FAST ADVERSARIAL ATTACKS WITH GRADIENT PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Generating adversarial examples at scale is a core primitive for robustness evaluation, adversarial training, and red-teaming, yet even "fast" attacks such as FGSM remain throughput-limited by the cost of a backward pass. We introduce a family of attacks that eliminates the backward pass by predicting the input gradient from forward-pass hidden states via a lightweight linear regression. Theoretically, we derive exact affine conditional gradient means, showing optimality in the (idealized) NTK regime. Empirically, our methods work when applied to practical finite-width models; we recover much of FGSM's attack performance while using only a small fraction of the time, corresponding to a 532% increase in throughput. These results suggest gradient prediction as a simple and general route to significantly faster adversarial generation under realistic wall-clock constraints.

1 INTRODUCTION

Adversarial examples expose small, structured perturbations that can reliably change a model's prediction (Szegedy et al., 2014). Beyond their diagnostic value, adversarial inputs are a practical tool for robustness evaluation and for training procedures that explicitly optimize against worst-case perturbations (Goodfellow et al., 2015; Madry et al., 2018). A recurring operational obstacle is *throughput*: modern robustness pipelines are often constrained by wall-clock time and accelerator availability, so the number of adversarial examples produced per unit time can matter as much as attack success rate.

Even "one-step" attacks inherit a fundamental bottleneck. FGSM uses a single update but still requires an input gradient of a chosen attack score (Goodfellow et al., 2015). For large networks (and especially large language models), this typically means an additional backward pass per example, which can dominate the cost of adversarial data generation.

We address this limitation by eliminating the backward pass. Our key idea is to *predict* the input gradient from quantities already produced during the forward pass: a hidden state (representation) vector. Concretely, we fit a lightweight linear regressor that maps hidden states to input gradients and then perform an FGSM-style update using the predicted gradient. This design is motivated by a kernel/NTK view of neural networks (Jacot et al., 2018; Lee et al., 2018; Matthews et al., 2018; Yang, 2019), and is closely related in spirit to synthetic gradient ideas (Jaderberg et al., 2017; Czarnecki et al., 2017) (here applied to *input* gradients rather than activation gradients).

Contributions. We propose a family of methods that replaces attack-time backward passes with a trained affine gradient predictor. Our objective is successful attacks per unit of generation time, with ASR as a complementary metric. For FGSM, successful-attack throughput increases by 532% at the cost of a drop in ASR (Table 1). Therefore GPR occupies part of the ASR-throughput Pareto frontier (Figure 1).

2 PRELIMINARIES

Adversarial attacks. We consider a fixed trained network f_θ. Let $x \in \mathbb{R}^d$ denote the (continuous) input we perturb; for language models, x can be understood as a flattened input embedding tensor. Let $z_\theta(x) \in \mathbb{R}^C$ denote the model's logits on x, and let $h_\theta(x) \in \mathbb{R}^{d_h}$ denote the hidden state.

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.

Given a target output t, define a scalar score

$$S_\theta(x, t) \in \mathbb{R},$$

which measures support for t. A logit-based classification score is $S_\theta(x, t) = z_{\theta,t}(x)$. The score is an implementation detail: we use a logit-based score for embedding attacks and a loss-based score for token-level attacks. We assume differentiability with respect to the continuous input and define the input gradient

$$g_\theta(x, t) \;\coloneqq\; \nabla_x S_\theta(x, t) \in \mathbb{R}^d.$$

A targeted FGSM-style perturbation takes a single step in the sign of the target-score gradient:

$$x_{\text{adv}} \;=\; x + \varepsilon\, \operatorname{sgn}\big(g_\theta(x, t)\big).$$

Gaussian processes (GPs) and the neural tangent kernel (NTK). A (vector-valued) Gaussian process (GP) is a collection of random vectors $\{f(x) \in \mathbb{R}^m : x \in \mathbb{R}^d\}$ such that for any finite set of inputs $\{x_1, \ldots, x_n\}$, the stacked vector $[f(x_1)^\top, \ldots, f(x_n)^\top]^\top$ is multivariate Gaussian (Williams & Rasmussen, 2006). A GP is determined by its mean function and its kernel (covariance) function. Wide neural networks with i.i.d. Gaussian weights converge, at random initialization (and with a suitable parametrization), to a GP: the joint distribution of outputs becomes (approximately) Gaussian as widths grow. As widths $\to \infty$ (with depth fixed and appropriate scaling), the network converges in finite-dimensional distribution to a GP with a deterministic kernel. This correspondence extends to broad architecture classes beyond fully-connected networks (Yang, 2019). The NTK perspective (Jacot et al., 2018) studies gradient-based training in the same wide-network scaling. In this regime, training dynamics under (suitably scaled) gradient descent are described by a kernel method with the limiting NTK computed at initialization (Jacot et al., 2018) and nearly constant during training. The limiting final outputs are Gaussian at initialization and remain Gaussian. Moreover, with suitable input differentiability, their input derivatives are jointly Gaussian with them (Appendix C), a fact our method crucially relies on.

Gaussian conditioning for gradients. Denote by $H(x)$ the vector random variable representing the hidden state and by $G(x)$ the network input gradient. If (H, G) is jointly Gaussian with means μ_H, μ_G and covariances $\Sigma_{HH} = \operatorname{Cov}(H, H)$ and $\Sigma_{GH} = \operatorname{Cov}(G, H)$, with Σ_{HH} invertible, then the conditional mean is affine:

$$\mathbb{E}[G \mid H] \;=\; \mu_G + \Sigma_{GH}\Sigma_{HH}^{-1}(H - \mu_H). \tag{1}$$

For the NTK specialization, take $H(x)$ to be the final hidden state and $G(x) = \nabla_x(c^\top H(x))$ for a fixed vector c. Under NTK training, we get joint Gaussianity of $(H(x), G(x))$; this follows from the GP derivative rule rather than being an extra Gaussianity assumption. Note that nonlinear activation functions inside the network are allowed. Appendix C gives more details.

Stationary kernels. A kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is *stationary* if $k(x + c, x' + c) = k(x, x')$ for all x, x', c, equivalently $k(x, x') = \psi(x - x')$ for some ψ. In our idealized analysis, joint stationarity of (H, Y), together with the stated moment and differentiability conditions, makes the relevant prior means and same-input covariances of $(H(x), \nabla_x Y(x))$ input-independent.[1]

Embedding vs token-level attacks (text models). Embedding-level attacks optimize continuous input embeddings directly, while token-level attacks operate in the discrete space of strings and typically use gradients only as guidance. The Greedy Coordinate Gradient (GCG) attack (Zou et al., 2023b) is a representative token-level method.

3 Our Method

Main idea. Our method eliminates the backward pass at attack time by predicting the input-gradient direction from a forward-pass hidden state. Concretely, we fit a linear predictor (with a bias term) on a small training set:

$$\nabla_x S_\theta(x, t) \approx\; A\, h_\theta(x) + b, \tag{2}$$

[1] See Section 3 for proof.

Algorithm 1 Grad-Predict FGSM (GPR-FGSM)

Require: Model f_θ with logits $z_\theta(\cdot)$, perturbation budget ε, inputs/targets $(x_i^\star, t_i^\star)$, predictor coefficients A, b (feature-standardized).

for $(x_i^\star, t_i^\star)$ **do**

$r_i \leftarrow h_\theta^{(\ell)}(x_i^\star)$ ▷ forward only

$\hat{g}_i \leftarrow A\, r_i + b$ ▷ predict grad. direction

$x_i^{\mathrm{adv}} \leftarrow x_i^\star + \varepsilon\, \mathrm{sgn}(\hat{g}_i)$

end for

Algorithm 2 Vanilla FGSM

Require: Model f_θ with logits $z_\theta(\cdot)$, perturbation budget ε, inputs/targets $(x_i^\star, t_i^\star)$.

for $(x_i^\star, t_i^\star)$ **do**

$g_i \leftarrow \nabla_x S_\theta(x_i^\star, t_i^\star)$ ▷ compute gradient

$x_i^{\mathrm{adv}} \leftarrow x_i^\star + \varepsilon\, \mathrm{sgn}(g_i)$

end for

where $h_\theta(x)$ denotes the chosen representation. The predictor is learned for a specified model, layer, objective, and prompt-target distribution. Targets are used when learning the predictor; we do not claim that arbitrary target changes can be inferred from early hidden states. We evaluate generalization to evaluation prompts (from the same distribution), we do not claim universal meta-predictions across models and objectives.

Proof structure. We analyze an idealized model in the NTK regime. Lemma 1 supplies stationarity for a trigonometric embedding, and Lemma 2 gives its constant cross-covariances; neither establishes stationarity in the fully general setting of real-world transformers. Gaussian conditioning then therefore gives an exact affine conditional mean at each input (Proposition 1). All theoretical results are proved in Appendix D. Our experiments test learned affine predictors using intermediate-layer representations and normalized targets beyond this idealization. Proofs appear in Appendix D.

Embedding choice. A sine-cosine embedding makes input translations equivalent to rotations of Gaussian coefficient pairs, giving a tractable stationary construction. This is an illustrative model, not a claim that real-world trained transformers are stationary in input-embedding space; Although RoPE also uses rotations (Su et al., 2024), it does it in a somewhat different setup. The following lemma (see appendix D for proof) shows that the sine-cosine embedding has a useful stationarity property.

Lemma 1 (Stationarity)**.** *Fix $d, m \in \mathbb{N}$. Let $\omega_j \sim p$ on $\mathbb{R}^d$ and $(u_j, v_j) \sim \mathcal{N}(0, I_2)$, independently across j and between frequencies and coefficient pairs. Define $e(x) = (e_1(x)^\top, \ldots, e_m(x)^\top)^\top$ by*

$$e_j(x) = \begin{bmatrix} \cos(\omega_j^\top x) & \sin(\omega_j^\top x) \\ -\sin(\omega_j^\top x) & \cos(\omega_j^\top x) \end{bmatrix} \begin{bmatrix} u_j \\ v_j \end{bmatrix}.$$

Let the remaining network parameters Θ be independent of the embedding parameters, and set

$$H(x) = R(e(x); \Theta), \qquad Y(x) = w(\Theta)^\top H(x) + \beta(\Theta),$$

where R is any measurable map into $\mathbb{R}^{d_h}$. Assume $H(x)$ and $Y(x)$ have finite second moments, and $Y(\cdot)$ is almost surely differentiable with integrable gradient $G(x) := \nabla_x Y(x)$.

