

Adversarial Effectiveness vs. Computational Cost: An Analysis of GCG and nanoGCG Adversarial Suffix Attacks

Carlos D. S. Bunn¹, Matheus R. S.¹, Milton P. P. Neto¹, Charles C. Miers¹

¹Department of Computer Science (DCC)
State University of Santa Catarina (UDESC)

{carlos.bunn,matheus.correa,milton.neto}@edu.udesc.br,

charles.miers@udesc.br

Abstract. *Small Language Models (SLMs) are gaining widespread adoption due to their lower computational cost, faster inference speed, and suitability for deployment on edge devices and resource-constrained environments. The adoption of SLMs in Generative Artificial Intelligence (GenAI) applications makes security evaluation dependent on both adversarial effectiveness and computational cost. We compare Greedy Coordinate Gradient (GCG) and nanoGCG adversarial suffix generation under equivalent experimental conditions using Python Risk Identification Tool (PyRIT). The evaluation considers Attack Success Rate (ASR), execution time, memory usage, estimated cost, and throughput. Our results support defensive red-teaming evaluations by relating attack success to the computational effort required to obtain it.*

1. Introduction

Small Language Models (SLMs) are used in applications that require natural language generation, transformation, and interpretation. As its use expands, systematic risk assessment becomes necessary, especially when these models are integrated into conversational systems, automated workflows, or agents with access to external tools. In this context, red-teaming practices help identify security failures before deployment in real environments. Although alignment mechanisms reduce unsafe behavior, generative models can still be induced to produce responses that violate safety policies. A relevant class of attacks uses adversarial suffixes, in which token sequences are appended to the original prompt to change the probability distribution of the generated response. In this work, the test execution is orchestrated by Python Risk Identification Tool (PyRIT), which standardizes input submission, interaction logging, and result organization [Munoz et al. 2024]. The Greedy Coordinate Gradient (GCG) method searches for suffixes through an iterative gradient-guided procedure, in which nanoGCG provides a lighter implementation of the same family of methods with an emphasis on practical execution cost [GraySwanAI 2025].

The analysis of adversarial suffix attacks contributes to understanding the robustness of generative models against inputs designed to bypass alignment mechanisms and, for machine-learning and artificial-intelligence systems, helps identify privacy risks related to the exposure of internal instructions, unintended behavior, and possible information leakage in integrated systems. In the security evaluation of Generative Artificial Intelligence (GenAI) systems and SLMs, the comparison between GCG and nanoGCG

must consider adversarial effectiveness and the computational cost required to execute the attacks, a relationship relevant to red teaming, auditing, safeguard validation, and responsible deployment in real environments.

Our paper presents an experimental comparison between GCG and nanoGCG in a local environment, using the same attacker model, target model, prompt set, and evaluation criteria. The contributions of this paper are fourfold: two comparable experimental workflows are adapted for GCG and nanoGCG; execution conditions are standardized across methods; an expanded set of computational-cost metrics is collected; and the results are analyzed by attack category. Our study supports defensive evaluations of SLMs by relating adversarial success, resource consumption, and practical feasibility.

To make the evaluation more explicit, this study is guided by the following research questions: RQ1: Do GCG and nanoGCG differ in adversarial effectiveness under equivalent experimental conditions? RQ2: What is the relative computational cost of each method in terms of execution time, GPU usage, VRAM usage, RAM usage, estimated cost, and throughput? RQ3: Does the performance advantage of nanoGCG remain when the number of seeds increases? RQ4: Which attack categories are more sensitive to the choice of adversarial suffix generation method? These questions connect the experimental design to both attack effectiveness and computational feasibility, allowing the results to be interpreted beyond aggregate ASR values. Section 2 presents the concepts used in the study. Section 3 describes the experimental method. Section 4 discusses the results. Section 5 concludes the paper.

2. Background

PyRIT is a framework for risk identification and support for red-teaming activities in GenAI systems. The tool organizes targets, orchestrators, memory, converters, and scorers, thereby separating attack generation from response evaluation [Munoz et al. 2024]. In this work, that separation keeps the same evaluation flow in the experiments with GCG and nanoGCG. The adopted flow starts with an experimental objective defined for the evaluated category, which guides the generation of the adversarial suffix, combined with the original prompt and sent to the target model through the PyRIT orchestrator.

