica relevante na literatura de automação de redes.

Para mitigar esse problema, diferentes trabalhos têm proposto metodologias baseadas em Inteligência Artificial (IA), que ajudam os usuários em tarefas complexas e em múltiplos domínios vêm ganhando destaque [Raiaan et al. 2024]. Os Modelos de LGE têm demonstrado capacidade de atuar como ”copilotos”, gerando códigos em diversas linguagens como Python e C++ a partir de descrições em linguagem natural [Soares et al. 2025]. Estudos indicam que ferramentas como o ChatGPT e o DeepSeek podem auxiliar na configuração de dispositivos e na lógica de controladores, sugerindo um caminho promissor para automatizar tarefas de engenharia de redes [Ifland et al. 2025, Cruz et al. 2025].

Contudo, a aplicação efetiva de LGEs na geração de *scripts* para simulações de redes, sobretudo as sem fio, ainda apresenta grandes desafios e é pouco explorada. Além disso, a confiabilidade desses modelos ainda permanece incerta, o que exige uma validação rigorosa para evitar pequenos erros ou alucinações de código [Soares et al. 2025]. Grande parte dos estudos realizados foca em configurações estáticas ou em controladores isolados, deixando uma lacuna na avaliação desses modelos para a geração de ambientes de simulação mais completos.

Diante disso, este trabalho propõe uma avaliação experimental da capacidade e da robustez de modelos de LGEs na geração de código Python para simulação e emulação de redes no Mininet-WiFi [Fontes 2025], considerando variações nos tipos de *prompts* fornecidos pelos usuários. O estudo contempla a análise dos modelos amplamente conhecidos das famílias GPT [OpenAI 2025], Gemini [Google 2025], Claude [Anthropic 2025], e DeepSeek [DeepSeek 2025].A abordagem metodológica foca na validação prática dos *scripts* gerados, verificando não apenas a sintaxe, mas a coerência da topologia, a execução e a necessidade de intervenção humana para corrigir alguma parte do script gerado.

As principais contribuições deste trabalho são:

- Análise de desempenho de modelos da familia GPT, Gemini, DeepSeek e Claude

na geração de código para Mininet-WiFi;
- Avaliação sistemática dos códigos gerados, considerando correção sintática, semântica e funcionalidade em ambiente emulado;
- Investigação de replicação de exemplos oficiais e uso de modelo local open-weight

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve a metodologia. A Seção 4 discute os resultados obtidos. A Seção 5 apresenta a comparação com exemplos oficiais e os testes com modelo local. Por fim, a Seção 6 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

No contexto da geração automática de código, modelos de LGE vêm sendo explorados em diferentes áreas de desenvolvimento de *software*, com foco tanto na redução do esforço de programação quanto na acessibilidade para usuários sem conhecimento [Brown et al. 2020]. Estudos recentes discutem a capacidade desses modelos de produzir trechos de código funcionais a partir de descrições mais simples em linguagem natural, destacando os ganhos de produtividade, mas também apontando desafios relacionados à correção sintática, à validação semântica e à qualidade em relação aos tipos de *prompts* de entrada [Wang and Zhu 2024]. Nesse contexto, diferentes abordagens têm sido propostas para avaliar o comportamento dos modelos em tarefas de programação, incluindo estudos de engenharia de *prompts*, análise de desempenho e mecanismos de validação de código, o que reforça a necessidade de investigações específicas em contextos de aplicação distintos, como os de automação residencial e simulação de redes de computadores.

A Tabela 1 apresenta, de forma resumida, uma perspectiva de trabalhos relacionados, destacando o domínio e as ferramentas usadas, a linguagem dos códigos gerados e os modelos utilizados e avaliados em cada proposta.

Tabela 1. Comparação de Trabalhos Relacionados

Referência	Domínio / Ferramenta	Linguagem	Modelos Avaliados
Soares et al. (2025)	SDN / POX Controller	Python	ChatGPT, Copilot, DeepSeek, BlackBox
Ahmed et al. (2025)	Simulação de Rede / ns-3	C++	GPT-4.1, Gemini 2.0, Qwen-3
Burfeid et al. (2025)	Emulação / Mininet	Python	Qwen 2.5 Coder (Fine-tuned)
Cruz et al. (2025)	Automação Residencial / Home Assistant	YAML	GPT-4o, GPT-4.5, Gemini 2.5 Flash, Gemini 2.5 Pro
Proposta Atual	Emulação Wireless / Mininet-WiFi	Python	Gemini 3, Gemini 3 Pro, Claude 4.5 Sonnet, DeepSeek-V3, GPT-5.1 thinking, GPT-5.2 thinking

Trabalhos recentes apresentam soluções baseadas em LGEs para a automação e geração de código em ambientes de redes de computadores. Entretanto, muitas delas apresentam algumas limitações, focando mais em redes cabeadas ou em simuladores de eventos, o que não reflete a dinâmica de emulação em tempo real. O framework GeNetE [Burfeid et al. 2025], por exemplo, propõe a construção de um *Specialized Small Language Model (SSLM)*, ajustando um modelo Qwen por *fine-tuning*, para gerar *scripts* de emulação de topologias específicas para o emulador Mininet padrão a partir de descrições em linguagem natural. Contudo, sua especialização está focada na API/sintaxe desse emulador cabeado. Assim, a adaptação para o Mininet-WiFi demandaria um novo tipo de revalidação de tarefas envolvendo Access Points (APs), estações e mobilidade, além de um provável reajuste do modelo para os parâmetros de redes sem fio.