Then (H, Y) is jointly stationary: its means are constant, and its auto- and cross-covariances depend only on $x - x'$. In particular, $k(x, x') := \mathbb{E}[Y(x)Y(x')]$ is stationary. The unconditional prior gradient mean $\mu_G(x) := \mathbb{E}[G(x)]$ is also constant in x.[2]

Constant covariances. In the following lemma (see appendix D for a proof), we identify conditions under which the centered covariances in (1) are independent of x.

Lemma 2 (Constant covariances)**.** *Let $H(x)$, $Y(x)$, and $G(x) = \nabla_x Y(x)$ be as in Lemma 1, and assume additionally that $\mathbb{E}\|G(x)\|^2 < \infty$ for every x. Then the same-input covariances*

$$\Sigma_{HH}(x) := \mathrm{Cov}(H(x), H(x)), \qquad \Sigma_{GH}(x) := \mathrm{Cov}(G(x), H(x))$$

are independent of x.

[2]This prior mean is not the conditional gradient prediction $\mathbb{E}[G(x) \mid H(x)]$, which need not be constant.

Main result. The following proposition derives the formula for the affine predictor, by characterizing an input-independent conditional mean in the idealized setting of joint Gaussianity and constant first and cross-covariance moments. Lemma 2 supplies the covariance condition. The proposition is proved in appendix D.

Proposition 1 (Input-independent affine gradient predictor)**.** *Suppose $H(x)$ and $G(x)$ have input-independent means μ_H, μ_G and same-input covariances $\Sigma_{HH} = \mathrm{Cov}(H(x), H(x))$ and $\Sigma_{GH} = \mathrm{Cov}(G(x), H(x))$, as supplied by Lemmas 1 and 2. Assume additionally that $(H(x), G(x))$ is jointly Gaussian for each x and that Σ_{HH} is invertible.*

Then

$$\mathbb{E}[G(x) \mid H(x)] = AH(x) + b, \qquad A = \Sigma_{GH}\Sigma_{HH}^{-1}, \quad b = \mu_G - A\mu_H,$$

with A, b independent of x. For each fixed x, this is the exact minimum-mean-squared-error predictor among all measurable functions of $H(x)$. It recovers $G(x)$ almost surely precisely when

$$\mathrm{Cov}(G(x), G(x)) - \Sigma_{GH}\Sigma_{HH}^{-1}\Sigma_{GH}^{\top} = 0.$$

The analysis above applies in the idealized NTK/Gaussian setup. Without Gaussianity, however, this predictor need not equal $\mathbb{E}[G \mid H]$: a nonlinear predictor may do better. However, we can show that the derived predictor is at least best in class of all affine predictors. We do that in the following observation (see appendix D) for a proof.

Observation 1 (Best affine predictor)**.** *Let H and G be any square-integrable random vectors with $\mathrm{Cov}(H, H)$ invertible. Then*

$$A = \mathrm{Cov}(G, H)\mathrm{Cov}(H, H)^{-1}, \qquad b = \mathbb{E}[G] - A\mathbb{E}[H]$$

minimize $\mathbb{E}\|G - AH - b\|_2^2$ over all affine predictors. Neither Gaussianity nor stationarity is required.

Predictor Training. The observation above applies to a fixed trained model with prompts sampled form a fixed distribution, with normalized gradient targets. In other words, the predictor has to be retrained for each use-case.

4 Related Work

Gradient-based attacks. Adversarial training aims to improve robustness by incorporating worst-case (or approximately worst-case) perturbations into the learning objective. Early work demonstrated that neural networks admit adversarial perturbations found via optimization (Szegedy et al., 2014). Goodfellow et al. (2015) introduced the Fast Gradient Sign Method (FGSM), enabling efficient single-step attacks using input gradients, while Madry et al. (2018) formulated adversarial robustness as a min–max problem and advocated multi-step projected gradient descent (PGD). A large body of follow-up work focuses on reducing the cost of these gradient computations. For example, adversarial training in the style of Shafahi et al. (2019) recycles the gradient computed for the weight update to also construct the adversarial perturbation, YOPO restricts backpropagation to the first layer (Zhang et al., 2019), randomly initialized FGSM can match PGD-based adversarial training when combined with appropriate training hyperparameters (Wong et al., 2020) and single-step methods with stabilization techniques (e.g., GradAlign) approximate multi-step robustness (Andriushchenko & Flammarion, 2020). Despite these improvements, such methods fundamentally rely on backpropagation to obtain input gradients. Our approach builds on this line of work, with the goal of making the methods faster by using synthetic gradients.

Gradient-free (black-box) attacks. An alternative paradigm eliminates backpropagation by estimating gradients through input perturbations. Zeroth-order optimization methods such as ZOO, NES and SPSA approximate $\nabla_x S(x)$, where $S(x)$ is the attack objective (e.g., a target logit or loss), using randomized queries (Chen et al., 2017; Ilyas et al., 2018; Uesato et al., 2018). These approaches are applicable in black-box settings but typically require hundreds to thousands of forward evaluations per example, trading gradient access for substantial query complexity. Like our approach, they avoid backpropagation; however, they estimate gradients via repeated input queries, whereas we predict them directly from forward-pass representations.

Gradient prediction. A separate line of work replaces gradient computation altogether by learning predictors of gradients. Synthetic gradient methods Jaderberg et al. (2017); Czarnecki et al. (2017) decouple neural network modules by predicting gradients during training. Our approach adapts this paradigm to adversarial attacks by predicting input gradients directly from forward-pass representations. In our setting, each gradient estimate uses forward-pass features and a linear map without backpropagation or query-based gradient estimation. Multi-step attacks repeat this operation, and GCG additionally evaluates candidate losses.

Steering. Steering methods seek to controllably alter a model's behavior at inference time, often by intervening on internal representations rather than updating weights. Inference-Time Intervention (ITI) (Li et al., 2023) learns a small set of activation-space directions (applied to selected attention heads) and shifts activations along these directions during the forward pass to elicit more truthful responses, revealing a tunable tradeoff between truthfulness and other desired behaviors. Representation Engineering (Zou et al., 2023a) advocates a top-down view in which representations encode high-level concepts; it develops practical procedures to discover, monitor, and manipulate directions relevant to transparency and safety (e.g., honesty or harmfulness). Activation Addition (Turner et al., 2024) provides a lightweight steering primitive that computes a steering vector from differences in intermediate activations induced by contrastive prompt pairs and then adds this vector during inference, enabling rapid, optimization-free control over attributes such as sentiment or toxicity while aiming to preserve performance on off-target tasks. Our approach is related in that it also leverages internal representations at inference time. However, whereas steering methods modify hidden states to control behavior, we instead use hidden states as predictors of input gradients. In this sense, steering performs representation-space intervention, while our method performs representation-based gradient inference.

Neural Tangent Kernel (NTK). The infinite-width description distinguishes the initialization covariance (NNGP) kernel from the NTK governing gradient-based training (Jacot et al., 2018). Lee et al. (2018) and Matthews et al. (2018) derive the GP limit of randomly initialized deep fully-connected networks, and Yang (2019) extends such results to broader architecture classes. For our theoretical analysis, we use the NTK approach, not Bayesian posterior inference with the NNGP kernel. Specifically, we leverage the fact that, under training on fixed data, in the NTK setting, the limiting output process remains Gaussian (Jacot et al., 2018). [3]

5 Experiments

Structure of Experiments We structure our experimental results into two sections, each considering a different type of attack on the LLM. Section 5.1 deals with attacks in a setting where we can perturb the input embedding, while section 5.2 focuses on attacks via exchanging input tokens.

LLMs and Datasets We wanted to evaluate for several different levels of model hardening quality so we used three LLMs: Llama 2 (released in 2023), Qwen 2.5 (released in 2024) and Qwen 3 (released in 2025). We tested all 3 models against JailbreakBench (Chao et al., 2024) and HarmBench (Mazeika et al., 2024) – both of which are leading open-source frameworks designed to evaluate and enhance the robustness of LLMs against jailbreak attacks.

Practical modifications. In practice, we use several simple technical modifications that substantially improve speed and stability. First, rather than extracting $h_\theta(x)$ from the final layer, we regress on an intermediate hidden state $h_\theta^{(\ell)}(x)$ and (at test time) obtain it via an early-exit forward pass, reducing compute while still preserving a strong signal for predicting input gradients. Second, we normalize regression targets per token to emphasize gradient direction rather than magnitude. Third, we standardize hidden-state features coordinatewise, which is standard practice for linear regression with heterogeneous feature scales. Finally, we fit the predictor with ridge regression, which both stabilizes the solution in the presence of collinearity and provides a simple form of regularization that improves generalization from small training sets.

[3] See appendix C for details in our notation.

Table 1: Attack success rate (ASR), perturbation throughput (Pert.), and success throughput (Succ.) for embedding-level attacks on held-out prompts from HarmBench. 95% bootstrap CIs from per-prompt resampling ($B = 10{,}000$).