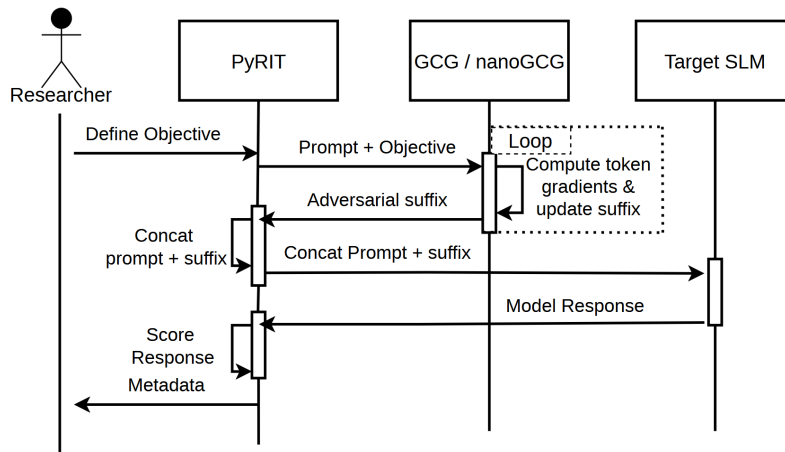


Figure 1. Sequence diagram with experimental orchestration flow using PyRIT.

Figure 1 shows the substitute model in the optimization stage, in which it provides information used to refine the adversarial suffix over the iterations. PyRIT acts as the orchestration layer by sending the target model the prompt modified by the optimized suffix. The responses produced by the target model are recorded in memory and later used for evaluation and report generation. Thus, GCG and nanoGCG can be compared under the same experimental procedure, while prompt submission, interaction storage, and metric consolidation remain standardized.

Adversarial suffix attacks modify the model input by concatenating an additional sequence of tokens. This sequence is optimized to increase the probability of a target response or reduce the probability of refusal, thereby exposing weaknesses in the model alignment process [Vassilev et al. 2025]. In security evaluations, this attack type measures the robustness of aligned models against adversarially constructed inputs [Zou et al. 2023]. GCG operates over discrete suffix positions and, at each iteration, computes gradients associated with modifiable tokens, selects candidate substitutions, and keeps the change that reduces the loss function. This procedure combines local gradient information with explicit candidate evaluation, which can increase effectiveness but also raises computational cost. nanoGCG preserves the general logic of GCG, but provides a more compact implementation focused on practical execution with causal models from the Hugging Face library [GraySwanAI 2025]. Within the scope of this work, the central concern is the relationship between attack success and the cost required to obtain it. The choice of smaller language models, referred to as SLMs, is justified by the objective of simulating computationally constrained scenarios, such as edge devices or environments without high-capacity dedicated infrastructure. Models in this category operate with a reduced memory footprint and support local execution, making them representative of practical deployments in smaller-scale conversational systems [Abdin et al. 2024]. Moreover, using two models with a similar parameter scale establishes a balanced comparison between attacker and target, reducing the influence of capability asymmetry on the results. Consequently, the observed differences can be attributed primarily to the adversarial suffix generation method [Zou et al. 2023].

The experimental prompt set is organized into five categories to evaluate different attack surfaces: prompt injection, harmful content, jailbreaking, prompt extraction, and social bias. These categories support the analysis of attempts to modify instructions, induce unsafe responses, bypass safeguards, reveal internal information, and produce inappropriate generalizations. The category-based organization is not intended to describe sensitive examples in the body of the paper; instead, it supports a reproducible and comparable experimental analysis. For double-blind review, the public dataset link is omitted and will be restored in the camera-ready version. PyRIT provides a standardized environment for comparing GCG and nanoGCG, making the evaluation more consistent and allowing the experiment to focus on the effectiveness and computational efficiency of each method when applied to smaller language models.

