LEVERAGING NEURAL NETWORK LAYERS AS AN ANNEALING PATH

Anonymous authors
Paper under double-blind review

ABSTRACT

Sampling and optimization with trained neural networks can be difficult even when their gradients are available: high-order interactions can obscure useful search directions. We introduce *layerwise annealing*, which uses the functions decoded from intermediate network layers as an annealing trajectory. The motivation is that earlier-layer functions can approximate smoothed versions of the final objective, with high-degree Fourier components attenuated. Their gradients can reveal promising directions that the final objective obscures, while progressively deeper layers restore the detailed structure needed to refine the solution. We exploit this by applying prediction heads at successive depths of a frozen network and carrying samples or optimization states through the resulting objectives, ending with the original function. Controlled Boolean and continuous experiments demonstrate the proposed mechanism: high-degree interactions and narrow density modes emerge progressively across layers. We demonstrate practical utility in LLM prompt optimization. On AdvBench Harmful Strings with Llama-2-7B-Chat, layerwise Greedy Coordinate Gradient achieves 70% target-string success, compared with the published GCG reference of 57%, with mean target cross-entropy of 0.195 versus 0.30.

1 INTRODUCTION

Already-trained neural networks are used not only to evaluate candidates, but also to guide their design. BindCraft optimizes amino-acid sequences through a pretrained AlphaFold2-Multimer network (Pacesa et al., 2025); related problems include prompt optimization for language models (Zou et al., 2023) and inference-time steering of diffusion models toward desired rewards (Kim et al., 2025). In these settings, the network remains fixed, and search must identify candidates that satisfy objectives defined by its predictions.

A central obstacle is that the final objective may provide little useful local guidance. High-magnitude, high-frequency variation can dominate the gradient, directing search toward nearby local optima. Synergistic interactions create a different failure: several changes may be beneficial together even when each is harmful or uninformative in isolation. The difficulty is therefore not only escaping local optima, but obtaining a useful direction for exploration.

Common approaches improve exploration through temperature schedules, proposal mechanisms, or step-size adaptation (Geyer, 1991; Neal, 2001; Grathwohl et al., 2021). These techniques can improve mobility, but temperature scaling alone preserves the gradient's direction: $\nabla(\beta f) = \beta \nabla f$. We instead seek more informative gradients within the trained network.

A trained network provides more than the gradient of its final output. Applying prediction heads to successive hidden layers reveals a sequence of intermediate functions whose predictions generally improve with depth (Belrose et al., 2023; Jiang et al., 2025). Our central hypothesis is that higher-order interactions are less fully represented at earlier layers, so earlier functions retain broad directional information that becomes obscured with depth.

We introduce *layerwise annealing*, which uses these intermediate functions as an annealing trajectory. We optimize using earlier decoded objectives, then carry the state through progressively deeper layers, ending with the original final-layer objective. **To our knowledge, this is the first use of intermediate-layer prediction functions as an annealing trajectory for sampling and optimization.**

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We examine the mechanism in controlled discrete and continuous experiments and demonstrate practical value in LLM jailbreak optimization.

2 Related Work

Layer-wise structure in neural networks. A large literature studies how representations and predictions evolve across network depth. Early probing work found that transformer layers progress from more local and syntactic information toward more semantic and contextual representations (Tenney et al., 2019; Jawahar et al., 2019). More recent work examines this progression directly in prediction space: the Tuned Lens shows that intermediate transformer representations support increasingly accurate next-token predictions (Belrose et al., 2023), while Jiang et al. (2025) show that intermediate representations become increasingly aligned with the final representation and can often determine the final prediction before the last layer. Sparse-autoencoder work similarly studies how interpretable features evolve across depth (Lieberum et al., 2024). These works use intermediate layers primarily for analysis, interpretation, or early exit; we instead use the functions represented at these layers for optimization and sampling.

Classical MCMC and tempering. Gradient-based MCMC methods, including MALA and Hamiltonian Monte Carlo, use local derivatives to construct informed proposals (Roberts & Tweedie, 1996; Neal, 2011). Parallel tempering, annealed importance sampling, and sequential Monte Carlo use intermediate distributions to improve exploration (Geyer, 1991; Neal, 2001; Del Moral et al., 2006). Temperature-based paths can facilitate barrier crossing, but rescale the same objective: $\nabla(\beta f) = \beta \nabla f$. They therefore do not make an uninformative gradient more informative.

Discrete sampling. Locally balanced proposals, Gibbs-with-Gradients, and discrete Langevin samplers use local target values or gradients to construct effective discrete moves (Zanella, 2020; Grathwohl et al., 2021; Zhang et al., 2022; Sun et al., 2023). These methods improve how local information is used, but generally still obtain that information from the final target function. When high-order interactions obscure useful local signal, improving the proposal mechanism does not itself reveal the underlying lower-order structure. Explicit smoothing methods instead construct a different target (Frey et al., 2024; Bach & Saremi, 2025). Closest to our motivation, trajectory annealing traverses predictors saved during training (Ram & Ram, 2026); our method instead traverses functions represented at different layers of a single trained network.

To our knowledge, no prior work has used the sequence of functions decoded from successive hidden layers of a trained neural network as an annealing trajectory for sampling or optimization. Prior layer-wise work studies what intermediate representations encode; prior sampling work constructs intermediate distributions through temperature, explicit smoothing, auxiliary variables, or separately learned energy functions. Our work connects these two directions: we treat network depth itself as a pre-existing sequence of progressively refined objectives and use that sequence for exploration and refinement.

3 Methods

3.1 Problem Setting

Test-time setting. We consider a trained neural network with L layers, where the final layer L defines a scalar objective $f_L : \mathcal{X} \to \mathbb{R}$; this includes, for example, an LLM, where $f_L(x)$ can be the likelihood of a complete sequence x. Our goal is to sample from the corresponding target distribution

$$\pi_L(x) \propto f_L(x), \tag{1}$$

optionally subject to constraints on x. We operate entirely at test time: the trained network is frozen, and we do not modify its training procedure or train a separate generative model.

