Locking Pretrained Weights via Deep Low-Rank Residual Distillation

Keitaro Sakamoto* **Pierre Ablin** **Federico Danieli** **Marco Cuturi**
Apple

Abstract

The quality of open-weight language models has dramatically improved in recent years. Sharing weights greatly facilitates model adoption by enabling their use across diverse hardware and software platforms. They also allow for more open research and testing, to the extent that users can use them as checkpoints, fine-tune them according to their needs, and potentially redistribute them. In some cases, however, concerns on modifying these weights towards unauthorized uses may outweigh the pros of giving users such freedom. Defending against such adaptation is non-trivial: since an adaptive attacker can observe all weights and architectures by definition, they can reverse simple structural defenses, and use optimization to defeat the simplest locking mechanisms. In this work, we exploit the inference–training asymmetry of automatic differentiation as a novel defense axis. We propose DLR-LOCK, a method where the purveyor of the model purposely replaces each pretrained MLP in their model with a deep low-rank residual network (DLR-Net) of comparable parameter count, forcing activation memory that grows linearly with depth during backpropagation. DLR-Nets are efficiently trained via module-wise regression. We show that, beyond this memory overhead, DLR-LOCK results in architectural mismatches that complicate the optimization landscape of standard fine-tuning, and a backward pass that incurs disproportionately more overhead than the forward pass. Our defense succeeds in withstanding adaptive attackers with full knowledge of the defense strategy while preserving the original model's capabilities. Experiments on LLMs validate these claims.

1 Introduction

Pretrained large language models represent a substantial investment of compute, data curation, and engineering effort. When released as open weights, the model provider loses control over how these weights are used: any party can adapt the model to new domains or remove usage restrictions, often at a fraction of the original training cost [Kapoor et al., 2024, Bommasani et al., 2024, Casper et al., 2026]. Open-weight licenses provide legal protection against such misuse, but no technical enforcement. These concerns motivate a fundamental scientific question: *can a neural network be locked, so that adaptation is inherently more expensive than adapting the original model, while retaining its inference time performance?* This pressing question is far from straightforward: simple structural defenses that exploit weight symmetries can be trivially reverted by an adaptive attacker who inspects the model. One might hope to use optimization to embed the defense more deeply into the loss landscape, but this strategy is inherently self-defeating: the defender must use optimization to find a solution that resists optimization, yet the same gradient information available to the defender is equally available to the attacker. Recent model locking methods [Rosati et al., 2025, Wang et al., 2026b, Rosati et al., 2026] face this fundamental difficulty.

*Work done during an internship at Apple.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

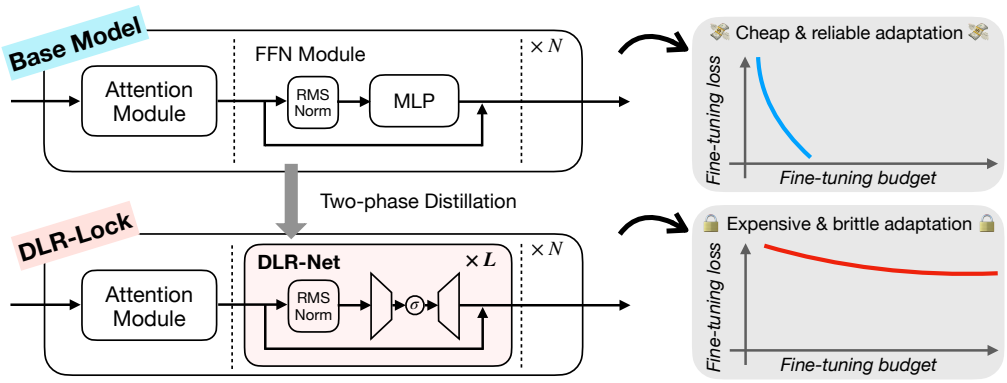


Figure 1: Overview of DLR-LOCK. Each MLP in the original model (top) is replaced with a functionally equivalent deep low-rank residual network (DLR-Net, bottom) via a two-phase distillation. The replacement introduces three locking mechanisms: (1) activation memory growth, (2) optimization instability from architectural mismatch, and (3) disproportionate backward-pass overhead.

In this work, we exploit a different defense axis: the inference–training asymmetry of automatic differentiation [Griewank and Walther, 2008]. Inference requires only a forward pass through the model, discarding all intermediate activations; training, by contrast, must store those intermediates for backpropagation. To exploit this weakness, we propose to replace each pretrained MLP in a large network with a deep low-rank residual network (DLR-Net) that matches the original input–output mapping via distillation (vector regression) with comparable parameter count.[2] The replacement introduces three locking mechanisms: (1) activation memory that grows linearly with depth; (2) architectural mismatch between the deep residual networks and the original MLPs destabilizes standard fine-tuning recipes; and (3) the backward pass is disproportionately slowed compared to the forward pass due to heavier memory access and more complex computation. The locked model thus preserves inference quality while imposing substantial additional cost on any party attempting to adapt it. Our locking mechanism is *fine-tuning-domain agnostic*: it raises the cost of fine-tuning on any target domain, not only on potentially harmful ones [Huang et al., 2024, Tamirisa et al., 2025]. On the one hand, users fine-tuning for inoffensive applications pay a higher cost. On the other hand, no malicious fine-tuning dataset avoids that cost. Open weights confer portability across hardware and software platforms, inspection and auditing, reuse as checkpoints, and fine-tuning; DLR-LOCK constrains only the last. Figure 1 illustrates the overall idea; related work is discussed in Appendix A. Our contributions are:

- We formalize the problem of *model locking* as adaptation cost amplification and establish an adaptive threat model grounded in Kerckhoffs's principle, which requires that security rely on structural properties rather than secrecy of the defense mechanism.
- We categorize existing locking approaches as symmetry-based, barrier-based, and penalty-based, and identify inherent limitations of each through analysis and controlled experiments.
- We show that the inference–training asymmetry of automatic differentiation provides a novel, optimizer-agnostic defense axis, and propose DLR-LOCK, a deep low-rank residual locking mechanism with a two-phase distillation pipeline that locks modules with limited quality loss, a residual gap that narrows as the distillation budget grows.
- We validate the approach on Qwen3 [Yang et al., 2025] and on models we pretrain ourselves, at three scales and across several fine-tuning domains, showing that the locked models preserve the original capabilities, and that a custom kernel brings inference latency close to the baseline, while fine-tuning costs more: beyond the increased activation memory expected by design, architectural mismatch destabilizes standard recipes, and the backward pass is disproportionately slowed.
- We evaluate adaptive attacks that exploit knowledge of the defense, including parameter freezing, low-rank adaptation, and reverse-distilling the locked modules back into ordinary MLPs, and find that none recovers cheap adaptation.

[2] Code is available at `https://github.com/apple-aiml-research/ml-dlr-lock`.

2 Model Locking: Formulation and Challenges

Modern large language models are built from a stack of N transformer blocks [Vaswani et al., 2017]. Each block consists of a multi-head self-attention module $\mathrm{Attn}(\cdot)$ followed by a feed-forward network (FFN) module, with a residual connection and RMSNorm [Zhang and Sennrich, 2019] applied before each module, following the pre-norm convention:

$$\boldsymbol{Z} = \boldsymbol{X} + \mathrm{Attn}\big(\mathrm{RMSNorm}(\boldsymbol{X})\big), \quad \boldsymbol{X}' = \boldsymbol{Z} + \mathrm{FFN}\big(\mathrm{RMSNorm}(\boldsymbol{Z})\big). \tag{1}$$

Here $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is a sequence of n token representations of dimension d, $\boldsymbol{Z}$ and $\boldsymbol{X}'$ are intermediate and output sequences of the same shape, and both FFN and RMSNorm act row-wise, i.e., independently on each token. In recent architectures such as LLaMA [Grattafiori et al., 2024] and Qwen [Yang et al., 2025], the FFN is a SwiGLU MLP [Shazeer, 2020]:

$$\mathrm{FFN}(\boldsymbol{x}) = \boldsymbol{W}_{\mathrm{down}}\big((\boldsymbol{W}_{\mathrm{up}}\boldsymbol{x}) \odot \sigma(\boldsymbol{W}_{\mathrm{gate}}\boldsymbol{x})\big),$$

where $\odot$ denotes element-wise multiplication, $\sigma(z) = z \cdot \mathrm{sigmoid}(z)$ is the SiLU activation [Elfwing et al., 2018], $\boldsymbol{W}_{\mathrm{gate}}, \boldsymbol{W}_{\mathrm{up}} \in \mathbb{R}^{d_{\mathrm{ff}}\times d}$, and $\boldsymbol{W}_{\mathrm{down}} \in \mathbb{R}^{d\times d_{\mathrm{ff}}}$. Our method replaces each FFN residual sub-block with a functionally equivalent architecture, leaving all other components unchanged.

2.1 Problem formulation

The term *model locking* was introduced by Rosati et al. [2025] to describe making pretrained weights resistant to fine-tuning; a related concept of *non-fine-tunability* was independently proposed by Wang et al. [2026b]. We formalize this setting below.

Definition 2.1 (Locking Mechanism)**.** Let $f(\cdot;\theta_0)$ be a model obtained by standard pretraining. A *locking mechanism* $\mathcal{M}$ with private information s produces a locked model $g(\cdot;\theta') = \mathcal{M}(f,\theta_0,s)$ that is expected to be harder to fine-tune than $f(\cdot;\theta_0)$.

Assumption 2.2 (Attacker Capabilities)**.** The attacker has: (i) access to the published weights θ', but not to s; (ii) knowledge of $\mathcal{M}$; (iii) a compute budget $C_{\mathrm{attack}} \ll \min\{C_{\mathrm{pretrain}}, C_{\mathrm{distill}}\}$, where C_{distill} is the cost of reconstructing a comparable unlocked model.

Condition (ii) follows Kerckhoffs's principle [Kerckhoffs, 1883]: the defense must remain effective even when the attacker knows exactly how it was constructed. The defender's advantage comes from structural properties of g, not secrecy of $\mathcal{M}$. Condition (iii) bounds what any locking mechanism can achieve: the attacker can pretrain an equivalent model outright at C_{pretrain}, or distill one from g's logits at C_{distill}, which needs only outputs from g. The two are of the same order [Busbridge et al., 2025], and no defense can withstand an attacker at that budget.

The attacker can employ various strategies to adapt the locked model g to a new, potentially harmful, task $\mathcal{T}$ under its budget constraint: fine-tuning the entire model, distilling individual modules, applying analytical transforms to preserve functional equivalence, or any combination thereof. Let $\mathfrak{A}$ denote the set of all feasible attack strategies available to the attacker. Each strategy $\mathcal{A} \in \mathfrak{A}$ incurs a cost $C(\mathcal{A})$ and yields a model $\mathcal{A}(g)$ with performance $\mathrm{perf}_{\mathcal{T}}(\mathcal{A}(g))$ on task $\mathcal{T}$. The *adaptation cost* for reaching a target performance level p is the minimum cost over all strategies that achieve it:

$$C_{\mathcal{T}}(g \to p) \;=\; \min_{\mathcal{A}\in\mathfrak{A}} C(\mathcal{A}) \quad \text{s.t.} \quad \mathrm{perf}_{\mathcal{T}}\big(\mathcal{A}(g)\big) \geq p. \tag{2}$$

Here $\mathcal{A}$ is a procedure the attacker can actually execute, not an oracle that supplies a good configuration: if reaching p requires a search over batch size, sequence length, or learning rate, that search belongs to $\mathcal{A}$ and its trials enter $C(\mathcal{A})$. We can now state the two requirements for model locking.

Definition 2.3 (Model Locking)**.** A defense mechanism $\mathcal{M}$ achieves *model locking* with locking factor $\kappa > 1$ and tolerance $\varepsilon > 0$ if the published model $g = \mathcal{M}(f,\theta_0,s)$ satisfies:

(i) **Utility preservation.** $\mathcal{L}(g) < \mathcal{L}(f) + \varepsilon$ for every downstream task loss $\mathcal{L}$.

(ii) **Adaptation cost amplification.** For any downstream task $\mathcal{T}$ and target performance level p,

$$C_{\mathcal{T}}(g \to p) \;\geq\; \kappa \cdot C_{\mathcal{T}}(f \to p),$$

where $C_{\mathcal{T}}(\cdot \to p)$ is measured in GPU-hours, what the attacker actually pays.

Condition *(i)* requires that the locked model retain general capabilities, not merely specific benchmark scores. Definition 2.3 places no constraint on the locking cost C_{lock}, since the defender can afford a one-time cost that is amortized over all deployments.

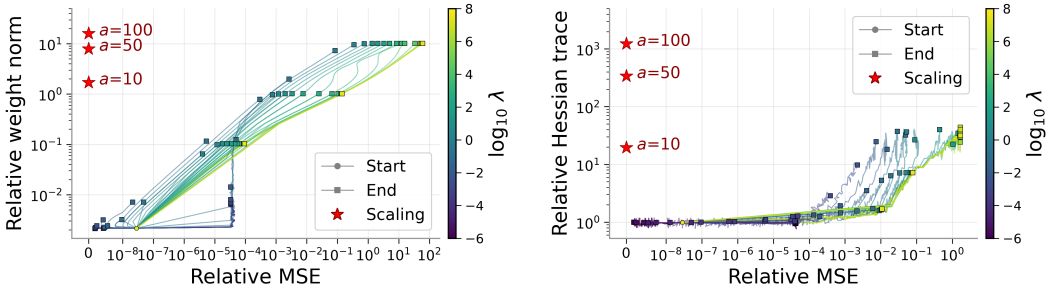


Figure 2: Locking solutions exist but gradient descent cannot reach them. We optimize $\min_{\Delta\theta} \mathrm{MSE} - \lambda\,\Omega$ on a two-layer ReLU MLP trained on MNIST, sweeping over 29 values of λ and three learning rates with SGD and gradient norm clipping. Each curve is one trajectory; color encodes $\log_{10}\lambda$. Red stars: weight-symmetry solutions $(a^{-1}\boldsymbol{W}_1,\, a\boldsymbol{W}_2)$. **Left**: $\Omega = \|\Delta\theta\|^2$; y-axis is $\|\Delta\theta\|/\|\theta_0\|$. **Right**: $\Omega = \mathrm{Tr}(H_{\mathrm{CE}})$; y-axis is relative Hessian trace $\exp\,\mathbb{E}_x[\log\,\mathrm{Tr}(H_i(\theta))/\mathrm{Tr}(H_i(\theta_0))]$ (H_i: per-sample CE Hessian). x-axis (both): relative MSE, $\exp\,\mathbb{E}_x[\log\,\|f(\theta,x)-f(\theta_0,x)\|^2/\|f(\theta_0,x)\|^2]$.

Remark 2.4 (Amortization across adaptations)**.** The factor κ compares single adaptations from the published checkpoint. In practice an attacker may adapt to many tasks, or several may cooperate, with one recovering an unlocked model and distributing it to the rest. Such a one-off recovery is then amortized rather than paid per task, a route we take up in Section 3.3.

2.2 Why model locking is fundamentally difficult

Before introducing our method, let us consider what approaches to model locking are available and why they face fundamental difficulties. We provide a detailed comparison with existing methods in Appendix A, in light of the discussion below. A natural starting point for model locking is to exploit *weight symmetries*: transformations of the parameters that preserve the network's input–output mapping while potentially disrupting optimization [Godfrey et al., 2022, Zhao et al., 2026].

Example 1: activation homogeneity. The ReLU activation is positively homogeneous: $\mathrm{ReLU}(a\boldsymbol{z}) = a\,\mathrm{ReLU}(\boldsymbol{z})$ for any $a > 0$. For any network layer of the form $\boldsymbol{W}_2\,\mathrm{ReLU}(\boldsymbol{W}_1\boldsymbol{x})$, this yields the identity $(a\boldsymbol{W}_2)\,\mathrm{ReLU}(a^{-1}\boldsymbol{W}_1\boldsymbol{x})$. The reparameterization preserves the function exactly, but creates an imbalance: the output becomes highly sensitive to small perturbations of $a^{-1}\boldsymbol{W}_1$, since these are amplified by $a\boldsymbol{W}_2$. This slows the optimization, but the defense is trivially broken: the attacker can inspect per-layer weight norms, detect the imbalance, and rebalance them.

Example 2: invertible matrix insertion. Between two consecutive linear layers, one can insert an invertible matrix $\boldsymbol{A}$ and its inverse: $(\boldsymbol{W}_2\boldsymbol{A}^{-1})(\boldsymbol{A}\boldsymbol{W}_1)$. In a transformer, this applies to consecutive linear maps such as the key–query or value–output projections, where an adversarial choice of $\boldsymbol{A}$ can significantly disturb the individual factors. A concrete example of matrix factorization is given in Appendix C.1. However, even in this case, the adaptive attacker can compute the singular value decomposition (SVD) of the product $(\boldsymbol{W}_2\boldsymbol{A}^{-1})(\boldsymbol{A}\boldsymbol{W}_1) = \boldsymbol{W}_2\boldsymbol{W}_1 = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$ and reparameterize as $(\boldsymbol{U}\Sigma^{1/2})(\Sigma^{1/2}\boldsymbol{V}^\top)$, a canonical factorization entirely independent of the defender's choice of $\boldsymbol{A}$.

From symmetry-based to optimization-based approaches. These examples illustrate a general difficulty: known symmetries are limited to simple patterns (scaling, permutation, invertible linear transforms) that an informed attacker can detect and reverse, at negligible additional cost in terms of equation 2. This motivates *optimization-based* approaches, where the defense is embedded through training and the configuration serves as secret information—even with full knowledge, exact unlearning requires solving a hard optimization problem. We discuss two possible families below.

(i) Barrier-based locking: disturbing the input Jacobian. The idea is to insert a barrier module B between layers whose input Jacobian $\partial B(\boldsymbol{h})/\partial\boldsymbol{h}$ causes gradients to explode or vanish, making fine-tuning difficult. Two constraints arise: the barrier must be inseparable from the model, and its form must be learned rather than hand-crafted, since an adaptive attacker can otherwise remove or reverse it. Replacing entire modules is a natural approach, but standard distillation produces smooth approximations, so achieving pathological Jacobians remains difficult.

Limitation: well-hidden barriers that sufficiently block gradient flow also make it more difficult for the defender to maintain model quality, as required by condition *(i)* of Definition 2.3.