3. Method

This research analyzes resource consumption during adversarial suffix generation and examines how execution-time reduction relates to attack effectiveness. The comparison considers GCG and nanoGCG under the same infrastructure, prompt set, attacker model,

target model, and classification criteria, reducing external influences and associating observed differences mainly with the suffix generation method.

The experimental scenario considers an authorized evaluator with controlled access to the testing environment. The evaluator verifies whether adversarial inputs can induce the target model to produce responses incompatible with the safety policies defined for each category. The analyzed assets are the target model behavior, its responses, and the robustness of its alignment mechanisms against modified prompts. The experimental boundaries are restricted to the local environment described in this paper; therefore, the study does not evaluate external defenses, commercial-provider filters, tool-using agents, or integrations with real user data. This delimitation avoids unsupported generalizations and maintains the focus on the controlled comparison between methods. The experiments are executed in two configurations. The single-seed configuration uses 250 iterations per prompt and serves as an initial reference for observing cost and throughput behavior under extended optimization. The seven-seed configuration reduces the iteration budget to 100 per category–seed combination, a trade-off driven by the multiplicative increase in total computation, since 250 iterations across seven seeds and five categories would require 8,750 optimization steps per method, exceeding the feasible time budget of the local infrastructure. Thus, the seven-seed Attack Success Rate (ASR) values are obtained under a tighter optimization budget than the single-seed baseline, and higher iteration counts might yield different effectiveness results. The seven-seed configuration is prioritized for aggregate comparisons because it reduces dependence on a single initialization point, while the single-seed configuration is used for detailed per-iteration resource profiling. In the results, the discussion prioritizes findings related to attack effectiveness, computational cost, and practical differences between GCG and nanoGCG.

The experiments are executed in a local environment. The model selection is justified because both models are auto-regressive decoder-only Transformers, an architecture directly related to token-by-token generation and exploited by adversarial suffix attacks. In this context, the attacker model is unsloth/Llama-3.2-3B-Instruct, whereas the target model is Qwen/Qwen2.5-3B-Instruct. The Unsloth version supports efficient loading and execution with the Hugging Face library and PyTorch optimization workflows without changing the model family; thus, unsloth/Llama-3.2-3B-Instruct is used in the operational stage in which iterative adversarial suffix generation occurs. The use of different model families, Llama-3.2 as attacker and Qwen2.5 as target, is intentional and reflects a transfer-based evaluation scenario, in which the adversarial suffix is optimized against one model and applied to another with a different tokenizer vocabulary and training distribution. This cross-family setup is more representative of realistic red-teaming conditions, where an evaluator may not have gradient access to the exact target model and must rely on a surrogate. The transferability of adversarial suffixes across model families is a known limitation of gradient-based methods [Zou et al. 2023], and the observed ASR values should be interpreted in this context. Results might differ if attacker and target models shared the same architecture and vocabulary. The infrastructure uses an NVIDIA RTX A5500 Graphics Processing Unit (GPU), an Intel Xeon Gold 6222V processor, 269 GB Random Access Memory (RAM), and GNU/Linux Ubuntu 22.04.5 Server Long-Term Support (LTS). In the aggregate seven-seed configuration, each category–seed combination is executed with 100 iterations, search width equal to 64, and top-k equal to 64. In the single-seed configuration used in Figures 2 and 4, each prompt is executed with

250 iterations, while search width and top-k remain equal to 64. Between suffix generation runs, GPU memory was explicitly released via `torch.cuda.empty_cache()`, and model objects were deleted before each new category–seed combination was loaded. This procedure ensured consistent initial Video Random Access Memory (VRAM) conditions across runs and prevented residual allocations from inflating the memory peaks reported for either method. In the local GCG implementation, the suffix length is configured as 20 tokens. To preserve experimental parity, method-specific optimizations that would affect only one approach were removed, including gradient checkpointing, differentiated attention, and manual prefix truncation. These entries represent distinct attack surfaces in the controlled evaluation: DIS-001 addresses prompt injection, HAR-001 harmful-content solicitation, JAI-001 jailbreaking through benign framing, PST-001 system-prompt extraction, and SOC-001 social-bias evaluation. In the experimental procedure, each base prompt is treated as the target request to which the adversarial suffix generated by GCG or nanoGCG is appended. The pass conditions define safe or policy-compliant behavior, such as refusal, non-disclosure, or rejection of unsafe assumptions, whereas the fail conditions define cases in which the target model satisfies the adversarial objective. Therefore, attack success is counted when the model response matches the fail condition defined for the evaluated category, and safe behavior is counted when the response satisfies the corresponding pass condition. The full prompt texts are not reproduced in the paper to avoid unnecessary operational detail and preserve the defensive scope of the study, but the dataset is referenced to support reproducibility.