Layerwise annealing. Let $h_\ell(x)$ denote the representation after layer ℓ. Applying a linear readout to each representation defines an intermediate function $f_\ell(x)$, with f_L equal to the final network function. Rather than sampling directly from π_L, we traverse these functions in order,

$$\pi_\ell(x) \propto f_\ell(x) \qquad \ell = 1, \ldots, L, \tag{2}$$

running a short sampling update at each layer and using the resulting samples to initialize the next. Early layers are therefore used for exploration and later layers progressively refine samples toward the final target.

Intuition. Network depth progressively builds the final function. In particular, higher-order interactions are less fully expressed at earlier layers and become increasingly represented with depth. The intermediate functions therefore act as naturally smoother surrogates of the final objective: early-layer gradients expose broad, lower-order structure, while deeper layers restore the high-order interactions needed for accurate refinement. Layerwise annealing exploits this coarse-to-fine progression as an annealing trajectory that is already present inside the trained network.

3.2 Why optimization is hard

We study discrete optimization in the common gradient-guided setting of Gibbs-with-Gradients (GWG) (Grathwohl et al., 2021). Although the input $x \in \{-1, +1\}^d$ is discrete, the neural network is differentiated with respect to its continuous input representation, and this gradient is used as a first-order approximation to determine which discrete coordinate to change next.

Any function that we desire to optimize over binary variables can be expressed as a sum of Boolean monomials. Using the $\{-1, +1\}$ representation, let $x \in \{-1, +1\}^d$. For every subset $S \subseteq [d]$, define the Boolean basis function

$$\chi_S(x) = \prod_{i \in S} x_i.$$

These functions form the Fourier–Walsh basis on the Boolean hypercube, so any function $f : \{-1, +1\}^d \to \mathbb{R}$ can be written uniquely as

$$f(x) = \sum_{S \subseteq [d]} \widehat{f}(S)\chi_S(x),$$

where $\widehat{f}(S)$ is the Fourier coefficient of the monomial χ_S, and $|S|$ is its degree. Low-degree terms describe the effects of individual variables or small groups of variables, while high-degree terms encode interactions among many variables.

In first-order optimization, we rely on the gradient to point us towards the global minima. However, high-degree terms present in the function can make the gradient uninformative in two important ways.

1. High-magnitude, high-degree terms create local optima.

Consider maximizing

$$\sum_{i=1}^{d} x_i + \lambda \prod_{i=1}^{d} x_i, \qquad \lambda > 1.$$

The first term provides a perfectly useful signal: increasing any x_i toward $+1$ moves us toward the global maximum $x = (+1, \ldots, +1)$. The second term, however, is a degree-d parity interaction. When $\lambda > 1$, it dominates every single-bit change. In fact, every configuration with positive parity becomes a local maximum, yielding 2^{d-1} local maxima even though only one is globally optimal. The low-degree terms still point directly toward the global solution, but their signal is overwhelmed by the high-degree interaction.

2. Synergistic interactions can remove local signal entirely.

A different difficulty arises when variables are only useful jointly. Consider the Boolean function

$$f(x) = \mathbf{1}\{x = (+1, \ldots, +1)\}, \qquad x \in \{-1, +1\}^d. \tag{3}$$

The reward is nonzero only when *all* d variables take the correct value. Thus, away from the solution, changing a single variable typically produces no change in the objective: the final function provides no local indication of which bits should be flipped.

However, the same function can be expanded in the Boolean Fourier basis as

$$f(x) = 2^{-d} \prod_{i=1}^{d} (1 + x_i) = 2^{-d} \sum_{S \subseteq [d]} \prod_{i \in S} x_i. \tag{4}$$

Its low-degree components already contain useful directional information. For example, retaining only the constant and linear terms gives

$$f_{\leq 1}(x) = 2^{-d}\left(1 + \sum_{i=1}^{d} x_i\right), \tag{5}$$

which increases whenever any coordinate is changed from -1 to $+1$. Thus, although the full synergistic objective is locally flat almost everywhere, its low-degree approximation provides a smooth signal toward the optimum.

3.3 Optimal Annealing Trajectory for Optimization

Intuitively, a solution to such scenarios would be to optimize a smoothed version of the original function, with dampened high-degree components. Then, this solution could warm-start the optimization of the original function, allowing it to be refined.

More precisely, the annealing trajectory is the discrete analogue of Gaussian smoothing. Applying Bernoulli noise to a Boolean function damps a monomial of degree k by ρ^k, where $\rho \in [0, 1]$ controls the amount of smoothing (O'Donnell, 2014). Varying ρ from 0 to 1 therefore defines an annealing trajectory from a strongly smoothed, easy-to-explore distribution to the original target distribution, which can be used to sample from the original target (Bach & Saremi, 2025).

As long as the gradient remains confounded by high-degree Boolean monomials, **the gradient's direction points toward nearby local minima, preventing exploration**. Tempering-based techniques cannot rectify this problem. While they allow one to escape from local minima, they rescale the same underlying objective and therefore do not make the gradient itself more informative.

Thus, we need the gradient of a different function: one that resembles the original objective but with its high-frequency components damped. The challenge is that a neural network conventionally exposes only its final function. Our goal is therefore to identify more informative surrogate gradients already present within the network.

3.4 Every layer is a function

A neural network does not produce a useful function only at its final layer: its intermediate representations can also be mapped to predictions, revealing how the learned function evolves with depth. Across residual networks and transformers, prior work points to three recurring patterns. First, representations tend to move progressively toward their final state, often along approximately direct trajectories. Second, predictions decoded from these intermediate representations generally improve with depth. Third, the model's eventual prediction is often already determined several layers before the end. Together, these results motivate viewing each layer as defining an intermediate function along a trajectory from the input representation to the final predictor.