O SIMCODE [Ahmed et al. 2025] propõe um *benchmark* padronizado para avaliar a capacidade de grandes modelos, como o GPT-4 e o Gemini, de gerar códigos em C++ para a simulação de redes sem fio a partir de descrições em linguagem natural. A

abordagem foca no simulador ns-3 e utiliza uma vasta coleção de 400 tarefas distribuídas em níveis introdutório, intermediário e avançado e, para cada tarefa, disponibiliza um *prompt*, uma solução de referência verificada e os casos de teste. A proposta tem a vantagem de estabelecer um padrão rigoroso de avaliação; no entanto, ainda foca em um tipo de simulação diferente, não é baseada em Python nem especificamente para Mininet-WiFi, além de revelar que modelos sofrem com alucinações em APIs de domínio específico, sem um contexto bem descrito.

O trabalho "Trust, But Verify" [Soares et al. 2025] realiza uma avaliação empírica de códigos gerados por diferentes ferramentas e modelos como o ChatGPT, o Copilot, o DeepSeek e o BlackBox.ai, para controladores SDN (POX). Os autores partem do diagnóstico de que a confiabilidade do código produzido por IA ainda é incerta e, portanto, o desenvolvedor não deve "confiar cegamente" na saída; assim, definem três tarefas de rede com complexidade crescente, submetendo o código gerado a análises de funcionalidade, correção e ajustes manuais. O estudo emprega estratégias de *prompting zero-shot* e *few-shot* para cada tarefa e valida os controladores em topologias emuladas no Mininet que variam de configurações simples a mais complexas, permitindo observar como mudanças no contexto do *prompt* e no cenário de teste impactam a robustez do código.

Em nosso trabalho anterior [Cruz et al. 2025], adotamos uma abordagem de avaliação de modelos generalistas, concentrando-nos na geração de arquivos de configuração YAML para automação residencial no *Home Assistant*. O trabalho apresenta uma análise fatorial relacionada ao impacto da complexidade do cenário e da especificidade dos *prompts* na qualidade do código gerado, oferecendo um referencial para estruturar avaliações comparativas de LGEs e para compreender como fatores de entrada afetam a qualidade das saídas. Apesar dessas contribuições, especialmente no uso de técnicas estatísticas para a caracterização de erros e a estratificação de níveis de dificuldade, o estudo foca na linguagem declarativa (YAML) para a área de automação residencial, não contemplando os desafios próprios de *scripts* em Python para instanciar topologias dinâmicas, configurar parâmetros de rede sem fio e modelar a mobilidade no Mininet-WiFi.

Dessa forma, o presente trabalho visa preencher uma lacuna ao propor uma avaliação experimental estruturada da capacidade de modelos de grande escala na geração de *scripts* completos em Python para o Mininet-WiFi, considerando não apenas a correção funcional e executabilidade dos cenários gerados, mas também a necessidade de ajustes, a complexidade e a manutenibilidade do código produzido, de modo a apoiar uma adoção mais segura e eficaz desses modelos na automação da construção de cenários de emulação de redes sem fio.

3. Metodologia

A metodologia proposta neste trabalho consiste em uma avaliação organizada em três fases principais, com o objetivo de comparar a capacidade e a robustez de diferentes modelos de LGE na geração automática de *scripts* em Python para a criação de cenários de emulação de redes no Mininet-WiFi, por meio da interpretação de *prompts* em linguagem natural.

A **Fase 1 (Instrução/Geração)** corresponde à etapa inicial de produção de dados. Nessa fase, os modelos avaliados foram preparados por meio de um *prompt* geral, que os contextualizou como especialistas em simulação/emulação de redes com Mininet-WiFi,

e então submetidos a uma série de *prompts* estruturados em três níveis de complexidade distintos. Como saída, cada LGE gerou *scripts* Python, compondo um banco de dados de códigos por modelo (Código BD), que serve de base para as etapas seguintes.

Na **Fase 2 (Validação)**, foi realizada a verificação funcional e operacional de cada *script* gerado. Cada código foi executado no ambiente Mininet-WiFi para avaliar sua execução, consistência com o cenário solicitado e a capacidade de suportar testes mínimos de rede, como conectividade e medições. Os *scripts* que apresentaram falhas por erros de importação, inconsistências de topologia, parâmetros inválidos, falhas de execução ou ausência de componentes essenciais, foram catalogados em um banco de dados de erros (Erros BD), juntamente com a caracterização do tipo de falha e o nível de complexidade associado. Por outro lado, os *scripts* executados com sucesso, ou que exigiram apenas ajustes mínimos para atingir comportamento correto, foram registrados como evidência de desempenho de determinado modelo. Essa fase é essencial para quantificar o desempenho comparativo entre modelos e analisar como a complexidade do cenário e a especificidade do *prompt* impactam a robustez do código gerado.