(ii) Penalty-based locking: optimizing for ill-conditioning. An alternative approach is to directly search for ill-conditioned solutions via optimization, rather than relying on known symmetries. One can formulate this as a constrained optimization problem:

$$\max_{\theta} \ \Omega(\theta) \quad \text{s.t.} \quad \mathcal{L}(\theta) < \mathcal{L}(\theta_0) + \varepsilon, \tag{3}$$

where Ω measures difficulty of fine-tuning. However, this formulation is self-defeating:

Limitation: the defender seeks to make optimization hard for the attacker, yet must itself rely on optimization to find such a solution, caught in a self-made trap.

To confirm this, we test a simplified instance of equation 3 on a two-layer ReLU MLP trained on MNIST [LeCun et al., 2010]. We optimize $\min_{\Delta\theta} \mathrm{MSE}(f_{\theta_0}, f_{\theta_0+\Delta\theta}) - \lambda\,\Omega(\theta_0 + \Delta\theta)$, where the MSE term preserves the full input–output mapping to maintain the original performance, sweeping over λ and learning rates with two proxy penalties: $\Omega = \|\Delta\theta\|_2^2$ (distance from initialization) and $\Omega = \mathrm{Tr}(\nabla_\theta^2 \mathcal{L}_{\mathrm{CE}})$ (sharpness via Hutchinson trace [Hutchinson, 1990]). Neither directly measures fine-tuning difficulty; if gradient descent cannot achieve even these simpler objectives under the constraint, the full problem equation 3 is unlikely to be easier. As shown in Figure 2, no trajectory reaches the high-Ω corner while maintaining the original function. Yet such solutions *do* exist: the weight symmetry in Example 1 preserves the function exactly while increasing both quantities. Gradient descent simply cannot reach them from the pretrained initialization (see also Appendix C.2).

3 DLR-LOCK: Deep Low-Rank Residual Locking

The approaches discussed in Section 2.2 all attempt to modify the loss landscape or gradient flow. We take a different route: we replace each SwiGLU MLP with a *DLR-Net* that computes the same function but is structurally expensive to differentiate. The key observation is that inference and training have fundamentally different memory requirements: users who run the model are unaffected, while those who update its weights face additional cost.

- **Inference** evaluates the model in a single forward pass, discarding intermediates.
- **Training** must store all intermediate activations for backpropagation.

This is not a property of the loss landscape or the optimizer but a structural property of automatic differentiation. An architecture that is cheap to evaluate but expensive to differentiate creates an inherent, optimizer-agnostic defense—one that does not depend on finding particular weights.

While the defender replaces each MLP independently without touching other components, the attacker optimizes all parameters jointly through $N \times L$ layers. Beyond the memory cost, two additional optimization challenges may arise: the effective depth may cause training instability, and standard fine-tuning recipes developed for the original architecture are unlikely to transfer, since the deep residual MLPs differ substantially from the MLP blocks they replace.

3.1 Deep low-rank residual network

A practical defense must not disadvantage legitimate users: the replacement must (i) match the original module's input–output behavior and (ii) not significantly increase the total parameter count or inference cost. We use *low-rank* weight matrices in each residual block: the bottleneck rank $r \ll d$ keeps the per-layer parameter count small, allowing the total budget to match (or only slightly exceed) that of the original SwiGLU MLP while distributing the parameters across many layers.

We replace the MLP residual block $\boldsymbol{x}' = \boldsymbol{z} + \mathrm{FFN}(\mathrm{RMSNorm}(\boldsymbol{z}))$ with hidden dimension d and intermediate dimension d_{ff} by a depth-L low-rank residual network:

$$\boldsymbol{h}_0 = \boldsymbol{z}, \quad \boldsymbol{h}_{i+1} = \boldsymbol{h}_i + \alpha_i \boldsymbol{U}_i\, \sigma\big(\boldsymbol{V}_i\, \mathrm{RMSNorm}(\boldsymbol{h}_i)\big), \quad \boldsymbol{x}' = \boldsymbol{h}_L, \tag{4}$$

where σ is the SiLU nonlinearity, $\boldsymbol{V}_i \in \mathbb{R}^{r\times d}$ and $\boldsymbol{U}_i \in \mathbb{R}^{d\times r}$ are the bottleneck projections, and α_i is a learnable ReZero [Bachlechner et al., 2021] scalar. Each layer costs $2dr$ parameters plus c_{norm} for normalization, so at a fixed budget P the depth is $L = \lfloor P/(2dr + c_{\mathrm{norm}}) \rfloor$. Smaller rank yields a

Algorithm 1 DLR-LOCK: model locking via deep low-rank residual distillation

Require: Pretrained model $\mathcal{M}$ with N transformer layers, each containing a normalization–MLP residual sub-block; bottleneck rank r; per-MLP parameter budget P; training data $\mathcal{D}$

Ensure: Locked model $\hat{\mathcal{M}}$ with MLP sub-blocks replaced by DLR-Nets

Phase 1: Module-wise regression
for each layer $l = 1, \ldots, N$ **in parallel do**
  Initialize DLR-Net $g^{(l)}$ with rank r and depth $L = \lfloor P/(2dr + c_{\text{norm}}) \rfloor$
  Train $g^{(l)}$ to minimize $\mathcal{L}_{\text{MSE}} = \frac{1}{n} \sum_i \|g^{(l)}(\boldsymbol{z}_i) - f_l(\boldsymbol{z}_i)\|^2 / \|f_l(\boldsymbol{z}_i)\|^2$
  Replace normalization, MLP, and skip connection with $g^{(l)}$ in $\hat{\mathcal{M}}$
end for
Phase 2: Logit distillation
Freeze all parameters in $\hat{\mathcal{M}}$ except DLR-Net weights
Train $\hat{\mathcal{M}}$ to minimize $\mathcal{L}_{\text{KD}} = \sum_{j \in \text{top-k}} p_j^{\text{teacher}} \log(p_j^{\text{teacher}} / p_j^{\text{student}})$ using $\mathcal{M}$ as teacher

deeper network at the same parameter cost. RMSNorm pre-normalization is essential to our design: it places a nonlinearity on the d-dimensional stream, so backpropagation retains $\Theta(d)$ activations per layer even with the DLR-Net frozen. Appendix F.1 reports these architectural ablations.

These per-layer costs accumulate across the full network. We quantify the cost amplification for the simplest attack strategy; stronger attacks are analyzed in Section 3.3:

Proposition 3.1 (Cost amplification under naive fine-tuning)**.** *Consider an attacker who uses naive fine-tuning $\mathfrak{A} = \{\mathcal{A}_{\text{ft}}\}$, i.e., gradient descent with all activations retained for the backward pass, and assume that* (i) *the number of tokens required to reach target performance p on task $\mathcal{T}$ is the same for f and g, and* (ii) *per-GPU throughput scales proportionally with the maximum feasible micro-batch size, up to constant factors. Then the GPU-hour cost of fine-tuning the locked model satisfies $\kappa = \Omega(d/r)$ when MLP activations dominate the memory budget.*

The bound is conservative: it assumes equal sample efficiency between f and g; in practice, the architectural mismatch introduced by the deep residual MLPs may further slow convergence, increasing the true κ. The detailed derivation, including per-layer memory analysis, is given in Appendix B.

3.2 Distillation procedure

We distill the pretrained MLP into the DLR-Net in two phases, summarized in Algorithm 1: a *module-wise* phase that approximates each module's input–output mapping independently, followed by a *logit distillation* phase that corrects the accumulated error by matching the full model's output distribution. Both phases aim to preserve the original function of the pretrained model. By construction, this satisfies condition *(i)* of Definition 2.3 for every downstream task, unlike methods that optimize a task-specific loss and only guarantee preservation on that objective.

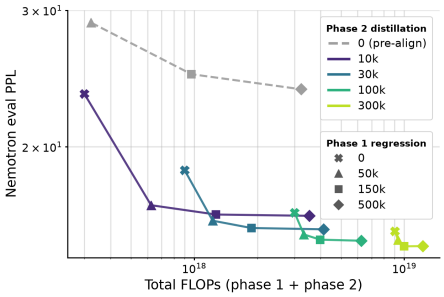


Figure 3: Compute-allocation Pareto frontier for the two-phase distillation of Qwen3-0.6B, sweeping a grid of Phase-1 regression and Phase-2 distillation steps. Skipping Phase 1 is Pareto-dominated at every compute budget. Appendix D gives the FLOPs accounting and hyperparameter configurations.

Phase 1: Module-wise regression. For each transformer layer l, we collect hidden states $\boldsymbol{z}_l$ (after the attention sub-layer) from the pretrained model. The DLR-Net $g^{(l)}$ is trained to replicate the full residual sub-block $f_l(\boldsymbol{z}) = \boldsymbol{z} + \text{FFN}_l(\text{RMSNorm}(\boldsymbol{z}))$ by minimizing the relative MSE, normalized by $\|f_l\|^2$ because output norms vary across layers.

Phase 2: Logit distillation. After all MLP layers are replaced, the small per-layer distillation errors compound across the full forward pass, degrading language modeling performance. We correct this by fine-tuning the DLR-Net parameters (MLP only; attention and embeddings are frozen) using the

pretrained model as teacher. The distillation loss is the top-k KL divergence between teacher and student output distributions, restricting the sum to the k tokens with highest teacher probability.

Phase 2 requires back-propagating through the full locked model, facing the same activation-memory barrier as the attacker; however, Phase 1 provides a good initialization, and the defender pays this cost only once. Figure 3 confirms that this two-phase design is both necessary and efficient.

3.3 Attack strategies

We consider strategies an attacker may use to fine-tune the locked model.

Gradient checkpointing. Gradient checkpointing [Chen et al., 2016] is the most direct countermeasure to activation memory overhead. It reduces peak memory from $O(NLd)$ to $O(Ld)$, at the cost of recomputing each MLP forward pass. The optimal trade-off between recomputation and saved activations is achieved at $O(\sqrt{NL}\,d)$ via evenly spaced checkpoints. If the attacker cannot afford even this memory, they must resort to hierarchical checkpointing, additional GPUs, gradient accumulation, or compromises such as shorter sequence lengths or reduced precision. This recomputation cost is unavoidable for any attacker who fine-tunes the locked model directly.

Frozen DLR-Nets. The attacker may freeze the DLR-Net parameters and train only the rest. However, freezing does not remove the activation memory: the RMSNorm backward still retains $\Theta(d)$ activations per layer, as noted in Section 3.1. If the attacker further applies `stop_grad` to avoid backpropagation entirely, gradients to earlier layers omit the main branch's contribution, producing a biased signal that degrades training. Restricting the trainable parameters also limits adaptation, so reaching a given target takes more steps than full fine-tuning, or may fail entirely.

Low-rank adaptation (LoRA). LoRA [Hu et al., 2022] reduces memory by freezing the base weights and training low-rank adapters. However, LoRA does not shorten the backward pass: backpropagation still traverses all L layers to compute gradients for the adapters. Moreover, it is non-trivial where to apply LoRA in the DLR-Net: since $\boldsymbol{U}_i \in \mathbb{R}^{d\times r}$ and $\boldsymbol{V}_i \in \mathbb{R}^{r\times d}$ are already low-rank, an adapter on one is nearly as large as the matrix itself, so adapting them all saves little over full fine-tuning. Appendix G.1 tests this attack: no fine-tuning rate closes the gap to the unlocked baseline.

Reverse distillation. Since we distill SwiGLU $\to$ DLR-Net, the attacker can run the same procedure in reverse, regressing each DLR-Net back into a standard MLP before fine-tuning the recovered model. It is the most important strategy here, since its cost is amortized rather than recurring per task, as in Remark 2.4. Appendix G.2 reports the reverse-distillation counterpart of Figure 3, confirming that recovery costs on the order of the locking budget itself, so unlocking the weights and passing them on is not a cheap operation. Additionally, one asymmetry here is structural: efficient recovery needs a corpus close to the data used to pretrain or lock the model, which only the provider holds.

In summary, the attacker's budget determines which route is rational:

Attacker budget	Rational strategy	Treated in
$C_{\text{attack}} \gtrsim \min\{C_{\text{pretrain}}, C_{\text{distill}}\}$	pretrain or distill a fresh model	Assumption 2.2
$C_{\text{attack}} \gtrsim C_{\text{lock}}$, several tasks	reverse distillation	Appendix G.2
$C_{\text{attack}} \lesssim C_{\text{lock}}$, or a single task	fine-tune the locked model directly	Section 4.4

4 Experiments

4.1 Experimental setup

We use Qwen3 [Yang et al., 2025] as our testbed. For distillation, we use the Nemotron-CC dataset [Su et al., 2025], a curated web text corpus. Weights are initialized as $\boldsymbol{U}_i \sim \mathcal{N}(0,\, 1/(d\sqrt{L}))$ and $\boldsymbol{V}_i \sim \mathcal{N}(0,\, 1/r)$, and $\alpha_i = 0$. For training, both phases use AdamW [Loshchilov and Hutter, 2019] with weight decay 10^{-5}, cosine learning-rate schedule with linear warmup (5% of total steps), and gradient clipping at norm 1. For Phase 1, each layer is trained independently for 500k steps with

Table 1: Quality preservation of Qwen3-0.6B. "Module-wise reg." replaces every SwiGLU MLP with a DLR-Net in Phase 1. "+Logit dist." applies logit distillation in Phase 2. Nemo = Nemotron, WT = WikiText-103, ARC-E/C = ARC Easy/Challenge, HSw = HellaSwag, WiGr = WinoGrande.

	Perplexity ↓		Accuracy (%) ↑						
	Nemo	WT	MMLU	ARC-E	ARC-C	HSw	WiGr	BoolQ	PIQA
Pretrained	14.9	21.5	40.3	55.9	33.7	47.3	56.4	63.8	67.3
Module-wise reg.	21.8	182	31.1	50.6	26.5	40.5	54.9	62.6	63.4
+Logit dist.	14.9	23.4	36.8	50.7	29.0	43.5	55.2	63.8	65.3

learning rate 10^{-3}, batch size 2, and sequence length 2048. The rank is chosen so that the depth falls in the range $L \in [140, 150]$ (e.g., $r = 32$ for 0.6B). For Phase 2, we train all DLR-Net parameters jointly for 100k steps with top-k=1000, learning rate 10^{-3}, batch size 4 and sequence length 2048. Appendix E.2 considers the provider's own setting: models pretrained from scratch on Nemotron-CC and locked on the same corpus. Baseline and locked arms always run on the same number and type of GPUs, so the wall-clock ratios we report are GPU-hour ratios in the sense of Definition 2.3.

4.2 Quality preservation

Since our algorithm replaces every MLP module, we first verify that the locked model retains the original model's capabilities, condition *(i)* of Definition 2.3, which concerns the published model before any fine-tuning. We evaluate on the Nemotron-CC validation set and WikiText-103 [Merity et al., 2016] for perplexity in and out of the distillation distribution, and on seven LM Evaluation Harness benchmarks [Gao et al., 2023] for downstream performance: MMLU [Hendrycks et al., 2021], ARC (Easy and Challenge) [Clark et al., 2018], HellaSwag [Zellers et al., 2019], WinoGrande [Sakaguchi et al., 2021], BoolQ [Clark et al., 2019], and PIQA [Bisk et al., 2020].

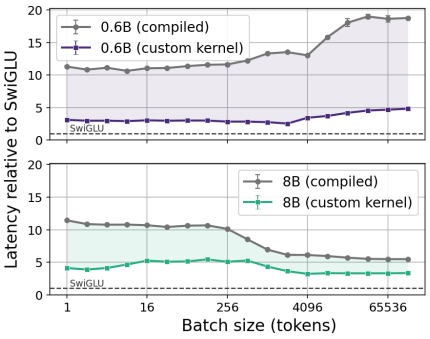
Table 1 shows the results. Regression alone leaves a noticeable gap in several metrics. Distillation recovers Nemotron perplexity almost completely, as expected for in-distribution data, and WikiText perplexity is also largely recovered since both are natural text corpora. On the remaining benchmarks, distillation closes much of the gap, bringing most metrics within a few points of the pretrained model. Matching the data to the model's own pretraining corpus, as a provider naturally would, closes more of this gap: the from-scratch models of Appendix E.2 stay closer to their pretrained baselines, at the same locking budget. Additionally, we used 100k Phase-2 steps as an initial setting, and if we extend the distillation, WikiText perplexity also matches the pretrained model (Appendix D).

4.3 Inference overhead

Replacing each MLP with a depth-L DLR-Net preserves total theoretical inference FLOPs (Section 3.1), but the sequential chain of small matrix multiplications adds wall-clock overhead from kernel launches and memory traffic. We measure it in `bfloat16` on H100 GPUs using PyTorch 2.9 [Ansel et al., 2024] across model scales. Two off-the-shelf levers remove most of the overhead: `torch.compile` with CUDA graph replay (`reduce-overhead`), and a custom kernel for the DLR-Net. Appendix H.1.1 gives the full kernel strategies and the measurement protocol.

Compilation and a custom kernel close the gap. Because the DLR-Net issues many small kernel launches per MLP, it benefits disproportionately from CUDA graph replay, which removes most of the eager-execution overhead. Our custom kernel closes most of the residual gap, bringing the single-MLP overhead to roughly 2–5× across scales (Figure 4, left; Appendix H.1.2 for the full four-scale sweep). The baseline is itself compiled with `reduce-overhead`, so these are ratios against an optimized SwiGLU. The residual gap shrinks with model size, as each residual block performs proportionally larger matrix multiplications, and with batch size, as they become compute-bound.

Defense–speed trade-off. In a full transformer, attention, embeddings, and the output projection are identical to the baseline and increasingly dominate at longer sequences, diluting the per-MLP overhead measured above. Figure 4 (right) shows the full-model prefill overhead on Qwen3-8B with the custom kernel: even the strongest configuration (L=143) reaches only 1.22× at 32k tokens,



Qwen3-8B, fixed n=32k tokens, prefill				
	Depth L			
Seq. len	143	71	35	17
1024	2.35×	1.63×	1.24×	1.09×
8192	1.64×	1.29×	1.11×	1.04×
32768	1.22×	1.10×	1.04×	1.02×

Figure 4: **Left:** Single-MLP latency of the DLR-Net relative to the SwiGLU MLP as a function of batch size, for 0.6B (top) and 8B (bottom). The SwiGLU baseline is compiled with `torch.compile` in `reduce-overhead` mode; relative to it we plot the DLR-Net under the same compilation (gray) and with our custom kernel (colored). The shaded band is the residual gap the kernel closes, and the dashed line at 1 is the baseline. **Right:** Full-model prefill overhead on Qwen3-8B with the custom kernel. Each column is a different depth L (rank $r = 128, 256, 512, 1024$).