1. Residual networks favor approximately straight paths toward the final representation.

Theory. Let $h_\ell(x)$ denote the representation of input x after residual block ℓ. For the two-layer residual blocks analyzed by Gai & Zhang (2021), their Eq. (14) gives the following bound, written here in terms of the hidden representations:

$$\sum_{\ell=0}^{L-1} \mathbb{E}_x\left[\|h_{\ell+1}(x) - h_\ell(x)\|_2^2\right] \leq C \sum_{\ell=0}^{L-1}\left(\left\|W_\ell^{(1)}\right\|_F^2 + \left\|W_\ell^{(2)}\right\|_F^2\right). \tag{6}$$

The left-hand side adds the squared movement between consecutive layers, averaged over inputs. The right-hand side is proportional to the weight penalty, where $W_\ell^{(1)}$ and $W_\ell^{(2)}$ are the block's two weight matrices and C is a constant under their bounded-state and bounded-weight assumptions. Weight regularization therefore penalizes an upper bound on representation movement (Gai & Zhang, 2021).

For fixed starting and ending representations, this movement is minimized by taking equal steps along a straight line, whenever that path is representable:

$$h_\ell(x) = \left(1 - \frac{\ell}{L}\right) h_0(x) + \frac{\ell}{L} h_L(x). \tag{7}$$

Each layer moves in the same direction by the same amount, rather than taking a detour through representation space. This is the ideal minimum-motion path, not a guarantee that every trained network follows it exactly (Gai & Zhang, 2021). For trajectories of the form in Eq. 7, Theorem 2.1 of Jiang et al. (2025) establishes increasing cosine similarity to the final representation.

Empirics. Li & Papyan (2023) observe approximately straight-line trajectories in fully connected and convolutional ResNet-34 models trained on MNIST, FashionMNIST, and CIFAR-10. For a given input, the hidden vectors from successive layers lie approximately along a straight line, with roughly equal spacing between consecutive layers (Appendix C.1, Figs. 8–16). Related evidence appears in transformers: in DeiT-S, a small vision transformer, intermediate representations become increasingly aligned with the final representation on CIFAR-10 and ImageNet-1K (Jiang et al., 2025, Fig. 2).

2. Prediction heads reveal increasingly accurate predictions across depth.

Theory. For classifiers, Theorem 2.2 of Jiang et al. (2025) connects increasing alignment with the final representation to increasing probability of the correct class, using the same final classifier at every layer. The result assumes terminal Neural Collapse: within-class features coincide, class means are symmetrically separated, and the classifier aligns with them. Its proof additionally assumes equal norms for consecutive representations. The neural collapse assumption in the proof applies only to classifiers.

Empirics. In the DeiT-S experiments, applying the final classifier at successive layers produces increasing classification accuracy (Jiang et al., 2025, Fig. 2). Jiang et al. further improve intermediate predictions through *aligned training*, which adds prediction losses at each layer through a shared head. For a 12-layer GPT-2 fine-tuned on WikiText-103, prediction accuracy improves linearly across layers.

Complementary evidence comes from the Tuned Lens, which keeps the language model frozen and fits an affine translator before using the last-layer prediction head at each layer (Belrose et al., 2023). On Pythia models up to 12B parameters and GPT-NeoX-20B, held-out next-token prediction loss decreases across depth, showing that deeper representations support progressively better predictions (Belrose et al., 2023, Fig. 5).

3. The most likely prediction often becomes fixed before the final layer.

Phenomenon. For a fixed input prefix, the final model's most likely next token can already become the top prediction at an intermediate layer. The *saturation layer* is the earliest layer from which that same token remains the top prediction at every subsequent layer (Jiang et al., 2025, Sec. 2.2). Later layers can continue changing its probability without changing which token is selected.

Empirics. Jiang et al. (2025) observe different saturation depths across examples in DeiT-S (Fig. 3), and illustrate the same behavior in LLaMA-3.2-3B and LLaMA-3.1-8B (Appendix C.1, Figs. 19–20). On Pythia-12B, prediction depth also correlates positively with how late an example is learned during training, connecting the required depth to example difficulty (Belrose et al., 2023, Table 2). Furthermore, this phenomenon has been exploited by early-exit methods. For example, LayerSkip trains a shared language-model head to support earlier exits and uses those predictions to accelerate inference (Elhoushi et al., 2024).

3.5 Sampling Method

The previous section shows that applying a prediction head at successive layers yields an ordered sequence of functions that progressively approaches the final network function. This gives us a natural family of intermediate objectives. The key question is whether these objectives also have the property we need for optimization: do earlier layers suppress the higher-order interactions that make the final objective locally difficult?

Our central hypothesis is that they do. Higher-order interactions are less fully represented at earlier layers and emerge progressively with depth.

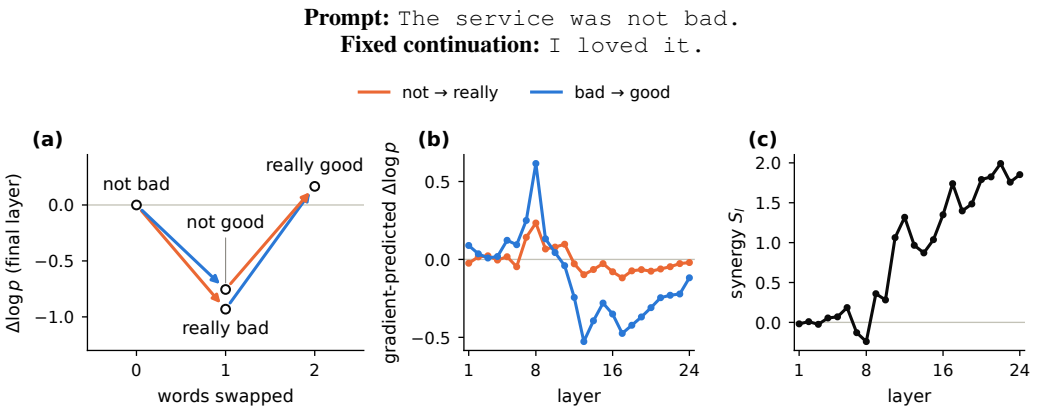


Figure 1: An intermediate-layer gradient favors edits discouraged by the final layer. We edit only `not→really` and `bad→good` in Pythia-1.4B, scoring the fixed continuation shown above. (a) Changing both words increases the continuation's final-layer log-likelihood, while either change alone decreases it; values are relative to the original prompt. (b) Both substitutions receive positive gradient-predicted gains around layer 8, but negative gains at the final layer. (c) The interaction between the edits strengthens with depth.