Já a **Fase 3 (Generalização/Reprodução)** foca em verificar se o melhor modelo, selecionado com base dos resultados das fases anteriores, mantém desempenho quando exposto a cenários reais e amplamente utilizados pela comunidade. Para isso, utilizaram-se os exemplos oficiais do Mininet-WiFi disponibilizados no repositório do projeto, que representam padrões de uso reais e validados por humanos. Alguns exemplos foram convertidos em um *prompt* descritivo, replicando o comportamento e os requisitos do *script* original. Em seguida, o *prompt* foi enviado ao modelo para que reproduzisse o *script* correspondente, e os códigos gerados foram submetidos ao mesmo processo de validação da Fase 2, permitindo mensurar a capacidade de generalização e a fidelidade aos padrões oficiais do Mininet-WiFi. Assim, essa fase complementa a avaliação baseada em cenários autorais ao verificar se o modelo consegue reproduzir implementações de referência, reduzindo o risco de que o bom desempenho esteja restrito a *prompts* "treinados".

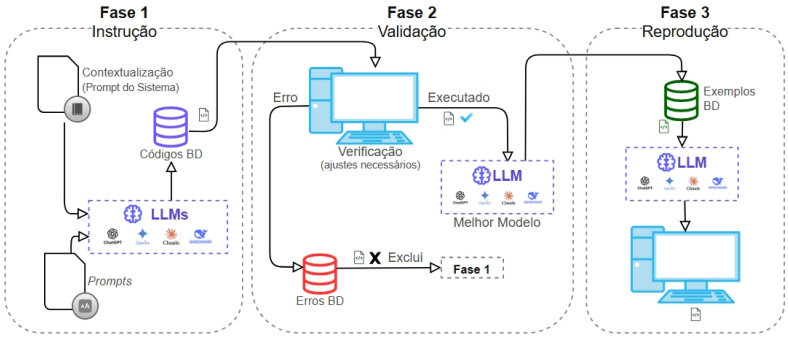

Figura 1. Fluxograma metodológico

3.1. Seleção dos Modelos

Para este estudo, foram selecionados seis modelos de linguagem de grande escala amplamente conhecidos e de diferentes empresas, sendo: **Gemini 3**, **Gemini 3 Pro**, **Claude Sonnet 4.5**, **DeepSeek-V3**, **GPT 5.1 Thinking** e **GPT 5.2 Thinking**. Os modelos foram acessados por meio de suas interfaces oficiais e executados com as mesmas instruções gerais para a geração de código em Python para o Mininet-WiFi.

3.2. Contextualização dos Modelos

Visando assegurar a consistência metodológica e orientar os modelos ao comportamento esperado, foi aplicada uma instrução de sistema padronizada, o *prompt* de contextualização, antes do envio dos *prompts* específicos para gerar os códigos. Essa contextualização delimita o domínio de atuação, induzindo os modelos a assumirem o papel de um especialista em redes de computadores e emulação/simulação com Mininet-WiFi, sendo ela:

Contextualização

"Você é um engenheiro de redes e especialista em simulações utilizando o Mininet-WiFi, uma ferramenta capaz de emular redes cabeadas e sem fio em um único ambiente computacional. Seu papel é gerar códigos completos em Python para a criação de topologias de rede com diferentes níveis de complexidade, utilizando os seguintes componentes: estações (addStation), pontos de acesso (addAccessPoint), switches (addSwitch), controladores SDN (addController), e os métodos essenciais para configuração e execução da rede, como configureWifiNodes(), mobility(), pingAll(), iperf(), net.start() e net.stop(). Todos os códigos devem ser completos e funcionais, contendo as importações necessárias, as configurações de topologia, os parâmetros de rede e as etapas de inicialização, teste e finalização. Nenhum trecho deve ser omitido nem indicado como "..." ou "código adicional". Os scripts devem representar com fidelidade o cenário solicitado, ser consistentes com a sintaxe do Mininet-WiFi, e adotar nomenclatura padronizada (por exemplo: sta1, sta2, ap1, s1, c0). Inclua comentários curtos e explicativos para cada bloco funcional, e garanta que o código esteja pronto para execução direta no terminal, sem necessidade de ajustes manuais. Sempre utilize o controlador SDN como elemento ativo da topologia e assegure que a rede possa ser iniciada e encerrada corretamente. Com base nisso, aguarde as próximas instruções para que você avalie cuidadosamente o cenário descrito e gere sempre o script completo dentro de um único bloco de código Python, pronto para execução, seguindo fielmente as especificações fornecidas."

Figura 2. Prompt de contextualização utilizado

3.3. Prompts utilizados

Com todos os modelos contextualizados, iniciamos o envio dos *prompts* de solicitação, que foram cuidadosamente estruturados, organizados em uma escala de complexidade composta por três níveis: básico, intermediário e avançado. Cada nível foi subdividido em duas variações complementares: uma versão simples e uma versão específica.

O *prompt* simples visa manter uma descrição mais generalista do cenário, sem um detalhamento de entidades e parâmetros, permitindo uma maior liberdade interpretativa do modelo e sua capacidade de definir os componentes essenciais do Mininet-WiFi. Em contrapartida, o *prompt* específico apresenta um maior detalhamento, descrevendo não apenas o objetivo do cenário, mas também fornecendo informações como quantidade e tipo de nós, nomes das entidades, parâmetros relevantes e condições de mobilidade e procedimentos mínimos de teste e validação. Ao todo, definimos seis *prompts* distintos, aplicados a cada modelo avaliado, sendo eles:

3.3.1. Cenário Básico

Simples: *Crie um script em Python para o Mininet-WiFi que simule uma rede com duas estações sem fio, um ponto de acesso e um controlador. As estações devem estar conectadas ao ponto de acesso via Wi-Fi e conseguir se comunicar entre si.*