	Qwen3-4B			Qwen2.5-3B			Llama-2-7B		
Method	ASR (%)	Pert. (ex/s)	Succ. (att/s)	ASR (%)	Pert. (ex/s)	Succ. (att/s)	ASR (%)	Pert. (ex/s)	Succ. (att/s)
Exact gradient									
FGSM	$12.6^{+3.9}_{-3.6}$	$5.1^{+0.0}_{-0.0}$	$0.65^{+0.20}_{-0.18}$	$24.5^{+4.8}_{-4.8}$	$6.3^{+0.0}_{-0.1}$	$1.55^{+0.31}_{-0.31}$	$18.6^{+4.6}_{-4.3}$	$7.4^{+0.0}_{-0.0}$	$1.37^{+0.35}_{-0.32}$
FGM	$9.7^{+3.6}_{-3.2}$	$5.1^{+0.0}_{-0.0}$	$0.50^{+0.18}_{-0.17}$	$6.1^{+3.1}_{-2.7}$	$6.1^{+0.0}_{-0.0}$	$0.38^{+0.19}_{-0.17}$	$10.0^{+3.6}_{-3.2}$	$7.4^{+0.0}_{-0.0}$	$0.74^{+0.26}_{-0.24}$
RS-FGSM	$11.0^{+3.6}_{-3.2}$	$5.1^{+0.0}_{-0.0}$	$0.56^{+0.18}_{-0.17}$	$22.8^{+4.8}_{-4.8}$	$6.3^{+0.0}_{-0.0}$	$1.43^{+0.30}_{-0.30}$	$18.6^{+4.6}_{-4.6}$	$7.3^{+0.0}_{-0.0}$	$1.35^{+0.34}_{-0.34}$
PGD-2	$16.5^{+4.2}_{-4.2}$	$2.6^{+0.0}_{-0.0}$	$0.42^{+0.11}_{-0.11}$	$31.0^{+5.4}_{-5.1}$	$3.1^{+0.0}_{-0.0}$	$0.97^{+0.17}_{-0.16}$	$39.3^{+5.7}_{-5.7}$	$3.7^{+0.0}_{-0.0}$	$1.44^{+0.21}_{-0.21}$
PGD-5	$32.4^{+5.5}_{-5.2}$	$1.0^{+0.0}_{-0.0}$	$0.33^{+0.06}_{-0.05}$	$59.9^{+5.5}_{-5.8}$	$1.3^{+0.0}_{-0.0}$	$0.76^{+0.07}_{-0.07}$	$41.4^{+5.7}_{-5.7}$	$1.5^{+0.0}_{-0.0}$	$0.61^{+0.08}_{-0.08}$
PGD-10	$35.6^{+5.5}_{-5.2}$	$0.5^{+0.0}_{-0.0}$	$0.18^{+0.03}_{-0.03}$	$72.1^{+5.1}_{-5.1}$	$0.6^{+0.0}_{-0.0}$	$0.46^{+0.03}_{-0.03}$	$49.3^{+5.7}_{-5.7}$	$0.7^{+0.0}_{-0.0}$	$0.36^{+0.04}_{-0.04}$
GPR (predicted)									
FGSM	$6.8^{+2.9}_{-2.6}$	$60.5^{+0.2}_{-0.2}$	$\mathbf{4.11}^{+1.77}_{-1.57}$	$11.6^{+3.7}_{-3.4}$	$73.4^{+0.2}_{-0.3}$	$\mathbf{8.49}^{+2.74}_{-2.51}$	$15.4^{+4.3}_{-3.9}$	$47.4^{+0.2}_{-0.2}$	$\mathbf{7.28}^{+2.04}_{-1.87}$
FGM	$4.2^{+2.3}_{-1.9}$	$59.6^{+0.2}_{-0.3}$	$\mathbf{2.51}^{+1.36}_{-1.16}$	$3.4^{+2.4}_{-2.0}$	$71.4^{+0.2}_{-0.2}$	$\mathbf{2.43}^{+1.69}_{-1.45}$	$2.1^{+1.8}_{-1.4}$	$47.0^{+0.1}_{-0.1}$	$\mathbf{1.01}^{+0.84}_{-0.67}$
RS-FGSM	$8.1^{+3.2}_{-2.9}$	$60.3^{+0.2}_{-0.2}$	$\mathbf{4.88}^{+1.92}_{-1.75}$	$13.6^{+4.1}_{-3.7}$	$72.3^{+0.3}_{-0.3}$	$\mathbf{9.84}^{+2.95}_{-2.71}$	$16.4^{+4.3}_{-4.3}$	$47.1^{+0.1}_{-0.1}$	$\mathbf{7.74}^{+2.04}_{-2.01}$
PGD-2	$6.5^{+2.9}_{-2.6}$	$30.8^{+0.1}_{-0.1}$	$\mathbf{2.00}^{+0.89}_{-0.80}$	$7.5^{+3.1}_{-2.7}$	$37.2^{+0.1}_{-0.1}$	$\mathbf{2.78}^{+1.15}_{-1.01}$	$13.6^{+3.9}_{-3.9}$	$23.9^{+0.1}_{-0.1}$	$\mathbf{3.25}^{+0.95}_{-0.94}$
PGD-5	$6.5^{+2.9}_{-2.6}$	$12.5^{+0.0}_{-0.0}$	$\mathbf{0.81}^{+0.36}_{-0.32}$	$17.3^{+4.4}_{-4.4}$	$15.2^{+0.0}_{-0.0}$	$\mathbf{2.63}^{+0.68}_{-0.67}$	$15.7^{+4.3}_{-3.9}$	$9.7^{+0.0}_{-0.0}$	$\mathbf{1.52}^{+0.42}_{-0.38}$
PGD-10	$3.9^{+2.3}_{-1.9}$	$6.3^{+0.0}_{-0.0}$	$\mathbf{0.25}^{+0.14}_{-0.12}$	$18.7^{+4.4}_{-4.4}$	$7.7^{+0.0}_{-0.0}$	$\mathbf{1.43}^{+0.34}_{-0.34}$	$15.4^{+4.3}_{-3.9}$	$4.9^{+0.0}_{-0.0}$	$\mathbf{0.75}^{+0.21}_{-0.19}$

5.1 Embedding-level attacks.

Setup In this set of experiments, we perturb the continuous token embeddings which the model receives as input. Given a prompt[4], we search for a small perturbation of its embedding sequence that causes the model to produce a harmful response instead of refusal. The perturbation is bounded by an ϵ-ball of the original embedding sequence, according to L_2 or L_∞ norm, depending on method. Our main optimization criterion is *successful attack throughput*, i.e., the number of prompts for which we successfully find an adversarial embedding perturbation per second. Note that with this criterion we may prefer a method that has lower *attack success rate (ASR)* (i.e., the proportion of prompts for which adversarial perturbation is found) over the method with better ASR, but lower processing speed. When measuring successful attack throughput, we only include time required to generate embedding perturbation.

Predictor Training Details The GPR predictor is a single linear map (with a bias feature) from a layer-L hidden state to the per-token, unit-normalized gradient $\hat{g}_t = g_t/\|g_t\|_2$ of the target logit with respect to the input embedding at token t; each input-token position contributes one training sample. For every training prompt we (i) run a full forward/backward pass and record layer-L input hidden states paired with $\hat{g}_t$, and (ii) take K FGSM augmentation steps in input-embedding space, recomputing the forward/backward at each step and recording the resulting $(h, \hat{g})$ pairs. Step-k samples are downweighted by γ^k. Hidden-state features are standardized per dimension using the empirical mean and std over all stored samples and then concatenated with a constant bias column. The weights are obtained in closed form by a single weighted ridge regression with fixed λ=1, solved via an eigendecomposition, fitting all d output dimensions jointly. The choice of hyperparameters is described in appendix B.1, together with a benchmark reporting how sensitive we are to hyperparameter choice.

Predictor Training Cost Gradient prediction removes backward passes at attack time, not when training the predictor. For example, for the Qwen2.5-3B FGM experiment, this takes 159.2-259.7 seconds (see B.3 for details). Comparing the wall-clock time of our method ($T_{\text{cal}} + N\bar{t}_{\text{GPR}}$) with the wall-clock time of the baseline($N\bar{t}_{\text{exact}}$) gives an equal-attempt runtime break-even point of between 1,280 and 2,113 attacked examples.

[4]We only consider prompts for which the zero-shot method doesn't work i.e., where the model refuses the harmful behavior without an intervention.

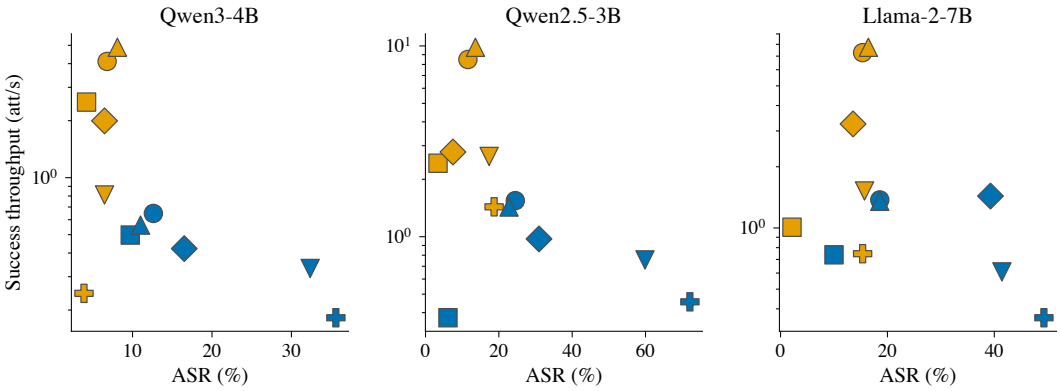
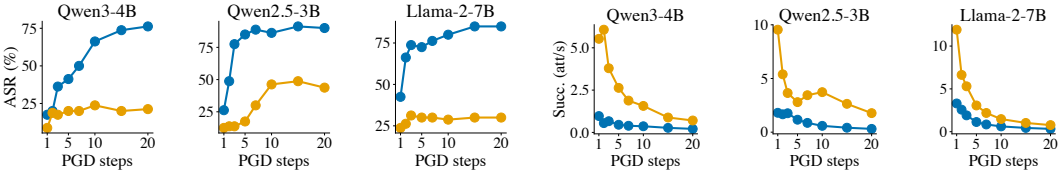


Figure 1: Pareto frontier of ASR vs. successful-attack throughput on HarmBench. Each point is a (method, model) configuration. Gradient source (color): ■ exact, ■ GPR. Attack (shape): ● FGSM, ■ FGM, ▲ RS-FGSM, ♦ PGD-2, ▼ PGD-5, + PGD-10.



(a) ASR vs. no. of PGD steps.

(b) Successful-attack throughput vs. no. of PGD steps.

Figure 2: PGD scaling on HarmBench. Curves: ■ PGD with exact gradient, ■ PGD-GPR

.

Main Results Table 1 reports results on model-specific non-trivial subsets of the 320-prompt HarmBench test split. GPR has higher successful-attack-throughput point estimates than the corresponding exact-gradient method in each reported HarmBench comparison, but the gains vary substantially and ASR is often lower. For example, FGSM on Qwen3-4B increases successful-attack throughput from 0.65 to 4.11 attacks/s, while ASR decreases from 12.6% to 6.8%. Appendix B.4 provides a direct throughput plot. Appendix B.3 reports cross-benchmark transfer, cosine/sign-agreement diagnostics, and predictor training cost for a controlled FGM configuration.

Throughput-ASR Pareto frontier. Figure 1 plots ASR against successful-attack throughput across all attack variants and models. It can be seen that GPR enables a new part of the Pareto frontier, with the largest gains for the cheap one-shot attacks (FGSM, RS-FGSM, FGM). The trade-off between ASR and throughput is further visible in figure 2, which plots both as a function of how many PGD steps the method takes. As can be expected, GPR works less well (but still has a higher throughput) for a larger number of steps. We theorize this is because of compounding of errors across several gradient steps.

5.2 Token-level attacks.

Setup. We evaluate whether gradient prediction can accelerate discrete prompt attacks, where the adversary optimizes an adversarial token suffix rather than a continuous perturbation. We use Greedy Coordinate Gradient (GCG) (Zou et al., 2023b), which iteratively uses token-embedding gradients to propose single-token replacements and then selects among those candidates using exact forward losses on a target continuation. Our variant, GCG-GPR, keeps the same GCG objective, candidate evaluation, and acceptance rule, but replaces each backward-computed suffix gradient with a predicted gradient from an intermediate hidden representation. Thus, the predictor only affects which candidate tokens are considered; the accepted update is still chosen using the true autoregressive loss. Appendix A gives the full objective and candidate-generation equations.