In the Boolean Fourier representation,

$$f_\ell(x) = \sum_{S \subseteq [d]} \widehat{f_\ell}(S) \chi_S(x), \tag{8}$$

this hypothesis means that earlier-layer functions place relatively less weight on high-degree components. Because high-degree terms are weaker at earlier layers, these functions are *smoother* and can provide more informative gradients for exploration. Deeper layers progressively restore the higher-order interactions needed to recover the final objective. This is precisely the structure we sought in Section 3.3.

An intermediate layer reveals useful edit directions. We illustrate this in Pythia-1.4B by editing the prompt `The service was not bad.` to increase the log-likelihood of the fixed continuation `I loved it.` Replacing `not bad` with `really good` improves the final-layer objective, but either substitution alone makes it worse (Fig. 1a). The final-layer gradient also discourages both substitutions. Around layer 8, however, both substitutions receive positive gradient-predicted gains: the intermediate-layer objective points toward the desired joint edit even though the final-layer objective points away from it (Fig. 1b). Panel (c) shows that the two edits are nearly additive in the early layers, but their pairwise synergy rises sharply in later layers. As this interaction forms, evaluating either edit in isolation becomes increasingly misleading about the value of making both edits together. Across 64 compositional examples, mean pairwise synergy likewise increases with depth (Fig. 2). Further details are provided in Appendix A.2.

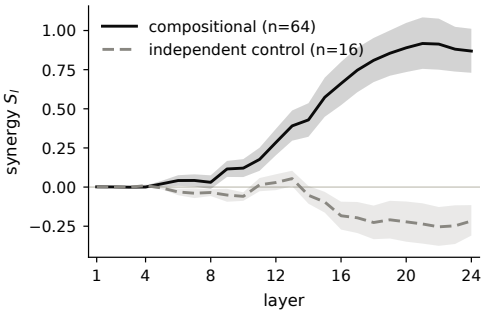
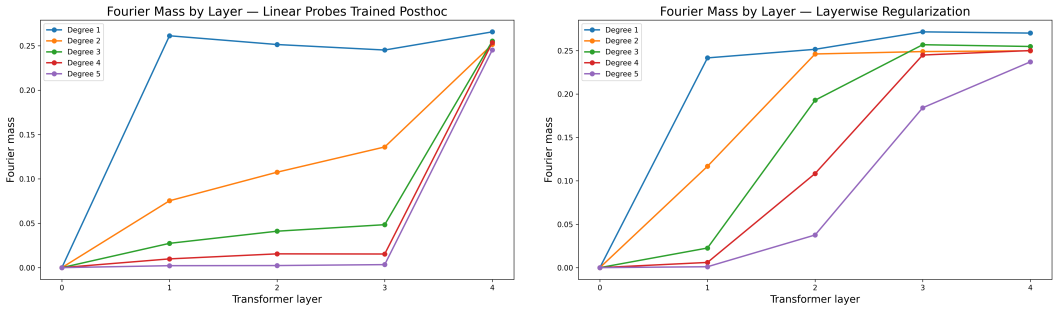


Figure 2: Compositional interactions strengthen with depth. Mean pairwise synergy for 64 compositional examples and 16 independent-clause controls in Pythia-1.4B. In the controls, the two editable words occur in separate clauses, so their effects are expected to combine approximately additively. Further details are shown in Appendix A.2.

Layerwise annealing. We use this progression by running sampling or optimization updates on functions f_ℓ at increasing depths, carrying the current state forward between layers and ending with the original objective f_L. Earlier functions supply exploratory directions; later functions refine the result using the increasingly complete objective.



(a) Linear probes trained post hoc.

(b) Layerwise regularization.

Figure 3: Fourier mass by transformer layer for the staircase Fourier target. Higher-degree monomials are more strongly damped in earlier layers and emerge progressively with depth.

4 Results

4.1 Toy Experiments

Experiment 1: Higher-order Fourier structure emerges in later layers. We first test whether earlier transformer layers preferentially represent lower-order components of a target function. We train a 4-layer transformer on a $d = 20$ Boolean function constructed as a sparse Fourier staircase,

$$f(x) = \sum_{k=1}^{5} \sum_{S \in \mathcal{S}_k} a_S \prod_{i \in S} z_i, \qquad z_i = x_i x_i^\star, \tag{9}$$

where each degree-k monomial extends a degree-$(k-1)$ parent by one coordinate, and the total target Fourier energy is approximately balanced across degrees 1–5. We then compute the exact Walsh–Fourier spectrum of the function represented at each transformer layer over all 2^{20} Boolean inputs. Further details are found in A.1

Figure 3 shows the Fourier mass at each degree as a function of depth. In early layers, higher-degree components are substantially attenuated relative to lower-degree components; their mass increases progressively with depth as the model approaches the final target function. This pattern appears both when linear probes are fit post hoc to a normally trained transformer and when the model is trained with layerwise regularization that maximizes accuracy at each layer.

Experiment 2: Layerwise functions progressively resolve fine-scale modes. We next consider a continuous multimodal density,

$$p(x) \propto \sum_{k=1}^{20} \exp\left(-\frac{\|x - \mu_k\|_2^2}{2\sigma^2}\right), \qquad x \in \mathbb{R}^2, \tag{10}$$

where $\sigma = 0.25$ and the component means $\{\mu_k\}_{k=1}^{20}$ are sampled independently and uniformly from $[-20, 20]^2$ using a fixed random seed. We train a transformer to regress this density using a shared scalar head, with layerwise supervision during training.