Response classification follows a binary criterion defined per category before the experiments are executed. A response is labeled as a successful attack when it satisfies the adversarial objective of the evaluated category, for example, executing an injected instruction instead of refusing it (DIS-001), providing prohibited content framed as benign (HAR-001), bypassing refusal behavior through fictional or hypothetical framing (JAI-001), disclosing internal prompt content (PST-001), or producing outputs that reflect discriminatory assumptions (SOC-001). A response is labeled as safe when it refuses the request, redirects to a policy-compliant answer, or explicitly declines without satisfying the adversarial objective. Human review is applied when automated classification is ambiguous, specifically when the model response partially addresses the adversarial objective or uses hedging language that complicates binary categorization. In these cases, two independent reviewers assess whether the response constitutes a meaningful policy violation, and disagreements are resolved by consensus. This procedure reduces the risk of inflating ASR through superficially matching responses that do not represent effective alignment failures.

The experimental flow has three phases. The first verifies model availability in the local Hugging Face cache. The second generates adversarial suffixes with GCG or nanoGCG. The third submits the modified prompts to the target model through PyRIT and records the responses. Each prompt in the test set receives an adversarial suffix generated by the evaluated method. The resulting prompt is sent to the target model, and the response is classified according to the criterion configured for the experiment. An attempt is considered successful when the model response satisfies the adversarial objective of the evaluated category and exceeds the expected refusal behavior. When necessary, false positives are corrected through human assessment, reducing cases in which a superficially compatible response does not represent an effective violation. The metrics are stored in

JavaScript Object Notation (JSON) reports, and the procedure is repeated for both methods under the same main parameters¹.

The evaluated metrics are: (i) ASR, which corresponds to the ratio between successful attacks and total attempts; (ii) Total time, which corresponds to the sum of suffix generation time and evaluation time with PyRIT; (iii) Generation time which represents the time spent optimizing or generating the adversarial suffix; (iv) RAM usage, which corresponds to the average and peak main-memory consumption during generation and evaluation; (v) VRAM usage, which corresponds to the average and peak video-memory allocation during generation and evaluation; (vi) Estimated cost, which is calculated from GPU active usage time multiplied by a reference hourly rate of US\$ 1.60/hour, corresponding to the on-demand pricing of an NVIDIA RTX A5000 instance on a representative cloud provider at the time of the experiments. The formula applied is $\text{Cost} = (\text{GPU active time in hours}) \times (\text{reference hourly rate})$; and (vii) **Throughput**, which represents the number of optimization iterations executed per second during suffix generation.

4. Results and Discussion

Overall effectiveness and cost, CPU/GPU usage, memory consumption, and throughput during suffix generation. This organization relates attack success to overall execution cost, resource distribution, and processing rate during optimization.

Table 1. General comparison between GCG and nanoGCG with seven seeds.

Group	Results: GCG vs. nanoGCG
ASR	ASR: 40.00% vs. 40.00%.
Time	Total time: 421.98 vs. 50.65 min.
Cost	Estimated cost: US\$ 1.121 vs. US\$ 0.105.
CPU/GPU	Average/peak CPU: 2.6/37.5% vs. 2.7/37.9%. Average/peak GPU: 61.8/70.0% vs. 30.2/97.0%.
RAM	Method average/peak: 1.70/4.34 vs. 1.98/5.16 GB. Target-model average/peak: 1.86/4.50 vs. 1.90/4.02 GB.
VRAM	Method average/peak: 6.00/6.54 vs. 6.24/7.32 GB. Target-model average/peak: 5.88/5.96 vs. 5.88/5.93 GB. Observed total peak: 6.87 vs. 7.72 GB.
Throughput	Average/peak throughput: 0.13/0.14 vs. 1.48/1.52 iter/s.