Table 2: Attack success rate (ASR), perturbation throughput (Pert.), and success throughput (Succ.) for prompt-level GCG attacks on held-out prompts from HarmBench (HB) and JailbreakBench (JBB). The blind random baseline independently samples one suffix per prompt and performs no loss-based search; its Pert. and Succ. entries are therefore left undefined. 95% bootstrap CIs from per-prompt resampling ($B = 10{,}000$).

		Qwen3-4B			Qwen2.5-3B			Llama-2-7B		
	Method	ASR (%)	Pert. (ex/s)	Succ. (att/s)	ASR (%)	Pert. (ex/s)	Succ. (att/s)	ASR (%)	Pert. (ex/s)	Succ. (att/s)
Blind random suffix (no search)										
HB	Random	$0.0^{+0.0}_{-0.0}$	—	—	$5.9^{+2.8}_{-2.5}$	—	—	$8.1^{+3.1}_{-2.8}$	—	—
JBB	Random	$0.0^{+0.0}_{-0.0}$	—	—	$1.0^{+2.0}_{-1.0}$	—	—	$6.0^{+5.0}_{-4.0}$	—	—
Exact gradient										
HB	GCG	$5.0^{+2.5}_{-2.2}$	$0.1^{+0.0}_{-0.0}$	$0.01^{+0.00}_{-0.00}$	$49.4^{+5.3}_{-5.3}$	$0.2^{+0.0}_{-0.0}$	$0.08^{+0.01}_{-0.01}$	$17.2^{+4.4}_{-4.1}$	$0.1^{+0.0}_{-0.0}$	$0.02^{+0.01}_{-0.01}$
JBB	GCG	$0.0^{+0.0}_{-0.0}$	$0.1^{+0.0}_{-0.0}$	$0.00^{+0.00}_{-0.00}$	$60.0^{+10.0}_{-10.0}$	$0.2^{+0.0}_{-0.0}$	$0.10^{+0.02}_{-0.02}$	$21.0^{+8.0}_{-8.0}$	$0.1^{+0.0}_{-0.0}$	$0.02^{+0.01}_{-0.01}$
GPR (predicted)										
HB	GCG	$4.4^{+2.5}_{-2.2}$	$0.3^{+0.0}_{-0.0}$	$0.01^{+0.01}_{-0.01}$	$50.9^{+5.6}_{-5.3}$	$0.3^{+0.0}_{-0.0}$	$\mathbf{0.18}^{+0.02}_{-0.02}$	$19.4^{+4.4}_{-4.4}$	$0.2^{+0.0}_{-0.0}$	$\mathbf{0.04}^{+0.01}_{-0.01}$
JBB	GCG	$0.0^{+0.0}_{-0.0}$	$0.3^{+0.0}_{-0.0}$	$0.00^{+0.00}_{-0.00}$	$69.0^{+9.0}_{-9.0}$	$0.3^{+0.0}_{-0.0}$	$\mathbf{0.24}^{+0.03}_{-0.03}$	$21.0^{+8.0}_{-8.0}$	$0.2^{+0.0}_{-0.0}$	$\mathbf{0.04}^{+0.01}_{-0.01}$

Experimental setting We compare exact-gradient GCG and GCG-GPR on HarmBench and JailbreakBench using the same suffix-optimization protocol. Both methods optimize the same target-continuation objective and use exact forward losses to select token substitutions; they differ only in whether candidate tokens are proposed using an exact or predicted gradient. We additionally evaluate a blind random-suffix baseline as a gradient-free sanity check. For each prompt, this baseline independently replaces every position in the GCG suffix region with a token sampled uniformly from the tokenizer's non-special vocabulary and uses the resulting suffix directly, without gradient computation, candidate-loss evaluation, or iterative search. [5] We fit the predictor on augmented suffix states generated from GCG trajectories, so that its training inputs resemble the iterates encountered during the attack. We report ASR for all methods and perturbation and successful-attack throughput for exact-gradient GCG and GCG-GPR. Throughput is left undefined for the blind random baseline because it performs no optimization. Appendix A.2 provides the complete experimental settings.

Results Table 2 reports ASR for the blind random baseline and ASR and throughput for exact-gradient GCG and GCG-GPR. The blind random suffix is substantially less effective in every setting in which GCG-GPR obtains nonzero ASR. This confirms that arbitrary, unoptimized suffix perturbations do not account for the effectiveness of the predicted-gradient attack.

Comparing the two optimized attacks, GCG-GPR has comparable ASR point estimates to exact-gradient GCG, with overlapping confidence intervals in the reported settings and higher point estimates on Qwen2.5-3B on both benchmarks. Replacing the backward-computed gradient increases perturbation throughput by roughly 1.5–$3\times$, yielding 2–$2.4\times$ higher successful-attack throughput in the nonzero-success regimes. On Qwen3-4B with JailbreakBench, all three methods obtain 0% ASR. Overall, predicted gradients produce substantially more effective suffixes than blind random sampling while retaining the efficiency advantage over exact-gradient GCG.

5.3 Sensitivity to gradient-prediction quality

The embedding-level main results in Section 5.1 use HarmBench. We additionally evaluate the same attacks on JailbreakBench (Appendix B.2, Table 3), obtaining less favorable results. On Qwen2.5-3B, FGM-GPR has observed ASR 22.0%, compared with 8.0% for FGM, while FGSM-GPR has ASR 4.0%, compared with 22.0% for FGSM. The effect of gradient prediction on ASR varies across the evaluated attack configurations. Low ASR in some exact-gradient baselines also highlights limitations of the underlying attacks.

[5] It tests whether arbitrary suffix perturbations suffice; it is not a GCG variant with random candidate proposals, since it does not retain exact-loss selection.

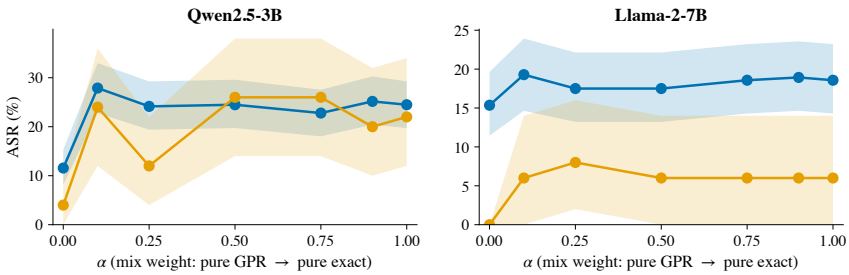


Figure 3: FGSM with mixed gradients (Section 5.3). $\alpha = 0$ uses predicted gradients and $\alpha = 1$ uses exact gradients, with the same attack settings throughout each curve. Evaluations use the same non-trivial prompt subsets as the corresponding tables. Appendix B.1 describes filtering and shared JailbreakBench judging. Shaded bands are 95% bootstrap CIs ($B = 10,000$). Curves: HarmBench, JailbreakBench.

To diagnose this, we run a mixed-gradient ablation: at each FGSM step we use $g_{\text{mix}} = (1-\alpha)\,\hat{g} + \alpha\, g_{\text{exact}}$ for $\alpha \in \{0, 0.1, 0.25, 0.5, 0.75, 0.9, 1\}$, holding all hyperparameters at their GPR-tuned values so that gradient quality is the only varying factor. Figure 3 plots ASR vs. α on HarmBench and JailbreakBench for Qwen2.5-3B and Llama-2-7B.

The effect of mixing depends on the model and benchmark. On Llama-2/JBB, $\alpha = 0.1$ gives observed ASR 6.0%, compared with 0.0% at $\alpha = 0$ and 6.0% at $\alpha = 1$; the Qwen2.5/JBB curve is less flat. Intermediate mixtures can be useful in the evaluated settings. Further evidence would be needed to establish a universal sign-agreement threshold. The influence of each term also depends on its gradient norm. This diagnostic experiment requires backpropagation to compute exact gradients. Figures 5a and 6, together with the cosine/sign-agreement results in Appendix B.3, provide complementary diagnostics.

6 Limitations

Our experiments cover three open LLMs and two jailbreak benchmarks. The theory is a conditional-mean characterization under idealized assumptions, not an exact-gradient guarantee for real-world transformers. Predictors are trained for a particular model, layer, objective, and prompt-target distribution; model updates or distribution changes may require retraining the predictor. The cost of such training must be amortized. Also, embedding perturbations are not directly deployable token attacks.

7 Conclusions

We presented a simple approach for accelerating adversarial example generation by replacing the per-example backward pass with a lightweight, learned linear predictor that maps forward-pass hidden states to input-gradient directions. Our method produces FGSM-style perturbations using only forward computation, substantially improving adversarial throughput under realistic wall-clock constraints. Theoretically, we derived an exact affine conditional-mean characterization for differentiable NTK-limit outputs. Empirically, the resulting gradient-prediction attack recovers a large fraction of the effectiveness of vanilla FGSM while using a small fraction of its compute, suggesting that representation-gradient relationships in modern networks are regular enough to exploit for fast adversarial generation.

Ethics Statement

This work lowers the computational cost of adversarial generation. It may support authorized robustness evaluation and red-teaming, but can also lower the cost of searching for harmful outputs or bypassing safeguards. All experiments use public models and datasets. The efficiency improvement therefore has dual-use implications, even though the underlying attack objectives already exist. The release is intended to support reproducible safety research.

AI USE STATEMENT

Large language models were used during the preparation of this work in the following ways: (1) as coding assistants to help generate implementation code for the experiments, (2) to polish the writing of the manuscript, (3) in the development of the proofs, (4) to suggest initial wording of some paragraphs (which were then edited manually), (5) for finding related work (and getting a detailed understanding of the NTK/NNGP literature), (6) for research ideation and helping decide what experiments to run next. In all cases, the human authors verified the output: code was tested and reviewed, written passages were checked and edited, and proof steps were independently checked for correctness before inclusion. The authors take full responsibility for the content of the paper.

REFERENCES

Maksym Andriushchenko and Nicolas Flammarion. Understanding and improving fast adversarial training. In *Annual Conference on Neural Information Processing Systems*, 2020.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J Pappas, Florian Tramer, et al. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. *Advances in Neural Information Processing Systems*, 37:55005–55029, 2024.

Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pp. 15–26, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450352024.

Wojciech Marian Czarnecki, Grzegorz Świrszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding synthetic gradients and decoupled neural interfaces. In *International Conference on Machine Learning*, pp. 904–912. PMLR, 2017.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, July 2018. URL https://arxiv.org/abs/1804.08598.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, pp. 1627–1635. PMLR, 2017.