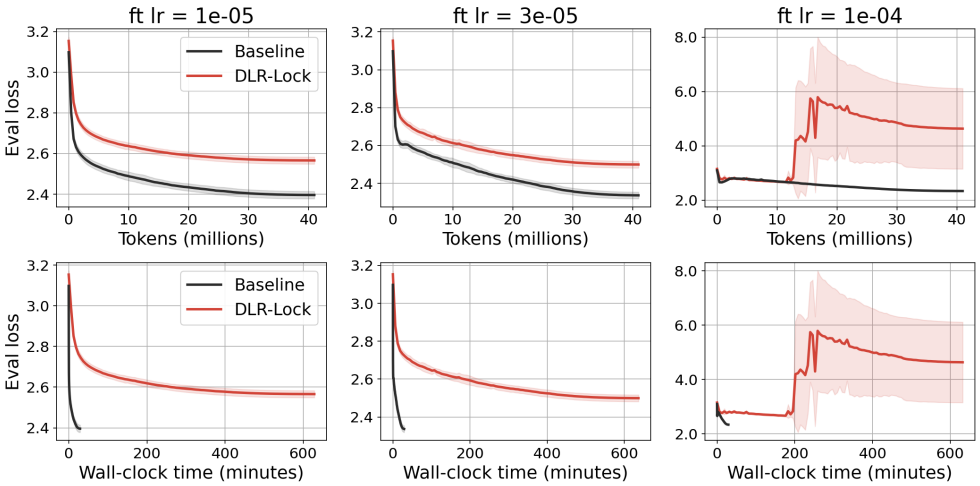


Figure 5: Fine-tuning convergence comparison between the pretrained baseline and the locked model on WikiText-103. **Top:** loss vs. number of training tokens (same batch size and sequence length for both); **Bottom:** loss vs. wall-clock time. Columns correspond to fine-tuning learning rates. The smallest rate swept, 3×10^{-6}, is omitted; it tracks 10^{-5} at a slower pace. Shaded regions indicate ±1 standard deviation across 3 seeds.

a moderate depth (L=35) is nearly transparent at $1.04\times$, and the shallowest (L=17) matches the SwiGLU baseline at $1.02\times$. The defender thus tunes this trade-off through the rank r and depth L (recall $2drL \approx 3d \cdot d_{\text{ff}}$ at fixed parameter count), choosing how much forward-pass latency to spend for locking strength (see also Appendix F.2).

4.4 Resistance to fine-tuning

We now evaluate the core claim: whether the locked model resists fine-tuning by an attacker. We use the same locked model as in Section 4.2, with gradient checkpointing enabled to reduce memory overhead. The attacker sweeps the fine-tuning learning rate over $\{3\times10^{-6}, 10^{-5}, 3\times10^{-5}, 10^{-4}\}$, each compared against fine-tuning the baseline unlocked model at the same rate.

Results. Figure 5 shows convergence curves at three of the four rates. The locked model starts at a similar loss to the baseline, so the final loss gap cannot be explained by the initial one alone. At lower rates (10^{-5}, 3×10^{-5}), the baseline steadily improves while the locked model converges more slowly and plateaus at a higher loss, leaving a persistent gap even at the attacker's best rate. At the largest

rate (10^{-4}), the locked model diverges entirely while the baseline trains normally, illustrating that our defense destabilizes the optimization itself, narrowing the attacker's effective range of learning rates. The same curves also bound the cost of reaching a fixed target, rather than the loss reached at a fixed budget. The locked model's best loss, reached at 3×10^{-5}, costs the baseline $\mathbf{4.8}\times$ fewer tokens at 10^{-5}, its cheapest route there; the baseline's own best loss the locked model never reaches. On top of that, the locked model alone must recompute activations at this batch, 8 rather than 6 FLOPs per parameter-token (Remark B.2). The floor is therefore $4.8 \times 4/3 = \mathbf{6.4}\times$: a FLOPs ratio at matched quality that no implementation removes, and a wall-clock floor too whenever the baseline uses the hardware no less efficiently than the locked model.

The bottom row bounds that cost ratio from above. In eager wall-clock, the pair differs by $\mathbf{109}\times$ in GPU-hours, a figure that also carries the forward slowdown and therefore overstates what an attacker facing an optimized deployment would pay. Decomposing one optimizer step (Appendix H.2) separates the per-step component from the optimization difference above: the training overhead accounts for $\mathbf{87}\%$ of the step for the locked model against 73% for the baseline, and it is $\mathbf{6.7}\times$ the forward pass, against $2.7\times$ for the baseline. In practice a provider can ship a model whose forward is served by the custom kernel of Section 4.3. The next paragraph shows that the backward admits no equivalent optimization, which further widens these ratios of training-related to inference-only cost.

Backward pass resists optimization. Let t_f denote the inference forward time, t_e one training step (forward, loss, and backward), and $t_b = t_e - t_f$ the training overhead over inference; primes denote the deployed setting, with t'_f our custom-kernel forward and t'_e the `reduce-overhead`-compiled training step, the cheapest speedup route open to an attacker. Table 2 isolates a single DLR-Net: at every n, optimization slashes the forward ($t'_f/t_f \ll 1$) yet reduces the training overhead by less ($t'_b/t_b > t'_f/t_f$), the same widening found at full-model scale (Appendix H.2). This gap is inherent, not an implementation artifact. *First*, the provider ships a forward-only kernel; having no backward, it cannot be reused for training, so the attacker falls back to generic compilation, or pays upfront to engineer a backward kernel of their own. *Second*, even if they take that route, a hand-written backward cannot recover the forward's speed either. Our kernel is fast because the hidden state stays on chip: each layer updates it in place, so the only traffic to memory is its own thin, rank-r weight. The backward cannot: each layer's $O(d)$ activation must be written out and read back, a far larger transfer than the thin weight the forward moves; its gradient, summed over tokens, also breaks the forward's free tiling. The defender thus gives ordinary users a near-free forward, while any attacker who computes gradients pays a disproportionate, unavoidable cost.

Table 2: Single DLR-Net, deployed setting: a custom kernel makes the forward near-free, but the backward pass speeds up consistently less.

	0.6B		8B	
n	t'_f/t_f	t'_b/t_b	t'_f/t_f	t'_b/t_b
4	0.02	**0.09**	0.04	**0.09**
512	0.02	**0.11**	0.08	**0.12**
8192	0.07	**0.39**	0.18	**0.40**

5 Conclusion

We introduced a model locking approach that replaces MLP layers with deep low-rank residual networks, exploiting the inference–training asymmetry of automatic differentiation. The locked model imposes three complementary barriers: activation memory that grows with depth, architectural mismatch destabilizing fine-tuning, and a backward pass disproportionately costlier than the forward pass. A two-phase distillation procedure preserves the original model's capabilities while achieving a measurable locking effect against adaptive attackers. Our custom kernel already removes much of the forward-pass overhead; narrowing what remains, through further kernel work or batched computation across residual layers, is a systems problem in its own right, which we leave to future work.

Acknowledgments and Disclosure of Funding

This work was supported by Apple. We thank the anonymous reviewers for their helpful feedback.

References

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 4895–4901, 2023.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703850. doi: 10.1145/3620665.3640366. URL `https://doi.org/10.1145/3620665.3640366`.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in artificial intelligence*, pages 1352–1361. PMLR, 2021.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

Rishi Bommasani, Sayash Kapoor, Kevin Klyman, Shayne Longpre, Ashwin Ramaswami, Daniel Zhang, Marietje Schaake, Daniel E Ho, Arvind Narayanan, and Percy Liang. Considerations for governing open foundation models. *Science*, 386(6718):151–153, 2024.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Dan Busbridge, Amitis Shidani, Floris Weers, Jason Ramapuram, Etai Littwin, and Russ Webb. Distillation scaling laws. *arXiv preprint arXiv:2502.08606*, 2025.

Stephen Casper, Kyle O'Brien, Shayne Longpre, Elizabeth Seger, Kevin Klyman, Rishi Bommasani, Aniruddha Nrusimha, Ilia Shumailov, Sören Mindermann, Steven Basart, Frank Rudzicz, Kellin Pelrine, Avijit Ghosh, Andrew Strait, Robert Kirk, Dan Hendrycks, Peter Henderson, J Zico Kolter, Geoffrey Irving, Yarin Gal, Yoshua Bengio, and Dylan Hadfield-Menell. Open technical problems in open-weight AI model risk management. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL `https://openreview.net/forum?id=8QyGLnFkzc`. Survey Certification.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Zhen Cheng, Hao-Bo Yang, Wan-Yi Huang, and Jin-Long Li. Attention editing: A versatile framework for cross-architecture attention conversion. *arXiv preprint arXiv:2604.05688*, 2026.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of machine learning research*, 24(240):1–113, 2023.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

Nolan Dey, Bin Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: Completep enables compute-efficient deep transformers. *Advances in Neural Information Processing Systems*, 38:137707–137739, 2026.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

Chongyu Fan, Jinghan Jia, Yihua Zhang, Anil Ramakrishna, Mingyi Hong, and Sijia Liu. Towards LLM unlearning resilient to relearning attacks: A sharpness-aware minimization perspective and beyond. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=zZjLv6F0Ks.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Charles Godfrey, Davis Brown, Tegan Emerson, and Henry Kvinge. On the symmetries of deep learning models and their internal representations. *Advances in Neural Information Processing Systems*, 35:11893–11905, 2022.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Shengyuan Hu, Yiwei Fu, Steven Wu, and Virginia Smith. Unlearning or obfuscating? jogging the memory of unlearned LLMs via benign relearning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=fMNRYBvcQN.

Tiansheng Huang, Sihao Hu, and Ling Liu. Vaccine: Perturbation-aware alignment for large language models against harmful fine-tuning attack. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=lpXDZKiAnt.

Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, and Ling Liu. Booster: Tackling harmful fine-tuning for large language models via attenuating harmful perturbation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tTPHgb0EtV.

M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, 1990. doi: 10.1080/03610919008812866. URL https://doi.org/10.1080/03610919008812866.

Jiaming Ji, Mickel Liu, Josef Dai, Xuehai Pan, Chi Zhang, Ce Bian, Boyuan Chen, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. Beavertails: Towards improved safety alignment of llm via a human-preference dataset. *Advances in Neural Information Processing Systems*, 36:24678–24704, 2023.

Wei Junhao, Yu Zhe, and Jun Sakuma. Disrupting model merging: A parameter-level defense without sacrificing accuracy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17698–17707, 2025.

Sayash Kapoor, Rishi Bommasani, Kevin Klyman, Shayne Longpre, Ashwin Ramaswami, Peter Cihon, Aspen Hopkins, Kevin Bankston, Stella Biderman, Miranda Bogen, et al. On the societal impact of open foundation models. *arXiv preprint arXiv:2403.07918*, 2024.

Auguste Kerckhoffs. La cryptographie militaire. *J. Sci. Militaires*, 9(4):5–38, 1883.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Bruce W. Lee, Addie Foote, Alex Infanger, Leni Shor, Harish K Kamath, Jacob Goldman-Wetzler, Bryce Woodworth, Alex Cloud, and Alexander Matt Turner. Distillation robustifies unlearning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=UTGjik64IK.

Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. Starcoder: may the source be with you! *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=KoFOg41haE. Reproducibility Certification.

Chen Liang, Jiahui Yu, Ming-Hsuan Yang, Matthew Brown, Yin Cui, Tuo Zhao, Boqing Gong, and Tianyi Zhou. Module-wise adaptive distillation for multimodality foundation models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL https://openreview.net/forum?id=JhQP33aMx2.

Chen Liang, Simiao Zuo, Qingru Zhang, Pengcheng He, Weizhu Chen, and Tuo Zhao. Less is more: Task-aware layer-wise distillation for language model compression. In *International Conference on Machine Learning*, pages 20852–20867. PMLR, 2023b.

Guozhi Liu, Weiwei Lin, Qi Mu, Tiansheng Huang, Ruichao Mo, Yuren Tao, and Li Shen. Targeted vaccine: Safety alignment for large language models against harmful fine-tuning via layer-wise perturbation. *IEEE Transactions on Information Forensics and Security*, 2025.

Ka Man Lo, Yiming Liang, Wenyu Du, Yuantao Fan, Zili Wang, Wenhao Huang, Lei Ma, and Jie Fu. m2mkd: Module-to-module knowledge distillation for modular transformers. *arXiv preprint arXiv:2402.16918*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Aengus Lynch, Phillip Guo, Aidan Ewart, Stephen Casper, and Dylan Hadfield-Menell. Eight methods to evaluate robust unlearning in llms. *arXiv preprint arXiv:2402.16835*, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025.

Quoc Minh Nguyen, Trung Le, Jing Wu, Anh Tuan Bui, and Mehrtash Harandi. Antibody: Strengthening defense against harmful fine-tuning for large language models via attenuating harmful gradient influence. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=qur2ef8MqQ.

NVIDIA Corporation. NVIDIA H100 Tensor Core GPU Architecture. Whitepaper, NVIDIA Corporation, 2022.

Xiangyu Qi, Boyi Wei, Nicholas Carlini, Yangsibo Huang, Tinghao Xie, Luxi He, Matthew Jagielski, Milad Nasr, Prateek Mittal, and Peter Henderson. On evaluating the durability of safeguards for open-weight LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=fXJCqdUSVG.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Yeonju Ro, Zhenyu Zhang, Souvik Kundu, Zhangyang Wang, and Aditya Akella. On-the-fly adaptive distillation of transformer to dual-state linear attention for long-context LLM serving. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=pqHWzviKKN.

Domenic Rosati, Jan Wehner, Kai Williams, Lukasz Bartoszcze, Robie Gonzales, carsten maple, Subhabrata Majumdar, Hassan Sajjad, and Frank Rudzicz. Representation noising: A defence mechanism against harmful finetuning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=eP9auEJqFg.

Domenic Rosati, Sebastian Dionicio, Xijie Zeng, Subhabrata Majumdar, Frank Rudzicz, and Hassan Sajjad. Locking open weight models with spectral deformation. In *ICML Workshop on Technical AI Governance (TAIG)*, 2025. URL https://openreview.net/forum?id=cjrm7bo6Eg.

Domenic Rosati, Xijie Zeng, Hong Huang, Sebastian Dionicio, Subhabrata Majumdar, Frank Rudzicz, and Hassan Sajjad. Limits of convergence-rate control for open-weight safety. *arXiv preprint arXiv:2602.18868*, 2026.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1631–1642, 2013.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2459–2475, 2025.

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

Rishub Tamirisa, Bhrugu Bharathi, Long Phan, Andy Zhou, Alice Gatti, Tarun Suresh, Maxwell Lin, Justin Wang, Rowan Wang, Ron Arel, Andy Zou, Dawn Song, Bo Li, Dan Hendrycks, and Mantas Mazeika. Tamper-resistant safeguards for open-weight LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FIjRodbW6.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yuhui Wang, Rongyi Zhu, and Ting Wang. Self-destructive language models. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=ERNpUGr8M5.

Zihao Wang, Enneng Yang, Lu Yin, Shiwei Liu, and Li Shen. Model unmerging: Making your models unmergeable for secure model sharing. *arXiv preprint arXiv:2509.01548*, 2025.

Ziyao Wang, Nizhang Li, Pingzhi Li, Guoheng Sun, Tianlong Chen, and Ang Li. Towards building non-fine-tunable foundation models. *arXiv preprint arXiv:2602.00446*, 2026b.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. Bert-of-theseus: Compressing bert by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869, 2020.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34:17084–17097, 2021.

Shuo Yang, Qihui Zhang, Yuyang Liu, Yue Huang, Xiaojun Jia, Kun-Peng Ning, Jia-Yu Yao, Jigang Wang, Dai Hailiang, Yibing Song, et al. Asft: Anchoring safety during llm fine-tuning within narrow safety basin. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 34322–34330, 2026.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Bo Zhao, Robin Walters, and Rose Yu. Symmetry in neural network parameter spaces. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=jLpWq5QY6I.

Yiran Zhao, Wenxuan Zhang, Yuxi Xie, Anirudh Goyal, Kenji Kawaguchi, and Michael Shieh. Understanding and enhancing safety mechanisms of LLMs via safety-specific neuron. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=yR47RmND1m.

Amber Yijia Zheng, Site Bai, Brian Bullins, and Raymond A. Yeh. Model immunization from a condition number perspective. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=uitj69FqD5.

Wangchunshu Zhou, Ronan Le Bras, and Yejin Choi. Modular transformers: Compressing transformers into modularized layers for flexible efficient inference. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10452–10465, 2023.

A Related Work

Preventing harmful model editing. A comprehensive survey of security risks in open-weight models is provided by Casper et al. [2026]. The first direction is preventing fine-tuning to harmful content [Huang et al., 2024, Tamirisa et al., 2025, Zhao et al., 2025, Huang et al., 2025, Zheng et al., 2025, Liu et al., 2025, Wang et al., 2026a, Yang et al., 2026, Nguyen et al., 2026] or robustifying unlearned models against relearning attacks [Lynch et al., 2024, Hu et al., 2025, Fan et al., 2025, Lee et al., 2025]. The major line of work takes a meta-learning approach, optimizing a bilevel objective or training the model under simulated tampering. Qi et al. [2025] show that existing objective-level defenses are broken by simple hyperparameter changes, motivating structural alternatives. Another approach modifies the model's internal representations [Rosati et al., 2024], which can be naturally combined with our method: representation modification alters the input–output relationship for safety, whereas our architectural replacement preserves it. The two mechanisms are complementary and can be applied jointly. A separate line of work targets model merging prevention [Junhao et al., 2025, Wang et al., 2025]: these methods reparameterize weight matrices using the symmetries discussed in Section 2.2, moving the model into a different loss basin to disrupt merging. However, such reparameterizations can be detected and reversed by an attacker who knows the defense strategy.