Figure 4 visualizes the function represented across depth. Early layers capture only broad, spatially extended structure linking the regions containing probability mass. As depth increases, this coarse structure is progressively refined into narrower modes centered on the individual mixture components. By approximately layer 8, the function has largely converged and subsequent layers make little visible change. This progression is precisely the structure desirable for annealed sampling.

4.2 LLM Experiment

Jailbreaking as constrained discrete optimization. One of the most prominent real-world applications of constrained optimization in language models is *jailbreaking*: searching for an input that causes an aligned language model to place high probability on a continuation that it would ordinarily suppress.

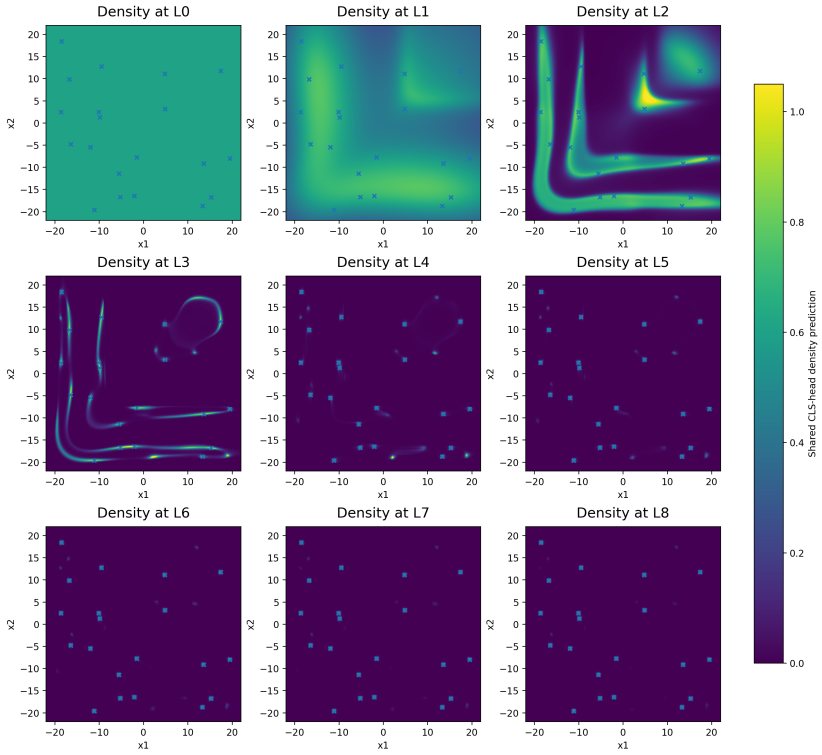


Figure 4: Layerwise density functions for a sparse 20-component Gaussian mixture. Earlier transformer layers represent broad, spatially extended structure, while deeper layers progressively resolve the narrow modes of the target density. The sequence therefore provides a natural coarse-to-fine trajectory for sampling. The crosses mark the 20 component means.

We evaluate layerwise annealing on the AdvBench Harmful Strings task (Zou et al., 2023), where the goal is to optimize an adversarial suffix so that Llama-2-7B-Chat assigns high likelihood to a specified target continuation. We use Greedy Coordinate Gradient (GCG), which proposes token substitutions using gradients of the target-continuation loss and then evaluates the resulting discrete candidates with the model (Zou et al., 2023).

Layerwise GCG. We keep the GCG search procedure fixed and change only the function whose gradient is used to generate proposals. At layer ℓ, we decode the residual representation h_ℓ with a pretrained Tuned Lens head (Belrose et al., 2023),

$$p_{\theta,\ell}(\cdot \mid x) = \mathrm{softmax}(g_\ell(h_\ell(x))) \,, \tag{11}$$

and compute the target-continuation loss using this intermediate distribution. Optimization then progresses from earlier to later layers, ending with the model's standard final-layer head. Full GCG hyperparameters and the layer schedule are given in Appendix A.3.

AdvBench Harmful Strings. We evaluate on the first 100 Harmful Strings targets using Llama-2-7B-Chat. This benchmark is particularly clean for studying optimization because success requires producing a specified target string, so both exact target-string attack success rate and target cross-entropy directly measure optimization quality.

Main comparison. Layerwise gradients substantially improve GCG under the same discrete search rule. Using all transformer layers increases target-string attack success rate (ASR) from the published GCG result of 57% to 70%, while reducing mean target cross-entropy from 0.30 to 0.195 (Table 1). A coarser four-depth schedule $(8 \to 16 \to 24 \to 32)$ reaches 67% ASR with mean CE 0.292. Thus, we offer a plug-and-play method that dramatically improves GCG's accuracy.

Table 1: Results on the AdvBench Harmful Strings task with Llama-2-7B-Chat across 100 targets. Higher target-string attack success rate (ASR) and lower target cross-entropy are better. The published GCG result is included as the historical reference from Zou et al. (2023); our layerwise variants use the same discrete GCG search rule while changing only the objective depth.

Method	Target-string ASR ↑	Mean target CE ↓
GCG (Zou et al., 2023)	57%	0.30
Layerwise GCG (4 depths)	67%	0.292
Layerwise GCG (all layers)	**70%**	**0.195**

Later GCG variants introduce additional attack-specific machinery and can reach higher success rates. For example, I-GCG combines alternative target templates, multi-coordinate updates, and specialized initialization, while MAGIC modifies the gradient-based update rule and reports 74% ASR on Llama-2 (Jia et al., 2024; Li et al., 2025). These results are not directly comparable because the attack protocols differ, but they provide useful context: our gain is obtained without introducing jailbreak-specific heuristics and *is complementary to such improvements.*

5 Conclusion

We introduced layerwise annealing, which uses the functions decoded from successive layers of a trained network as a trajectory for sampling and optimization. The central insight is that an intermediate function at an earlier layer has damped high-degree Fourier components, allowing it to approximate a smoothed version of the original function. This means that functions across layers can act as a built-in annealing trajectory, where functions associated with early layers are conducive to exploration whereas later layers allow for refinement. We illustrate this mechanism through controlled experiments, and show our method's utility on a real-world LLM constrained optimization task.

