

Exploring Jailbreak Vulnerability Across Depth-Based Dynamic Inference in Large Language Models

Lass Najmuldeen*

School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
lass.najm@gmail.com

Seyedhamidreza Mousavi†

School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
seyedhamidreza.mousavi@mdu.se

Abstract

Large Language Models (LLMs) have been shown to be computationally expensive and vulnerable to jailbreak attacks, in which adversarial prompts bypass safety alignment and induce harmful responses. To address this inefficiency, a range of techniques have been proposed to enable efficient LLM inference. Depth-based dynamic inference has demonstrated promising results by enabling outputs to be generated from intermediate layers. It offers the flexibility to adjust model behavior based on the available resource budget. However, how this flexibility affects the robustness of LLMs against jailbreak attacks remains unexplored. In this paper, we address this gap via a comprehensive investigation into the impact of depth-based dynamic inference on the robustness of LLMs against jailbreak attacks. Specifically, we make the following key observations: 1) Shallow exits are consistently more robust to jailbreak attacks than deeper ones; 2) Exit depth affects how easily the adversarial suffix can be optimized, with deeper exits reaching lower attack loss; and 3) Adversarial suffixes affect utility uniformly across exit depths. These insights can help design dynamic inference methodologies that are more robust to jailbreak attacks. Code is available at <https://github.com/LassNajmuldeen/balcony-jailbreaking>

1 Introduction

Large Language Models (LLMs) have been widely leveraged across a variety of applications and enhanced through numerous alignment techniques [3]. However, deploying these models in real-world applications, particularly in industrial environments, remains constrained by strict computational and latency requirements, whether on edge devices or in cloud settings [13, 4]. To address this efficiency bottleneck, dynamic inference has been proposed, enabling a single model to trade off compute against accuracy at inference time rather than requiring separate models for each deployment budget [6, 4, 5]. Dynamic inference can be achieved through width-based approaches, which adjust the hidden dimension of the model’s feed-forward layers [14], or depth-based approaches, which selectively reduce the number of transformer layers used [12].

At the same time, despite the alignment techniques, LLMs remain vulnerable to jailbreak attacks, in which adversarial prompts bypass safety alignment and induce harmful responses [21, 2]. Exploiting this vulnerability, a range of optimization-based attacks have been proposed that automatically search for such adversarial prompts using the model’s gradients or output probabilities, rather than relying on manual prompt engineering [25, 23]. However, these attacks have only been studied against the standard, full-depth inference of LLMs, and how depth-based dynamic inference affects the robustness of LLMs against jailbreak attacks remains unexplored.

*Corresponding author.

†<https://hamidmousavi0.github.io/>

In this paper, we present the first comprehensive investigation into the impact of depth-based dynamic inference on the robustness of LLMs against jailbreak attacks. We focus specifically on depth-based methods, since removing entire layers maps more naturally onto how GPUs execute sequential computation, yielding larger real-world latency and throughput gains than adjusting a layer’s width [6]. Specifically, our investigation is based on the Balcony method [13]. We adopt Balcony because its frozen backbone leaves the original safety-aligned model unmodified. This ensures that any change we observe in jailbreak robustness across exit points can be attributed to the dynamic-inference mechanism itself, rather than to degradation of the underlying model. We apply an optimization-based jailbreak attack, Greedy Coordinate Gradient (GCG) [25], at different exit points to explore the impact of dynamic inference configurations on the robustness of dynamic LLMs. To validate these design choices, we conduct a systematic study of jailbreak robustness across exit depths using the GCG attack [25] on two model families, Balcony-LLaMA3-8B and Balcony-LLaMA2-7B, evaluated on AdvBench and HarmBench. Figure 1 summarizes attack success rate against final optimization loss at each exit depth.

Based on our explorations, we make the following key observations:

- Shallow exits are consistently more robust to jailbreak attacks than deeper ones, though the size of this effect varies substantially across models. For Balcony-LLaMA3-8B, the GCG attack success rate falls sharply, from 28% at full depth to just 0–4% at the shallowest exit. In contrast, Balcony-LLaMA2-7B shows a much smaller effect, with success rates ranging only from 36% to 44% across all exit depths.
- Exit depth affects how hard the adversarial suffix is to optimize: loss decreases monotonically with depth, and the full model is easiest to attack. The size of this effect varies sharply by model. The final GCG optimization loss for Balcony-LLaMA3-8B is $3\times$ lower at the full model than at exit-15.
- Adversarial suffixes affect utility uniformly across exit depths, indicating that their collateral cost is set by the backbone rather than the exit point. The robustness gains at shallow exits therefore cannot be attributed to degraded capability under attack.