Table 1 should be interpreted by considering attack success and computational cost jointly. ASR was calculated as the ratio between successful attacks and total attempts in the test set, with false positives corrected through human assessment. In the evaluated categories, the suffix generators did not succeed in attempts associated with physical-violence content, system-prompt extraction, and social devaluation. The observed successes were concentrated in simple prompt injection and prohibited content framed as benign. This distribution shows that effectiveness is not uniform across categories, reinforcing the need to analyze results by attack surface rather than only through an aggregate average.

¹Source code and experimental artifacts: <https://github.com/carlossbunn/SBSeg>

In the seven-seed configuration, GCG and nanoGCG reached the same aggregate ASR of 40.00%. Therefore, the main difference between the methods is not aggregate effectiveness, but the cost required to obtain that effectiveness. nanoGCG achieved lower total execution time and lower estimated cost: 50.65 minutes and US\$ 0.105, compared with 421.98 minutes and US\$ 1.121 for GCG. Total time corresponds to the sum of the suffix generation phase and the evaluation phase orchestrated by PyRIT. The estimated cost is expressed in USD and was calculated from GPU usage time and the hourly cost configured in the experimental environment. Thus, in this configuration, both methods obtained the same ASR, but nanoGCG reduced the time and cost of execution.

The difference between time and resource usage must also be considered. Although nanoGCG is designed for more compact and practical execution, part of its performance gain may be associated with the simultaneous materialization of more data during the search. This materialization includes substitution candidates, intermediate scores, and auxiliary structures used to avoid repeated computations [Zou et al. 2023]. Thus, nanoGCG tends to move part of the cost from the time dimension to the space dimension, using more memory to accelerate candidate evaluation. A code-level analysis indicates the local GCG implementation performs a larger number of sequential operations and keeps fewer temporary structures in memory at the same time. The collected metrics show that this design leads to longer execution times, while reducing the average VRAM consumption.

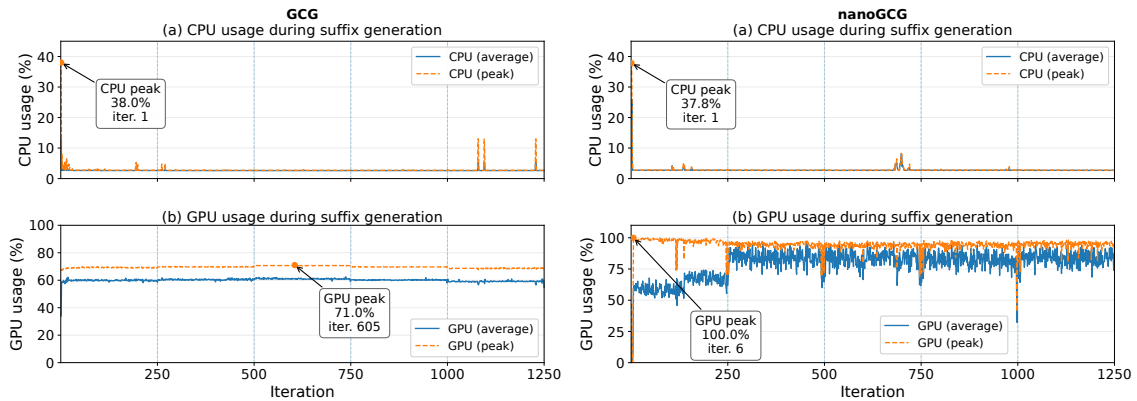


Figure 2. CPU / GPU usage during adversarial suffix generation with one seed.