Específico: *Gere um código completo em Python para o Mininet-WiFi que construa uma topologia simples composta por duas estações sem fio (sta1 e sta2), um ponto de acesso (ap1) e um controlador SDN padrão (c0). O ponto de acesso deve operar no canal 1, com alcance de 30 metros, e ambas as estações devem estar conectadas a ele via Wi-Fi. Inclua todas as importações necessárias, bem como a criação da rede, a configuração dos nós (addStation, addAccessPoint, addController), a função net.configureWifiNodes() para habilitar os parâmetros sem fio, e a ligação entre os elementos.*

3.3.2. Cenário Intermediário

Simples: *Crie um script em Python para o Mininet-WiFi que simule uma rede com dois pontos de acesso Wi-Fi, quatro estações sem fio e um controlador. Cada ponto de acesso deve ter duas estações conectadas, e ambos devem estar interligados por um switch central.*

Específico: *Gere um código completo em Python para o Mininet-WiFi que simule uma rede composta por dois pontos de acesso Wi-Fi (ap1 e ap2), quatro estações sem fio (sta1 a sta4), um switch central (s1) e um controlador SDN (c0). Cada ponto de acesso deve estar conectado ao switch central, e cada um deve atender a duas estações, de modo que ap1 sirva às estações sta1 e sta2 e ap2 sirva às estações sta3 e sta4. Configure o canal dos APs para operar de forma não interferente (por exemplo, canal 1 e canal 6) e atribua posições distintas às estações para simular distribuição espacial. Inclua as importações, a configuração dos nós (addStation, addAccessPoint, addSwitch, addController), a chamada net.configureWifiNodes() para aplicar as configurações sem fio, e a interligação entre todos os componentes.*

3.3.3. Cenário Avançado

Simples: *Crie um script em Python para o Mininet-WiFi que simule uma rede híbrida com dois pontos de acesso Wi-Fi, seis estações sem fio e um controlador. As estações devem estar distribuídas entre os dois pontos de acesso, com possibilidade de mobilidade entre eles. A rede deve permitir comunicação entre todas as estações, mesmo quando se movem entre áreas de cobertura diferentes.*

Específico: *Gere um código completo em Python para o Mininet-WiFi que implemente uma topologia híbrida composta por dois pontos de acesso Wi-Fi (ap1 e ap2), seis estações sem fio (sta1 a sta6) e um controlador SDN remoto (c0). Cada ponto de acesso deve estar conectado a um switch intermediário (s1), que integra toda a rede. As estações devem ser inicialmente associadas de forma equilibrada entre os dois APs e configuradas com endereços IP automáticos. A rede deve suportar mobilidade, permitindo que as estações se movam entre as áreas de cobertura dos APs, utilizando o recurso net.mobility() para definir posições iniciais e finais com tempos distintos de início e parada. Configure os APs em canais diferentes (por exemplo, 1 e 11), defina alcance de 40 metros, e habilite net.configureWifiNodes() para aplicar os parâmetros físicos.*

4. Avaliação

As respostas geradas por cada modelo em cada cenário e tipo de *prompt* foram avaliadas por meio de um processo de validação funcional e operacional dos *scripts* em Python. Os testes foram realizados em uma instância do Mininet-WiFi 2.6[1]. Cada código gerado foi iniciado e testado de forma padronizada, permitindo verificar sua execução, a presença dos componentes da topologia e realizar verificações mínimas de rede, como testes de conectividade e de tráfego. Em paralelo, foi realizada uma análise qualitativa da estrutura das respostas e dos códigos, considerando critérios de completude, legibilidade, aderência à API do Mininet-WiFi, complexidade e o nível de intervenção ou ajustes necessários

[1]Disponível em: `https://github.com/intrig-unicamp/mininet-wifi`

para correção. Por fim, foram aplicadas análises estatísticas para quantificar e comparar o desempenho dos modelos, considerando as taxas de sucesso e as variações de resultado por cenário e por nível de complexidade definido na metodologia.

4.1. Análise Funcional

A primeira etapa da avaliação consistiu na execução direta de cada código Python gerado, observando se o comportamento resultante estava de acordo com cenários de rede solicitados. Esse procedimento prático não se restringiu à verificação da correção sintática dos *scripts*, mas também permitiu uma análise quantitativa de sua eficácia operacional, isto é, se a topologia era criada corretamente, se os nós conectavam-se como esperado e se os testes de conectividade eram concluídos com sucesso. Para tornar essa avaliação objetiva, foram definidos dois critérios principais, ambos mensuráveis e adequados a uma análise sistemática dos resultados:

A **Correção Funcional**, foi o primeiro critério definido de forma binária (Sim/Não) com o intuito de indicar se o código produzido executava corretamente já na primeira tentativa no ambiente do Mininet-WiFi. Um resultado "Sim" indica que o *script* foi capaz de inicializar a emulação, instanciar os elementos de rede, como estações, pontos de acesso, links, controladores, e concluir os testes previstos sem erros, caracterizando um código pronto para uso, sem necessidade de ajustes iniciais.