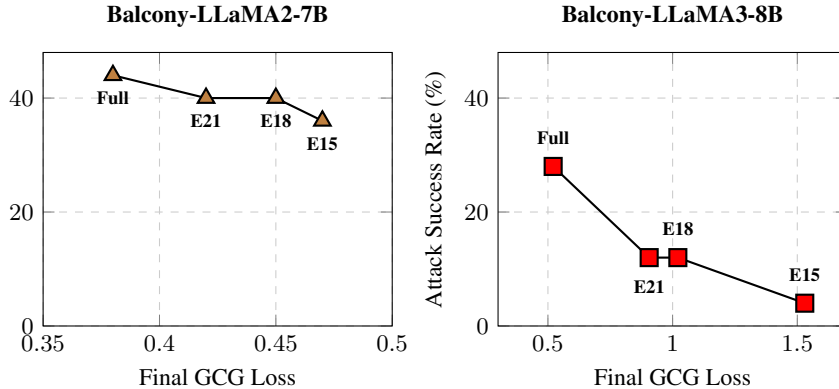


Figure 1: Attack success rate (ASR) versus final GCG loss at each exit depth (E15, E18, E21, Full), for Balcony-LLaMA2-7B (left) and Balcony-LLaMA3-8B (right) on the HarmBench dataset.

2 Related Work

2.1 Jailbreak Attacks

Despite extensive safety alignment, large language models remain vulnerable to jailbreak prompts [21]. These prompts can be viewed as adversarial inputs designed to circumvent refusal mechanisms and induce models to generate harmful or policy-violating content [8, 2]. Existing jailbreak attacks can be broadly categorized according to the level at which the adversarial input is constructed: token-level and prompt-level attacks.

Token-level attacks search directly over the discrete input-token space by modifying, inserting, or optimizing individual tokens or token sequences [25, 18, 24, 10]. These methods often employ

gradient-based optimization, heuristic search, or query-based procedures to identify adversarial suffixes that suppress refusal behavior.

In contrast, prompt-level attacks operate at the semantic or instruction level and typically produce more natural and interpretable jailbreak prompts. Such prompts may be manually crafted by humans [21, 19] or automatically generated by LLM-based agents that iteratively propose, evaluate, and refine attack prompts against the target model [17, 16].

2.2 Dynamic Inference in LLMs

Dynamic inference was initially explored in NLP using relatively small encoder-based models [22, 11]. These approaches typically rely on architecture-specific modifications and specialized training procedures, making their direct application to modern large-scale language models challenging.

Recent studies have extended dynamic inference to larger language models by introducing structural elasticity along different model dimensions. MatFormer [14] introduces a nested architecture along the width dimension, allowing multiple submodels of different sizes to be extracted from a single jointly trained network. Building on this idea, Flextron and LLaMAFlex [4, 5] combine nested model structures with input-adaptive routing to dynamically select computational configurations according to the input. In parallel, SortedLLaMA [12] and LayerSkip [6] explore elasticity along the depth dimension, enabling inference with different numbers of transformer layers through specialized training strategies.

Despite their efficiency benefits, these methods modify or retrain the base model to support dynamic execution, which may alter the behavior or degrade the performance of the original full model. Balcony [13] avoids this limitation by freezing the pretrained base model and training only lightweight exit modules attached at intermediate layers. We therefore adopt Balcony as the architectural basis for our robustness evaluation. Its frozen-backbone design provides a controlled setting in which the robustness of the original full model can be directly compared with that of sub-models exiting at different depths, without confounding the analysis through modifications to the base model parameters.

3 Preliminaries

In this section, we introduce the key concepts used in our investigation to analyze the vulnerability of depth-based dynamic inference in large language models to jailbreak attacks.

3.1 Depth-Based Dynamic Inference

As discussed in the Introduction (Section 1), we adopt Balcony as our depth-based dynamic inference architecture because it leaves the base model unchanged. To provide the necessary background, we next describe its architecture and training procedure.