Module replacement via distillation. Building on knowledge distillation [Hinton et al., 2015], several works replace individual transformer modules with compact alternatives for model compression [Xu et al., 2020, Zhou et al., 2023, Liang et al., 2023a,b, Lo et al., 2024, Ro et al., 2025]. Closest to our approach, Cheng et al. [2026] replace attention modules with alternatives such as multi-head latent attention via module-wise distillation without retraining from scratch.

Model locking. Model locking aims to make pretrained weights resistant to *any* form of adaptation—not only harmful fine-tuning. While methods preventing harmful editing apply to both the open-weight and fine-tuning-as-a-service (FTaaS, where the provider performs the fine-tuning) settings, model locking specifically targets the open-weight case, where the attacker controls the full training pipeline. SpecDef [Rosati et al., 2025] and Wang et al. [2026b] are the closest works to ours; we discuss them in detail in the following paragraphs. Concurrently, Rosati et al. [2026] study spectral weight decomposition as a defense but acknowledge that adaptive attackers can trivially revert it. Additionally, they show that any spectral deformation can be diluted by decomposing it across consecutive layers, motivating a shift from weight-space defenses to our structural approach. Our work differs in that we formalize the problem (Definitions 2.1–2.3) and propose an architectural defense with a quantifiable locking factor κ.

Spectral Deformation (SpecDef; Rosati et al. 2025). SpecDef maximizes the top-k singular values of weight matrices as an instance of equation 3 with $\Omega(\theta) = \sum_l \sum_{i=1}^{k} \sigma_i(\boldsymbol{W}_l(\theta))$. This is a post-hoc method: it can be applied to any pretrained model without retraining from scratch. We note several open challenges: maximizing Ω while minimizing the task loss $\mathcal{L}$ is a multi-objective problem subject to the self-defeating structure discussed in Section 2.2 (Figure 2); moreover, using the task loss as the faithfulness constraint risks overfitting to that specific task, with no guarantee that general capabilities are preserved—in contrast, distillation-based approaches maintain the input–output relationship of the original model and thus preserve performance across all tasks in principle. Additionally, the experimental evaluation is conducted at moderate scale, leaving scaling behavior as an open question.

Non-Fine-Tunable Foundation Models [Wang et al., 2026b]. This method embeds a lottery-ticket-style binary mask during pretraining: a subset of weights is designated as "locked" (masked) and the remaining weights are trained to compensate. An attacker unaware of the mask structure optimizes all weights, entering a different optimization space from the one the model was trained in. We note several open challenges: the binary mask reduces effective model capacity since masked weights do not contribute to the forward computation; the mask structure may be detectable via statistics such as weight magnitude or second-order derivatives, in addition to the gradient norm they investigated; the experimental evaluation is conducted at small scale; and most importantly, the comparison between masked and unmasked models reports similar benchmark performance, but both achieve scores near chance level on binary tasks (e.g., $\sim 50\%$), making it difficult to conclude that the proposed locked method does not sacrifice the original performance.

B Activation memory analysis

This section provides the detailed activation memory analysis underlying Proposition 3.1.

Proposition B.1 (DLR-Net activation memory)**.** *For a depth-L network of the form equation 4:*

(a) ***Full training.*** *Each layer stores: the RMSNorm input $\boldsymbol{h}_i$ (d), the RMSNorm output (d), the SiLU pre-activation (r), the SiLU output (r), and the block output (d) for the α_i gradient. Per-layer cost: $3d + 2r \approx 3d$. Total: $\Theta(dL)$.*

(b) ***Frozen parameters.*** *Each layer stores: the RMSNorm input $\boldsymbol{h}_i$ (d, needed because the backward Jacobian depends on $\boldsymbol{h}_i$) and the SiLU pre-activation (r). Per-layer cost: $d + r \approx d$. Total: $\Theta(dL)$.*

Case (a) is the standard training cost. Importantly, case (b) shows that this overhead cannot be avoided even when the attacker freezes all DLR-Net parameters and trains only other components.

Using this result, we now prove the cost amplification bound stated in the main text.

Proof of Proposition 3.1. Recall that the pretrained transformer has N layers, hidden dimension d, and FFN intermediate dimension d_{ff}. The locked model replaces all N SwiGLU MLPs with depth-L DLR-Nets of rank r, where $L = \lfloor 3d_{\text{ff}}/(2r) \rfloor$ matches the original parameter count. By Proposition B.1(a), the total per-token activation memory for the MLP layers increases from $\Theta(Nd_{\text{ff}})$ (one SwiGLU per layer) to $\Theta(NdL)$ (depth-L DLR-Net per layer), an amplification factor of $\Theta(d/r)$ at matched parameter count.

On a GPU with memory capacity M, let M_{fixed} denote the memory that does not scale with the micro-batch: weights, gradients and optimizer state. Since the DLR-Net is parameter-matched to the MLP it replaces, M_{fixed} agrees between the two models up to constant factors. The largest micro-batch that fits is $B = \lfloor (M - M_{\text{fixed}})/m \rfloor$, where m is the per-token activation memory. Let a_{attn} denote the per-token per-layer activation memory for the attention blocks (unchanged), and let B_f, B_g denote the largest micro-batches for the original and locked models respectively. Their ratio satisfies

$$\frac{B_g}{B_f} \lesssim \frac{a_{\text{attn}} + d_{\text{ff}}}{a_{\text{attn}} + dL}.$$

The attacker must process the same number of tokens T, by assumption *(i)*. A smaller micro-batch does not change the total arithmetic work: the DLR-Net is FLOP-matched to the SwiGLU MLP it replaces, so the same T tokens require the same total FLOPs however they are batched. What changes is the rate at which that arithmetic runs. By assumption *(ii)*, per-GPU throughput is proportional to B up to constant factors, so the GPU-hours required to process T tokens scale as T/B. Hence,

$$\kappa \gtrsim \frac{B_f}{B_g} \gtrsim \frac{a_{\text{attn}} + dL}{a_{\text{attn}} + d_{\text{ff}}}.$$

At matched parameter count, $L = \Theta(d_{\text{ff}}/r)$, hence $dL = \Theta(d\, d_{\text{ff}}/r)$. When MLP activations dominate the memory budget, $a_{\text{attn}} = O(d_{\text{ff}})$, so the denominator is $\Theta(d_{\text{ff}})$, while the numerator is $\Theta(dL)$ since $r \ll d$. Therefore, $\kappa = \Omega(d/r)$. □

Remark B.2 (Gradient checkpointing)**.** Proposition 3.1 considers naive fine-tuning with all activations retained for the backward pass. Gradient checkpointing falls outside this restricted attack: it trades activation memory for recomputation, as discussed in Section 3.3. For example, recomputing each forward activation once increases the standard training cost from approximately 6 to 8 FLOPs per parameter-token, a $4/3$ increase. More aggressive checkpointing can further reduce memory at the expense of additional recomputation.

C Ill-conditioning from weight symmetries

C.1 A matrix-factorization example

We illustrate the ill-conditioning caused by the weight symmetries discussed in Section 2.2 on the matrix factorization problem

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{d \times d}} \ell(\boldsymbol{W}_1, \boldsymbol{W}_2) \;=\; \|\boldsymbol{W}_1 \boldsymbol{W}_2 - \boldsymbol{M}\|_F^2, \tag{5}$$

where $\boldsymbol{M} \in \mathbb{R}^{d\times d}$ is a fixed target. This simple setting captures the structure of Example 2: for any invertible $\boldsymbol{A} \in \mathbb{R}^{d\times d}$, the reparameterization $(\boldsymbol{W}_1, \boldsymbol{W}_2) \mapsto (\boldsymbol{W}_1\boldsymbol{A}^{-1},\ \boldsymbol{A}\boldsymbol{W}_2)$ preserves the product $\boldsymbol{W}_1\boldsymbol{W}_2$ exactly while changing the individual factors and hence the optimization landscape. We study the scalar case $\boldsymbol{A} = a\boldsymbol{I}$ with $a > 0$, giving $(\boldsymbol{W}_1, \boldsymbol{W}_2) \mapsto (a^{-1}\boldsymbol{W}_1,\ a\boldsymbol{W}_2)$.

Before presenting the experimental results, we show that such a scaling worsens the condition number of the Hessian, leading to slower convergence.

Proposition C.1 (Ill-conditioning under scaling)**.** *Let $\ell(\boldsymbol{W}_1, \boldsymbol{W}_2) = \|\boldsymbol{W}_1\boldsymbol{W}_2 - \boldsymbol{M}\|_F^2$ with $\tilde{\boldsymbol{W}}_1 = a^{-1}\boldsymbol{W}_1$ and $\tilde{\boldsymbol{W}}_2 = a\boldsymbol{W}_2$, and suppose that $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are full-rank. Then, the condition number of the Hessian of ℓ with respect to the parameters $(\tilde{\boldsymbol{W}}_1, \tilde{\boldsymbol{W}}_2)$ satisfies*

$$\operatorname{cond}(\tilde{\boldsymbol{H}}) \ \geq\ a^4 \cdot \frac{\sigma_{\max}^2(\boldsymbol{W}_2)}{\sigma_{\min}^2(\boldsymbol{W}_1)}, \tag{6}$$

where $\tilde{\boldsymbol{H}}$ is that Hessian, $\operatorname{cond}(\cdot)$ denotes the condition number, and $\sigma_{\max}(\cdot)$, $\sigma_{\min}(\cdot)$ denote the largest and smallest singular values, so that $\operatorname{cond}(\tilde{\boldsymbol{H}}) = \Omega(a^4)$. We write $\operatorname{cond}(\cdot)$ rather than κ here, since κ denotes the locking factor of Definition 2.3 throughout the paper.

Proof. Since $\ell = \|\tilde{\boldsymbol{W}}_1\tilde{\boldsymbol{W}}_2 - \boldsymbol{M}\|_F^2$, the gradients with respect to each weight matrix are given by

$$\nabla_{\tilde{\boldsymbol{W}}_1}\ell = 2(\tilde{\boldsymbol{W}}_1\tilde{\boldsymbol{W}}_2 - \boldsymbol{M})\tilde{\boldsymbol{W}}_2^\top, \quad \nabla_{\tilde{\boldsymbol{W}}_2}\ell = 2\tilde{\boldsymbol{W}}_1^\top(\tilde{\boldsymbol{W}}_1\tilde{\boldsymbol{W}}_2 - \boldsymbol{M}). \tag{7}$$

Let $\operatorname{vec}(\cdot)$ denote column-major vectorization of a matrix. Differentiating again gives

$$\tilde{\boldsymbol{H}}_{1,1} \ =\ \frac{\partial^2\ell}{\partial\operatorname{vec}(\tilde{\boldsymbol{W}}_1)^2} \ =\ 2\,(\tilde{\boldsymbol{W}}_2\tilde{\boldsymbol{W}}_2^\top)\otimes\boldsymbol{I}_d \ =\ 2a^2\,(\boldsymbol{W}_2\boldsymbol{W}_2^\top)\otimes\boldsymbol{I}_d, \tag{8}$$

where $\otimes$ denotes the Kronecker product. Similarly, we have

$$\tilde{\boldsymbol{H}}_{2,2} \ =\ \frac{\partial^2\ell}{\partial\operatorname{vec}(\tilde{\boldsymbol{W}}_2)^2} \ =\ 2\,\boldsymbol{I}_d\otimes(\tilde{\boldsymbol{W}}_1^\top\tilde{\boldsymbol{W}}_1) \ =\ \tfrac{2}{a^2}\,\boldsymbol{I}_d\otimes(\boldsymbol{W}_1^\top\boldsymbol{W}_1). \tag{9}$$

Therefore, the largest singular value of the Hessian $\tilde{\boldsymbol{H}}$ with respect to $(\tilde{\boldsymbol{W}}_1, \tilde{\boldsymbol{W}}_2)$ is given by

$$\sigma_{\max}(\tilde{\boldsymbol{H}}) \ =\ \max_{\|\boldsymbol{v}\|_2=1}\boldsymbol{v}^\top\tilde{\boldsymbol{H}}\boldsymbol{v} \ \geq\ \max_{\|\boldsymbol{v}\|_2=1}\boldsymbol{v}^\top\tilde{\boldsymbol{H}}_{1,1}\boldsymbol{v} \ =\ 2a^2\sigma_{\max}\left((\boldsymbol{W}_2\boldsymbol{W}_2^\top)\otimes\boldsymbol{I}_d\right), \tag{10}$$

where the inequality follows by restricting the last d^2 entries to zero, and the last equality is from equation 8. Using the property of the Kronecker product, we have

$$\sigma_{\max}(\tilde{\boldsymbol{H}}) \ \geq\ 2a^2\sigma_{\max}\left(\boldsymbol{W}_2\boldsymbol{W}_2^\top\right)\sigma_{\max}\left(\boldsymbol{I}_d\right) \ =\ 2a^2\sigma_{\max}\left(\boldsymbol{W}_2\right)^2. \tag{11}$$

Similarly, equation 9 leads to

$$\sigma_{\min}(\tilde{\boldsymbol{H}}) \ \leq\ \frac{2}{a^2}\sigma_{\min}\left(\boldsymbol{W}_1\right)^2. \tag{12}$$

Combining with equation 11 gives the desired result. □

This result shows that the condition number is worsened at the rate of a^4. The full-rank assumption in Proposition C.1 is satisfied almost surely under standard random initialization.

Remark C.2 (Connection to classical convergence bounds)**.** Proposition C.1 only shows that the reparameterization worsens conditioning in this matrix-factorization example. Independently of this example, classical first-order optimization theory provides several conditions in which a larger condition number leads to slower convergence [Boyd and Vandenberghe, 2004]. As one representative case, we focus on the classical μ-strongly convex and β-smooth setting, writing β for the smoothness constant since L denotes the DLR-Net depth elsewhere in the paper; gradient descent with stepsize smaller than $1/\beta$ has the linear convergence rate $1 - \mu/\beta$, yielding the ϵ-iteration complexity $t = O((\beta/\mu)\log(1/\epsilon)) = O(\mathrm{cond}\cdot\log(1/\epsilon))$, where $\mathrm{cond} = \beta/\mu$ is the condition number. Therefore, the larger condition number increases the complexity bound, which implies a slower convergence.

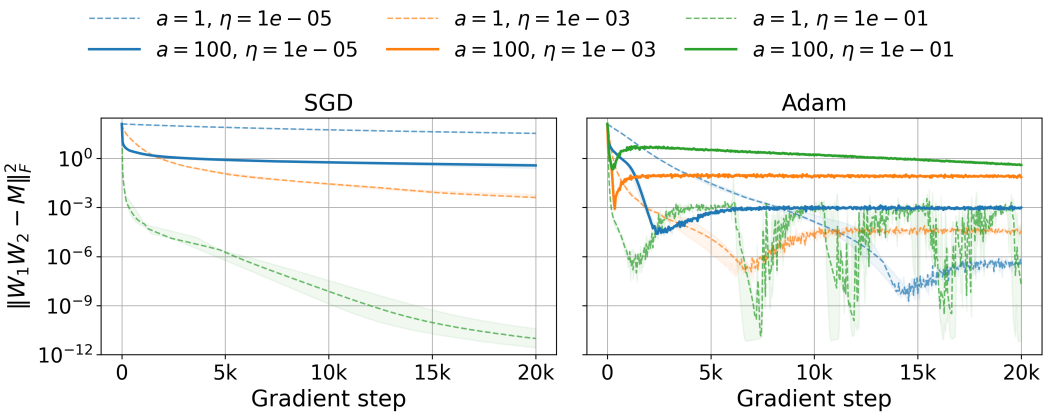


Figure 6: Training loss for matrix factorization $\|\boldsymbol{W}_1\boldsymbol{W}_2 - \boldsymbol{M}\|_F^2$ from balanced ($a = 1$, dashed) and scaled ($a = 100$, solid) initialization. **Left:** SGD. Under the scaling, the two larger learning rates diverge (not shown). Only the smallest learning rate remains stable, yet it plateaus orders of magnitude above the $a = 1$ optimum. **Right:** Adam. All learning rates converge at $a = 100$, albeit slower than at $a = 1$. Shaded regions show the interquartile range over five seeds.

Experimental verification. We verify the above discussion empirically by training the matrix factorization problem equation 5 with and without the scaling reparameterization. We set $d = 64$ with entries of $\boldsymbol{M}$, $\boldsymbol{W}_1$, $\boldsymbol{W}_2$ drawn i.i.d. from $\mathcal{N}(0, 1/d)$, apply the scaling with $a = 100$, and train with both SGD and Adam across three learning rates each.

Figure 6 shows that the bad condition number actually leads to a slower convergence. For SGD at $a = 100$, only the smallest learning rate is stable; and the convergence is extremely slow because the gradient of the enlarged factor $\tilde{\boldsymbol{W}}_2 = a\boldsymbol{W}_2$ scales as a^{-1}, so it barely updates while $\tilde{\boldsymbol{W}}_1$ converges to its conditionally optimal value. Larger learning rates immediately diverge. Adam does not exhibit the same divergence because its element-wise normalization makes the effective step size approximately scale-invariant. However, it still converges slower than at $a = 1$, confirming that the ill-conditioning affects optimization even with adaptive methods.

C.2 Perturbation limit: Hessian scope comparison

Figure 2 (right panel) computes the Hessian trace over all parameters. A natural question is whether restricting the penalty to a subset of parameters makes the optimization easier—for example, concentrating the sharpness penalty on a single layer or a random subset might reduce interference between the MSE and Ω objectives. This setting directly corresponds to increasing the condition number of the Hessian: high curvature in a subset of directions makes gradient-based fine-tuning over the full parameter space difficult. Figure 7 tests four scopes: all parameters, the first layer only, a random 10% mask, and a random 50% mask. The result is consistent across all scopes: trajectories move rightward (increasing MSE) as λ grows, but fail to reach the high-Hessian-trace region occupied by scaling-symmetry solutions. This confirms that the optimization difficulty observed in the main text is not an artifact of the penalty scope.

D Compute allocation between regression and distillation phases

Section 3.2 specifies our two-phase distillation procedure but treats its compute allocation only qualitatively. Here we quantify how the model provider—the defender, who pretrains and then releases the model—should divide a fixed compute budget between Phase 1 (module-wise regression) and Phase 2 (end-to-end logit distillation).