Jaehoon Lee, Jascha Sohl-dickstein, Jeffrey Pennington, Roman Novak, Sam Schoenholz, and Yasaman Bahri. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1EA-M-0Z.

Kenneth Li, Oam Patel, Fernanda B. Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/81b8390039b7302c909cb769f8b6cd93-Abstract-Conference.html.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018.

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.

Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free!, 2019. URL https://arxiv.org/abs/1904.12843.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6199.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. Activation addition: Steering language models without optimization. 2024.

Jonathan Uesato, Brendan O'Donoghue, Pushmeet Kohli, and Aaron van den Oord. Adversarial risk and the dangers of evaluating against weak attacks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5025–5034. PMLR, 10–15 Jul 2018.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006.

Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJx040EFvH.

Greg Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. *Advances in neural information processing systems*, 32, 2019.

Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle, 2019. URL https://arxiv.org/abs/1905.00877.

Andy Zou, Long Phan, Sarah Li Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, J. Zico Kolter, and Dan Hendrycks. Representation engineering: A top-down approach to AI transparency. *CoRR*, abs/2310.01405, 2023a. doi: 10.48550/ARXIV.2310.01405. URL https://doi.org/10.48550/arXiv.2310.01405.

Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *CoRR*, abs/2307.15043, 2023b. doi: 10.48550/ARXIV.2307.15043. URL https://doi.org/10.48550/arXiv.2307.15043.

A EXPERIMENTAL DETAILS - TOKEN-LEVEL ATTACKS

This appendix gives the full token-level attack setup used in Section 5.2.

A.1 SETUP AND METHOD DETAILS

Setup. We evaluate token-level attacks using Greedy Coordinate Gradient (GCG) (Zou et al., 2023b), a discrete suffix-optimization attack for autoregressive language models. Let $q_{1:m}$ denote the fixed user prompt and let $a_{1:\ell} \in [V]^\ell$ denote an adversarial suffix of ℓ tokens, where V is the vocabulary size. The full model input is the concatenation

$$x(q, a) \;=\; [q_{1:m}\,, a_{1:\ell}].$$

For each prompt, GCG specifies a short target continuation $t_{1:H}$, typically an affirmative prefix that begins the response in a compliant form, like "Sure, here is". The attack objective is the teacher-forced negative log-likelihood of this target continuation:

$$\mathcal{L}_\theta(q, a, t) \;=\; -\sum_{h=1}^{H} \log P_\theta(t_h \mid x(q, a), t_{<h})\,. \tag{3}$$

Thus, the token-level adversarial problem is

$$\min_{a_{1:\ell} \in [V]^\ell} \; \mathcal{L}_\theta(q, a, t). \tag{4}$$

In the universal setting, the same suffix is optimized across prompts, and optionally across models with a shared tokenizer, by minimizing an aggregate objective

$$\mathcal{L}(a) \;=\; \sum_{\theta' \in \mathcal{M}} \sum_{j=1}^{J} \lambda_{\theta',j}\, \mathcal{L}_{\theta'}\left(q^{(j)}, a, t^{(j)}\right), \tag{5}$$

where $\mathcal{M}$ is the set of attacked models and $\lambda_{\theta',j}$ are optional weights.

Greedy Coordinate Gradient. GCG optimizes (4) by repeatedly proposing single-token substitutions. Let $E \in \mathbb{R}^{V \times d_e}$ be the token embedding matrix, and let $u_r = E_{a_r} \in \mathbb{R}^{d_e}$ be the embedding of the r-th suffix token. If $a_{r \leftarrow v}$ denotes the suffix obtained by replacing token a_r with candidate token v, then a first-order approximation gives

$$\mathcal{L}(a_{r \leftarrow v}) - \mathcal{L}(a) \;\approx\; (E_v - E_{a_r})^\top \nabla_{u_r} \mathcal{L}(a). \tag{6}$$

Equivalently, writing $\mathbf{e}_{a_r} \in \{0, 1\}^V$ for the one-hot representation of a_r, GCG computes

$$d_r \;=\; \nabla_{\mathbf{e}_{a_r}} \mathcal{L}(a) \in \mathbb{R}^V,$$

and ranks candidate replacements by the most negative coordinates of this gradient. For each suffix position r, it forms a candidate set

$$\mathcal{C}_r \;=\; \mathrm{TopK}_{v \in [V]} \left(-(E_v - E_{a_r})^\top \nabla_{u_r} \mathcal{L}(a) \right), \tag{7}$$

or, equivalently, the top-k entries of $-d_r$ up to the constant corresponding to the current token. GCG then samples a batch of candidate substitutions (r, v) with $v \in \mathcal{C}_r$, evaluates the true loss (3) for each substituted suffix using forward passes, and commits the replacement with the smallest exact loss. The gradient is therefore used only to construct a promising discrete candidate set; the final greedy step is still selected by evaluating the actual autoregressive objective.

GCG-GPR. Our token-level method keeps the GCG loss and greedy candidate evaluation unchanged, but replaces the backward-computed gradient in (7) with a predicted gradient. Specifically, from the same teacher-forced forward pass used to score the target continuation, we extract an intermediate hidden representation $h_\theta^{(\ell_0)}(q, a, t)$ and use our learned gradient predictor to estimate the suffix-embedding loss gradient:

$$\widehat{G}_\theta(q, a, t) \;=\; A\, \widetilde{h}_\theta^{(\ell_0)}(q, a, t) + b \;\approx\; \mathcal{N}_{\mathrm{tok}}(\nabla_{u_{1:\ell}} \mathcal{L}_\theta(q, a, t))\,, \tag{8}$$

where $\widetilde{h}_\theta^{(\ell_0)}$ denotes the standardized hidden-state feature vector and $\widehat{G}_\theta(q,a,t) \in \mathbb{R}^{\ell \times d_e}$ is reshaped into per-position predictions $\widehat{g}_r \in \mathbb{R}^{d_e}$. Here $\mathcal{N}_{\text{tok}}$ normalizes each nonzero token-gradient row to unit Euclidean norm and maps zero rows to zero. This preserves the exact within-position candidate ranking under positive rescaling. We then replace the exact GCG candidate score with

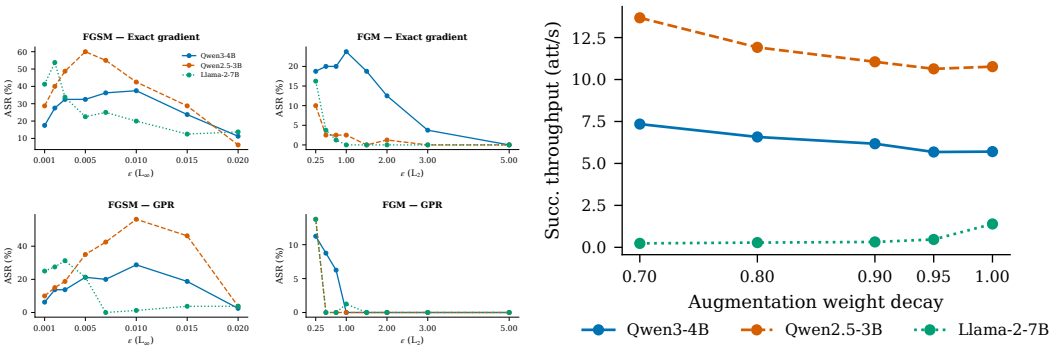
$$\widehat{\Delta}_{r,v} \;=\; (E_v - E_{a_r})^\top \widehat{g}_r, \qquad \widehat{\mathcal{C}}_r \;=\; \mathrm{TopK}_{v\in[V]}\left(-\widehat{\Delta}_{r,v}\right). \tag{9}$$

After this substitution, the rest of GCG is unchanged: we sample candidate token replacements from $\{\widehat{\mathcal{C}}_r\}_{r=1}^{\ell}$, evaluate their exact losses by forward pass, and accept the best replacement. This makes GCG-GPR a drop-in acceleration of GCG's candidate-generation step. Standard GCG requires a backward pass at every iteration to obtain $\nabla_{u_r}\mathcal{L}(a)$ or $\nabla_{\mathbf{e}_{a_r}}\mathcal{L}(a)$; GCG-GPR instead obtains an approximate gradient from a forward-pass representation and a lightweight linear map. Importantly, because the final token replacement is still chosen using the true loss, the method does not require the predicted gradient to be exact. It only needs to rank sufficiently many useful replacements into the top-k candidate sets.

A.2 Experimental hyperparameters

The following settings specify the token-level experiments reported in Section 5.2.

- **Data and splits.** For HarmBench, predictor fitting uses the validation split, which contains 80 prompts, and evaluation uses the 320 held-out test prompts. For JailbreakBench, predictor fitting uses the 300-example judge-comparison split; this split is used only for judge benchmarking and is not used as the test-time jailbreak set. JailbreakBench evaluation uses the 100-prompt harmful test set. Prompt sampling uses seed 42.
- **GCG attack settings.** We initialize each attack with the dummy suffix used by the original GCG setup, namely ten repetitions of `!  `. We run GCG for 25 optimization steps, use the top 64 candidate tokens per suffix position, and evaluate candidate substitutions with the default batch size from the GCG implementation. The target continuation is `Sure, here is`. We do not apply additional token filtering or allowed-token constraints.
- **Predictor-training augmentation.** On HarmBench, each validation prompt is first paired with the same dummy suffix used to initialize the exploit, namely ten repetitions of `!` tokens. We then run GCG for 25 steps from this initial suffix and record the initial suffix state together with each subsequent GCG iterate, giving 26 suffix states per prompt and $80 \times 26 = 2{,}080$ trajectory examples. To further enlarge the predictor-training set, for each trajectory example we generate seven additional variants by randomly changing one suffix token, while also keeping the original trajectory example. This gives $8 \times 2{,}080 = 16{,}640$ HarmBench predictor-training examples.
- **GCG-GPR predictor settings.** We use hidden layer $\ell_0 = 18$ for gradient prediction. The predictor normalizes each true suffix gradient to unit norm per token, flattens hidden states and gradients across token positions, standardizes hidden-state features coordinatewise, appends a bias feature, and fits a full-rank weighted linear ridge regression. All stored examples use unit sample weight. The ridge parameter is 100. The augmentation procedure is GCG-trajectory augmentation plus random one-token suffix perturbations as described above.
- **Blind random baseline.** For each evaluation prompt, we draw one suffix by independently sampling every token in the GCG suffix region uniformly from the tokenizer's non-special vocabulary supported by the target model. We use this suffix directly, with no gradient computation, target-loss evaluation, candidate selection, or iterative search. We report ASR only; perturbation and successful-attack throughput are left undefined.
- **Evaluation and timing.** We use the HarmBench custom judge for HarmBench and JailbreakJudge for JailbreakBench; JailbreakJudge is Llama-3-70B-Instruct quantized and served with Ollama. Attacked-model responses use default generation parameters. ASR is the number of successful attacks divided by the total number of evaluation prompts. We use the default timing boundary from the attack implementation, without an additional timing exclusion. Experiments run on A100 GPUs using bfloat16 precision. Confidence intervals use bootstrap resampling with $B = 10{,}000$.