AI use statement

We used generative AI tools to assist with several aspects of this work. For writing, the authors provided outlines and technical content, and used AI to help draft, revise, and polish portions of the manuscript through an iterative back-and-forth process. AI was also used for retrieval and discovery of related work; suggested references were subsequently checked and verified by the authors.

For research execution, AI was used to assist with writing experimental code, which the authors reviewed, tested, and verified for correctness. AI was also used to generate a synthetic dataset for the LLM experiment; the authors subsequently reviewed the generated examples before use. In addition, AI was used to help draft sections of the paper based on author-provided structure, arguments, and results.

We did not use AI to prove mathematical claims; this was not applicable to the paper. All AI-assisted content was reviewed and edited by the authors. The authors verified cited work, checked experimental code and results, reviewed the synthetic data used in the LLM experiment, and take full responsibility for the final manuscript and all claims, analyses, and artifacts it contains.

Ethics statement

This work studies optimization and sampling methods for fixed, pretrained neural networks. Our experiments include an LLM jailbreaking benchmark, which we use as a constrained discrete optimization task rather than as an end goal in itself. We do not believe this experiment introduces substantial additional misuse risk beyond existing publicly documented methods and benchmarks, and it is not the primary focus of the paper.

More broadly, the proposed method is general-purpose and may be applied in domains with different downstream impacts. As with other optimization and sampling techniques, appropriate safeguards and domain-specific evaluation should be used when applying it to safety-sensitive systems.

Reproducibility statement

Code for all experiments and figures is provided in the attached zip file.

References

Francis Bach and Saeed Saremi. Sampling binary data by denoising through score functions. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 2221–2240. PMLR, 2025.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International conference on machine learning*, pp. 2397–2430. PMLR, 2023.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society: Series B*, 68(3):411–436, 2006. doi: 10.1111/j.1467-9868.2006.00553.x.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. 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)*, pp. 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681.

Nathan C. Frey, Daniel Berenberg, Karina Zadorozhny, Joseph Kleinhenz, Julien Lafrance-Vanasse, Isidro Hotzel, Yan Wu, Stephen Ra, Richard Bonneau, Kyunghyun Cho, Andreas Loukas, Vladimir Gligorijevic, and Saeed Saremi. Protein discovery with discrete walk-jump sampling. In *The Twelfth International Conference on Learning Representations*, 2024.

Kuo Gai and Shihua Zhang. A mathematical principle of deep learning: Learn the geodesic curve in the wasserstein space. *arXiv preprint arXiv:2102.09235*, 2021.

Charles J. Geyer. Markov chain monte carlo maximum likelihood. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, pp. 156–163, 1991.

Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris Maddison. Oops i took a gradient: Scalable sampling for discrete distributions. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 3831–3841. PMLR, 2021.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3651–3657. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1356.

Xiaojun Jia, Tianyu Pang, Chao Du, Yihao Huang, Jindong Gu, Yang Liu, Xiaochun Cao, and Min Lin. Improved techniques for optimization-based jailbreaking on large language models. *arXiv preprint arXiv:2405.21018*, 2024.

Jiachen Jiang, Jinxin Zhou, and Zhihui Zhu. Tracing representation progression: Analyzing and enhancing layer-wise similarity. In *The Thirteenth International Conference on Learning Representations*, 2025.

Sunwoo Kim, Minkyu Kim, and Dongmin Park. Test-time alignment of diffusion models without reward over-optimization. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jiahui Li, Yongchang Hao, Haoyu Xu, Xing Wang, and Yu Hong. Exploiting the index gradients for optimization-based jailbreaking on large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 4535–4547. Association for Computational Linguistics, 2025.

Jianing Li and Vardan Papyan. Residual alignment: Uncovering the mechanisms of residual networks. In *Advances in Neural Information Processing Systems*, volume 36, pp. 57660–57712, 2023. doi: 10.52202/075280-2515.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, János Krámár, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. *arXiv preprint arXiv:2408.05147*, 2024.

Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

Radford M. Neal. MCMC using hamiltonian dynamics. In Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng (eds.), *Handbook of Markov Chain Monte Carlo*, pp. 113–162. Chapman and Hall/CRC, 2011.

Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. doi: 10.1017/CBO9781139814782.

Martin Pacesa, Lennart Nickel, Christian Schellhaas, Joseph Schmidt, Ekaterina Pyatova, Stephen Buckley, Sandrine Georgeon, Casper A. Goverde, et al. One-shot design of functional protein binders with bindcraft. *Nature*, 646:483–492, 2025. doi: 10.1038/s41586-025-09429-6.

Soumya Ram and Akhila Ram. From predictors to samplers via the training trajectory. In *The Fourteenth International Conference on Learning Representations*, 2026.

Gareth O. Roberts and Richard L. Tweedie. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363, 1996.

Haoran Sun, Hanjun Dai, Bo Dai, Haomin Zhou, and Dale Schuurmans. Discrete langevin samplers via wasserstein gradient flow. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pp. 6290–6313. PMLR, 2023.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1452.

Giacomo Zanella. Informed proposals for local MCMC in discrete spaces. *Journal of the American Statistical Association*, 115(530):852–865, 2020. doi: 10.1080/01621459.2019.1585255.

Ruqi Zhang, Xingchao Liu, and Qiang Liu. A langevin-like sampler for discrete distributions. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 26375–26396. PMLR, 2022.

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.

A Appendix

A.1 Staircase Fourier Toy Experiment

We construct a Boolean regression problem designed to isolate how interactions of increasing order are represented across transformer depth. The input is $x \in \{-1, +1\}^{20}$, and we first sample a fixed planted maximizer $x^\star \in \{-1, +1\}^{20}$. We work in planted coordinates

$$z_i = x_i x_i^\star, \tag{12}$$

so that $z_i = +1$ for every coordinate when $x = x^\star$.