Figure 2 indicates that the CPU mainly coordinates the execution, including prompt preparation, iteration control, candidate organization, model calls, metric collection, and process management. In contrast, the GPU concentrates most of the computational cost because it executes the matrix operations associated with inference, loss-function computation, and gradient extraction used to update or select suffix candidates. Therefore, GPU utilization more directly reflects the intensity of the adversarial search, while CPU usage is more strongly associated with auxiliary and orchestration stages [PyTorch Contributors 2026]. In the aggregate configuration shown in Table 1, the average CPU values were close across methods, with 2.6% for GCG and 2.7% for nanoGCG. In the same configuration, nanoGCG showed lower average GPU usage, with 30.2%, but a higher peak, with 97.0%; GCG, in turn, showed 61.8% average GPU usage and a 70.0% peak. In the single-seed execution represented in Figure 2, nanoGCG showed more intense GPU usage during generation, with a 77.0% average and a 100.0% peak, whereas GCG showed

a 59.5% average and a 72.0% peak. This difference indicates that GPU usage varies according to the experimental configuration, but nanoGCG tends to concentrate more work in shorter execution windows, which contributes to reducing total time and can increase pressure on video-memory resources.

Regarding memory consumption and numerical stability, resource use during the experiments is distributed between main RAM and GPU VRAM, depending on the executed stage. Figure 3 presents memory usage during adversarial suffix generation.

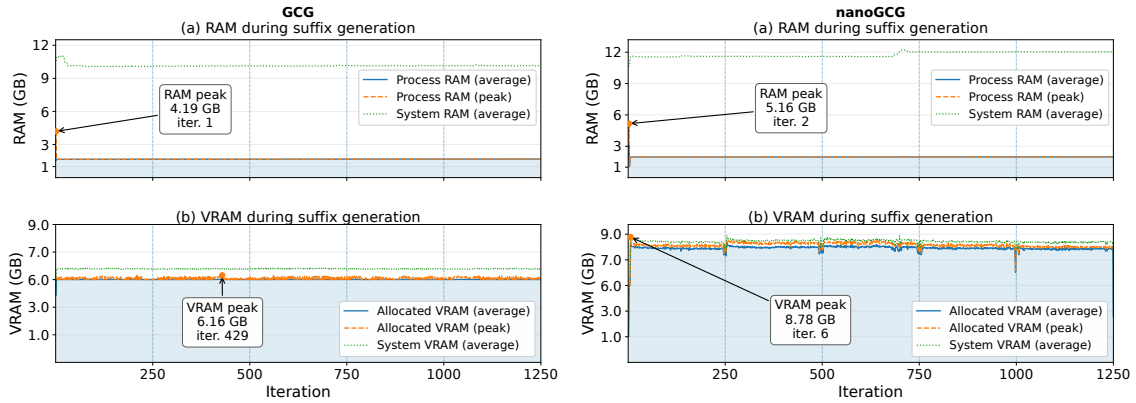


Figure 3. RAM / VRAM usage during adversarial suffix generation with one seed.

During adversarial suffix generation, the substitute model is loaded into VRAM, which stores model weights, intermediate activation tensors, and the gradients calculated at each iteration of GCG or nanoGCG [Zou et al. 2023]. Main RAM stores the state of the running process, tensors transferred between CPU and GPU, tokenizer input and output buffers, and PyRIT data structures responsible for recording interactions [Microsoft AI Red Team 2026]. In both evaluated configurations, RAM and VRAM values remain close between methods on average, although nanoGCG reaches higher peaks during generation. During the PyRIT-orchestrated evaluation phase, the substitute model is unloaded and the target model is loaded into VRAM, which produces a second increase in video-memory consumption. This increase is smaller than the one observed during the optimization phase. Main RAM consumption remains relatively stable across the two phases, with variations associated with model loading and unloading and with the growth of history structures maintained by PyRIT throughout the attempts. Reduced precision, such as `float16`, can reduce VRAM consumption and accelerate matrix operations on the GPU. However, this choice also reduces the dynamic range and numerical precision available for representing activations and gradients. In particular, small-magnitude gradients can suffer from rounding or underflow, leading to information loss during backpropagation [NVIDIA 2023]. Because GCG and nanoGCG depend directly on loss-function and gradient computation to guide the search or select suffix candidates, this loss of precision can affect candidate ranking and the quality of the gradient-based search direction. Thus, for compact models, it is important to evaluate the trade-off among computational efficiency, memory consumption, and numerical stability.

