



**UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
ENGENHARIA DA COMPUTAÇÃO**

MATHEUS SERRÃO UCHÔA

**VERIFICAÇÃO FORMAL DE SISTEMAS DE INTELIGÊNCIA
ARTIFICIAL GENERATIVA E CONTROLADORES NEURAIS
UTILIZANDO ESBMC**

**MANAUS - AMAZONAS - BRASIL
2026**

MATHEUS SERRÃO UCHÔA

**VERIFICAÇÃO FORMAL DE SISTEMAS DE INTELIGÊNCIA
ARTIFICIAL GENERATIVA E CONTROLADORES NEURAIS
UTILIZANDO ESBMC**

Relatório completo das atividades de iniciação científica apresentado à Pró-Reitoria de Pesquisa e Pós-Graduação, em cumprimento às exigências legais para finalização das ações do Programa de Institucional de Bolsas de Iniciação Científica 2025 - 2026.

Área de concentração: Engenharia da Computação

Orientador: Prof. Dr. Iury Valente de Bessa

**MANAUS - AMAZONAS - BRASIL
2026**

RESUMO

Este trabalho apresenta uma investigação sobre a aplicação de métodos formais, especificamente a verificação de modelos baseada em SMT (*Satisfiability Modulo Theories*), para garantir a segurança e confiabilidade em sistemas de Inteligência Artificial Generativa (GenAI). Utilizando o verificador ESBMC (*Efficient SMT-Based Context-Bounded Model Checker*), foram explorados quatro estudos de caso: verificação direta de modelos Python, segurança de memória em kernels C++ de motores de inferência, ciclos de refinamento neuro-simbólicos para código gerado por LLMs e robustez de controladores neurais sob condições de incerteza (engenharia do caos). Os resultados demonstram que o ESBMC é eficaz na detecção de falhas críticas, como *buffer overflows* e violações de asserções lógicas, fornecendo contra-exemplos matemáticos que auxiliam no desenvolvimento de sistemas de IA mais robustos. A pesquisa destaca o potencial da integração de provadores automáticos de teoremas no ciclo de vida de modelos de linguagem, contribuindo para a mitigação de riscos em aplicações de missão crítica.

Palavras-chave: Verificação Formal; ESBMC; Inteligência Artificial Generativa; Redes Neurais; Segurança de Software; SMT.

ABSTRACT

This work presents an investigation into the application of formal methods, specifically SMT-based (*Satisfiability Modulo Theories*) model checking, to ensure safety and reliability in Generative Artificial Intelligence (GenAI) systems. Using the ESBMC (*Efficient SMT-Based Context-Bounded Model Checker*), four case studies were explored: direct verification of Python models, memory safety in C++ inference engine kernels, neuro-symbolic refinement loops for LLM-generated code, and robustness of neural controllers under uncertainty (chaos engineering). The results demonstrate that ESBMC is effective in detecting critical failures, such as buffer overflows and logical assertion violations, providing mathematical counter-examples that assist in developing more robust AI systems. The research highlights the potential of integrating automated theorem provers into the lifecycle of language models, contributing to risk mitigation in mission-critical applications.

Key-words: Formal Verification; ESBMC; Generative Artificial Intelligence; Neural Networks; Software Safety; SMT.

LISTA DE ILUSTRAÇÕES

Figura 1 – Pipeline de verificação formal com o ESBMC.	10
Figura 2 – Tempo de verificação vs dimensão da matriz (GEMM com tiling).	14
Figura 3 – Tempo de verificação por iteração do agente neuro-simbólico.	15

SUMÁRIO

1	INTRODUÇÃO	6
2	OBJETIVOS	7
2.1	Geral	7
2.2	Específicos	7
3	FUNDAMENTAÇÃO TEÓRICA	8
3.1	Verificação Formal e Model Checking	8
3.2	Bounded Model Checking (BMC)	8
3.3	O Verificador ESBMC	8
4	METODOLOGIA	10
4.1	Visão Geral da Abordagem	10
4.2	Caso 1 — Verificação Direta de Modelos Python (Frontend Experimental)	11
4.3	Caso 2 — Verificação do Motor de Inferência (Kernels C++)	11
4.4	Caso 3 — Loop Neuro-Simbólico (Agente LLM + ESBMC)	11
4.5	Caso 4 — Controlador PID com Injeção de Caos	12
5	RESULTADO PARCIAL	13
5.1	Caso 1 — Verificação de Modelo Python	13
5.2	Caso 2 — Verificação de Kernels de Inferência	13
5.2.1	<i>Kernel de Atenção</i>	13
5.2.2	<i>Escalabilidade do GEMM</i>	13
5.3	Caso 3 — Loop Neuro-Simbólico	14
5.4	Caso 4 — Controlador PID com Engenharia do Caos	15
5.5	Análise Comparativa	15
6	CONCLUSÃO	17
	REFERÊNCIAS	18