A **Necessidade de Ajuste** foi o segundo critério, também binário (Sim/Não), e teve como foco identificar a necessidade de algum tipo de alteração manual no código. Considerou-se 'ajuste mínimo' qualquer modificação feita em uma única linha ou parâmetro, como a correção do nome de uma classe ou de um valor de atributo, que não alterava a lógica geral do script. Casos que exigiram o reenvio de prompts ao modelo ou a modificação maiores foram classificados como falha. A detecção dos tipos de erro foi realizada pela execução direta do script no terminal do Mininet-WiFi: erros de sintaxe foram identificados por exceções de parse ou importação, erros de lógica, por comportamento divergente da topologia em relação ao *prompt*, e alucinações, pela presença de métodos ou parâmetros inexistentes na documentação da ferramenta.

4.2. Análise Fatorial

Com o objetivo de analisar, de forma estruturada, como os critérios avaliados variam sob diferentes condições experimentais, adotou-se um planejamento fatorial do tipo 2^k. Essa abordagem clássica permite investigar, de maneira sistemática, tanto os efeitos principais quanto as interações entre múltiplos fatores, a partir de um conjunto reduzido e controlado de experimentos. Especificamente, foi empregado um planejamento fatorial 2^3, no qual três fatores de interesse foram definidos e avaliados em dois níveis cada, possibilitando uma caracterização abrangente do espaço experimental considerado.

Desse modo, foram analisados três aspectos: a Complexidade do Cenário de Rede (intermediária ou avançada), o Tipo de *prompt* (simples ou específico) e a Necessidade de um Ajuste Simples no código após a primeira execução (com ajuste ou sem ajuste). A análise foi conduzida separadamente para cada um dos modelos avaliados, considerando os fatores e seus níveis para capturar sua influência sobre o sucesso dos *scripts* Python gerados e executados no Mininet-WiFi.

Tabela 2. Resultados por cenário e tipo de *prompt*.

Cenário	*Prompt*	Gemini 3		Gemini 3 Pro		GPT 5.1 Thinking		GPT 5.2 Thinking		DeepSeek-V3		Claude Sonnet 4.5	
		Func.	Ajuste	Func.	Ajuste	Func.	Ajuste	Func.	Ajuste	Func.	Ajuste	Func.	Ajuste
Básico	Simples	✓	✓	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗
Básico	Específico	✓	✓	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗
Intermediário	Simples	✗	✓	✓	✗	✗	✓	✓	✓	✗	✓	✓	✗
Intermediário	Específico	✗	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✗
Avançado	Simples	✗	✓	✓	✗	✗	✓	✓	✓	✗	✓	✗	✓
Avançado	Específico	✓	✓	✓	✗	✓	✓	✓	✗	✗	✓	✗	✓

Símbolos ✓= sim, ✗= não, indicam se cada modelo gerou código funcional e se funcionou após ajuste.

4.2.1. Principais Efeitos

A Tabela 2 sintetiza o desempenho dos modelos nos diferentes cenários avaliados. Os resultados indicam que o nível de complexidade do cenário é o fator que exerce maior impacto sobre o sucesso do código gerado, apresentando uma correlação negativa com o funcionamento correto das soluções produzidas. À medida que a complexidade do cenário aumenta, observa-se uma maior incidência de erros e de inadequações em relação aos requisitos especificados. Em contrapartida, nos cenários de nível básico, tecnicamente mais simples, todos os modelos alcançaram sucesso, o que sugere que, frente a demandas menos desafiadoras, tendem a apresentar comportamentos semelhantes.

À medida que a complexidade aumenta para cenários intermediários, as primeiras diferenças de desempenho começam a aparecer. Em cenários de maior complexidade, essas diferenças tornam-se ainda mais claras. No Gemini 3 Pro e no GPT 5.2 Thinking, todos os códigos produzidos funcionaram (100% de sucesso) sem a necessidade de qualquer ajuste, demonstrando maior consistência e confiabilidade em topologias e lógicas de rede mais exigentes, enquanto o Gemini 3 e o GPT 5.1 Thinking obtiveram um desempenho intermediário, com cerca de metade dos casos bem-sucedidos. Já o modelo DeepSeek-V3 apresenta resultados insatisfatórios nesse nível, o que evidencia uma limitação importante na sua capacidade de lidar com problemas de emulação de rede mais gerais e complexos.

O tipo de *prompt* mostrou-se, em geral, um fator benéfico, indicando que descrições mais específicas e detalhadas tendem a elevar a taxa de acerto. Essa relação entre nível de detalhe e desempenho sugere que dedicar esforço à formulação cuidadosa dos *prompts* é uma prática decisiva para melhorar a qualidade das respostas em sistemas baseados em LGE. Em contrapartida, a necessidade de ajuste frequentemente apareceu associada a falhas, sugerindo que a intervenção humana atua como um sinal claro de baixa qualidade inicial do código gerado.

Dessa forma, a análise fatorial indica que o desempenho dos modelos depende da interação entre esses fatores, e não de um único fator isolado. A complexidade do cenário assume papel central ao elevar a incidência de erros à medida que a emulação de rede se torna mais complexa, enquanto *prompts* mais detalhados melhoram a probabilidade de sucesso do código gerado.