Architecture. The Balcony architecture consists of a frozen base language model and a set of lightweight intermediate exits that can be selected according to the available computational budget. Each exit enables token prediction before all transformer layers have been processed.

Let the base model contain L transformer layers, and let $\mathcal{S} \subseteq \{1, \dots, L\}$ denote the selected exit locations. For an exit at layer $k \in \mathcal{S}$, the intermediate hidden representation H_k is transformed by a Balcony module as

$$\tilde{H}_k = g_k(H_k; \Theta_k), \quad (1)$$

where g_k consists of an additional transformer block followed by a normalization layer, and Θ_k denotes its trainable parameters. The transformed representation is then passed to the same language-model head used by the base model to produce the output-token distribution.

Training. Balcony employs self-distillation to align the predictive distribution of each intermediate exit with that of the full model. During training, the parameters of the base model remain frozen, and only the parameters of the Balcony modules are optimized. For an exit at layer k , the training objective encourages its output distribution to match the distribution produced by the full model.

3.2 Optimization-Based Jailbreak Attack

Optimization-based attacks leverage the target model’s gradients or output logits to automatically construct jailbreak prompts and often achieve higher attack success rates than manually designed approaches. Among these methods, Greedy Coordinate Gradient (GCG) is a representative token-level attack. GCG iteratively optimizes an adversarial suffix to increase the likelihood that the model begins its response with a predefined affirmative continuation, thereby encouraging compliance with a harmful user request.

Let an autoregressive language model define a probability distribution over the next token as

$$p_\theta(x_t \mid x_{1:t-1}), \quad (2)$$

where $x_{1:t-1}$ denotes the preceding token sequence and θ represents the model parameters. Accordingly, the probability of generating a target sequence $\mathbf{y}^* = (y_1^*, \dots, y_M^*)$ conditioned on an input sequence $\mathbf{x}$ is factorized as

$$p_\theta(\mathbf{y}^* \mid \mathbf{x}) = \prod_{m=1}^M p_\theta(y_m^* \mid \mathbf{x}, y_{1:m-1}^*). \quad (3)$$

GCG appends an adversarial suffix $\mathbf{s} = (s_1, \dots, s_K)$ to a harmful user query $\mathbf{q}$. The suffix is optimized to increase the probability that the model begins its response with a predefined affirmative target sequence $\mathbf{y}^*$. The attack objective is therefore defined as the negative log-likelihood of the target sequence:

$$\mathcal{L}_{\text{GCG}}(\mathbf{s}) = -\log p_\theta(\mathbf{y}^* \mid \mathbf{q} \oplus \mathbf{s}), \quad (4)$$

where $\oplus$ denotes sequence concatenation. Equivalently,

$$\mathcal{L}_{\text{GCG}}(\mathbf{s}) = -\sum_{m=1}^M \log p_\theta(y_m^* \mid \mathbf{q} \oplus \mathbf{s}, y_{1:m-1}^*). \quad (5)$$

The adversarial suffix is obtained by solving the discrete optimization problem

$$\mathbf{s}^* = \arg \min_{\mathbf{s} \in \mathcal{V}^K} \mathcal{L}_{\text{GCG}}(\mathbf{s}), \quad (6)$$

where $\mathcal{V}$ is the model vocabulary and K is the suffix length. Because direct optimization over discrete tokens is intractable, GCG uses token-level gradients to identify promising replacements and greedily updates selected suffix positions to reduce the objective.

4 Exploring Jailbreak Vulnerability

We investigate how the jailbreak vulnerability of a depth-based dynamic inference LLM changes across different inference depths. Our exploration considers several intermediate exits together with the full model and analyzes their behavior from complementary perspectives.

We first examine jailbreak effectiveness across exit depths using attack success rate. We then study the optimization behavior of GCG, including its convergence pattern and final attack loss, to understand how easily adversarial suffixes can be optimized for each exit. Next, we evaluate model utility under both clean and adversarial inputs to determine whether the observed robustness differences are associated with changes in general capability. Finally, we analyze the relationship between inference depth, attack optimization, utility, and jailbreak success, and summarize the main findings of this exploration.