Setup. We sweep the number of Phase-1 regression steps (0, 50k, 150k, 500k) against the number of Phase-2 distillation steps (0, 10k, 30k, 100k, 300k) on Qwen3-0.6B, where a step count of 0 skips that phase; the cell that skips both is excluded, as it leaves the DLR-Nets untrained. Following the training configuration of Section 4.1 (rank r=32, distillation learning rate 10^{-3}, top-k=1000, cosine

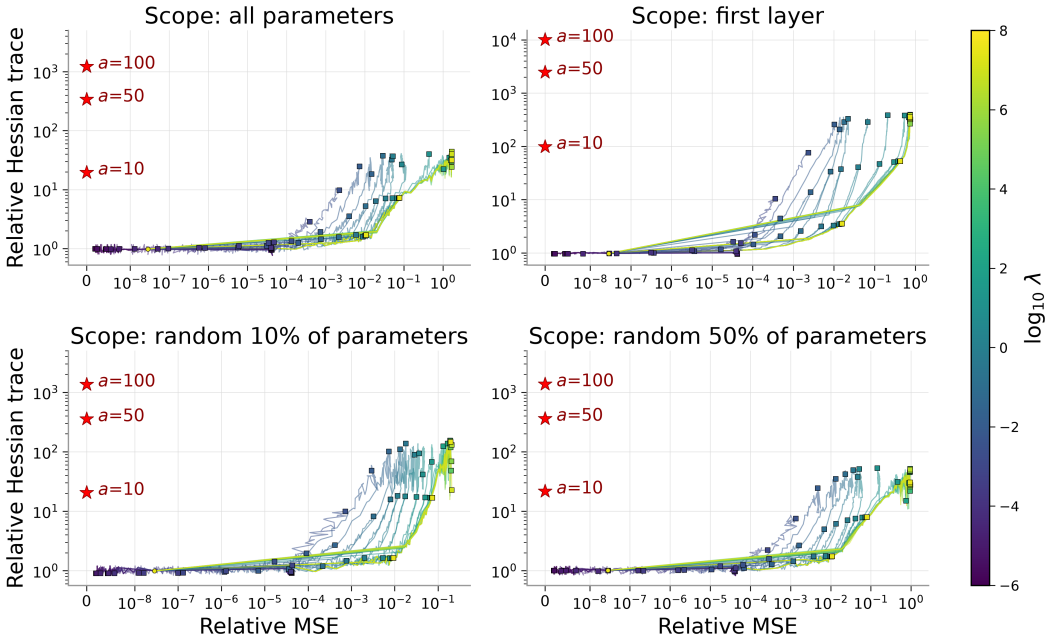


Figure 7: Hessian-trace penalty across four parameter scopes. Same setup as Figure 2 (right panel), but each panel restricts the Hessian trace $\Omega = \mathrm{Tr}(H_{\mathrm{CE}})$ to a different parameter subset. Each curve is one trajectory; color encodes $\log_{10} \lambda$ (dark = small, bright = large). Circles: start; squares: end. Red stars: weight-symmetry solutions $(a^{-1}\boldsymbol{W}_1,\, a\boldsymbol{W}_2)$. x-axis: relative MSE; y-axis: relative Hessian trace over the corresponding subset. Across all scopes, no trajectory reaches the low-MSE, high-Hessian-trace corner occupied by scaling solutions.

schedule with 5% warmup), we report perplexity on the Nemotron-CC evaluation set. Our default configuration, used throughout the paper, is 500k regression and 100k distillation steps.

FLOPs accounting. The x-axis of Figure 3 is the total FLOPs of the distillation procedure across both phases, excluding the teacher forward pass (deterministic in the inputs, hence cacheable). We count gradient-checkpointing recompute where it occurs: Phase 2 back-propagates through the full locked model and, facing the very activation-memory barrier our defense imposes, must recompute each decoder block, whereas Phase 1 trains each DLR-Net independently and needs none.

Per Phase-1 step, we train all $N_{\mathrm{mlp}} \approx 263\times10^6$ DLR-Net parameters (the 28 module-wise MLP replacements) on $T_1 = 4096$ student tokens each, at 6 FLOPs per parameter-token for one forward and backward pass:

$$F_{1,\mathrm{step}} = 6\, N_{\mathrm{mlp}}\, T_1 \approx 6.5 \times 10^{12} \text{ FLOPs.}$$

Per Phase-2 step, we back-propagate through the full model ($N_{\mathrm{total}} \approx 596\times10^6$ parameters) over $T_2 = 8192$ tokens, with only the 28 DLR-Net MLPs (N_{mlp} parameters) trainable; let $N_{\mathrm{block}} \approx 380\times10^6$ denote the attention and MLP matmul parameters across the decoder blocks. Activation gradients traverse the full frozen backbone while weight gradients cover only the trainable MLPs:

$$F_{2,\mathrm{step}} = \underbrace{2N_{\mathrm{total}}T_2}_{\text{fwd}} + \underbrace{2N_{\mathrm{block}}T_2}_{\text{block recompute (GC)}} + \underbrace{2N_{\mathrm{total}}T_2}_{\text{act-grads}} + \underbrace{2N_{\mathrm{mlp}}T_2}_{\text{weight-grads (MLP-only)}} \approx 3.0 \times 10^{13} \text{ FLOPs.}$$

A configuration's total distillation FLOPs is then its Phase-1 step count times $F_{1,\mathrm{step}}$ plus its Phase-2 step count times $F_{2,\mathrm{step}}$.

Implication: DLR-LOCK must be post-hoc. Figures 3 and 8 show the same Nemotron-CC perplexity frontier from two angles, grouped by distillation length and by regression length. This speaks to whether DLR-LOCK could be folded into pretraining rather than applied post-hoc. Training the DLR-Net from scratch as a drop-in SwiGLU replacement, with no per-layer teacher, is exactly

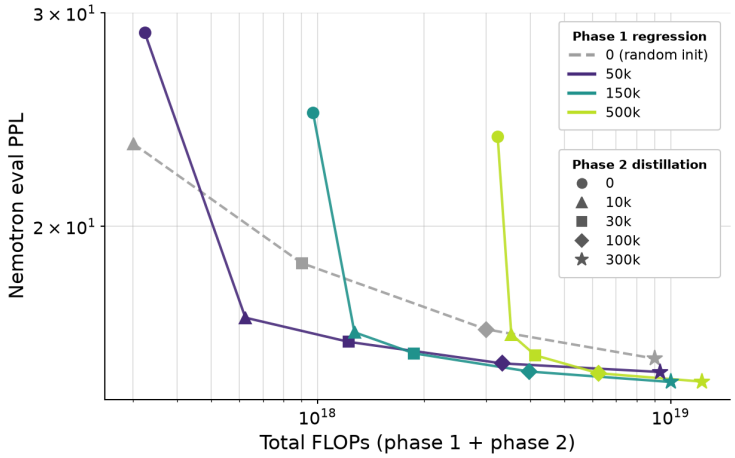


Figure 8: The same compute-split grid as Figure 3, now grouped by Phase-1 regression length. Color encodes the number of Phase-1 regression steps (0, 50k, 150k, 500k), with 0 (gray dashed) the random-initialized DLR-Nets that skip module-wise regression entirely; marker shape encodes Phase-2 distillation steps (0, 10k, 30k, 100k, 300k). The paper-original configuration uses 500k regression and 100k distillation steps.

Table 3: Extending Phase 2 at our default Phase-1 budget (500k regression steps), Qwen3-0.6B. At 300k distillation steps, three times the budget used in the main text, WikiText-103 perplexity reaches that of the pretrained model (21.5, Table 1). A Phase-2 budget of 0 is the Phase-1-only model. The 100k column is an independent rerun of the default configuration and reproduces Table 1's 23.4. Total FLOPs follow the accounting above.

Phase-2 steps	0	10k	30k	100k	300k
Total FLOPs	3.2×10^{18}	3.5×10^{18}	4.1×10^{18}	6.2×10^{18}	1.2×10^{19}
WikiText-103 PPL	182	26.9	25.1	23.5	**21.5**

the zero-regression setting, which is Pareto-dominated at every compute budget: the module-wise regression targets of Phase 1 supply a supervision signal that end-to-end logit distillation alone cannot match at comparable compute. DLR-LOCK is therefore only viable as a *post-hoc* distillation from a pretrained backbone, not a from-scratch pretraining objective.

Implication: the residual quality gap is a budget choice. A second, incidental finding of this sweep concerns the quality gap of Table 1. Holding Phase 1 at our default 500k steps and extending Phase 2 past the 100k steps used throughout the paper keeps improving quality on held-out natural text: as shown in Table 3, WikiText-103 perplexity falls from 23.5 to 21.5, matching the pretrained model's 21.5. Phase 2 is expensive by construction, since the defender must back-propagate through the very model it is locking. Its cost is nonetheless one the defender can pay: as noted in Section 2.1, Definition 2.3 constrains the attacker's adaptation cost but places no constraint on the locking cost, which is paid once and amortized over every deployment of the released weights. The gap that further distillation closes is therefore not an intrinsic limit of DLR-LOCK, but a budget allocation under the defender's control.

E DLR-LOCK across model settings

This section extends the study of DLR-LOCK beyond the main text along two axes: model scale and deployment setting. Appendix E.1 keeps the main-text model, Qwen3, and its WikiText-103 fine-tuning target, extending both claims to larger scales and adding a data-limited fine-tuning setting.

In the main text we use Qwen3 because it is a strong, widely adopted open model, which makes our results easy to reproduce and compare. However, Qwen3 was pretrained by a different provider on a proprietary corpus, so locking it with Nemotron-CC leaves a domain gap, and the degradation we

Table 4: Quality preservation of Qwen3 at 1.7B and 8B before and after DLR-LOCK, at k=1000 and each model's best alignment learning rate: η=10^{-4} for 1.7B and 8B, versus 10^{-3} at 0.6B, since larger models favor a smaller alignment rate. "+DLR-Lock" replaces every SwiGLU MLP with a DLR-Net via the two-phase distillation pipeline. The 0.6B result is in Table 1. Nemo = Nemotron-CC, WT = WikiText-103, ARC-E/C = ARC Easy/Challenge, HSw = HellaSwag, WiGr = WinoGrande.

		Perplexity ↓		Accuracy (%) ↑						
Model	Variant	Nemo	WT	MMLU	ARC-E	ARC-C	HSw	WiGr	BoolQ	PIQA
1.7B	Pretrained	11.95	17.13	55.4	70.2	43.0	60.4	61.5	77.6	72.3
	+DLR-Lock	11.94	17.19	48.5	68.1	38.9	56.2	60.0	78.1	70.9
8B	Pretrained	8.57	9.83	73.0	80.9	56.7	74.9	68.1	86.6	77.8
	+DLR-Lock	9.06	12.13	58.1	76.9	49.7	70.1	68.2	83.5	75.7

report is a *conservative* estimate. In practice, **DLR-LOCK is applied by a model provider to its own model**: the party that pretrains a model and releases its weights is also the one that locks it, so it can run DLR-LOCK's regression and distillation on the *same data it pretrained on*. Appendix E.2 realizes that setting: we pretrain our own models on Nemotron-CC and lock them on the same corpus, so the pretraining and locking data coincide by construction and the domain-mismatch confound of the Qwen3 setting disappears. There we study both central claims of DLR-LOCK, quality preservation and resistance to fine-tuning, across model scales and fine-tuning domains.

E.1 Pretrained Qwen3

This is the setting studied in the main text: we take a publicly released pretrained model, Qwen3, and lock it with DLR-LOCK using Nemotron-CC for regression and distillation. Every experiment here fine-tunes on WikiText-103, as the main text does. Appendix E.2 carries both the data-matched version of these quality numbers and the other fine-tuning domains, since we pretrained those models ourselves and can evaluate them on a new target without a domain gap.

Larger model scales. Table 4 extends the main-text 0.6B result of Table 1 to 1.7B and 8B. Following the rank rule of Section 4.1, the bottleneck widens with the model so that depth stays in the target range: r=64 and L=142 at 1.7B, r=128 and L=143 at 8B, against r=32 and L=141 at 0.6B. Depth is thus essentially constant across scales: the growing parameter budget is absorbed by a wider bottleneck rather than a longer chain. We hold L in this range throughout for comparability, but it is a free parameter rather than a constraint. At a fixed budget rank trades directly against depth, and Appendix F.2 follows that ladder to much shallower networks, which cost less forward latency and lock correspondingly less. The 8B model's Phase-2 batch is halved by memory, so it is distilled for twice as many steps. All scales therefore see the same 0.82B tokens. Even so, the 8B knowledge gap (e.g. MMLU) remains larger than at 1.7B. This is what a fixed budget predicts: the token budget is the same at every scale, so a model with an order of magnitude more parameters receives proportionally less distillation, and closing the gap would likely require scaling the defender's budget with the model rather than holding it constant. At 0.6B, the compute-allocation frontier of Figure 3 in Appendix D shows quality still improving as Phase-2 distillation is extended, so our operating point is not a ceiling, and we expect the same to hold at 8B. Additionally, two mechanisms contribute to this gap and can be separated: the pretraining/locking distribution shift, and a distillation budget that does not scale with the model. The from-scratch setting of Appendix E.2 removes the first by construction, and the gap there is smaller at every scale, leaving the budget as the remaining cause. One column matters especially for the fine-tuning experiment below, which fine-tunes on WikiText-103: at 1.7B that perplexity is preserved to within 0.06, whereas at 8B it rises from 9.83 to 12.13, so the two arms would not start fine-tuning from the same loss. We therefore study fine-tuning resistance at 1.7B.

Fine-tuning resistance at larger scale. The protocol is the main text's, unchanged in every other respect; 8B is covered instead by the data-matched from-scratch setting of Appendix E.2, where the corresponding gap is smaller. The arms start together, and Figure 9 shows the locked model converging more slowly and plateauing higher at every rate. Read as a cost to reach a fixed target, the locked model's best loss, reached at 10^{-4}, costs the baseline $\mathbf{10.0}\times$ fewer tokens at 10^{-5}, its cheapest route there; the baseline's own best loss the locked model never reaches. The locked model

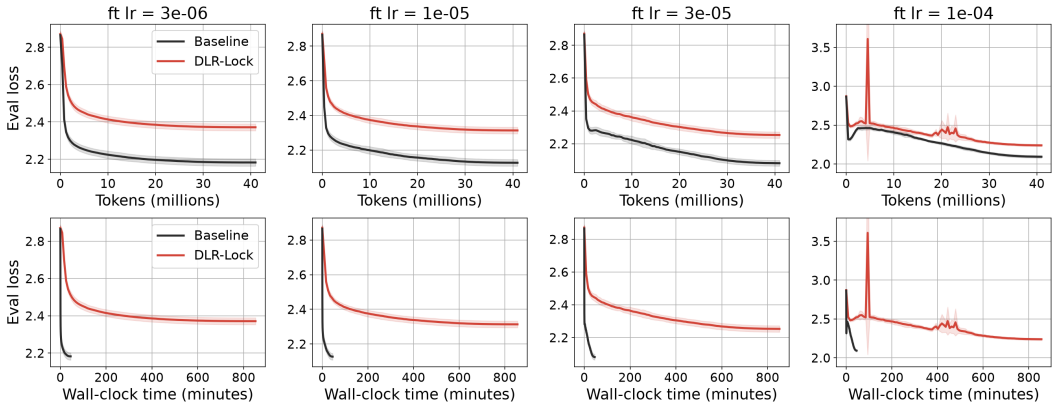


Figure 9: Fine-tuning resistance on WikiText-103 at Qwen3-1.7B, the scale extension of Figure 5, whose protocol it follows. **Top:** loss vs. number of training tokens; **Bottom:** loss vs. wall-clock time. Columns correspond to fine-tuning learning rates; unlike the main-text figure, all four swept rates are shown. The vertical axis of the 10^{-4} column is clipped, where the locked model spikes transiently before recovering. Shaded regions indicate ± 1 standard deviation across 3 seeds.

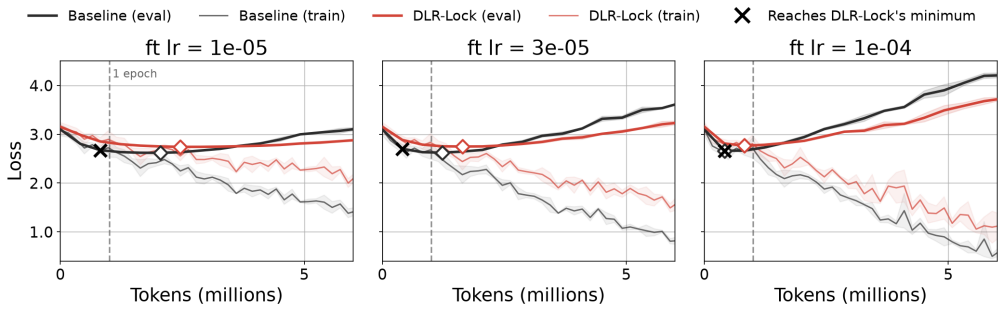
alone recomputes activations at this batch (Remark B.2), a further factor of $4/3$, so the FLOPs floor is $\mathbf{13.3}\times$. In eager wall-clock the pair differs by $\mathbf{189}\times$, which as in the main text carries the forward slowdown too and so bounds the ratio from above. Both ratios exceed their 0.6B counterparts ($4.8\times$ and $109\times$), so the effect does not weaken with scale. The instability does not carry over: the largest rate diverges at 0.6B but here only spikes transiently, to 5.8 in one seed, before 10^{-4} becomes the locked model's best rate. The claim at this scale therefore rests on the persistent loss gap, not on a narrowed range of usable rates.

Fine-tuning under a fixed token budget. The WikiText-103 experiment in the main text (Figure 5) streams a fresh batch at every step, so neither model ever re-sees a token. A realistic attacker often has only a small corpus and must repeat it. We therefore cap the fine-tuning data at 1M and 5M tokens and loop over it for the same 5000-step budget, holding everything else identical to the main-text setup. Figure 10 shows the result. Eval loss falls during the first sweep over the data and rises again as the model overfits, a reading confirmed by the train loss, which continues to fall past the eval-loss minimum. The locked model takes much longer in wall-clock to reach a worse optimum than the baseline does: at $\eta{=}10^{-5}$ its best eval loss is higher by about 0.12 at both caps, and the wall-clock multiplier at the U-bottom is roughly $25\times$ with 1M tokens, growing to $\sim 35\times$ with 5M. Read as a cost to reach a fixed target, the locked model's minimum loss costs the baseline $\mathbf{3.0}\times$ fewer tokens with the 1M cap and $\mathbf{8.3}\times$ fewer with the 5M cap (the cross), growing to $\mathbf{66}\times$ and $\mathbf{181}\times$ in wall-clock. At the 1M cap the two reach their optima at comparable token counts, so the additional cost is paid in compute time, not data. The panels plot loss against tokens, so the gap they show is an optimization effect measured at matched data; the recomputation and backward-pass costs enter only through the wall-clock multipliers.