(a) ASR vs. ϵ for exact and GPR variants.

(b) Augmentation weight-decay ablation.

Figure 4: Sensitivity to perturbation budget ϵ and predictor-training augmentation weight decay.

B Experimental details - embedding attacks

B.1 Hyperparameters

Hyperparameter Selection All hyperparameters are selected on a disjoint dataset split, not used for testing. Tuning proceeds greedily through three phases, each fixing the winners of the previous one and selecting by GPR successful-attack throughput (ASR$\times|\mathcal{D}|$/wall-time). **Phase 1** (predictor quality) sweeps the joint grid target layer $\times$ aug. steps $\times$ aug. weight decay $\in$ $9, 12, 15, 18, 21, 24 \times 0, 5, 10, 15, 20, 30 \times 0.7, 0.8, 0.9, 0.95, 1.0$ at fixed ϵ=0.005, pruning the redundant decay axis when aug. steps$=0$. **Phase 2** (perturbation budget) sweeps $\epsilon \in 0.001, 0.002, 0.003, 0.005, 0.007, 0.01, 0.015, 0.02$ for ℓ_∞ methods and $\epsilon \in 0.25, 0.5, 0.75, 1, 1.5, 2, 3, 5$ for ℓ_2 methods, choosing the best ϵ per norm. **Phase 3** (method-specific HPs) then tunes (a) the PGD inner step size over $\epsilon/N, 2\epsilon/N, 10^{-3}, 2\times10^{-3}$ with N=5, independently for vanilla PGD and PGD-GPR; (b) the RS-FGSM step multiplier $\alpha/\epsilon \in 0.5, 0.75, 1, 1.25, 1.5, 2, 2.5$; and (c) the PGD step-count Pareto frontier $N \in 1, 2, 3, 5, 7, 10, 15, 20$.

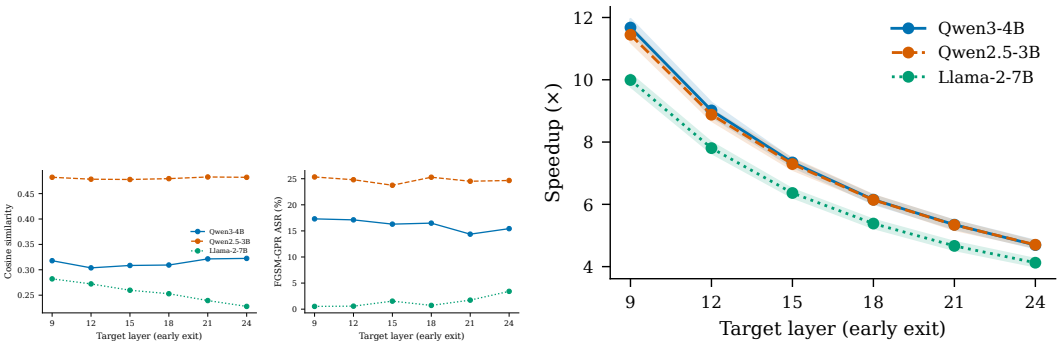
Hyperparameter Sensitivity. Figure 4a ablates the perturbation budget ϵ and Figure 4b ablates the augmentation weight decay used during predictor training; both knobs trade off ASR against gradient-prediction quality.

Mixed-gradient evaluation protocol. Figure 3 holds predictor-fitting and attack settings fixed across mixing weights. HarmBench uses the same non-trivial subsets as Table 1: 294 prompts for Qwen2.5-3B and 280 for Llama-2-7B, excluding prompts for which the unperturbed model already succeeds. JailbreakBench uses the same held-out prompts as Table 3. JailbreakBench evaluations use a shared Llama-3-70B-Instruct Q4_K_M judge with fixed decoding settings. Each unique behavior–response pair receives one cached verdict, reused across the table and mixed-gradient curves. Saved responses and perturbation-generation timings are retained, and statistics are recomputed using the corresponding prompt subsets and shared verdicts. Both curve endpoints and their ASR confidence intervals match the corresponding table entries. Bootstrap intervals quantify prompt-resampling uncertainty with judge verdicts held fixed.

Layer choice. The early-exit layer ℓ trades depth (richer representation) against speed. Figure 5a shows predictor cosine similarity and resulting ASR as a function of ℓ, and Figure 5b shows the corresponding speedup over full backpropagation. Cosine similarity is not a perfect proxy for ASR (Figure 6), motivating hyperparameter selection by successful-attack throughput.

B.2 Embedding-Level JailbreakBench Results

See Table 3 for the JailbreakBench results.



(a) Predictor cosine and ASR vs. target layer ℓ.

(b) Speedup over full backprop vs. target layer ℓ.

Figure 5: Effect of the early-exit layer ℓ.

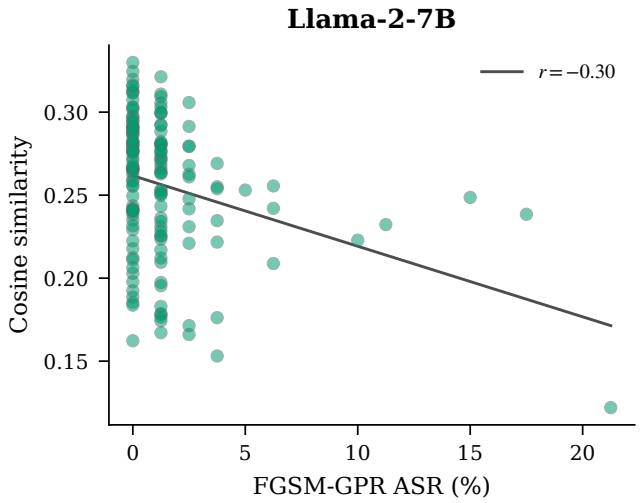


Figure 6: Cosine similarity of predicted gradients vs. resulting ASR on Llama-2-7B. The relationship is noisy, showing that cosine isn't the best choice for tuning hyperparameters.

B.3 Cross-benchmark transfer and predictor training cost

We test whether the continuous-gradient predictor transfers beyond its training distribution in a controlled FGM experiment on Qwen2.5-3B. The JailbreakBench (JBB) harmful set is shuffled with seed 0: the first 50 prompt–target pairs are used for training and the remaining 50 for evaluation. The HarmBench (HB) predictor is trained on its 80 validation prompts and evaluated on the 294 non-trivial Qwen2.5 test prompts. Within each evaluation set, the attack configuration is fixed (layer 9, 20 augmentation steps, augmentation weight decay 1, and ℓ_2 budget $\epsilon = 0.25$); only the predictor-training benchmark changes. Timing uses one explicit untimed warm-up step before all measured examples. Confidence intervals are computed by 10,000 per-prompt bootstrap resamples.

Transfer is asymmetric in this setting. On JBB, in-domain training gives higher ASR than the HB-trained predictor; on HB, the JBB-trained predictor gives higher ASR than the in-domain predictor despite lower cosine similarity and sign agreement. Thus these predictor-quality diagnostics should be interpreted alongside direct measurements of attack performance. Both transferred predictors nevertheless achieve substantially higher successful-attack throughput than exact FGM in these evaluations. This experiment probes transfer across the joint prompt–target distributions of two benchmarks for one fixed model and attack configuration. Broader evaluations would be needed to assess model-, objective-, or target-agnostic gradient prediction. The table also makes explicit that GPR is intended for workloads large enough to amortize its one-time fitting cost.

Table 3: Attack success rate (ASR), perturbation throughput (Pert., examples/s), and success throughput (Succ.) on the non-trivial subset of JailbreakBench held-out prompts (Qwen3-4B: 50, Qwen2.5-3B: 50, Llama-2-7B: 50). Saved responses are rejudged with shared per-pair verdicts (Appendix B.1). 95% bootstrap CIs from per-prompt resampling ($B = 10{,}000$).