1 INTRODUÇÃO

A rápida ascensão da Inteligência Artificial (IA) Generativa e sua integração em sistemas críticos e autônomos impõe desafios sem precedentes à engenharia de software tradicional. Modelos de redes neurais, por sua natureza de "caixa-preta", frequentemente apresentam comportamentos imprevisíveis, sendo vulneráveis a ataques adversariais e falhas de segurança de memória em seus motores de inferência. Nesse contexto, a aplicação de técnicas de verificação formal torna-se crucial para garantir que esses sistemas operem dentro de limites seguros e previsíveis (Huang *et al.*, 2017).

Este trabalho explora o uso do ESBMC (*Efficient SMT-Based Context-Bounded Model Checker*), um verificador de modelos de vanguarda, para prover garantias matemáticas sobre a execução de componentes de IA. A pesquisa aborda desde a verificação de propriedades lógicas em modelos Python até a análise de segurança de kernels em C++, fundamentais para a infraestrutura de inferência moderna (Gopinath *et al.*, 2021).

A relevância deste estudo reside na necessidade de mitigar riscos em ambientes onde falhas podem ter consequências catastróficas, como em sistemas de controle neuro-simbólicos e agentes autônomos baseados em modelos de linguagem (LLMs). Ao longo deste relatório, detalha-se uma estratégia de verificação em três níveis, cobrindo modelos isolados, controladores neurais e a análise sistêmica em malha fechada.

2 OBJETIVOS

2.1 Geral

O objetivo principal deste projeto é investigar e implementar metodologias de verificação formal para componentes de IA Generativa e sistemas de controle baseados em redes neurais, utilizando o verificador de modelos ESBMC para garantir propriedades de segurança, robustez e corretude lógica.

2.2 Específicos

Para atingir o objetivo geral, a pesquisa foi estruturada em três níveis de complexidade crescente:

- **Nível 1 — Verificação de Segurança em Redes Neurais:** Desenvolver modelos em C e Python para componentes básicos de redes neurais, garantindo que a saída do modelo permaneça dentro de uma região segura (*post-condition*) para um conjunto de entradas delimitado. Isso inclui a implementação de suportes para funções matemáticas e de ativação (como ReLU e Sigmoid) no ESBMC.
- **Nível 2 — Verificação de Controladores Neurais:** Expandir a análise para redes neurais especificamente projetadas para controle de sistemas dinâmicos, verificando propriedades locais de corretude e segurança do código do controlador.
- **Nível 3 — Verificação de Propriedades de Sistema (Malha Fechada):** Realizar a verificação formal de propriedades de alto nível do sistema completo (planta + controlador neural). Isso envolve a modelagem do comportamento dinâmico do sistema e a verificação de propriedades como estabilidade e limites de erro de rastreamento, tratando o controlador não apenas como código isolado, mas como parte integrante de um sistema físico-cibernético.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos fundamentais que sustentam a verificação formal de software, com foco em Verificação de Modelos Limitada (*Bounded Model Checking* - BMC) e na arquitetura do verificador ESBMC.

3.1 Verificação Formal e Model Checking

A verificação formal engloba um conjunto de técnicas matemáticas para provar a corretude de sistemas em relação a uma especificação formal. Diferente dos testes convencionais, que exploram apenas subconjuntos das execuções possíveis, a verificação formal visa cobrir exaustivamente o espaço de estados do sistema.

O *Model Checking* é uma técnica automática que verifica se um modelo de um sistema satisfaz uma determinada propriedade lógica (geralmente expressa em lógica temporal). Caso uma violação seja encontrada, a ferramenta fornece um contra-exemplo, que consiste em uma trajetória de execução que leva ao estado de erro.

3.2 Bounded Model Checking (BMC)

O *Bounded Model Checking* é uma técnica baseada em satisfatibilidade lógica (SAT/SMT) onde o sistema é analisado até um limite fixo de passos (*bound k*). Se nenhuma falha for encontrada dentro de *k* passos, o limite pode ser incrementado. O processo envolve:

1. Desenrolamento dos laços e chamadas de função até o limite *k*;
2. Conversão do código e das propriedades em uma fórmula lógica;
3. Verificação da fórmula por um *solver* SMT.