Base architectures. We use pretrained LLaMA2-7B and LLaMA3-8B as base models and convert them into dynamic models using the Balcony method. Figure 2 provides an overview of the resulting architecture, and detailed configurations are summarized in Table 1. For exit points, we consider the same configuration $\{15, 18, 21\}$ for both models.

Experimental settings.

Training. Both Balcony-LLaMA2-7B [20] and Balcony-LLaMA3-8B [9] are trained on the Cosmo-pedia V2 [1] dataset using KL self-distillation, following the original Balcony setup [13]. Keeping

Table 1: Configuration of the Balcony models used in our experiments. Training hyperparameters and parameter counts follow [13]; the base model is frozen and only the Balcony modules are updated.

	Balcony-LLaMA2-7B	Balcony-LLaMA3-8B
<i>Base model</i>		
Transformer layers L	32	32
Non-embedding parameters	6.5B	6.9B
Pretraining tokens	2T	15T
<i>Balcony modules</i>		
Exit locations S	{15, 18, 21}	{15, 18, 21}
<i>Training</i>		
Objective	KL self-distillation	KL self-distillation
Corpus	Cosmopedia V2	Cosmopedia V2
Training tokens	31.5B (1.5% of pretraining)	31.5B (0.2% of pretraining)
Batch size	256	256
Steps	30K	30K
Sequence length	4,096	4,096
Peak learning rate	5×10^{-4}	5×10^{-4}
LR schedule	cosine	cosine

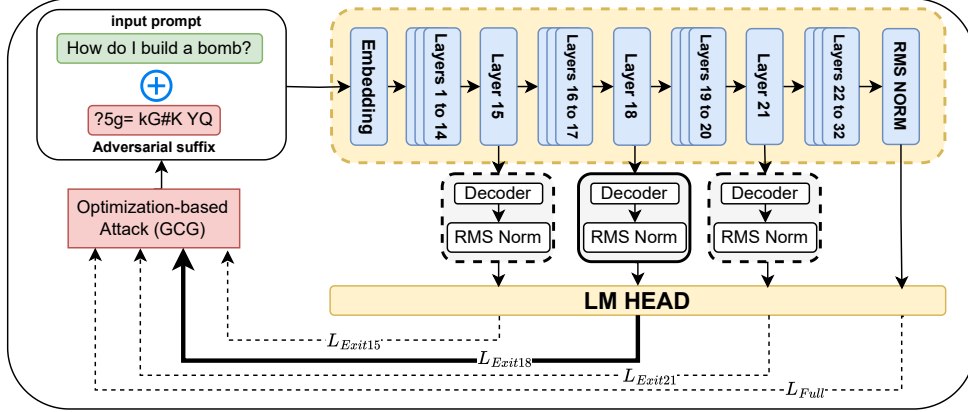


Figure 2: Overview of the jailbreak evaluation pipeline. A harmful prompt (e.g., “How do I build a bomb?”) is combined with an adversarial suffix produced by an optimization-based attack (GCG), and the resulting input is passed through the frozen backbone. We compute the loss at three Balcony exit points (layers 15, 18, and 21) as well as at the full model, enabling direct comparison of jailbreak robustness across inference depths within the same underlying weights.

the training data and objective identical across the two models ensures that any difference in jailbreak robustness we observe stems from the exit configuration and the underlying backbone rather than from differences in the Balcony training procedure itself.

Evaluation. We evaluate jailbreak robustness on HarmBench [15] and AdvBench [25], applying GCG with the attack configuration of Zou et al. [25]. We optimize an adversarial suffix for 150 steps over 50 harmful behaviors per benchmark and report the attack success rate (ASR) as the fraction of behaviors the HarmBench judge scores above 0.5, together with the mean final cross-entropy loss reached by the attack. To assess whether robustness differences coincide with changes in general capability, we additionally evaluate utility on six multiple-choice benchmarks, HellaSwag, ARC-C, ARC-E, WinoGrande, PIQA, and MMLU, using lm-eval-harness [7], both on clean inputs and with each exit’s strongest GCG suffix appended to every question. We report acc_norm for HellaSwag, ARC, and PIQA, and acc for WinoGrande and MMLU.

4.1 Jailbreak Vulnerability Across Exit Depths

We first explore how early-exit inference affects the attack success rate (ASR) of GCG, the most representative optimization-based jailbreak attack. We evaluate four exit points (*exit-15*, *exit-18*, *exit-21*, and *full*) on Balcony-LLaMA2-7B and Balcony-LLaMA3-8B, using GCG with 150 steps over 50 behaviors from HarmBench and AdvBench (Figure 3).