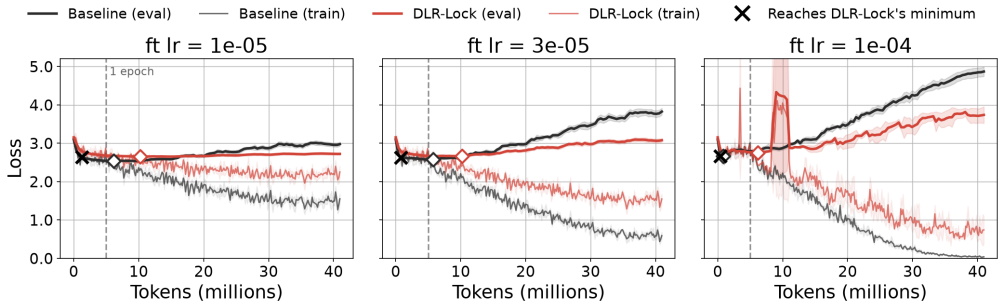
E.2 Models pretrained from scratch

In this setting we control the entire pipeline: we pretrain a family of decoder-only Transformers from scratch on Nemotron-CC and then lock them with DLR-LOCK on the same corpus.

Architecture. The models follow Qwen3's high-level design: a pre-norm decoder-only Transformer with RMSNorm ($\epsilon{=}10^{-5}$), rotary position embeddings (RoPE [Su et al., 2024], base frequency $5{\times}10^5$), SwiGLU feed-forward blocks with SiLU activation, per-head query/key normalization, and the SwiGLU expansion ratio fixed at $d_{\mathrm{ff}}/d = 3$. The main difference is in the attention layer: whereas Qwen3 uses grouped-query attention [Ainslie et al., 2023] with fewer key/value heads than query heads, our models use standard multi-head attention with a fixed head dimension of 128, so the number of heads is $d/128$. We pretrain three scales mirroring the Qwen3 sizes used in the main text: 0.6B ($d{=}1024$, 28 blocks, 8 heads), 1.7B ($d{=}2048$, 28 blocks, 16 heads), and 8B ($d{=}4096$, 32 blocks, 32 heads).



(a) 1M tokens



(b) 5M tokens

Figure 10: Fine-tuning convergence against tokens on a fixed slice of WikiText-103, looped for 5000 steps. Same models and protocol as Figure 5; only the dataset size is capped. Diamonds mark the bottom of each eval U-curve; the cross marks where the baseline first reaches the locked model's minimum during its own descent. Shaded regions: ± 1 standard deviation across 3 seeds.

Pretraining. All models are trained autoregressively on Nemotron-CC with token packing at sequence length 4096, using the tiktoken `cl100k_base` tokenizer (vocabulary size 100,277). We use AdamW (β_1=0.9, β_2=0.95, ϵ=10^{-8}) with weight decay 0.2, gradient-norm clipping at 1.0, a 2000-step warmup followed by a constant-then-decay learning-rate schedule, and the maximal-update parameterization (μP) [Yang et al., 2021] combined with CompleteP [Dey et al., 2026] at base width 512 and base depth 32, with base maximum learning rate 3.79×10^{-3} at every scale, so the effective rate on hidden and readout weights scales as $512/d$. Training uses BF16 precision with an auxiliary z-loss [Chowdhery et al., 2023] (weight 10^{-4}), and each model sees 140B tokens.

Quality preservation. Table 5 reports language-modeling perplexity and zero-shot benchmark accuracy for each scale, before and after locking. Because the locking corpus is identical to the pretraining corpus, this setting removes the distribution-shift penalty that inflates the off-the-shelf gaps of Table 4, leaving the fixed distillation budget as the remaining confound. Perplexity rises by 0.4 to 0.7 on Nemotron-CC and 1.0 to 1.9 on WikiText, and the zero-shot accuracies move by at most 2.9 points, most of them by under 1.5. These models see only 140B tokens, so MMLU and ARC-Challenge sit near chance for both variants and cannot register a drop; on the benchmarks where they are clearly above chance, ARC-Easy, HellaSwag and PIQA, locking costs at most 1.7 points. The absence of a large MMLU gap is therefore consistent with the off-the-shelf drop stemming from the pretraining/locking data mismatch, though the near-chance baseline means this setting alone cannot establish it.

WikiText-103 (general text). We begin with the main text's fine-tuning target, so these curves can be read against Figure 5. Both arms run for 41M tokens at sequence length 2048 (5000 steps at batch 4 for 0.6B and 1.7B, 20,000 steps at batch 1 for the 8B), sweeping $\eta \in \{3\times10^{-6}, 10^{-5}, 3\times10^{-5}, 10^{-4}\}$ over three seeds. The 8B model does not fit on one device in either arm, so we fine-tune it on 2 GPUs with the model split across devices and report wall-clock time under identical hardware for a fair comparison. The locked model is shown at the rate that gives

Table 5: Quality preservation of our from-scratch models (0.6B, 1.7B, 8B) before and after DLR-LOCK, with Phase-2 distillation at k=1000 and learning rate 10^{-4} at every scale, chosen on quality preservation. "+DLR-Lock" is the locked model. At 8B, memory limits the Phase-2 distillation batch to 2 rather than 4, so we run Phase-2 for 200k steps rather than 100k; every scale then sees the same 0.82B distillation tokens. GSM8K [Cobbe et al., 2021] is reported at 2-shot with 128-token generation (uniformly across scales) for tractability. Nemo = Nemotron-CC, WT = WikiText-103, ARC-E/C = ARC Easy/Challenge, HSw = HellaSwag, WiGr = WinoGrande.

		Perplexity ↓		Accuracy (%) ↑							
Model	Variant	Nemo	WT	MMLU	ARC-E	ARC-C	HSw	WiGr	BoolQ	PIQA	GSM8K
0.6B	Pretrained	13.47	26.29	24.8	52.1	27.2	40.9	51.3	59.1	67.5	1.5
	+DLR-Lock	14.20	28.14	23.1	50.5	26.6	39.3	51.5	57.3	66.2	1.5
1.7B	Pretrained	9.92	18.15	24.7	60.4	32.6	54.1	54.1	54.6	71.3	1.1
	+DLR-Lock	10.35	19.20	24.2	60.1	31.4	52.4	54.4	51.7	70.1	2.0
8B	Pretrained	11.29	21.54	24.7	56.4	29.4	46.1	53.7	57.7	69.8	2.4
	+DLR-Lock	11.78	22.66	24.4	56.2	30.5	44.7	52.2	59.4	69.4	1.9

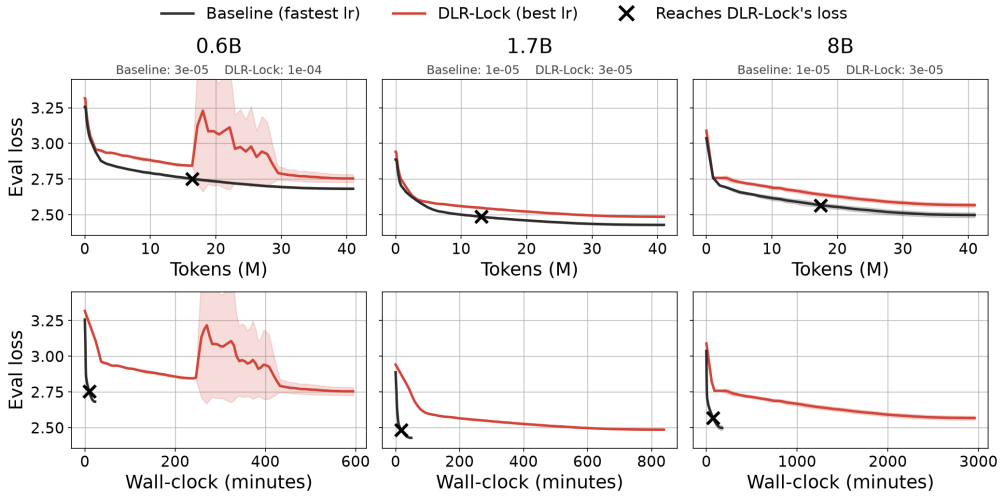


Figure 11: Fine-tuning resistance on WikiText-103 for the from-scratch models at 0.6B, 1.7B and 8B. **Top:** loss vs. number of training tokens; **Bottom:** loss vs. wall-clock time. The rate shown for each arm is annotated above the panel, and the cross marks where the baseline first reaches the locked model's final loss. Shaded regions indicate ± 1 standard deviation across 3 seeds. The transient spike in the 0.6B locked curve is one seed at 10^{-4}, which is nonetheless the locked model's lowest-final-loss rate.

it the lowest final loss, the baseline at whichever of its own rates reaches that loss in the fewest tokens. The arms start within 0.07 at all three scales, including 8B, so the ratios below are not inherited from a pre-existing quality gap. Figure 11 shows the locked model converging more slowly and plateauing higher at every scale. Read as a cost to reach a fixed target, the locked model's best loss costs the baseline $\mathbf{2.5}\times$, $\mathbf{3.1}\times$ and $\mathbf{2.4}\times$ fewer tokens at 0.6B, 1.7B and 8B on its cheapest route there; the locked model alone recomputes activations (Remark B.2), so the FLOPs floors are $\mathbf{3.3}\times$, $\mathbf{4.2}\times$ and $\mathbf{3.1}\times$. In eager wall-clock the pair differs by $\mathbf{62}\times$, $\mathbf{48}\times$ and $\mathbf{40}\times$, which carries the forward slowdown too and so bounds the ratio from above. The baseline's own best loss the locked model never reaches at any scale. These token ratios show no trend with scale. They are below the $4.8\times$ and $10.0\times$ measured off the shelf at 0.6B and 1.7B, but these models are pretrained on only 140B tokens and are correspondingly weaker at equal size (Table 5 against Table 4), so the ratios of the two settings are not directly comparable.

Alpaca (instruction following). For a task-formatted counterpart to the plain-text target above, we fine-tune on Alpaca [Taori et al., 2023], a 52k-prompt instruction dataset, with the prompt masked

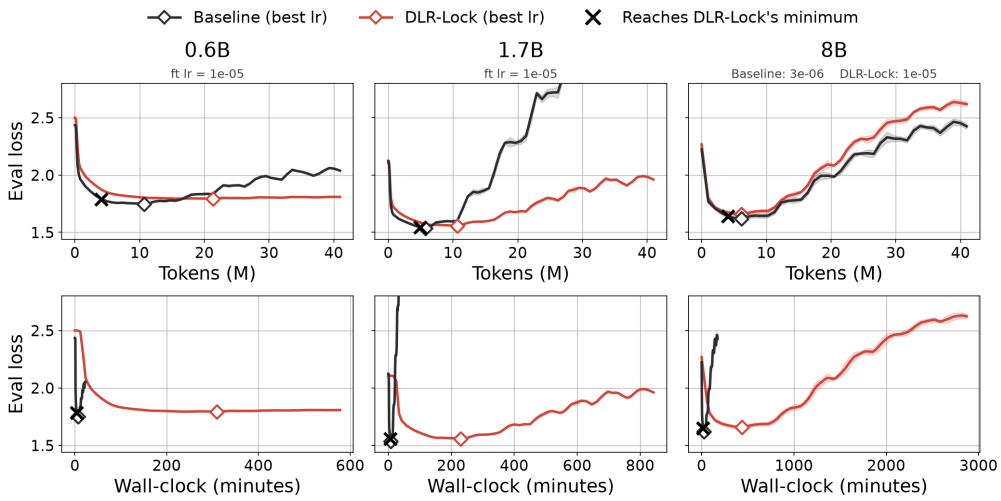


Figure 12: Fine-tuning resistance on Alpaca for the from-scratch models at 0.6B, 1.7B and 8B. **Top:** loss vs. number of training tokens; **Bottom:** loss vs. wall-clock time. Both curves are U-shaped: this corpus is small enough that the baseline and the locked model alike eventually overfit it, so each arm is compared at its own minimum, the diamonds, rather than at the end of the budget. Each arm is shown at the rate that gives it the lowest loss it attains, annotated above the panel; the cross marks where the baseline first reaches the locked model's minimum during its own descent. Shaded regions indicate ± 1 standard deviation across 3 seeds.

from the loss as detailed in Appendix E.3. Both arms run under the same protocol as WikiText-103 above: 41M tokens, the same sweep, three seeds and identical hardware. Both also cycle over this small corpus and eventually overfit, so the curves of Figure 12 are U-shaped and we compare each arm at its own best rate and its own minimum loss, where an attacker monitoring eval loss would stop. The locked model's minimum sits 0.022 to 0.045 above the baseline's, a small but persistent quality ceiling. Read as a cost to reach a fixed target, that minimum costs the baseline $\mathbf{5.2}\times$, $\mathbf{2.2}\times$ and $\mathbf{1.5}\times$ fewer tokens at 0.6B, 1.7B and 8B en route to its own, lower minimum (the cross), shrinking with scale. At each arm's own minimum the token cost is $2.0\times$, $1.9\times$ and $1.0\times$ and the wall-clock cost $\mathbf{46.9}\times$, $\mathbf{27.2}\times$ and $\mathbf{16.5}\times$: at 8B the two arms bottom out after the same number of tokens, so what resists there is the quality ceiling and the wall-clock rather than the token count.

BeaverTails (harmful instruction-following). As a safety-relevant complement to the instruction-following domain above, we also fine-tune on BeaverTails [Ji et al., 2023], a corpus of prompts paired with responses that a majority of crowd annotators flagged unsafe; formatting and masking follow the same convention as Alpaca (Appendix E.3). After deduplication and the majority-unsafe filter this corpus is small, so both arms cycle over it many times to reach the same 41M-token budget as the other domains, and both eventually overfit: Figure 13 shows eval loss falling, reaching a minimum, then rising again at every scale, for the baseline and the locked model alike. We therefore compare each arm at its own best rate and its own minimum loss (the diamonds) rather than a fixed target, since an attacker monitoring eval loss would stop at that point regardless of arm. The locked model's minimum sits 0.046–0.053 above the baseline's at every scale, a small but persistent quality ceiling on this specific objective. Read as a cost to reach a fixed target, the locked model's minimum loss costs the baseline $\mathbf{3.2}\times$, $\mathbf{2.2}\times$ and $\mathbf{1.9}\times$ fewer tokens at 0.6B, 1.7B and 8B en route to its own, lower minimum (the cross), shrinking with scale as on Alpaca. Wall-clock resistance at each arm's own minimum remains large, at $\mathbf{39.4}\times$, $\mathbf{17.2}\times$ and $\mathbf{16.9}\times$, driven by the same per-step overhead as every other domain rather than anything specific to this one. The same mechanism resists adaptation here as on every other domain in this section. This is the practical content of our domain-agnostic claim: since DLR-LOCK is not designed around any particular target domain, resistance to harmful fine-tuning follows as a special case of the same, broader guarantee. Eval loss on held-out unsafe responses is the same adaptation-cost lens used throughout this section, and we report it here for consistency. It does not, however, directly measure harmfulness: it does not indicate whether the

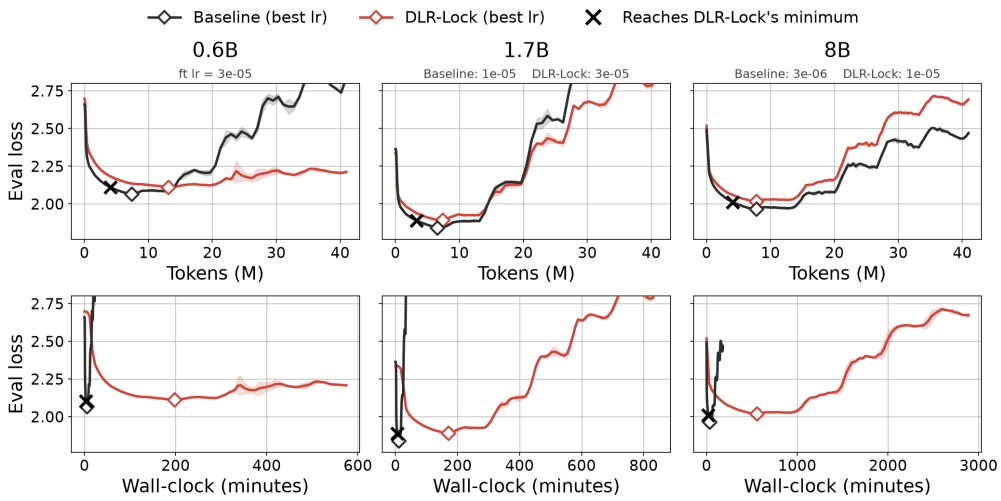
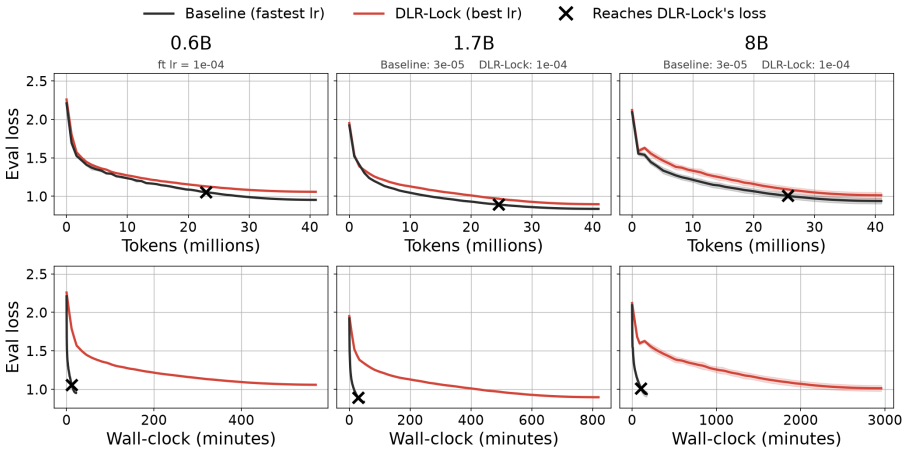


Figure 13: Fine-tuning resistance on BeaverTails for the from-scratch models at 0.6B, 1.7B and 8B. **Top:** loss vs. number of training tokens; **Bottom:** loss vs. wall-clock time. Both curves are U-shaped: repeated cycling of this small corpus eventually overfits, so "best" here is each arm's own minimum rather than its final value. Each arm is shown at the rate that gives it the lowest loss it attains, annotated above the panel; the cross marks where the baseline first reaches the locked model's minimum during its own descent. Shaded regions indicate ± 1 standard deviation across 3 seeds.