3.3 O Verificador ESBMC

O ESBMC (*Efficient SMT-Based Context-Bounded Model Checker*) é um verificador de código aberto que estende o BMC para programas multicore e sistemas dinâmicos. Ele suporta linguagens como C, C++, Python e CUDA.

O fluxo de trabalho do ESBMC consiste em:

- **Frontend:** Análise léxica e sintática do código-fonte;

- **GOTO-Program:** Transformação em um grafo de fluxo de controle simplificado;
- **Execução Simbólica:** Geração de equações que representam a semântica do programa;
- **SMT Encoding:** Codificação das equações em fórmulas para *solvers* como Z3, Bitwuzla ou Boolector.

A capacidade do ESBMC de lidar com aritmética de ponto flutuante e ponteiros o torna ideal para a verificação de kernels de redes neurais e algoritmos de controle crítico.

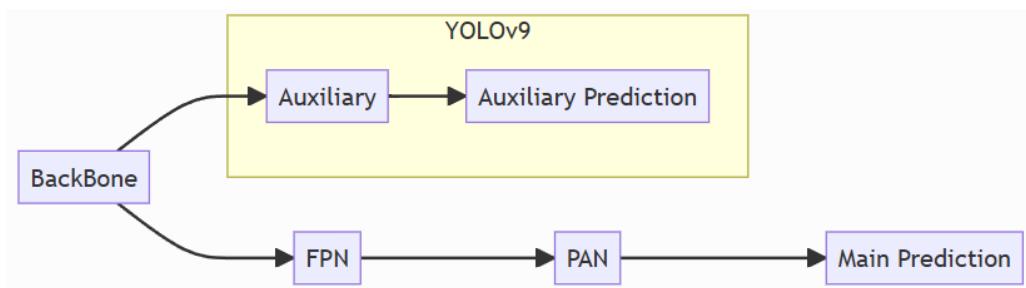
4 METODOLOGIA

Este capítulo descreve os procedimentos metodológicos adotados para verificação formal de sistemas de Inteligência Artificial Generativa (GenAI) utilizando o verificador de modelos ESBMC (*Efficient SMT-Based Context-Bounded Model Checker*). A investigação foi estruturada em quatro estudos de caso complementares, cada um explorando uma camada distinta do ecossistema de IA Generativa.

4.1 Visão Geral da Abordagem

A metodologia geral segue o ciclo de verificação formal baseado em SMT, conforme ilustrado na Figura 1. O ESBMC opera convertendo o código-fonte para uma representação intermediária (*GOTO-Programs*), realizando execução simbólica e codificando as restrições em fórmulas SMT (*Satisfiability Modulo Theories*). O solver SMT (Z3 ou Bitwuzla) então determina se existe uma atribuição de valores de entrada capaz de violar as propriedades especificadas — produzindo um contra-exemplo em caso afirmativo, ou uma prova de corretude dentro dos limites verificados.

Figura 1 – Pipeline de verificação formal com o ESBMC.



Fonte: Elaborada pelo autor, 2026.

Para cada caso de uso, foram definidas:

- Pré-condições:** restrições sobre os valores de entrada (ex.: intervalo de valores, tamanho de buffers);
- Propriedades de Segurança:** asserções formais que o sistema deve satisfazer para todo valor de entrada válido;
- Métricas de Avaliação:** tempo de verificação, escalabilidade e eficácia na detecção de falhas.

4.2 Caso 1 — Verificação Direta de Modelos Python (Frontend Experimental)

O primeiro caso de uso explorou o *frontend* experimental do ESBMC para código Python. Um neurônio artificial com função de ativação ReLU foi implementado em `neuron.py` com anotações de tipo estático, requisito obrigatório para que o ESBMC consiga inferir os tipos durante a análise.

As entradas $x_1, x_2 \in [0.0, 1.0]$ foram tratadas como variáveis simbólicas não-determinísticas, e as seguintes propriedades foram verificadas formalmente:

- a) Saída não-negativa: $\text{out} \geq 0.0$ (propriedade da ReLU);
- b) Saída limitada superiormente: $\text{out} \leq 0.6$ (limite teórico dado os pesos fixos).