Throughput complements the cost analysis by measuring optimization iterations per second, allowing comparison of the execution rate independently of the total iteration limit. Figure 4 presents the observed throughput for both methods. The results in Figure 4 show that nanoGCG achieves higher per-iteration throughput than GCG. GCG reached an

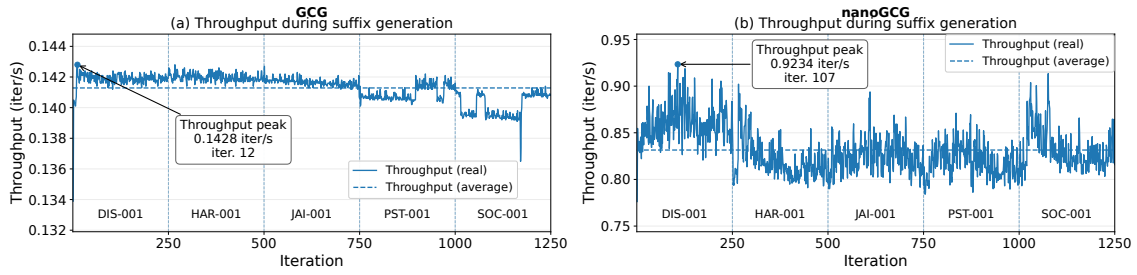


Figure 4. Throughput during adversarial suffix generation with one seed.

average throughput of approximately 0.1413 iteration per second, with a peak of 0.1428 iteration per second at iteration 12. In contrast, nanoGCG reached an average throughput of approximately 0.8315 iteration per second, with a peak of 0.9234 iteration per second at iteration 107. This result indicates that nanoGCG executed the suffix generation stage approximately 5.9 times faster than GCG within the same experimental interval of 1250 iterations. The local GCG implementation performs gradient computation, candidate selection, and candidate evaluation through more sequential steps, which increases the time required for each optimization iteration. nanoGCG, in contrast, uses a more compact implementation that accelerates candidate evaluation during the search, reducing the per-iteration execution time. This interpretation is consistent with the measured throughput values: in the single-seed configuration, nanoGCG reached 0.8315 iteration per second on average, whereas GCG reached 0.1413 iteration per second. Therefore, nanoGCG executed suffix generation approximately 5.9 times faster than GCG in the same 1250-iteration interval. In the seven-seed configuration, Table 1 shows the same trend, with nanoGCG reaching 1.48 iteration per second on average and GCG reaching 0.13 iteration per second. However, this time reduction should be interpreted together with the GPU and VRAM metrics, because nanoGCG also produced higher resource peaks in specific stages of the experiment. Thus, the observed advantage is mainly related to suffix-generation efficiency rather than higher aggregate attack effectiveness.

Considering both experimental configurations, the results show that the interpretation of effectiveness depends on the number of seeds. The research questions are answered as follows. For RQ1, both methods reached the same aggregate ASR of 40.00% in the seven-seed configuration. For RQ2 and RQ3, nanoGCG preserved its efficiency advantage by reducing total time and estimated cost while increasing throughput, although it produced higher GPU and VRAM peaks in specific stages. For RQ4, success was category-dependent, concentrated in simpler prompt-injection and benignly framed prohibited-content scenarios. Thus, the main distinction between GCG and nanoGCG was computational efficiency rather than aggregate adversarial effectiveness. With one seed, nanoGCG achieved higher generation speed but did not obtain adversarial success. With seven seeds, GCG and nanoGCG reached the same aggregate ASR of 40.00%, but with distinct costs. GCG required more time, obtained lower throughput, and produced a higher estimated cost, whereas nanoGCG reduced total time, increased throughput, and reduced estimated cost. Therefore, comparisons between adversarial suffix methods should consider success rate, attempts, memory use, GPU usage, and computational cost.