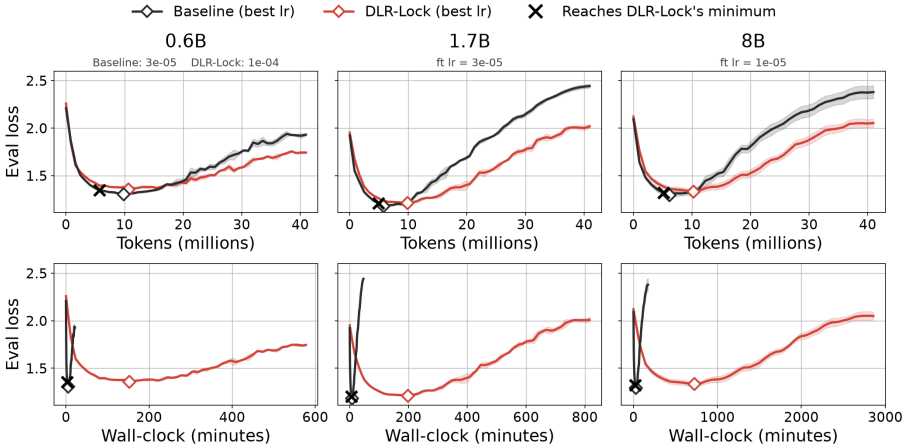
fine-tuned model's own generations are actually more toxic. That would require scoring sampled completions, e.g. with an LLM judge, which we leave to future work.

OpenMathReasoning (mathematical reasoning). We next move away from natural prose to mathematical reasoning traces, using a decontaminated and compliance-filtered copy of OpenMath-Reasoning [Moshkov et al., 2025]. The corpus is consumed as raw text under the same next-token objective as WikiText-103, with no prompt template, and we run two settings: streaming fresh tokens for the section's 41M-token budget, and spending that budget cycling a fixed 5M-token subset, the data-limited regime of Appendix E.1. Streaming, in Figure 14a, both arms descend monotonically and the locked model stays above the baseline throughout, ending 0.105, 0.061 and 0.075 higher at 0.6B, 1.7B and 8B. These gaps sit above the baseline's own spread across seeds, which stays under 0.04 at every scale, so the ceiling is resolved here rather than inferred. Read as a cost to reach a fixed target, the locked model's final loss costs the baseline $\mathbf{1.8}\times$, $\mathbf{1.7}\times$ and $\mathbf{1.6}\times$ fewer tokens, and $\mathbf{47.7}\times$, $\mathbf{28.8}\times$ and $\mathbf{28.0}\times$ less wall-clock. Capping the data at 5M tokens, in Figure 14b, turns both curves into the U-shape of the other fixed-corpus domains, and the picture does not change: the minima sit 0.050, 0.032 and 0.040 apart, the locked model needs $\mathbf{1.9}\times$, $\mathbf{2.0}\times$ and $\mathbf{2.0}\times$ the tokens the baseline needs to reach its minimum, and $\mathbf{29.0}\times$, $\mathbf{30.0}\times$ and $\mathbf{27.4}\times$ the wall-clock at each arm's own minimum. Restricting the attacker's data therefore does not weaken the defense on this domain; if anything the token ratio grows.

StarCoderData (source code). Our last loss-based domain is source code, using StarCoderData [Li et al., 2023], which also carries issue and pull-request discussions alongside code in many languages, again as raw text under the two settings above. This is the one domain where the quality ceiling disappears. Streaming, in Figure 15a, the arms end 0.074, 0.047 and 0.037 apart, but the baseline's own spread across seeds is 0.15 to 0.18, several times those gaps, so we claim no ceiling on code. Under the 5M-token cap, in Figure 15b, the two minima coincide within that noise, 2.92 against 2.96 at 0.6B and within 0.03 at the two larger scales, where the locked model's minimum is in fact the lower of the two. What survives is the cost of getting there. Streaming, the locked model's final loss costs the baseline $\mathbf{1.5}\times$, $\mathbf{1.6}\times$ and $\mathbf{1.6}\times$ fewer tokens and $\mathbf{39.7}\times$, $\mathbf{26.1}\times$ and $\mathbf{26.6}\times$ less wall-clock; under the cap, where the two arms bottom out within 1.1 to $1.3\times$ the same number of tokens, the wall-clock ratios at each arm's own minimum are $\mathbf{38.0}\times$, $\mathbf{20.2}\times$ and $\mathbf{23.8}\times$. Because the two arms reach the same loss here, these are the section's cleanest iso-quality costs: no part of the ratio can be attributed to the locked model settling for a worse optimum.



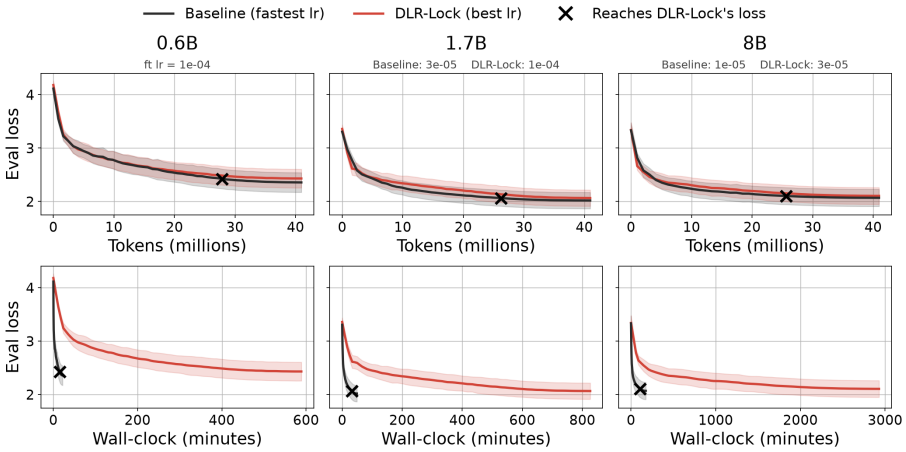
(a) Streaming



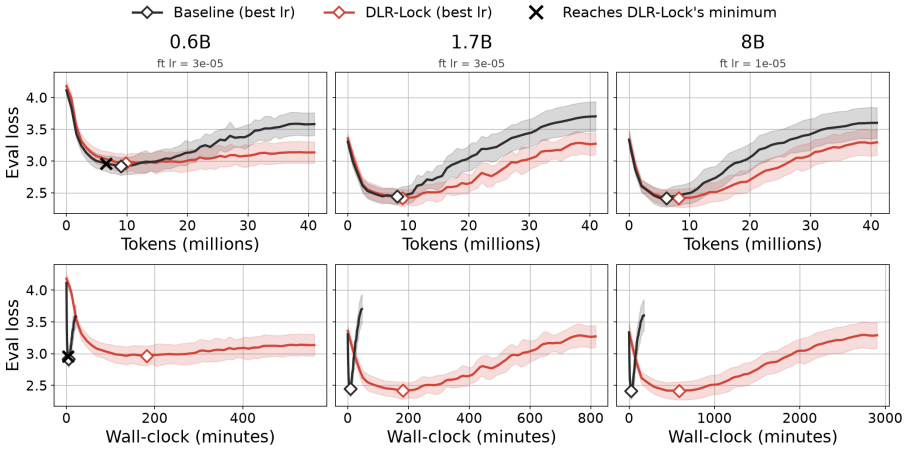
(b) 5M-token cap

Figure 14: Fine-tuning resistance on OpenMathReasoning for the from-scratch models at 0.6B, 1.7B and 8B, streaming fresh tokens in (a) and cycling a fixed 5M-token subset for the same budget in (b). In each, **top:** loss vs. tokens; **bottom:** loss vs. wall-clock time. Rates are annotated above each panel: in (a) the locked model's lowest-final-loss rate and the baseline's fastest rate to that loss; in (b), where both arms overfit the subset, each arm's lowest-minimum rate, with diamonds at those minima. The cross marks where the baseline first reaches the locked model's loss. Shaded regions indicate ± 1 standard deviation across 3 seeds.

SST-2 (sentiment classification). We close with the sparsest supervision in this section: SST-2 [Socher et al., 2013], a binary task in which each example contributes roughly two supervised tokens (Appendix E.3). Both arms are fine-tuned with batch size 32, sequence length 512, and prompt-masked cross-entropy on the label tokens. For each model size we show each arm at its own best rate: the learning rate from $\{10^{-5}, 3 \times 10^{-5}, 10^{-4}\}$ that reaches 90% accuracy in the fewest optimizer steps, averaged over three seeds. Figure 16 shows the two arms reaching 90% accuracy within ten optimizer steps of each other at every scale, and at 8B the locked model gets there first. Only the per-step cost survives, at $\mathbf{29.1}\times$, $\mathbf{20.8}\times$ and $\mathbf{14.7}\times$ longer in wall-clock to the same threshold. This is the domain where we would expect the least resistance: binary sentiment classification is plausibly close to a readout problem, since the pretrained model already computes features that largely separate the two classes, so adapting little more than the output layer suffices and little pressure falls on the feed-forward blocks that DLR-LOCK replaces.



(a) Streaming



(b) 5M-token cap

Figure 15: Fine-tuning resistance on StarCoderData, with the layout and conventions of Figure 14: streaming in (a), a fixed 5M-token subset in (b). Shaded regions indicate ± 1 standard deviation across 3 seeds, and they are wide on this domain: the baseline's spread across seeds exceeds every arm-to-arm gap we measure here. Only the 0.6B panel of (b) carries a cross, since at 1.7B and 8B the baseline never falls to the locked model's minimum, which is in fact marginally lower than its own. Past that minimum the locked model's loss rises more slowly than the baseline's, the same slower adaptation seen from the other side.

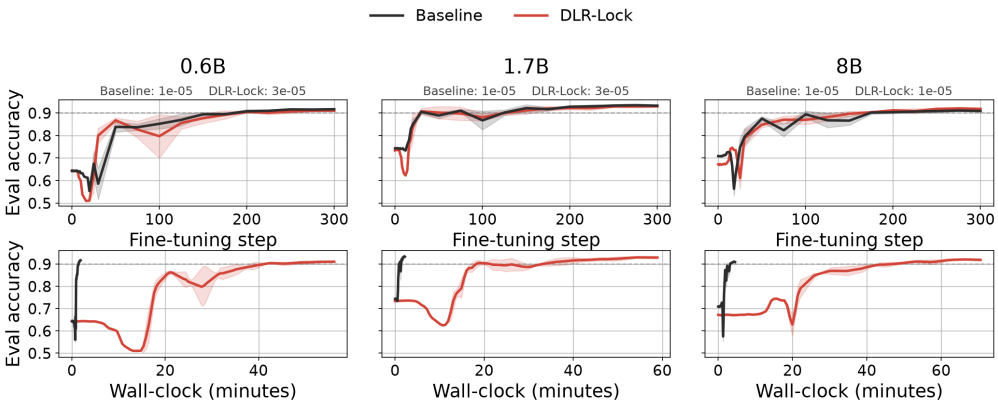


Figure 16: Fine-tuning resistance on SST-2 for the from-scratch models at 0.6B, 1.7B and 8B. **Top:** eval accuracy vs. fine-tuning step; **Bottom:** eval accuracy vs. wall-clock time. The dashed line marks 90% accuracy, and the rate at which each arm reaches it fastest is annotated above the panel. Reaching it takes 185 steps for the baseline against 191 for the locked model at 0.6B, 29 against 29 at 1.7B, and 174 against 164 at 8B. Shaded regions indicate ± 1 standard deviation across 3 seeds.

E.3 Prompt templates and verbalizers

This subsection collects the exact prompt templates and verbalizer sets used by the instruction-tuning (Figure 12), harmful-instruction (Figure 13), and classification (Figure 16) experiments of Appendix E.2. All experiments use the same templates across the baseline and DLR-Lock arms.

Alpaca (instruction tuning). We use the original Stanford Alpaca template [Taori et al., 2023] in two variants depending on whether the example carries an optional `input` field. For examples with an `input` field:

```
Below is an instruction that describes a task, paired with an input that provides
further context. Write a response that appropriately completes the request.

### Instruction:
{instruction}

### Input:
{input}

### Response:
{response}
```

For examples without an `input` field (the majority):

```
Below is an instruction that describes a task. Write a response that
appropriately completes the request.

### Instruction:
{instruction}

### Response:
{response}
```

The cross-entropy loss is masked on all template + instruction (+ input) tokens; only the `{response}` tokens (followed by a single end-of-text marker) contribute to the gradient. Per-example supervised token counts therefore range from a few dozen to a few hundred. Examples are packed into `seq_len=2048` chunks separated by end-of-text tokens; the prompt mask is preserved across example boundaries within a packed chunk. Reported eval loss is the same prompt-masked cross-entropy on a held-out 5% split of Alpaca, evaluated on response tokens only.

BeaverTails (harmful instruction-following). We use BeaverTails [Ji et al., 2023], a corpus of (prompt, response) pairs each annotated by multiple crowdworkers for safety. We aggregate annotations per unique pair and keep only those where a majority of annotators flagged the response `unsafe`, then format each surviving pair with the same no-`input` Alpaca template above, with `{instruction}` set to the prompt and `{response}` to the flagged-unsafe response. Loss masking and packing follow the same convention as Alpaca: the cross-entropy loss is masked on all template and prompt tokens, only response tokens (followed by an end-of-text marker) contribute to the gradient, and examples are packed into `seq_len=2048` chunks with the prompt mask preserved across example boundaries. Train and eval pairs come from the dataset's own disjoint `330k_train` and `330k_test` splits, respectively; reported eval loss is the same prompt-masked cross-entropy on `330k_test`, evaluated on response tokens only.

SST-2 (sentiment classification, 2-way). Each example renders as

```
Sentence: {sentence}
Sentiment:
```

Predictions are obtained as $\arg\max$ over the first BPE token IDs of the verbalizer set {" positive", " negative"} (note the leading space) computed at the token position immediately following the prompt. The cross-entropy loss is computed only on the label tokens (the prompt is masked); this restricts the supervised signal to $\sim$2 tokens per example (the label token plus an end-of-text marker).

F DLR-LOCK ablations

This appendix ablates two axes of the defender's architecture knob: Appendix F.1 covers the choice of block form and nonlinearity placement, and Appendix F.2 covers the rank–depth trade-off within that block form.

F.1 Architecture ablation

Much of the DLR-Net's shape is forced rather than chosen. Depth is what raises activation memory, while utility preservation caps the parameter count and the inference cost; together these force a low-rank bottleneck and fix the depth from the budget (Section 3.1). What is left open is the nonlinearity, and it does double duty: the activation is what keeps a depth-L stack from collapsing to a single linear map, while a nonlinearity acting on the d-dimensional stream, the RMSNorm in our block, is what obliges the attacker to store $\Theta(d)$ per layer rather than the $\Theta(r)$ the activation alone would need. The remaining choices, the form of the residual block and the exact activation and normalization, do not follow from that reasoning, and we settled them empirically.

Synthetic sweep. The design was originally fixed on a synthetic sweep over the block form, residual scaling, normalization, initialization scale and learning rate, distilling a randomly initialized SwiGLU teacher at d=128. At rank r=32, depth 192 and matched settings our block reached 0.355 relative MSE against 0.395 for $\boldsymbol{h} + \sigma(\boldsymbol{U}\boldsymbol{V}\,\mathrm{RMSNorm}(\boldsymbol{h}))$, which moves the nonlinearity outside the low-rank map, and giving each its own best setting moved both by under 2.5% without changing the ordering. The placement of the nonlinearity therefore dominated, while residual scaling and initialization scale were second-order at 1–4%, and a low-rank adapter baseline with no depth was uniformly worse at 0.53–0.72. This sweep is the only evidence we have for the scaling and initialization axes, which the real-data study below holds fixed.

Real Phase-1 setting. We therefore repeat the comparison in the actual module-wise regression setting, distilling one real Qwen3-0.6B SwiGLU residual sub-block into each variant at a matched parameter budget, which at r=32 gives depth L=141. Each variant trains for 50k steps, a tenth of the 500k Phase 1 uses in the pipeline, on Nemotron-CC activations at three learning rates bracketing the value used for module-wise regression, $\{3\times10^{-4}, 10^{-3}, 3\times10^{-3}\}$, and we report the best of the three. At this budget that best rate is usually the top of the bracket, so a wider sweep would likely lower the absolute values somewhat; what the comparison rests on is the ranking, and the two penalties we draw conclusions from below hold at all three rates and all three layers. All six variants for a given layer share one teacher forward and the same activation batches, which makes the comparison paired. We run this at an early, a middle and a late layer, so that the ordering is not read off one depth. A low-rank SwiGLU block, $\boldsymbol{h} + \boldsymbol{U}\big((\boldsymbol{V}\tilde{\boldsymbol{h}}) \odot \sigma(\boldsymbol{W}\tilde{\boldsymbol{h}})\big)$ with $\tilde{\boldsymbol{h}} = \mathrm{RMSNorm}(\boldsymbol{h})$, is not among the variants: its third matrix per layer buys only two thirds the depth at a fixed parameter budget, L=95 against 141 at r=32, and depth is what carries the locking, so it is ruled out by the budget rather than by regression fidelity. Table 6 reports the result.