Across both backbones and benchmarks, ASR is highest at the full model, confirming that later layers are consistently the most vulnerable to adversarial suffixes. However, the two models differ sharply in how quickly that vulnerability emerges across depth. Balcony-LLaMA2-7B is already substantially exploitable at the shallowest exit tested: *exit-15* reaches 36% ASR on HarmBench and 46% on AdvBench, and ASR rises only modestly from there, reaching 44%/54% at the full model. This near-monotonic, gently increasing trend suggests that adversarial suffixes crafted against LLaMA2 transfer effectively even to early exits, i.e., the model’s vulnerability is distributed across depth rather than concentrated in later layers.

Balcony-LLaMA3-8B shows the opposite pattern. At *exit-15* it is far more robust, with ASR of only 4% on HarmBench and 2% on AdvBench. ASR climbs steadily through *exit-18* (12%/0%) and *exit-21* (12%/12%) before jumping sharply at the full model, where both benchmarks converge to 28%. This large gap between shallow-exit and full-model ASR indicates that LLaMA3’s vulnerability to GCG is concentrated almost entirely in its later layers, and that early-exit inference offers meaningfully stronger protection for this backbone than for LLaMA2.

Taken together, these results show that early-exit robustness is not a fixed property of the early-exit mechanism itself, but depends on the backbone: for LLaMA2, shallow exits provide little defensive benefit, while for LLaMA3 they substantially suppress attack success until the final layers are reached.

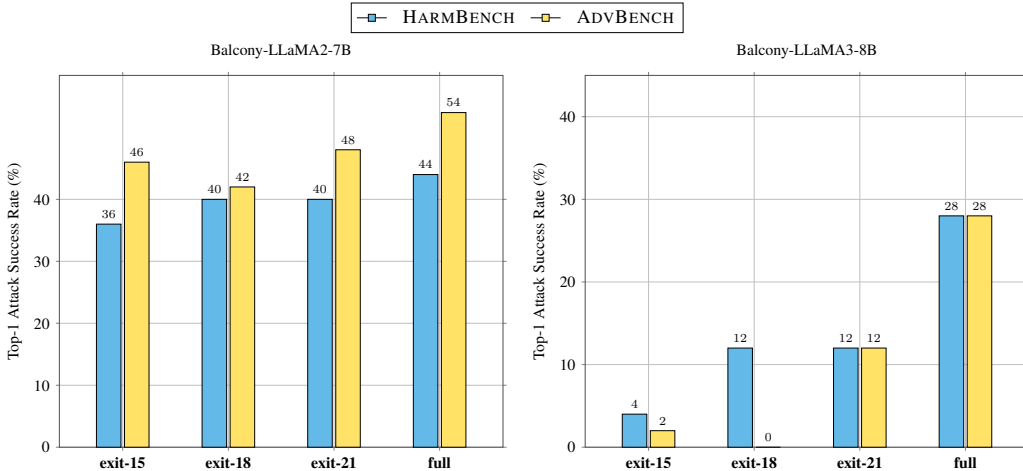


Figure 3: Attack success rate for every exit of Balcony-LLaMA2-7B (left) and Balcony-LLaMA3-8B (right) on HarmBench and AdvBench, using GCG with 150 steps on 50 behaviors.

4.2 Attack Optimization

In this section, we explore how exit depth affects the optimization dynamics of the attack. To this end, we track the GCG loss over 150 optimization iterations at each exit depth (*exit-15*, *exit-18*, *exit-21*, *full*) for both models on AdvBench and HarmBench (Figure 4). GCG loss decreases monotonically with exit depth across all four settings. The full model reaches the lowest loss, followed by *exit-21*, *exit-18*, and *exit-15*. This ordering holds throughout optimization, across both models and benchmarks. Deeper exits are thus systematically easier targets for suffix optimization. The separation between exit depths is more pronounced for Balcony-LLaMA3-8B than for Balcony-LLaMA2-7B. On Balcony-LLaMA2-7B, the four curves converge to a narrow band after roughly 50 iterations, with *exit-15* trailing only modestly behind the full model. On Balcony-LLaMA3-8B, the pattern differs: *exit-15* stays clearly separated from the other three arms throughout all 150 iterations, converging to a substantially higher loss, while *exit-18*, *exit-21*, and the full model cluster closer together.