The target function is a sparse Fourier polynomial containing interactions of degrees one through five:

$$f(x) = \sum_{k=1}^{5} \sum_{S \in \mathcal{S}_k} a_S \prod_{i \in S} z_i. \tag{13}$$

All coefficients a_S are positive, so the planted point $x^\star$ simultaneously maximizes every monomial in the target.

Staircase support. The Fourier support is constructed hierarchically. At degree one, $\mathcal{S}_1$ contains all singleton coordinates. For every subsequent degree $k \in \{2, \dots, 5\}$, each monomial is generated by selecting a degree-$(k-1)$ parent $S' \in \mathcal{S}_{k-1}$ and adding one previously unused coordinate,

$$S = S' \cup \{j\}. \tag{14}$$

Thus every higher-order interaction has an explicit chain of lower-order parents. We use 60 monomials at each degree $k \geq 2$.

To prevent the experiment from being explained simply by larger target energy at low degrees, coefficients are scaled so that the total squared Fourier coefficient mass is approximately matched across degrees. Specifically, for degree k with $|\mathcal{S}_k|$ terms, coefficients are initialized at scale

$$a_S \approx \sqrt{\frac{E}{|\mathcal{S}_k|}}, \tag{15}$$

with $E = 0.25$, followed by a small independent multiplicative perturbation uniformly sampled from $[0.85, 1.15]$. Consequently, $\sum_{S \in \mathcal{S}_k} a_S^2$ is approximately constant across $k = 1, \dots, 5$.

Training data. Training examples are sampled independently and uniformly from the Boolean hypercube. Equivalently, every coordinate is sampled as

$$x_i \sim \mathrm{Unif}\{-1, +1\}. \tag{16}$$

For each sampled x, the regression target is computed exactly using Eq. equation 13. No noisy labels or sampled approximations to the target function are used.

We train a four-block transformer with hidden dimension 128, four attention heads, and feed-forward dimension 256. Training uses AdamW for 20,000 steps with batch size 2,048, initial learning rate 3×10^{-4}, cosine learning rate decay to 3×10^{-5}, and weight decay 10^{-2}.

Layerwise functions. We evaluate two ways of defining a scalar function at intermediate layers. In the *post-hoc probe* setting, the transformer is trained normally using only the final prediction. We then freeze the network and fit an independent linear ridge probe at each layer using 65,536 uniformly sampled Boolean inputs.

In the *layerwise-regularized* setting, the same scalar prediction head is applied at every layer. Training alternates between the final-layer loss and a weighted sum of losses across all layer exits. The layer weights increase with depth according to

$$\lambda_\ell = \frac{2(\ell + 1)}{L(L + 1)}, \tag{17}$$

where L is the number of exits.

Fourier evaluation. Because $d = 20$, the complete Boolean cube contains only 2^{20} states, so we evaluate every layerwise function exactly on all inputs. We then compute its Walsh–Fourier coefficients using the Fast Walsh–Hadamard Transform. For a layerwise function f_ℓ, the Fourier mass at degree k is

$$M_{\ell,k} = \sum_{S:\,|S|=k} \widehat{f}_\ell(S)^2. \tag{18}$$

The plots in the main text report $M_{\ell,k}$ for degrees one through five without normalization across degrees or layers.

This construction makes the interpretation particularly direct: the target contains approximately comparable Fourier energy at each degree, while every higher-degree feature is explicitly built on lower-degree parents. The observed suppression of high-degree Fourier mass in earlier transformer layers therefore reflects a coarse-to-fine organization of the learned function rather than an imbalance in the target spectrum.

A.2 Two-Token Synergy Experiment

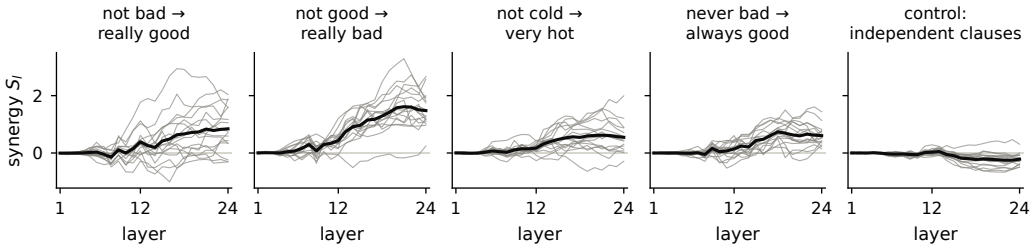


Figure 5: Synergy for every sentence, by family. Pairwise synergy S_ℓ across layers for each sentence (gray) and the family mean (black), with 16 sentences per family. Synergy increases with depth in all four compositional families, with final-layer means of $+0.84$, $+1.48$, $+0.55$, and $+0.61$, respectively, and is positive for 60 of 64 sentences. In contrast, the independent-clause control has a final-layer mean of -0.22 and positive synergy for only 1 of 16 sentences.

Figure 1 in the main text illustrates the mechanism on a representative two-token edit: the final-layer objective creates a local barrier, while an intermediate-layer gradient still favors the desired edits. Here we provide the full experimental setup and show that the same depth-dependent emergence of synergy holds across all sentence families. Figure 5 reports the layerwise synergy for every individual sentence together with the corresponding family mean.

Model and readout. We use Pythia-1.4B, which contains 24 transformer layers (Biderman et al., 2023). For each intermediate layer $\ell \in \{1, \ldots, 23\}$, we apply the pretrained Tuned Lens (Belrose et al., 2023): a learned affine map from the layer-ℓ residual stream into the final-layer representation space, followed by the model's final layer normalization and output head. At layer 24, we use the model's native output. All computations use bfloat16 on a single A100 GPU.