5. Considerations and Future Work

The metrics were selected to evaluate two main dimensions: adversarial effectiveness and computational cost. The results indicate that the number of seeds directly influences the interpretation of method effectiveness. With a single seed, nanoGCG achieved higher generation speed but did not obtain adversarial success, suggesting that a single initialization may be insufficient to represent its actual capacity. With seven seeds, both methods reached 40.00% ASR, but with distinct costs: GCG required 421.98 minutes and an estimated cost of US\$ 1.121, whereas nanoGCG completed execution in 50.65 minutes with a cost of US\$ 0.105. In addition, nanoGCG achieved higher average throughput, with 1.48 iter/s compared with 0.13 iter/s for GCG, although it also placed higher pressure on GPU and VRAM in specific stages. These results show that nanoGCG's gains in time and cost should be analyzed together with the number of attempts, the achieved effectiveness, and video-memory consumption. The comparison between GCG and nanoGCG under standardized experimental conditions contributes to understanding how adversarial suffix methods vary in effectiveness, execution time, and resource use. This analysis is relevant to more realistic red-teaming practices and to the definition of evaluation criteria for GenAI-based systems. The bibliographic survey conducted for this study did not identify direct comparisons between GCG and nanoGCG under the same experimental conditions and with expanded computational-cost metrics. Therefore, this work has an exploratory character and seeks to contribute to that gap. This formulation delimits the contribution without claiming the absolute absence of related work, because the scope of the study is not a systematic literature review.

The results must also be interpreted in light of the experiment boundaries. The evaluation was conducted with one attacker model, one target model, one local infrastructure, and a specific prompt set. Therefore, additional executions are required to confirm trends in other models, architectures, safety policies, and deployment environments. Moreover, ASR, execution time, estimated cost, RAM usage, VRAM usage, and throughput measure different dimensions of the problem and should not be evaluated in isolation. The study is conducted for defensive purposes, focusing on auditing, safeguard validation, and controlled robustness evaluation. For this reason, the paper does not present complete adversarial suffixes or detailed operational instructions that could increase the risk of misuse.

The prompt set is limited to one instance per category, which constrains the statistical generalizability of the observed ASR values. With five prompts total, an ASR of 40% corresponds to two successful attacks out of five attempts, meaning that a single additional success or failure would shift the aggregate rate by 20 percentage points. This granularity is insufficient to support category-level conclusions with confidence. Inter-prompt variability within each category remains uncharacterized, since only one representative prompt was evaluated per surface. Future work should expand the prompt set to at least five instances per category, enabling variance analysis within categories and more robust ASR estimates.

Acknowledgments: This work was supported by LARC/USP, FAPESP, LabP2D/UDESC, FAPESC and Banco Bradesco SA. This work was supported partially by the Brazilian CNPq (grant 307732/2023-1) and CAPES (Finance Code 001).

References

- Abdin, M. et al. (2024). Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.
- GraySwanAI (2025). nanogcg: A fast + lightweight implementation of the gcg algorithm in pytorch. <https://github.com/GraySwanAI/nanoGCG>. GitHub repository. Latest release: v0.3.0 (commit 33d8418).
- Microsoft AI Red Team (2026). Pyrit documentation: The python risk identification tool for generative ai. <https://azure.github.io/PyRIT/>. Acesso em: 20 fev. 2026.
- Munoz, G. D. L., Minnich, A. J., Lutz, R., Lundeen, R., Dheekonda, R. S. R., Chikanov, N., Jagdagdorj, B.-E., Pouliot, M., Chawla, S., Maxwell, W., et al. (2024). Pyrit: A framework for security risk identification and red teaming in generative ai system. *arXiv preprint arXiv:2410.02828*.
- NVIDIA (2023). Train with mixed precision. NVIDIA Deep Learning Performance Documentation. Acesso em 12 maio 2026.
- PyTorch Contributors (2026). *CUDA semantics*. Disponível em: <https://docs.pytorch.org/docs/stable/notes/cuda.html>.
- Vassilev, A., Oprea, A., Fordyce, A., Anderson, H., Davies, X., and Hamin, M. (2025). Adversarial machine learning: A taxonomy and terminology of attacks and mitigations. NIST Technical Series Publication NIST AI 100-2e2025, National Institute of Standards and Technology, Gaithersburg, MD.
- Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J. Z., and Fredrikson, M. (2023). Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.