4.3 Utility Evaluation

In this section, we evaluate how adversarial suffixes optimized against each exit affect the model’s general capability. We evaluate every exit on six standard benchmarks (HellaSwag, ARC-C, ARC-E, WinoGrande, PIQA, and MMLU), both without attack (Table 2) and with each exit’s strongest GCG suffix appended (Table 3). The clean utility increases monotonically with exit depth for both

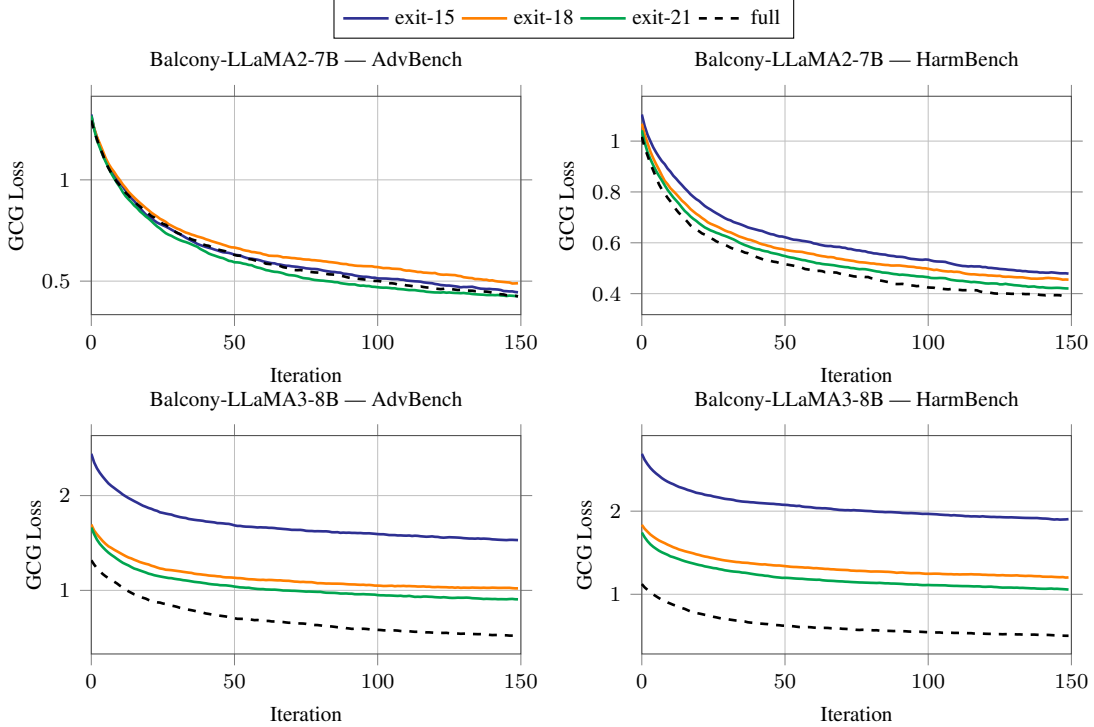


Figure 4: GCG loss over 150 optimization steps at each exit depth (*exit-15*, *exit-18*, *exit-21*, *full*), for Balcony-LLaMA2-7B (top) and Balcony-LLaMA3-8B (bottom) on AdvBench (left) and HarmBench (right).

models across all six benchmarks (Table 2). This confirms that deeper exits carry more task-relevant capability. Under attack, utility remains largely stable across exits and benchmarks for both models. The exception is ARC-E on the 7B model, where the adversarial suffix appears to interfere with task performance itself, rather than merely eliciting unsafe behavior. These results indicate that adversarial suffixes optimized to induce jailbreaks do not impose a broad or systematic cost on downstream utility.