4.3. Análise Qualitativa de erros

Além da análise quantitativa de sucesso, foi realizada uma análise qualitativa para classificar os tipos de erros encontrados nos códigos que falharam ou que necessitaram de algum ajuste para funcionar. O objetivo foi identificar padrões de falha específicos de

cada modelo, para compreender melhor suas limitações. Os erros foram categorizados da seguinte forma:

Os principais erros observados foram classificados em três categorias: **Erro de Sintaxe**, quando o código viola regras da linguagem ou da estrutura esperada, por exemplo, ao usar nomes de classes incorretos ou criar conexões inválidas; **Erro de Lógica**, quando o script é sintaticamente correto, mas não faz o que foi descrito no prompt, falhando na execução da lógica desejada; e **Erro de Alucinação**, quando o modelo introduz configurações, funcionalidades ou parâmetros que não existem na documentação do mininet-wifi.

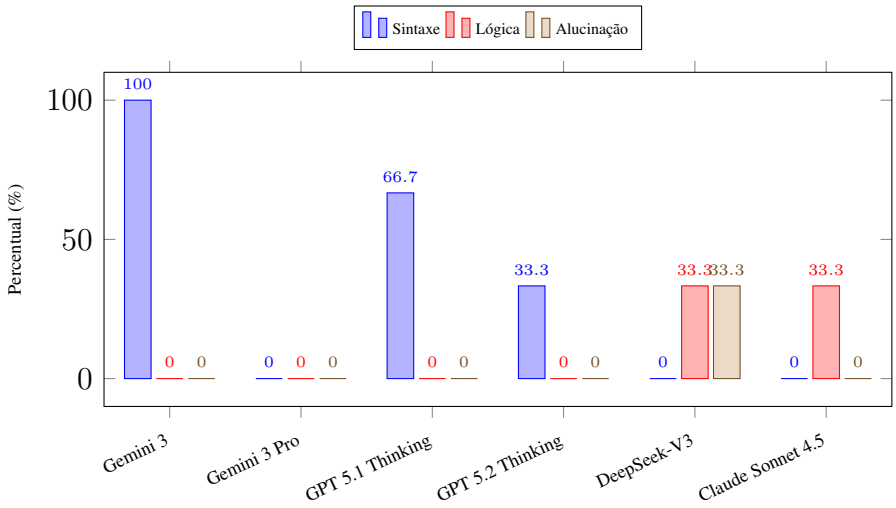

Figura 3. Distribuição dos tipos de erro por modelo de IA.

A Figura 3 mostra que o erro de sintaxe foi o tipo de falha mais frequente, em grande medida devido ao desempenho dos modelos *Gemini 3* e *GPT 5.1 Thinking*. Esse resultado indica que, embora os códigos gerados sejam próximos do esperado, ainda apresentam erros relevantes. o modelo *Gemini 3*, por exemplo, exibiu uma taxa de 100% de erros de sintaxe, produzindo scripts que falhavam na definição do nome da classe principal em todos os cenários avaliados.

O modelo *DeepSeek-V3* foi o único a apresentar simultaneamente erros de lógica e alucinação de parâmetros, principalmente pela invenção de configurações inexistentes, não suportadas pela ferramenta. Já o modelo *Claude Sonnet 4.5* apresentou apenas erros de lógica, nos cenários de maior complexidade. Nesse caso, embora o código gerado tenha sido próximo do esperado e chegue a executar, ele falha na implementação da mobilidade exigida pelo cenário, ao tentar acessar coordenadas por meio de atributos que já não são compatíveis com a versão atual da biblioteca utilizada, demonstrando uma interpretação inadequada de restrições técnicas da API.

4.4. Avaliação por Análise de Variância

Para avaliar de forma quantitativa o impacto dos fatores considerados sobre a geração de códigos Python funcionais no Mininet-WiFi, foi empregada uma Análise de Variância (ANOVA) com três fatores. A ANOVA é uma técnica estatística utilizada para comparar a variância entre as médias de diferentes grupos com a variância observada dentro de cada

grupo. Quando a variância entre grupos se for significativamente maior do que a variância interna, conclui-se que existem diferenças estatisticamente significativas entre as médias analisadas [Montgomery 2017].

Na presente análise, a ANOVA foi empregada para investigar, de forma conjunta, o impacto de três fatores sobre o sucesso dos códigos Python executados no Mininet-WiFi: O fator "Modelo" compreendeu os diferentes LGEs avaliados, enquanto o fator "Cenário" representou os níveis de dificuldade das topologias e das tarefas de rede, e o fator "Tipo de Prompt" diferenciou descrições mais simples de instruções mais específicas e detalhadas.

Tabela 3. Análise de variância dos fatores avaliados

Fonte de Variação	Soma dos Quadrados	Graus de Liberdade	Estatística F	Valor-p
C(Modelo)	1.555556	5	2.290909	0.073767
C(Cenário)	1.555556	2	5.727273	**0.008445**
C(Tipo_Prompt)	0.444444	1	3.272727	0.081589
Residual	3.666667	27	–	–

Ao analisar os resultados, a ANOVA indicou que apenas o fator associado à complexidade do cenário apresentou efeito estatisticamente significativo, com valor-p de 0,008445, inferior ao nível de significância padrão ($\alpha = 0.05$). Isso evidencia que o grau de complexidade influencia de maneira decisiva a probabilidade de obtenção de *scripts* funcionais na primeira execução. Em contraste, os efeitos associados ao modelo e ao tipo de *prompt* não se destacaram tanto.