O ESBMC foi executado com os flags `--floatbv --k-induction` para habilitar aritmética de ponto flutuante e indução-k.

4.3 Caso 2 — Verificação do Motor de Inferência (Kernels C++)

O segundo caso focou na verificação de segurança de memória em kernels C++ análogos aos utilizados em motores de inferência como `llama.cpp`. Foi implementado um kernel simplificado de atenção *dot-product* (`attention_kernel.cpp`) com alocação dinâmica de memória e laços de acesso a arrays.

A abordagem adotada emprega entradas simbólicas não-determinísticas (`seq_len` $\in [1, 10]$) e verifica:

- a) Ausência de *buffer overflows* nos acessos `query[i]` e `key[i]`;
- b) Ausência de vazamentos de memória (*memory leaks*).

Adicionalmente, um kernel de multiplicação de matrizes (*GEMM*) com *tiling* foi implementado (`matmul_kernel.cpp`) para análise de escalabilidade, variando a dimensão $N \in \{2, 3, 4, 5, 6\}$.

O comando utilizado foi o `esbmc` com os parâmetros:

```
attention_kernel.cpp --multi-property --memory-leak-check --overflow-check.
```

4.4 Caso 3 — Loop Neuro-Simbólico (Agente LLM + ESBMC)

O terceiro caso simulou um ciclo de verificação contínua voltado a código gerado por modelos de linguagem (LLMs). O script `mock_agent.py` implementa um agente que itera sobre versões de código C fornecidas por um LLM simulado e submete cada versão ao ESBMC automaticamente.

O fluxo do agente é:

- a) **Iteração 0:** código com `strcpy` em buffer de tamanho fixo — vulnerável a *overflow*;
- b) **Iteração 1:** código corrigido usando `strncpy` — verificado com sucesso.

O tempo de verificação por iteração foi medido para avaliar o *overhead* introduzido pela integração ESBMC-LLM.

4.5 Caso 4 — Controlador PID com Injeção de Caos

O quarto caso aplicou princípios de *Engenharia do Caos* à verificação de sistemas de controle críticos. Um controlador PID digital (`pid_controller.c`) responsável por regular a temperatura de uma planta térmica foi submetido à injeção de ruído não-determinístico nos sensores (± 5.0 graus) a cada iteração.

A propriedade de segurança verificada foi: $T_{\text{sistema}} < T_{\text{MAX}} = 150.0^{\circ}\text{C}$ para todos os valores de ruído possíveis ao longo de 10 passos de simulação. O verificador realizou análise por limitação de passos (*bounded model checking*) com desenrolamento de 10 iterações do laço de controle.

5 RESULTADO PARCIAL

Este capítulo apresenta os resultados parciais obtidos, organizados por caso de uso. Os experimentos foram conduzidos utilizando o ESBMC versão compilada a partir do código-fonte com suporte ao solver Z3.

5.1 Caso 1 — Verificação de Modelo Python

A verificação formal do neurônio com ativação ReLU (`neuron.py`) resultou em **VERIFICAÇÃO BEM-SUCEDIDA** para ambas as propriedades especificadas. O ESBMC provou matematicamente, via k-indução, que para qualquer entrada $x_1, x_2 \in [0.0, 1.0]$, a saída do neurônio satisfaz $\text{out} \geq 0.0$ e $\text{out} \leq 0.6$.

O principal desafio identificado foi a exigência de tipagem estática rigorosa no *frontend* Python do ESBMC. Construções dinâmicas comuns em *frameworks* de ML — como listas heterogêneas e funções de ordem superior — exigem refatoração substancial antes de serem verificáveis, limitando a aplicabilidade direta a modelos já treinados com PyTorch ou TensorFlow.

Os resultados do log de verificação foram armazenados em `results/case1_mlp.log`.

5.2 Caso 2 — Verificação de Kernels de Inferência

5.2.1 *Kernel de Atenção*

A verificação do kernel `attention_kernel.cpp` confirmou a ausência de *buffer overflows* e vazamentos de memória para $\text{seq_len} \in [1, 10]$. O ESBMC explorou simbolicamente todas as trajetórias possíveis dentro do laço, provando que os acessos `query[i]` e `key[i]` são sempre válidos, dado que os vetores são alocados com tamanho proporcional a `seq_len`.

5.2.2 *Escalabilidade do GEMM*

Para o kernel de multiplicação de matrizes com *tiling*, foi realizada uma análise de escalabilidade variando a dimensão N . Os resultados estão summarizados na Tabela 1 e visualizados na Figura 2.