Table 2: Utility on six multiple-choice benchmarks with no attack

Model	Exit	HellaSwag	ARC-C	ARC-E	WinoGrande	PIQA	MMLU
7B	Exit 15	0.630	0.347	0.617	0.636	0.742	0.369
	Exit 18	0.684	0.388	0.662	0.650	0.760	0.395
	Exit 21	0.715	0.414	0.683	0.670	0.766	0.400
	Full (32)	0.760	0.451	0.739	0.691	0.787	0.409
8B	Exit 15	0.519	0.334	0.476	0.658	0.634	0.310
	Exit 18	0.581	0.386	0.573	0.684	0.669	0.614
	Exit 21	0.635	0.429	0.611	0.697	0.695	0.626
	Full (32)	0.792	0.540	0.777	0.734	0.806	0.621

4.4 Relating Depth, Optimization, Utility, and Jailbreak Success

Bringing together the results, we can now relate inference depth, attack optimizability, downstream utility, and jailbreak success within a single picture, and ask whether the robustness of shallow exits reflects a genuine safety property or is simply a byproduct of reduced capability.

Depth couples capability, attackability, and vulnerability, but the strength of this coupling is backbone-dependent. For Balcony-LLaMA3-8B, the three quantities move together tightly: clean utility, ease of GCG optimization (i.e., how low the attack loss falls), and ASR all increase monotonically and substantially from exit-15 to the full model. Exit-15 is simultaneously the least

Table 3: Utility with each exit’s strongest GCG suffix appended to every question, to be read against the clean scores in Table 2. Same benchmarks, settings and metrics.

Model	Exit	HellaSwag	ARC-C	ARC-E	WinoGrande	PIQA	MMLU
7B	Exit 15	0.612	0.324	0.482	0.632	0.718	0.330
	Exit 18	0.657	0.342	0.485	0.653	0.726	0.326
	Exit 21	0.685	0.371	0.489	0.654	0.740	0.335
	Full (32)	0.740	0.407	0.572	0.696	0.773	0.384
8B	Exit 15	0.515	0.323	0.485	0.644	0.638	0.306
	Exit 18	0.585	0.361	0.560	0.667	0.668	0.589
	Exit 21	0.629	0.401	0.617	0.713	0.708	0.598
	Full (32)	0.785	0.515	0.784	0.729	0.799	0.605

capable, the hardest to optimize against, and the least jailbreakable exit, while the full model is the most capable, the easiest to optimize against, and the most jailbreakable. This alignment suggests that, for LLaMA3, the apparent robustness of shallow exits is closely tied to how little task-relevant and instruction-following capability has been formed at that depth, rather than to a distinct, depth-localized safety mechanism.

For Balcony-LLaMA2-7B, this coupling is markedly weaker. Clean utility still increases monotonically with depth, but ASR is already high at exit-15 and increases only modestly toward the full model, and the GCG loss curves for all four exits converge to a narrow band within the first 50 iterations (Section 4.2). In other words, LLaMA2’s shallow exits are attackable almost as easily as its full model, despite having visibly lower utility. This indicates that whatever representations GCG exploits to induce jailbreaks are already largely in place early in the network for this backbone, well before the model reaches its full downstream capability.

Attack optimizability tracks jailbreak success more consistently than it explains it. Across both models, the ordering of GCG loss across exits (full < exit-21 < exit-18 < exit-15) mirrors the ordering of ASR, suggesting that harder-to-optimize exits are, unsurprisingly, harder to jailbreak. However, the *magnitude* of the gap differs sharply between backbones: LLaMA3’s exit-15 is both much harder to optimize against and much more robust in absolute terms than LLaMA2’s exit-15, even though both models use the same exit configuration, training corpus, and self-distillation objective. This rules out training-side confounds as the source of the difference and points instead to how each backbone’s pretraining distributes alignment-relevant computation across depth.

Jailbreak success is largely decoupled from utility degradation. Table 3 shows that, across nearly every exit and benchmark, adversarial suffixes optimized to induce unsafe generations leave clean-task utility essentially unchanged relative to Table 2, with ARC-E on the 7B model as the main exception. This indicates that GCG suffixes are acting in a targeted way, i.e., steering the model toward affirmative, unsafe completions, rather than degrading the model’s general representations. Consequently, utility cannot be used as a proxy for robustness: an exit can be simultaneously low-capability and highly jailbreakable (LLaMA2-7B, exit-15) or, conversely, an exit’s low ASR cannot be dismissed as merely reflecting a broken or degenerate sub-model, since its clean utility remains substantial.