4.5. Análise de Resposta dos Modelos

Por fim, foi realizada uma análise qualitativa das respostas dos modelos, que mostrou diferenças no estilo e na utilidade prática dos códigos Python gerados para o Mininet-WiFi. Os modelos das famílias GPT e Gemini responderam de forma mais objetiva, retornando apenas o *script* solicitado, com foco em entregar a solução, mas com pouca ou nenhuma contextualização adicional fora do código. Em contrapartida, o modelo DeepSeek apresentou um comportamento mais "explicativo", pois além do código em si, o modelo acrescentou comentários no script, além de instruções de execução, características da topologia, especificações técnicas e cenários de testes extras.

Por outro lado, a mesma análise qualitativa mostrou que o DeepSeek-V3 também foi o modelo que mais concentrou problemas estruturais em cenários avançados. Em alguns *scripts*, observou-se alucinações em relação à API do Mininet-WiFi, como a criação incorreta de pontos de acesso, além de falhas na lógica de engenharia de rede. Esses casos ilustram que comentários e dicas de uso, embora valiosos, não compensam erros conceituais e de aderência à API, reforçando a necessidade de uma etapa de revisão especializada mesmo quando o modelo se mostra mais robusto e didático em suas respostas.

5. Comparação com Exemplos Oficiais e Testes com Modelo Local

Com base nas análises realizadas, identificamos o Gemini 3 Pro como o modelo de melhor desempenho global por ser o único que não demandou ajustes manuais nos códigos, mantendo uma taxa de sucesso elevada mesmo em topologias mais complexas. Além disso, os *scripts* produzidos por esse modelo apresentavam comentários explicativos, descrevendo trechos específicos do código, o que favorece a compreensão da lógica por pessoas sem

domínio na área. O projeto avançou para uma fase de comparação com exemplos oficiais do Mininet-WiFi e de testes com um modelo de Linguagem de Pequena Escala Local (LPE) executado localmente.

Inicialmente, avaliamos a capacidade do *Gemini 3 Pro* de replicar um exemplo de referência de complexidade intermediária: o *script* `handover.py`, disponível no repositório oficial do Mininet-WiFi. Esse exemplo demonstra um cenário de *handover* entre dois *Access Points* operando em canais distintos, com mobilidade programada das estações, uso de um modelo de propagação *logDistance* em regime severo e um *backbone* cabeado entre os APs, sob controle de um controlador SDN. O objetivo dessa etapa foi verificar se o modelo online seria capaz de gerar um código estruturalmente semelhante ao exemplo oficial, replicando os principais elementos da topologia, os parâmetros de mobilidade e os mecanismos de *handover* presentes no *script* de referência.

Em seguida, investigamos se um modelo local, de menor exigência computacional, poderia apresentar sinais de reprodução de códigos de qualidade comparável. Para isso, selecionamos o *gpt-oss-20B*, modelo open-weight da família GPT-OSS que apresenta arquitetura otimizada para raciocínio e geração de código [Bi et al. 2025]. O modelo foi obtido e inicializado localmente por meio do *Ollama*, uma plataforma de código aberto que simplifica a execução de Modelos de Linguagem de Pequena Escala Local diretamente na máquina, oferecendo interface própria e suporte a execução totalmente local [Ollama 2025].

Inicializamos o *Gemini 3 Pro* e adaptamos o *gpt-oss-20B* via *few-shot in-context learning*, fornecendo o mesmo *prompt* de contextualização das fases anteriores e incorporando apenas alguns exemplos representativos via prompt. Após essa etapa, ambos os modelos receberam um *prompt* solicitando a geração de um *script* capaz de implementar uma rede equivalente ao exemplo *handover.py*, preservando topologia lógica, parâmetros relevantes e o comportamento de mobilidade.

Os resultados indicam que o *script* gerado pelo *Gemini 3 Pro* consegue reproduzir o comportamento do exemplo selecionado. Embora existam diferenças sintáticas e de organização do código, os dois *scripts* cumprem o mesmo objetivo, instanciando a mesma topologia lógica e aplicando regras físicas de propagação compatíveis, o que permite validar o mecanismo de *handover* entre pontos de acesso em um ambiente SDN. Já os testes realizados com o *gpt-oss-20B* indicaram que o modelo local também foi capaz de produzir *scripts* funcionais para o cenário de *handover*, ainda que com maior variação na estrutura do código e necessidade de inspeção mais cuidadosa. Em ambos os casos, a topologia e o comportamento de mobilidade foram executados, sugerindo que modelos locais open-weight, orientados via *few-shot in-context learning*, podem ter qualidade similar ao modelos online de alto desempenho para tarefas de geração de código em Mininet-WiFi, oferecendo uma alternativa mais acessível do ponto de vista computacional e de controle sobre o ambiente de execução.

6. Conclusão e trabalhos futuros

Este estudo apresentou uma investigação sistemática sobre o uso de LGEs na geração de código em Python para a emulação de redes no Mininet-WiFi, cobrindo diferentes cenários de complexidade e dois tipos de *prompt*. As análises mostraram que a complexidade do cenário é o fator que mais impacta o sucesso dos códigos produzidos, enquanto

o tipo de *prompt* atua como mecanismo de melhoria e a necessidade de ajuste funciona como indicador prático da qualidade inicial dos *scripts*. Em simulações simples, todos os modelos foram capazes de produzir códigos funcionais, sugerindo que, em cenários de baixa complexidade, as diferenças entre os modelos tendem a ser menores.