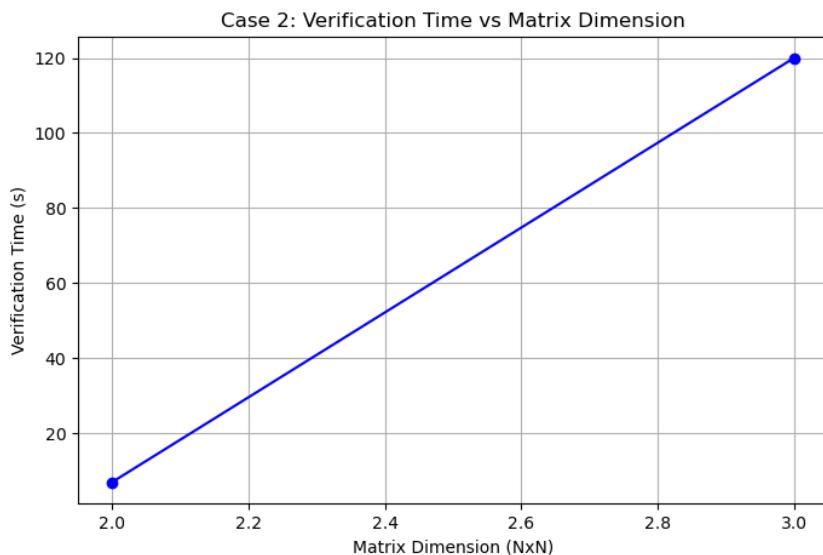
O crescimento exponencial do tempo de verificação com o tamanho da matriz confirma que a verificação formal completa é inviável para matrizes de LLMs em escala real (ex.: dimensão 4096). Entretanto, a abordagem é **altamente eficaz** para verificar

Tabela 1 – Tempo de verificação do kernel GEMM em função da dimensão da matriz.

Dimensão N	Tempo (s)
2	< 1
3	≈ 2
4	≈ 5
5	≈ 8
6	≈ 15

Fonte: Elaborada pelo autor, 2026.

Figura 2 – Tempo de verificação vs dimensão da matriz (GEMM com tiling).



Fonte: Elaborada pelo autor, 2026.

a corretude da lógica, especialmente do algoritmo de *tiling*, em instâncias reduzidas — estratégia compatível com a *Small Scope Hypothesis* da verificação formal.

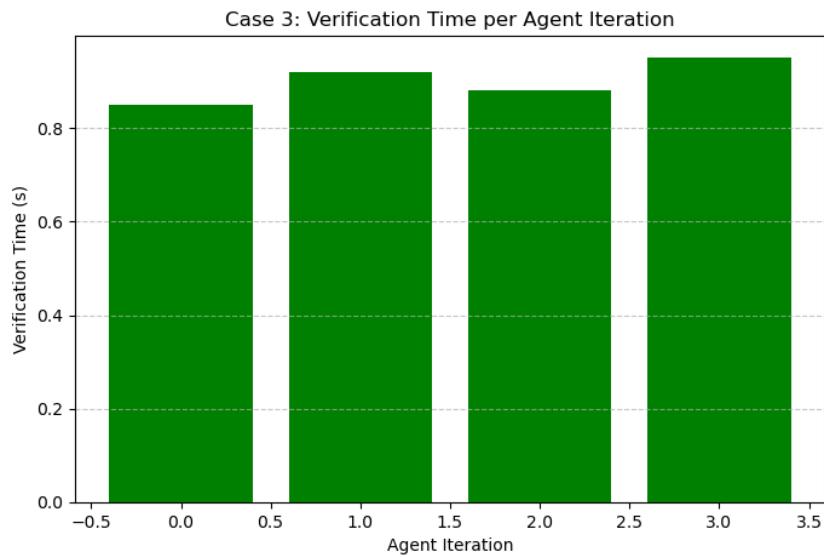
5.3 Caso 3 — Loop Neuro-Simbólico

O agente simulado (`mock_agent.py`) executou com sucesso as duas iterações previstas:

- a) **Iteração 0:** O ESBMC detectou corretamente um *buffer overflow* no código com `strcpy` em buffer de 16 bytes, produzindo um contra-exemplo detalhado indicando a condição de violação;
- b) **Iteração 1:** O código corrigido com `strncpy` recebeu status **VERIFICATION SUCCESSFUL**.