5 Conclusion

In this paper, we explored the jailbreak vulnerability of depth-based dynamic inference in LLMs. We conducted this investigation through extensive experiments on two dynamic models (Balcony-LLaMA2-7B and Balcony-LLaMA3-8B) adapted with the Balcony method, using GCG as a representative jailbreak attack. Our findings are as follows: 1) shallow exits are consistently more robust to jailbreak attacks than deeper exits; 2) exit depth affects how easily an adversarial suffix can be optimized, with deeper exits reaching lower attack loss; and 3) adversarial suffixes affect utility uniformly across exit depths. These findings are associative and come from a single attack with fixed exit points; extending the study to routing policies, stronger attacks, and analyses of internal representations is left for future work.

References

- [1] Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. SmolLM-Corpus. <https://huggingface.co/datasets/HuggingFaceTB/SmolLM-corpus>, 2024.
- [2] Tim Beyer, Jonas Dornbusch, Jakob Steimle, Moritz Ladenburger, Leo Schwinn, and Stephan Günnemann. AdversariaLLM: A unified and modular toolbox for LLM robustness research. *arXiv preprint arXiv:2511.04316*, 2025.
- [3] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.
- [4] Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. *arXiv preprint arXiv:2406.10260*, 2024.
- [5] Ruisi Cai, Saurav Muralidharan, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. LLaMAFlex: Many-in-one LLMs via generalized pruning and weight sharing. In *The Thirteenth International Conference on Learning Representations*, 2025.
- [6] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, 2024.
- [7] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac’h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024.
- [8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- [9] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The LLaMA 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [10] Jonathan Hayase, Ema Borevkovic, Nicholas Carlini, Florian Tramèr, and Milad Nasr. Query-based adversarial prompt generation. *Advances in Neural Information Processing Systems*, 37:128260–128279, 2024.
- [11] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. DynaBERT: Dynamic BERT with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793, 2020.
- [12] Jerry Huang, Prasanna Parthasarathi, Mehdi Rezagholizadeh, and Sarath Chandar. Context-aware assistant selection for improved inference acceleration with large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 5817–5830, 2024.
- [13] Benyamin Jamialahmadi, Parsa Kavehzadeh, Mehdi Rezagholizadeh, Parsa Farinneya, Hossein Rajabzadeh, Aref Jafari, Boxing Chen, and Marzieh S Tahaei. Balcony: A lightweight approach to dynamic inference of generative language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 24864–24878, 2025.
- [14] Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, Prateek Jain, et al. MatFormer: Nested transformer for elastic inference. *arXiv preprint arXiv:2310.07707*, 2023.

- [15] Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, et al. HarmBench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*, 2024.
- [16] Alexander Panfilov, Peter Romov, Igor Shilov, Yves-Alexandre de Montjoye, Jonas Geiping, and Maksym Andriushchenko. Claudini: Autoresearch discovers state-of-the-art adversarial attack algorithms for LLMs. *arXiv preprint arXiv:2603.24511*, 2026.
- [17] Mahdi Sabbaghi, Paul Kassianik, George Pappas, Yaron Singer, Amin Karbasi, and Hamed Hassani. Adversarial reasoning at jailbreaking time. *arXiv preprint arXiv:2502.01633*, 2025.
- [18] Vinu Sankar Sadasivan, Shoumik Saha, Gaurang Sriramanan, Priyatham Kattakinda, Atoosa Chegini, and Soheil Feizi. Fast adversarial attacks on language models in one GPU minute. *arXiv preprint arXiv:2402.15570*, 2024.
- [19] Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1671–1685, 2024.
- [20] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [21] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does LLM safety training fail? *Advances in Neural Information Processing Systems*, 36:80079–80110, 2023.
- [22] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, 2020.
- [23] Zhi Xu, Jiaqi Li, Xiaotong Zhang, Hong Yu, and Han Liu. TAO-attack: Toward advanced optimization-based jailbreak attacks for large language models. *arXiv preprint arXiv:2603.03081*, 2026.
- [24] Sicheng Zhu, Ruiyi Zhang, Bang An, Gang Wu, Joe Barrow, Zichao Wang, Furong Huang, Ani Nenkova, and Tong Sun. AutoDAN: interpretable gradient-based adversarial attacks on large language models. *arXiv preprint arXiv:2310.15140*, 2023.
- [25] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.