	Qwen3-4B			Qwen2.5-3B			Llama-2-7B		
Method	ASR (%)	Pert. (ex/s)	Succ. (att/s)	ASR (%)	Pert. (ex/s)	Succ. (att/s)	ASR (%)	Pert. (ex/s)	Succ. (att/s)
Exact gradient									
FGSM	$6.0^{+8.0}_{-6.0}$	$6.0^{+0.0}_{-0.0}$	$\mathbf{0.36^{+0.48}_{-0.36}}$	$22.0^{+12.0}_{-10.0}$	$7.5^{+0.1}_{-0.1}$	$1.65^{+0.89}_{-0.75}$	$6.0^{+8.0}_{-6.0}$	$8.8^{+0.1}_{-0.1}$	$\mathbf{0.53^{+0.69}_{-0.53}}$
FGM	$0.0^{+0.0}_{-0.0}$	$5.6^{+0.1}_{-0.2}$	$\mathbf{0.00^{+0.00}_{-0.00}}$	$8.0^{+8.0}_{-6.0}$	$6.9^{+0.2}_{-0.4}$	$0.55^{+0.58}_{-0.42}$	$0.0^{+0.0}_{-0.0}$	$8.9^{+0.1}_{-0.1}$	$\mathbf{0.00^{+0.00}_{-0.00}}$
RS-FGSM	$2.0^{+4.0}_{-2.0}$	$5.9^{+0.1}_{-0.1}$	$\mathbf{0.12^{+0.24}_{-0.12}}$	$22.0^{+12.0}_{-10.0}$	$7.4^{+0.1}_{-0.1}$	$1.63^{+0.89}_{-0.75}$	$12.0^{+10.0}_{-8.0}$	$8.5^{+0.1}_{-0.1}$	$\mathbf{1.03^{+0.85}_{-0.68}}$
PGD-2	$4.0^{+6.0}_{-4.0}$	$3.0^{+0.0}_{-0.1}$	$\mathbf{0.12^{+0.18}_{-0.12}}$	$10.0^{+8.0}_{-8.0}$	$3.7^{+0.0}_{-0.0}$	$\mathbf{0.37^{+0.30}_{-0.30}}$	$14.0^{+10.0}_{-8.0}$	$4.3^{+0.0}_{-0.0}$	$\mathbf{0.60^{+0.43}_{-0.35}}$
PGD-5	$6.0^{+8.0}_{-6.0}$	$1.2^{+0.0}_{-0.0}$	$\mathbf{0.07^{+0.10}_{-0.07}}$	$40.0^{+14.0}_{-14.0}$	$1.5^{+0.0}_{-0.0}$	$0.60^{+0.21}_{-0.21}$	$24.0^{+12.0}_{-12.0}$	$1.7^{+0.0}_{-0.0}$	$\mathbf{0.42^{+0.21}_{-0.21}}$
PGD-10	$30.0^{+14.0}_{-12.0}$	$0.6^{+0.0}_{-0.0}$	$\mathbf{0.18^{+0.08}_{-0.07}}$	$60.0^{+14.0}_{-14.0}$	$0.8^{+0.0}_{-0.0}$	$0.45^{+0.10}_{-0.11}$	$26.0^{+12.0}_{-12.0}$	$0.9^{+0.0}_{-0.0}$	$\mathbf{0.23^{+0.11}_{-0.10}}$
GPR (predicted)									
FGSM	$0.0^{+0.0}_{-0.0}$	$59.4^{+0.6}_{-0.7}$	$0.00^{+0.00}_{-0.00}$	$4.0^{+6.0}_{-4.0}$	$94.6^{+1.9}_{-2.1}$	$\mathbf{3.78^{+5.75}_{-3.78}}$	$0.0^{+0.0}_{-0.0}$	$70.2^{+0.8}_{-0.8}$	$0.00^{+0.00}_{-0.00}$
FGM	$0.0^{+0.0}_{-0.0}$	$55.7^{+0.9}_{-1.1}$	$0.00^{+0.00}_{-0.00}$	$22.0^{+12.0}_{-12.0}$	$64.6^{+1.0}_{-1.1}$	$\mathbf{14.21^{+7.70}_{-7.60}}$	$0.0^{+0.0}_{-0.0}$	$46.1^{+0.5}_{-0.5}$	$0.00^{+0.00}_{-0.00}$
RS-FGSM	$0.0^{+0.0}_{-0.0}$	$59.0^{+0.7}_{-0.9}$	$0.00^{+0.00}_{-0.00}$	$8.0^{+8.0}_{-6.0}$	$94.8^{+1.5}_{-1.6}$	$\mathbf{7.58^{+7.65}_{-5.70}}$	$0.0^{+0.0}_{-0.0}$	$70.0^{+0.9}_{-1.1}$	$0.00^{+0.00}_{-0.00}$
PGD-2	$0.0^{+0.0}_{-0.0}$	$31.2^{+0.3}_{-0.4}$	$0.00^{+0.00}_{-0.00}$	$0.0^{+0.0}_{-0.0}$	$51.6^{+0.6}_{-0.7}$	$0.00^{+0.00}_{-0.00}$	$0.0^{+0.0}_{-0.0}$	$37.6^{+0.3}_{-0.3}$	$0.00^{+0.00}_{-0.00}$
PGD-5	$0.0^{+0.0}_{-0.0}$	$13.0^{+0.1}_{-0.1}$	$0.00^{+0.00}_{-0.00}$	$4.0^{+6.0}_{-4.0}$	$22.3^{+0.2}_{-0.2}$	$\mathbf{0.89^{+1.35}_{-0.89}}$	$0.0^{+0.0}_{-0.0}$	$15.7^{+0.1}_{-0.1}$	$0.00^{+0.00}_{-0.00}$
PGD-10	$0.0^{+0.0}_{-0.0}$	$6.6^{+0.0}_{-0.0}$	$0.00^{+0.00}_{-0.00}$	$16.0^{+10.0}_{-10.0}$	$11.3^{+0.1}_{-0.1}$	$\mathbf{1.81^{+1.15}_{-1.13}}$	$0.0^{+0.0}_{-0.0}$	$8.0^{+0.1}_{-0.1}$	$0.00^{+0.00}_{-0.00}$

Evaluation	Predictor training	ASR (%)	Succ. (att/s)	Cosine	Sign agreement (%)
JailbreakBench ($n = 50$)	Exact gradient	$8.0^{+8.0}_{-6.0}$	$0.58^{+0.60}_{-0.44}$	–	–
	JBB (in-domain)	$22.0^{+12.0}_{-12.0}$	$16.30^{+8.97}_{-8.63}$	$0.43^{+0.02}_{-0.02}$	$64.5^{+0.5}_{-0.5}$
	HB (transferred)	$16.0^{+10.0}_{-10.0}$	$12.18^{+7.70}_{-7.59}$	$0.37^{+0.02}_{-0.02}$	$61.3^{+0.5}_{-0.5}$
HarmBench ($n = 294$)	Exact gradient	$6.1^{+3.1}_{-2.7}$	$0.45^{+0.22}_{-0.20}$	–	–
	HB (in-domain)	$12.6^{+3.7}_{-3.7}$	$9.61^{+2.88}_{-2.84}$	$0.39^{+0.01}_{-0.01}$	$61.0^{+0.2}_{-0.2}$
	JBB (transferred)	$20.7^{+4.8}_{-4.4}$	$15.79^{+3.67}_{-3.43}$	$0.33^{+0.01}_{-0.01}$	$59.4^{+0.2}_{-0.2}$

Evaluation	Dataset	Predictor Training (s)	Break-even examples
JailbreakBench	JBB (in-domain)	159.2	1280
JailbreakBench	HB (transferred)	259.7	2082
HarmBench	HB (in-domain)	259.7	2113
HarmBench	JBB (transferred)	159.2	1296

Table 4: Controlled cross-benchmark transfer for continuous FGM (top) and its one-time predictor-training cost (bottom). Successful-attack throughput is the number of successful attacks divided by total perturbation time. Cosine similarity and sign agreement are reported for predicted-gradient conditions, comparing each prediction with its corresponding exact gradient. The break-even count is $T_{\text{cal}}/(\bar{t}_{\text{exact}} - \bar{t}_{\text{GPR}})$. Intervals are 95% bootstrap CIs ($B = 10{,}000$).

B.4 Further Details on Throughput

Throughput comparison. Figure 7 contrasts per-method successful-attack throughput between exact and GPR variants on a linear scale.

C Additional background on GP derivatives

This appendix records a standard GP derivative-conditioning formula used as motivation in the main text. Let $f : \mathbb{R}^d \to \mathbb{R}^m$ be a (vector-valued) Gaussian process with mean $m(\cdot)$ and kernel $K(\cdot, \cdot)$ so that $K(x, x') = \mathrm{Cov}(f(x), f(x'))$ (Williams & Rasmussen, 2006). Assume m is continuously differentiable and K is twice continuously differentiable (entrywise). Then the partial-derivative process $g_j(x) \coloneqq \partial f(x)/\partial x_j \in \mathbb{R}^m$ exists (in a mean-square sense) and (f, g_j) is jointly Gaussian (Williams & Rasmussen, 2006, Chapter 9.4). Given a finite input set $X = \{x_1, \ldots, x_n\}$ and noiseless

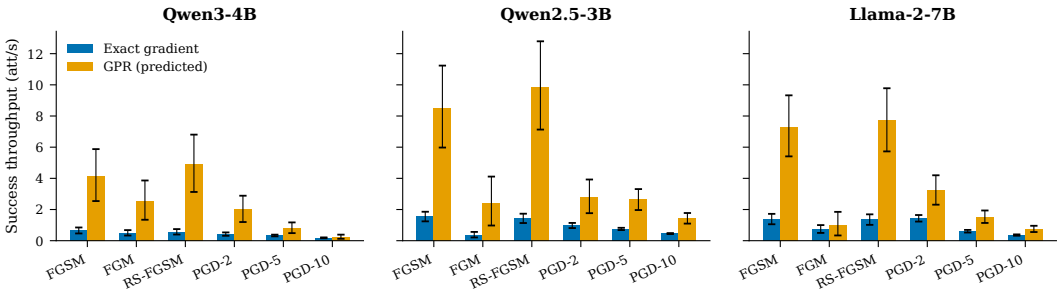


Figure 7: Per-method successful-attack throughput on HarmBench.

observations $y = f(X)$, assume $K_{ff}(X, X)$ is invertible. Conditioning the joint Gaussian then yields the posterior mean of the derivative at $x_\star$:

$$\mathbb{E}[g_j(x_\star) \mid f(X) = y] = m_{g_j}(x_\star) + K_{g_j f}(x_\star, X)\, K_{ff}(X, X)^{-1}\big(y - m_f(X)\big), \tag{10}$$

with $m_f(X)$ the stacked mean at X, $m_{g_j}(x_\star) = \partial m(x_\star)/\partial x_j$, and $\big[K_{g_j f}(x_\star, X)\big]_i = \frac{\partial}{\partial x_{\star,j}} K(x_\star, x_i)$ (blockwise).

Exact conditional means in the NTK limit. Fix regression data $\mathcal{D} = (X, y)$, with all output coordinates stacked in y. Let $\mathcal{K}$ be the deterministic limiting NTK and let the centered initialization f_0 have GP covariance K_0. Assume $\mathcal{K}_{XX} := \mathcal{K}(X, X)$ is invertible. Squared-loss gradient flow in the NTK limit gives (Jacot et al., 2018, Section 5)

$$f_\tau(x) = f_0(x) + Q_\tau(x)\big(y - f_0(X)\big), \qquad Q_\tau(x) = \mathcal{K}(x, X)\mathcal{K}_{XX}^{-1}\big(I - e^{-\tau \mathcal{K}_{XX}}\big).$$

The infinite-time limit replaces the last factor by I. For fixed data, this is an affine transformation of f_0, so f_τ is a GP over random initializations. Its mean is $Q_\tau(x)y$; an individual trained function also retains the centered random term $f_0(x) - Q_\tau(x)f_0(X)$. Its covariance is induced by K_0 and Q_τ, and is generally not $\mathcal{K}$. Assume K_0 and $\mathcal{K}$ are twice continuously differentiable and take $H_\tau(x) = f_\tau(x)$ and $G_\tau(x) = \nabla_x(c^\top f_\tau(x))$ for fixed c. Their joint Gaussianity follows from mean-square differentiation of this limiting process. Writing their pointwise means and centered covariance blocks as $\mu_{H,\tau}(x)$, $\mu_{G,\tau}(x)$, $C_{HH,\tau}(x)$, and $C_{GH,\tau}(x)$ gives

$$\mathbb{E}[G_\tau(x) \mid H_\tau(x)] = A_\tau(x)H_\tau(x) + b_\tau(x),$$

$$A_\tau(x) = C_{GH,\tau}(x)C_{HH,\tau}(x)^{-1}, \qquad b_\tau(x) = \mu_{G,\tau}(x) - A_\tau(x)\mu_{H,\tau}(x),$$

where $C_{HH,\tau}(x)$ is invertible; a Moore–Penrose inverse gives the corresponding conditional mean when it is singular. This is an exact pointwise conditional-mean result. Input-independent coefficients follow under the additional moment conditions of Proposition 1. Stationarity of the initialization or NTK kernel alone does not establish these conditions after training on fixed data. Expectations here are over initialization, not over prompts for a single fixed trained network.