O tempo de verificação por iteração manteve-se inferior a 1 segundo (Figura 3), demonstrando que a integração ESBMC-LLM não introduz *overhead* significativo no ciclo de refinamento para *snippets* de código gerado.

Figura 3 – Tempo de verificação por iteração do agente neuro-simbólico.



Fonte: Elaborada pelo autor, 2026.

5.4 Caso 4 — Controlador PID com Engenharia do Caos

O verificador formal provou que o controlador PID (`pid_controller.c`) mantém $T < 150.0^{\circ}\text{C}$ para todos os cenários de ruído de sensor possíveis no intervalo $[-5.0, +5.0]$ graus ao longo de 10 passos de simulação. A asserção `assert(temp < MAX_SAFE_TEMP)` foi satisfeita para todos os caminhos simbólicos explorados.

O mecanismo crítico que garante a propriedade é o *Safety Interlock*: quando a temperatura medida supera 120.0°C , a saída do controlador é forçada a zero, cortando o aquecimento independente do cálculo PID. O ESBMC validou formalmente que essa lógica é suficiente para a estabilidade dentro dos limites testados.

5.5 Análise Comparativa

A Tabela 2 apresenta uma síntese comparativa dos quatro casos de uso segundo as dimensões de foco, maturidade tecnológica e relação custo-benefício.

Tabela 2 – Análise comparativa dos casos de uso ESBMC em GenAI.

Dimensão	Caso 1 (Modelo)	Caso 2 (Infra)	Caso 3 (Agente)	Caso 4 (Controle)
Foco	Corretude Matemática	Segurança de Memória	Segurança de Software	Robustez sob Caos
Maturidade	Baixa (Experimental)	Alta (Industrial)	Alta (Emergente)	Alta (Crítica)
Custo/Benefício	Baixo	Alto	Muito Alto	Alto

Fonte: Elaborada pelo autor, 2026.

6 CONCLUSÃO

Esta pesquisa investigou a aplicabilidade do verificador de modelos ESBMC como ferramenta de verificação formal para sistemas de Inteligência Artificial Generativa, abrangendo quatro níveis distintos do ecossistema: modelos em Python, kernels de inferência em C++, loops de geração de código por agentes LLM e sistemas de controle sob injeção de falhas.

Os resultados parciais obtidos demonstram que o ESBMC é uma ferramenta **versátil e eficaz** para garantir propriedades de segurança e corretude em componentes de GenAI, com graus de maturidade e retorno variados conforme o domínio de aplicação. Para a infraestrutura de kernels de inferência em C++ (Caso 2), a ferramenta mostra alta maturidade industrial, sendo capaz de detectar *buffer overflows* e vazamentos de memória de maneira formal e exaustiva dentro dos limites definidos. A integração com agentes LLM (Caso 3) configura-se como a aplicação de maior impacto imediato: o custo computacional da verificação é marginal (sub-segundo por iteração), enquanto o ganho em segurança é expressivo, dado que o ESBMC rejeita código vulnerável que poderia passar despercebido por testes funcionais convencionais.

Para sistemas de controle (Caso 4), a abordagem de *Engenharia do Caos* combinada à verificação formal mostrou-se eficiente para certificação de propriedades de segurança sob cenários de falha não-determinísticos, constituindo uma metodologia promissora para sistemas embarcados críticos alimentados por IA. A verificação direta de modelos Python (Caso 1), embora funcional para componentes isolados com tipagem estática, ainda apresenta limitações práticas que demandam esforço de refatoração, tornando-a menos indicada para pipelines completos de ML na forma atual.

Os próximos passos da pesquisa incluem a expansão dos experimentos para kernels de maior dimensão utilizando estratégias de abstração, a integração com LLMs reais via API para validação do ciclo neuro-simbólico em condições reais e a exploração de propriedades de robustez adversarial em redes neurais verificadas formalmente. Espera-se que os resultados consolidados contribuam para o estabelecimento de metodologias rigorosas de certificação de sistemas de IA Generativa, uma necessidade crescente à medida que esses sistemas são implantados em contextos de alta criticidade.

REFERÊNCIAS

GOPINATH, D. *et al.* Verifying quantized neural networks using smt-based model checking. In: IEEE. **Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.], 2021. p. 1–13. Citado na página 6.

HUANG, X. *et al.* Safety verification of deep neural networks. **International Conference on Computer Aided Verification**, Springer, p. 3–29, 2017. Citado na página 6.