No conjunto de modelos avaliados, o *Gemini 3 Pro* destacou-se por não exigir grandes ajustes manuais e, ainda assim, gerar códigos funcionais em todos os cenários, incluindo os mais complexos. A comparação com o exemplo oficial `handover.py`, do repositório do Mininet-WiFi, indicou que o modelo é capaz de produzir um *script* funcional e equivalente ao exemplo de referência, preservando topologia lógica, parâmetros de mobilidade e mecanismos de *handover*, ainda que com diferenças de estilo e organização. A etapa adicional com o modelo local *gpt-oss-20B*, executado via *Ollama* e adaptado por *few-shot in-context learning*, mostrou que, com pequenos ajustes e uma inspeção cuidadosa, um LPE também pode gerar *scripts* funcionais para o cenário de *handover*, aproximando-se do comportamento esperado.

Como trabalhos futuros, pretende-se ampliar o conjunto de cenários avaliados, incluindo topologias maiores e modelos de mobilidade mais realistas bem como investigar estratégias automáticas de validação dos *scripts* gerados, combinando testes de execução, análise estática e métricas de qualidade de código para reduzir a dependência de inspeção manual. Outra direção promissora é a comparação entre modelos online e modelos locais especializados (via fine-tuning) especificamente para o domínio de redes, avaliando o desempenho, custo computacional, privacidade e reprodutibilidade dos experimentos, além da extensão da abordagem para outros frameworks de simulação/emulação (como ns-3 ou OMNeT++).

Disponibilidade de Artefatos

Em aderência aos princípios da Ciência Aberta, o código-fonte e o dataset utilizados neste trabalho podem ser acessados em: `https://github.com/bps90/llms-mininet-wifi`.

Agradecimentos

Gostaríamos de agradecer ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), ao Comitê Gestor da Internet no Brasil (CGI.br), à Fundação de Amparo à Pesquisa do Estado da Bahia (FAPESB) e aos auxílios 444077/2024-3, 407568/2025-5, 2018/23097-3, 2020/05182-3, TIC 002/2015.

Referências

Ahmed, T., Azwad, M. M., and Choudhury, S. (2025). Simcode: A benchmark for natural language to ns-3 network simulation code generation. *ArXiv*, abs/2507.11014.

Anthropic (2025). Models overview. `https://docs.anthropic.com/en/docs/about-claude/models/overview`. Acesso em: 12 dez. 2025.

Bi, Z., Chen, K., Tseng, C.-Y., Zhang, D., Wang, T., Luo, H., Chen, L., Huang, J., Guan, J., Hao, J., et al. (2025). Is gpt-oss good? a comprehensive evaluation of openai's latest open source models. *arXiv preprint arXiv:2508.12461*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Burfeid, S., Rettore, P. H. L., Zißner, P., Sevenich, P., Celes, C., and Santos, B. P. (2025). Genete - generative network environments: A specialized small language model. In *MILCOM 2025 - 2025 IEEE Military Communications Conference (MILCOM)*, pages 1462–1467.

Cruz, A., Rettore, P. H., and Santos, B. P. (2025). Análise do uso de modelos de linguagem de grande escala na geração de códigos para automação residencial. In *Brazilian Symposium on Multimedia and the Web (WebMedia)*, pages 48–56. SBC.

DeepSeek (2025). Lists models. `https://api-docs.deepseek.com/quick_start/pricing`. Acesso em: 12 dez. 2025.

Fontes, R. (2025). Mininet-wifi: Emulation platform for software-defined wireless networks. `https://mininet-wifi.github.io/`. Acessado em: 14 dez. 2025.

Fontes, R. and Rothenberg, C. (2019). Mininet-wifi: Plataforma de emulação para redes sem fio definidas por software. In *Anais Estendidos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 201–208, Porto Alegre, RS, Brasil. SBC.

Google (2025). Gemini models. https://ai.google.dev/gemini-api/docs/models. Acesso em: 12 dez. 2025.

Ifland, B., Krief, R., Zilberman, A., Duani, E., Ohana, M., Murillo, A., Manor, O., Lavi, O., Hikichi, K., Shabtai, A., et al. (2025). Genet: A multimodal llm-based co-pilot for network topology and configuration. In *2025 IEEE 45th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 117–122. IEEE.

Montgomery, D. C. (2017). *Design and Analysis of Experiments*. John Wiley & Sons, Hoboken, NJ, 9 edition.

Ollama (2025). Ollama. `https://ollama.com`. Acesso em: jan. 2025.

OpenAI (2025). Models. https://platform.openai.com/docs/models. Acesso em: 12 dez. 2025.

Raiaan, M. A. K., Mukta, M. S. H., Fatema, K., Fahad, N. M., Sakib, S., Mim, M. M. J., Ahmad, J., Ali, M. E., and Azam, S. (2024). A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access*, 12:26839–26874.

Soares, F. A., Franco, M. F., Scheid, E. J., and Granville, L. Z. (2025). Trust, but verify: An empirical evaluation of ai-generated code for sdn controllers.

Tiamiyu, O. A., Onidare, S. O., Akande, H. B., Ajayi, O. T., and Ogbotobo, A. O. (2025). Implementation and Comparison of Software-Defined Network Controllers in various Simulated Network Environments. *FUDMA JOURNAL OF SCIENCES*, 9(3):154–163.

Wang, X. and Zhu, D. (2024). Validating llm-generated programs with metamorphic prompt testing. *arXiv preprint arXiv:2406.06864*.