Exact recovery in a fixed-feature NTK model. Separately, let $\phi(x) = (\cos x, \sin x)^\top$ and let the final regression outputs be $f_B(x) = B\phi(x) \in \mathbb{R}^2$, with only $B \in \mathbb{R}^{2\times 2}$ trained. Its matrix NTK is $\mathcal{K}(x, x') = \cos(x - x')I_2$, independent of B; thus squared-loss training is exactly fixed-kernel training. For any fitted B that is invertible, set $H(x) = f_B(x)$ and $J = \left[\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right]$. Then

$$\frac{df_B(x)}{dx} = BJ\phi(x) = BJB^{-1}H(x).$$

Hence each output's input derivative is recovered exactly by a model-specific linear map of the final outputs. The identity follows from $\phi' = J\phi$ and holds after training whenever B remains invertible. This is a restricted exact-recovery example, not a consequence of Gaussianity alone or a guarantee for arbitrary NTK architectures or finite-sample ridge estimates.

D PROOFS

Proof of Lemma 1. Write $\eta = \{(\omega_j, u_j, v_j)\}_{j=1}^m$ for the embedding parameters. Fix a translation $c \in \mathbb{R}^d$ and define

$$Q(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}, \qquad \begin{bmatrix} u_j' \\ v_j' \end{bmatrix} = Q(\omega_j^\top c) \begin{bmatrix} u_j \\ v_j \end{bmatrix}.$$

Let $\eta' = \{(\omega_j, u_j', v_j')\}_{j=1}^m$. The identity $Q(\alpha+\delta) = Q(\alpha)Q(\delta)$ gives

$$\begin{aligned} e_j(x+c;\eta) &= Q(\omega_j^\top x + \omega_j^\top c) \begin{bmatrix} u_j \\ v_j \end{bmatrix} \\ &= Q(\omega_j^\top x) \begin{bmatrix} u_j' \\ v_j' \end{bmatrix} = e_j(x;\eta'). \end{aligned}$$

Thus $e(x+c;\eta) = e(x;\eta')$ simultaneously for all x. Conditional on the frequencies, the transformation applies deterministic orthogonal rotations to independent standard Gaussian pairs. Their joint law is unchanged. Hence $\eta' \overset{d}{=} \eta$, and independence from Θ gives

$$(\eta', \Theta) \overset{d}{=} (\eta, \Theta).$$

Because the network depends on x only through $e(x)$,

$$H(x+c;\eta,\Theta) = H(x;\eta',\Theta), \qquad Y(x+c;\eta,\Theta) = Y(x;\eta',\Theta).$$

These identities use the same transformed parameters for every input. Equality of the parameter laws therefore proves joint stationarity of (H, Y).

Finite second moments give constant means and, for $U, V \in \{H, Y\}$,

$$\mathrm{Cov}(U(x), V(x')) = \mathrm{Cov}(U(x-x'), V(0)).$$

Thus all auto- and cross-covariances depend only on $x - x'$. Likewise,

$$k(x+c, x'+c) = \mathbb{E}[Y(x+c)Y(x'+c)] = \mathbb{E}[Y(x)Y(x')] = k(x, x').$$

Finally, differentiate the pathwise identity $Y(x+c;\eta,\Theta) = Y(x;\eta',\Theta)$ with respect to x. Since η' depends on c but not on x, almost-sure differentiability yields

$$G(x+c;\eta,\Theta) = G(x;\eta',\Theta).$$

Equality of the parameter laws and integrability now imply

$$\mu_G(x+c) = \mathbb{E}[G(x+c;\eta,\Theta)] = \mathbb{E}[G(x;\eta',\Theta)] = \mu_G(x).$$

Thus μ_G is constant. This expectation is the unconditional prior mean over (η, Θ), not the conditional prediction $\mathbb{E}[G(x) \mid H(x)]$. No interchange of differentiation and expectation is needed. □

Proof of Lemma 2. Fix a translation c. The coefficient rotation in the proof of Lemma 1 replaces the embedding parameters η by η' without changing their joint law with Θ, and gives

$$H(x+c;\eta,\Theta) = H(x;\eta',\Theta), \qquad Y(x+c;\eta,\Theta) = Y(x;\eta',\Theta).$$

Since η' does not depend on x, differentiating the second identity gives

$$G(x+c;\eta,\Theta) = G(x;\eta',\Theta).$$

Consequently,

$$(H(x+c), G(x+c)) \overset{d}{=} (H(x), G(x)).$$

The finite second moments ensure that the covariances exist, and equality of these joint distributions makes both $\Sigma_{HH}(x)$ and $\Sigma_{GH}(x)$ translation-invariant, hence constant. □

Proof of Proposition 1. Fix x. Gaussian conditioning gives

$$\mathbb{E}[G(x) \mid H(x)] = \mu_G + \Sigma_{GH}\Sigma_{HH}^{-1}(H(x) - \mu_H) = AH(x) + b.$$

The means and covariance blocks are input-independent, so the same A, b work for every x. The conditional mean minimizes squared error among all measurable predictors based on $H(x)$.

The residual $G(x) - AH(x) - b$ has mean zero and covariance

$$\mathrm{Cov}(G(x), G(x)) - \Sigma_{GH}\Sigma_{HH}^{-1}\Sigma_{GH}^{\top}.$$

It vanishes almost surely exactly when this covariance is zero, which proves the exact-recovery criterion. □

Proof of observation 1. Write

$$\mu_H = \mathbb{E}[H], \qquad \mu_G = \mathbb{E}[G], \qquad \Sigma_{HH} = \mathrm{Cov}(H, H), \qquad \Sigma_{GH} = \mathrm{Cov}(G, H).$$

The proposed coefficients and their prediction residual are

$$A = \Sigma_{GH}\Sigma_{HH}^{-1}, \qquad b = \mu_G - A\mu_H, \qquad \varepsilon = G - AH - b.$$

All expectations below exist because H and G have finite second moments.

Step 1: the residual has mean zero. The coefficients A and b are deterministic, so linearity of expectation gives

$$\begin{aligned}\mathbb{E}[\varepsilon] &= \mathbb{E}[G - AH - b] \\ &= \mathbb{E}[G] - A\mathbb{E}[H] - b \\ &= \mu_G - A\mu_H - (\mu_G - A\mu_H) \\ &= 0.\end{aligned}$$

Thus the choice of intercept b ensures that the prediction has the same mean as G.

Step 2: the residual is orthogonal to every coordinate of H. Substituting the definition of b, we can write the residual using centered variables:

$$\begin{aligned}\varepsilon &= G - AH - (\mu_G - A\mu_H) \\ &= (G - \mu_G) - A(H - \mu_H).\end{aligned}$$

Also, $H = (H - \mu_H) + \mu_H$. Therefore,

$$\begin{aligned}\mathbb{E}[\varepsilon H^{\top}] &= \mathbb{E}[\varepsilon(H - \mu_H)^{\top}] + \mathbb{E}[\varepsilon]\mu_H^{\top} \\ &= \mathbb{E}[\varepsilon(H - \mu_H)^{\top}] \\ &= \mathbb{E}\big[\big((G - \mu_G) - A(H - \mu_H)\big)(H - \mu_H)^{\top}\big] \\ &= \mathbb{E}[(G - \mu_G)(H - \mu_H)^{\top}] - A\mathbb{E}[(H - \mu_H)(H - \mu_H)^{\top}] \\ &= \Sigma_{GH} - A\Sigma_{HH} \\ &= \Sigma_{GH} - \Sigma_{GH}\Sigma_{HH}^{-1}\Sigma_{HH} \\ &= 0.\end{aligned}$$

The second equality uses Step 1, and the fifth uses the definition of covariance. In particular, $\mathbb{E}[\varepsilon_i H_j] = 0$ for every pair of coordinates i, j. This is an orthogonality statement, not a claim that ε and H are independent.

Step 3: compare with any other affine predictor. Let $CH + v$ be an arbitrary competing affine predictor, with deterministic coefficients C, v. Define the difference between the proposed prediction and the competing prediction by

$$\Delta(H) := (AH + b) - (CH + v) = (A - C)H + (b - v).$$

The competing predictor's error is then

$$\begin{aligned}G - CH - v &= (G - AH - b) + (AH + b - CH - v) \\ &= \varepsilon + \Delta(H).\end{aligned}$$

Expanding its expected squared norm gives

$$\begin{aligned}\mathbb{E}\|G - CH - v\|_2^2 &= \mathbb{E}\big[(\varepsilon + \Delta(H))^\top(\varepsilon + \Delta(H))\big] \\ &= \mathbb{E}[\varepsilon^\top\varepsilon] + \mathbb{E}[\varepsilon^\top\Delta(H)] + \mathbb{E}[\Delta(H)^\top\varepsilon] + \mathbb{E}[\Delta(H)^\top\Delta(H)] \\ &= \mathbb{E}\|\varepsilon\|_2^2 + 2\mathbb{E}[\varepsilon^\top\Delta(H)] + \mathbb{E}\|\Delta(H)\|_2^2.\end{aligned}$$

The two cross terms are equal because they are transposes of the same real scalar.

To see explicitly why their expectation is zero, write $G, \varepsilon \in \mathbb{R}^d$ and $H \in \mathbb{R}^{d_h}$. Then

$$\begin{aligned}\mathbb{E}[\varepsilon^\top\Delta(H)] &= \mathbb{E}[\varepsilon^\top(A - C)H] + \mathbb{E}[\varepsilon^\top(b - v)] \\ &= \sum_{i=1}^{d}\sum_{j=1}^{d_h}(A - C)_{ij}\mathbb{E}[\varepsilon_i H_j] + \sum_{i=1}^{d}(b - v)_i\mathbb{E}[\varepsilon_i] \\ &= 0.\end{aligned}$$

The first sum vanishes by Step 2, and the second by Step 1.

Substituting this into the error expansion yields

$$\begin{aligned}\mathbb{E}\|G - CH - v\|_2^2 &= \mathbb{E}\|\varepsilon\|_2^2 + \mathbb{E}\|\Delta(H)\|_2^2 \\ &= \mathbb{E}\|G - AH - b\|_2^2 + \mathbb{E}\|(A - C)H + (b - v)\|_2^2 \\ &\geq \mathbb{E}\|G - AH - b\|_2^2.\end{aligned}$$

The additional term is nonnegative and equals zero when $C = A$ and $v = b$. Hence $AH + b$ minimizes expected squared error among all affine predictors. No Gaussianity or stationarity assumption is used. □