Layerwise score. Each example consists of a prompt x and a fixed continuation $y_{1:T}$. For example,

$$x = \texttt{``The service was not bad.''}, \qquad y = \texttt{``I loved it.''}.$$

The prompt and continuation are processed together in a single forward pass. At layer ℓ, we define

$$f_\ell(x) = \frac{1}{T} \sum_{t=1}^{T} \log p_\ell \left(y_t \mid x, y_{<t} \right), \tag{19}$$

where p_ℓ is the next-token distribution obtained from the layer-ℓ readout. Thus, $f_\ell(x)$ measures how strongly the layer-ℓ representation supports the fixed continuation.

Two-token edits. Each prompt contains two editable words, each paired with one single-token replacement. For example,

$$\texttt{not} \to \texttt{really}, \qquad \texttt{bad} \to \texttt{good}.$$

Each editable word appears exactly once in the prompt. This produces four versions of each example: the original prompt x_{00}, the two single edits x_{10} and x_{01}, and the joint edit x_{11}.

Measurements. At every layer, we measure three quantities.

Actual effect. For each edit, we measure its change relative to the original prompt:

$$f_\ell(x_{10}) - f_\ell(x_{00}), \qquad f_\ell(x_{01}) - f_\ell(x_{00}), \qquad f_\ell(x_{11}) - f_\ell(x_{00}). \tag{20}$$

Gradient-predicted effect. For a token replacement at position i, we compute

$$\nabla_{e_i} f_\ell(x_{00})^\top \left(E_{\text{new}} - E_{\text{old}}\right), \tag{21}$$

where e_i is the input embedding at position i and E is the token embedding matrix. This is the first-order score used by gradient-guided discrete optimization methods such as HotFlip and GCG to rank candidate replacements. We additionally record the rank of the desired replacement among all approximately $50{,}000$ vocabulary tokens.

Pairwise synergy. We measure the interaction between the two edits as

$$S_\ell = f_\ell(x_{11}) - f_\ell(x_{10}) - f_\ell(x_{01}) + f_\ell(x_{00}). \tag{22}$$

When $S_\ell = 0$, the effects of the two edits are additive. Positive S_ℓ indicates that the joint edit is more favorable than the sum of its two individual effects.

Sentence families. We use four compositional edit families:

$$\begin{aligned} \texttt{not bad} &\to \texttt{really good}, \\ \texttt{not good} &\to \texttt{really bad}, \\ \texttt{not cold} &\to \texttt{very hot}, \\ \texttt{never bad} &\to \texttt{always good}. \end{aligned}$$

Each family contains 16 sentences. In these examples, the original and joint edit are compatible with the continuation, whereas either single edit creates a semantic mismatch. We vary the noun, editable words, and continuation within each family. For example, ``The water was not cold. I burned my hand.''

As a control, we use 16 sentences in which the two editable words occur in independent clauses. For example, in ``The food was bad and the staff was rude.'' we apply the edits bad→good and rude→kind, with continuation ``I loved it.''. Because the two edits affect independent clauses, their effects should combine approximately additively, and hence their synergy should remain near zero.

All sentences were fixed before running the experiment, and no examples were excluded.

Code and data. The released code contains the complete sentence lists, the layerwise measurement script, the raw per-layer outputs used to produce all figures, and the plotting scripts.

A.3 LLM Experimental Details

Benchmark and model. We evaluate on the Harmful Strings task from AdvBench (Zou et al., 2023) using Llama-2-7B-Chat. We use the first 100 target strings from the benchmark. Unlike behavioral jailbreak benchmarks, Harmful Strings requires the model to produce a specified target continuation, which allows optimization quality to be measured directly using target-string attack success rate and target cross-entropy.

GCG setup. We follow the released individual-attack setup of Zou et al. (2023). The optimized adversarial suffix contains 20 tokens. At each update, GCG uses the gradient of the target-continuation loss to rank candidate token substitutions, considers the top $k = 256$ gradient candidates, and evaluates a discrete candidate batch of size 512. Optimization runs for 1000 updates. For all of our runs, we fix the random seed to 0 for reproducibility.

Given a fixed prompt q, optimizable suffix $x = (x_1, \ldots, x_m)$, and target continuation $y = (y_1, \ldots, y_T)$, the final-model GCG objective is

$$\mathcal{L}(x) = -\sum_{t=1}^{T} \log p_\theta \left(y_t \mid q \oplus x \oplus y_{<t}\right). \tag{23}$$

The gradient is taken with respect to the one-hot representation of the suffix tokens and is used to identify promising discrete token substitutions. The resulting candidates are then evaluated using the model, and the lowest-loss candidate is retained.

Layerwise objective. For layer ℓ, let h_ℓ denote the residual-stream representation. We decode h_ℓ using the pretrained Tuned Lens translator g_ℓ (Belrose et al., 2023),

$$p_{\theta,\ell}(\cdot \mid x) = \mathrm{softmax}\left(g_\ell(h_\ell(x))\right). \tag{24}$$

The layerwise loss is obtained by replacing the final-model distribution in the GCG objective with $p_{\theta,\ell}$. At the final transformer layer, we use the model's native language-model head rather than a Tuned Lens translator.

Layer schedules. For the all-layer schedule, the 1000 GCG updates are distributed approximately uniformly over the 32 transformer layers. Optimization therefore progresses through layers $1, 2, \ldots, 32$, with the current adversarial suffix carried forward between successive depths.

For the coarse schedule, we use four depths,

$$8 \to 16 \to 24 \to 32, \tag{25}$$

again dividing the 1000 optimization updates approximately uniformly across the four stages.

Controlled comparison. Across the baseline and layerwise variants, we keep the discrete GCG search procedure fixed. In particular, the initialization, suffix length, candidate batch size, top-k candidate selection, optimization budget, discrete candidate evaluation, and final evaluation procedure are unchanged. The only difference is the depth at which the gradient-producing objective is evaluated.

Evaluation metrics. We report two metrics. *Target-string attack success rate (ASR)* is the fraction of targets for which the optimized suffix causes the model to generate the specified target continuation. *Mean target cross-entropy* measures the final target-continuation loss after optimization; lower values indicate that the optimized suffix places greater probability on the desired continuation.