Findings. The residual connection is the one load-bearing choice: removing it is the only variant that fails outright, at 76–2000$\times$ the reference. Moving the nonlinearity outside the low-rank map costs 1.13–$2.65\times$ and replacing it with the identity costs 1.39–$3.28\times$, in both cases with the widest margin at the middle layer and the narrowest at layer 2. Removing RMSNorm is the one axis fidelity does not settle: it costs $4.4\times$ at layer 2 and $1.11\times$ at layer 14, but at layer 27 it edges ahead of the reference. Two limits are worth stating. A single distillation seed per cell means differences under roughly 10% should not be read as real, which is the range the RMSNorm variants fall in. And this measures Phase-1 fidelity on one module, not end-to-end quality after Phase 2 or locking strength after fine-tuning, so it settles the block form rather than the whole design.

Table 6: Architecture ablation in the module-wise regression setting, reporting relative MSE $\times 10^3$ when regressing one Qwen3-0.6B SwiGLU residual sub-block for 50k steps at the best of three learning rates, lower is better. Each column is one layer, and the reference is the locking configuration at r=32, L=141. Depth follows from the parameter budget, so the norm-free variant, which no longer spends parameters on L scale vectors, runs two layers deeper at L=143. The residual-removed variant also drops ReZero, since a zero-initialized α_i with no residual path emits exactly zero at zero gradient, so no single-variable version of it exists.

Variant	Layer 2	Layer 14	Layer 27
Ours, $\boldsymbol{h} + \boldsymbol{U}\sigma(\boldsymbol{V}\,\mathrm{RMSNorm}(\boldsymbol{h}))$	0.50	0.47	5.91
Nonlinearity outside the low-rank map, $\boldsymbol{h} + \sigma(\boldsymbol{U}\boldsymbol{V}\,\mathrm{RMSNorm}(\boldsymbol{h}))$	0.57	1.24	9.93
RMSNorm removed	2.18	0.52	5.20
One RMSNorm shared across all L layers	0.63	0.50	5.48
Identity activation, $\sigma = \mathrm{id}$	0.69	1.53	12.85
Residual connection removed	996	945	450

F.2 Rank–depth ladder: locking strength vs. forward latency

The DLR-Net replacement is parameterized by a single shape knob, the low-rank dimension r; the depth L is then chosen so that the DLR-Net matches the original MLP's parameter count (Section 3). A smaller r buys a deeper stack at the same parameter budget, which should make the locked model harder to fine-tune (more residual layers to backpropagate through, more activation memory) but also slower at inference: each of the L layers depends on the previous one's output, so the forward pass is an irreducible chain of L sequential steps. A well-optimized kernel may reduce this overhead to a negligible level, but Section 4.3 shows it still grows with L in practice, even with compilation and a custom kernel tried in this paper. Note that matching the original parameter count is a choice, not a requirement: relaxing it opens a larger design space, trading network expressivity, locking strength, and inference overhead against each other, along with weight memory and the defender's locking compute budget; we study only the parameter-matched slice here. This ablation traces that frontier empirically across the rank ladder.

Setup. We use the from-scratch models of Appendix E.2: the locking corpus and the pretraining corpus are identical by construction, so any change in locking strength as r varies is attributable to the architectural knob rather than to a shifting domain gap. Concretely, we lock the from-scratch 1.7B model at seven rank–depth trade-off points, $r \in \{64, 128, 256, 512, 1024, 2048, 4096\}$ with depths $L \in \{142, 71, 35, 17, 8, 4, 2\}$, plus an r=9216, L=1 no-depth control: a single SiLU-activated layer parameter-matched to SwiGLU, which needs hidden width $\frac{3}{2}d_{\mathrm{ff}}$ since it has two matrices where SwiGLU has three. Module-wise regression and end-to-end logit distillation follow the same protocol as Section 3, using the same distillation configuration as the fine-tuning experiments of Appendix E.2 (lr=10^{-4}, k=1000) at every rung of the ladder. Locking strength is measured by the WikiText-103 fine-tuning curves (same protocol as Figure 5), and forward latency by the single-MLP overhead benchmark of Appendix H.1.

Quality preservation across the ladder. Table 7 shows the quality of the locked 1.7B model at every rung of the ladder. Within this locking budget, capabilities are flat across all seven rank–depth points and the no-depth control, within noise of the pretrained baseline and of each other: the defender can move along r without any quality cost. A larger locking budget might create a difference among them in performance, but it is already small enough that this does not matter here. This is the precondition for the trade-off question to be well-posed: since quality is essentially flat along r, any difference in fine-tuning resistance or inference latency below is attributable to the rank–depth knob itself rather than to a confounding capabilities gap.

Locking strength vs. forward latency. Figure 17 shows the resulting WikiText-103 fine-tuning curves for every rung of the ladder. Locking strength is essentially depth-invariant across the ladder. In the left panel, all eight locked curves collapse into a narrow band that plateaus clearly above the pretrained baseline, and the single-layer degenerate control at L=1, parameter-matched to the SwiGLU MLP it replaces, reaches the same plateau as the deepest configuration at L=142: depth is not what carries the fine-tuning resistance. In the right panel, the ladder produces the expected

Table 7: Quality preservation across the rank–depth ladder for our 1.7B model, pretrained from scratch on Nemotron-CC, after model locking, averaged over 3 distillation seeds. All DLR-Locked rows use the same logit distillation configuration (lr=10^{-4}, k=1000). r = rank, L = depth: higher L means stronger locking but more inference overhead. †The r=9216, L=1 row is the no-depth control: a single SiLU-activated layer, parameter-matched to SwiGLU. WT = WikiText-103, out-of-distribution for this model; ARC-E/C = ARC Easy/Challenge; HSw = HellaSwag; WiGr = WinoGrande.

			Perplexity ↓	Accuracy (%) ↑							
	r	L	WT	MMLU	ARC-E	ARC-C	HSw	WiGr	BoolQ	PIQA	GSM8K
Pretrained	—	—	18.2	24.7	60.4	32.6	54.1	54.1	54.6	71.3	2.7
DLR-Locked	64	142	19.2	24.6	59.8	31.6	52.4	53.7	52.1	70.0	2.1
	128	71	19.2	25.0	59.8	31.1	52.5	55.6	52.2	70.0	2.2
	256	35	19.2	24.5	60.1	31.6	52.3	54.3	52.7	70.6	1.8
	512	17	19.2	25.4	60.0	31.2	52.3	54.8	51.0	70.4	2.4
	1024	8	19.4	24.8	59.7	31.9	52.2	54.9	52.1	69.8	2.2
	2048	4	19.4	24.7	60.0	32.2	52.3	53.5	52.4	69.7	1.9
	4096	2	19.3	25.3	60.2	32.3	52.1	54.3	51.5	70.0	2.4
	9216†	1	19.0	25.1	60.4	31.5	52.3	54.8	50.9	70.5	1.6

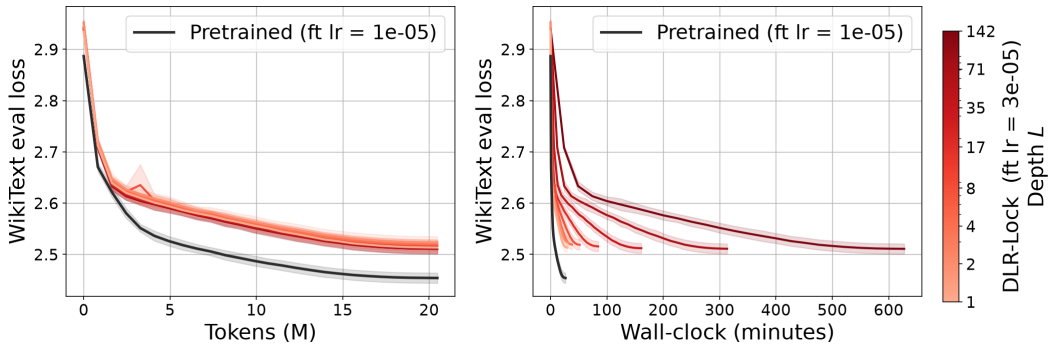


Figure 17: Locking strength is depth-invariant across the rank–depth ladder of Table 7, on the from-scratch 1.7B model, fine-tuned on WikiText-103. Mean ±1 standard deviation across 3 seeds. **Left:** loss vs. training tokens. **Right:** loss vs. wall-clock time. Each arm is shown at its own attacker-favorable learning rate: pretrained at 10^{-5}, locked at 3×10^{-5} for every depth.

spread: deeper networks take longer per step, so higher L moves the curve to the right, but every locked curve still plateaus above the baseline. Because the architectural mismatch that drives this resistance (Section 4.4) does not depend on depth—even the single-layer control already differs structurally from the original SwiGLU MLP—the defender can choose L freely to fit an inference-latency budget from Figure 4 without a resistance cost. Given that budget, the defender can still choose a deeper L to strengthen the other two mechanisms of Section 4.4, activation memory growth and the disproportionate backward-pass overhead, both of which scale with depth.

G Attack strategies

This section reports the empirical counterparts of the strategies analyzed in Section 3.3, each one a way the attacker might reduce the cost the defense imposes. Appendix G.3 closes with one knob that section does not analyze, the fine-tuning schedule.

G.1 Low-rank adaptation (LoRA) on attention

Section 3.3 argued that LoRA does not circumvent the defense: it does not shorten the backward pass, and adapters on the DLR-Net's already thin factors save so little that covering the stack approaches the cost of full fine-tuning. We test it here on the main-text setting, Qwen3-0.6B on WikiText-103. The attacker freezes the base weights and the DLR-Nets and applies LoRA

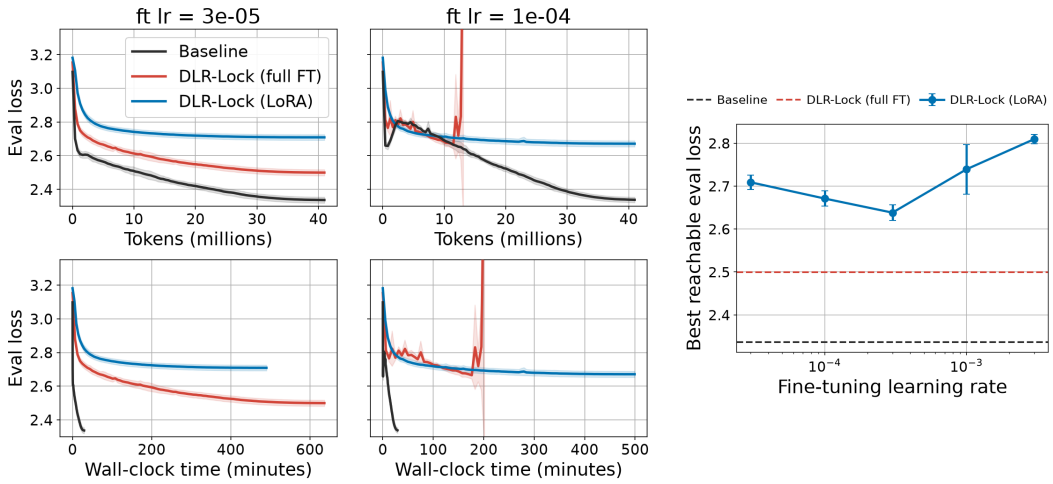


Figure 18: LoRA attack on the DLR-Locked Qwen3-0.6B, fine-tuned on WikiText-103 with the streaming cosine schedule of Figure 5. Adapters are placed on the attention projections $q/k/v/o$ with the base weights and the DLR-Nets frozen, and bands span ± 1 standard deviation over three seeds. **Left:** eval loss vs. tokens in the top row and wall-clock time in the bottom row, at matched fine-tuning rates $\eta \in \{3{\times}10^{-5}, 10^{-4}\}$, for the baseline, DLR-Lock under full fine-tuning, and DLR-Lock under the LoRA attack. The vertical axis is shared and clipped, so DLR-Lock under full fine-tuning leaves the frame when it diverges at $\eta{=}10^{-4}$. **Right:** best reachable eval loss vs. fine-tuning rate across the full sweep $\{3{\times}10^{-5}, \ldots, 3{\times}10^{-3}\}$; dashed lines mark the best baseline and full-fine-tuning losses.

of rank $\rho{=}16$ and scaling $\alpha{=}32$ to the attention projections. The protocol matches Figure 5: 5000 steps, batch 4×2048, cosine schedule, three seeds. We sweep the fine-tuning rate η over $\{3{\times}10^{-5}, 10^{-4}, 3{\times}10^{-4}, 10^{-3}, 3{\times}10^{-3}\}$, extending above the full-fine-tuning range since LoRA tolerates larger rates.

With gradient checkpointing. At matched fine-tuning rates, on the left of Figure 18, the LoRA attack converges to a higher eval loss than the unlocked baseline and never reaches it, since it adapts only the attention projections and only with a low-rank update. It is faster than full fine-tuning of the locked model, which also updates the deep residual stacks, yet it stays far slower than fine-tuning the unlocked model: reaching the attention adapters still forces the backward pass to traverse every depth-L residual stack, and gradient checkpointing recomputes each stack's forward along the way. Sweeping the fine-tuning rate upward, on the right, does not close the gap either: the best reachable loss improves and then turns up before reaching the baseline, bottoming out at 2.638 at $\eta{=}3{\times}10^{-4}$, against 2.499 for full fine-tuning of the locked model and 2.337 for the baseline.

Without gradient checkpointing. Even with LoRA's small optimizer footprint, fine-tuning without checkpointing exhausts an 80 GB H100 at the main-text batch of 4×2048: every run fails during the forward pass, independent of the fine-tuning rate. LoRA sheds optimizer state and frozen-weight gradients but not activations: the frozen DLR-Net stacks alone still retain $\Theta(T\,N\,L\,d)$ of them, on the order of 70 GB at this batch (Proposition B.1(b)), and the model weights and attention activations then push the total past the 80 GB budget. The attacker is therefore forced either to checkpoint, paying the recompute cost above, or to shrink the effective batch, the same activation-memory amplification of Proposition 3.1.

G.2 Reverse distillation

Of the strategies in Section 3.3, reverse distillation matters most, since its cost is amortized across adaptations rather than recurring per task (Remark 2.4). Three things stand in the attacker's way, one a matter of cost and two structural. First, running our pipeline in reverse costs on the order of the locking run itself, some 50 times a single fine-tuning budget. Second, every weight of the locked model is public, but the corpus it was pretrained and locked on is not. An attacker without it must distill from substitute corpora, which recovers less efficiently at every budget we ran. Third, because

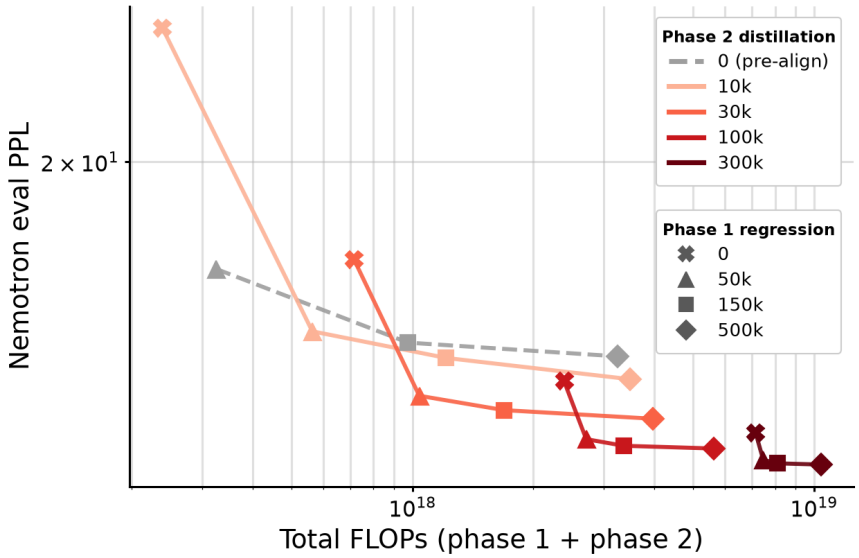


Figure 19: Reverse distillation of the locked Qwen3-0.6B, over a grid of module-wise regression and logit distillation steps. Color encodes the number of distillation steps and marker shape the number of regression steps, and each line holds the distillation length fixed while the regression length increases, matching Figure 3. The dashed line is the warm start reached by regression alone, before any distillation. Cost excludes the teacher forward on both sides, as in Figure 3; the attacker's cost per distillation step is a factor 1.26 below the defender's, since a shallow student needs no gradient-checkpoint recompute.

DLR-LOCK's Phase 2 lets the locked modules co-adapt, they no longer sit near the original MLPs they replaced, so the attacker's regression phase has neither a certified target nor a stopping rule. A final paragraph steps outside full replacement to a further attack strategy: recovering only part of the stack.

Setup. The attacker runs our pipeline in reverse, as Section 3.3 describes, copying every non-MLP weight from the locked checkpoint verbatim and relearning only the MLP sub-block from random initialization. The two phases mirror ours: module-wise regression onto each locked module's residual map, then top-k logit distillation against the locked model's outputs. We sweep the number of regression steps (0, 50k, 150k, 500k) against the number of distillation steps (0, 10k, 30k, 100k, 300k) on the locked model of Table 1, the only checkpoint and recovery pipeline we study here, where a step count of 0 skips that phase, each cell a dedicated run with its own cosine schedule at the same per-step token counts as the defender's phases (Appendix D), and evaluate every cell on one fixed held-out sample of 819k tokens. None of DLR-LOCK's locking mechanisms discussed in the main text applies to this route. The locked model enters only as a frozen forward-only teacher. What the attacker pays is the volume of distillation that recovery needs.

Recovery cost. Figure 19 maps the resulting frontier on the same accounting as the defender's Figure 3, so the two horizontal axes mean the same thing. The frontier is the set of allocations that no other beats on both cost and perplexity. The cheapest allocation in our grid to reach its lowest perplexity band, 50k regression steps with 300k distillation steps, costs 7.5×10^{18} floating point operations, against 6.3×10^{18} for the locking run it attacks, 500k Phase-1 steps followed by 100k Phase-2 steps. Reaching that quality therefore costs slightly more than locking the model did, and roughly 50 times the 5000-step baseline fine-tuning run of Section 4.4. A single-task attacker has no reason to take this route, and it pays only for one who reuses the recovered model across many adaptations or passes it on, the regime of Remark 2.4. Perplexity is still falling at the largest budget we ran, so no allocation shows recovery having converged, and the attacker cannot read the frontier as the price of a finished job. The allocation above is about a fifth of a Chinchilla compute-optimal pretraining budget at this model size [Hoffmann et al., 2022]. Being both the cheapest in our grid and short of convergence, it is a lower bound: recovery costs at least that much, and so

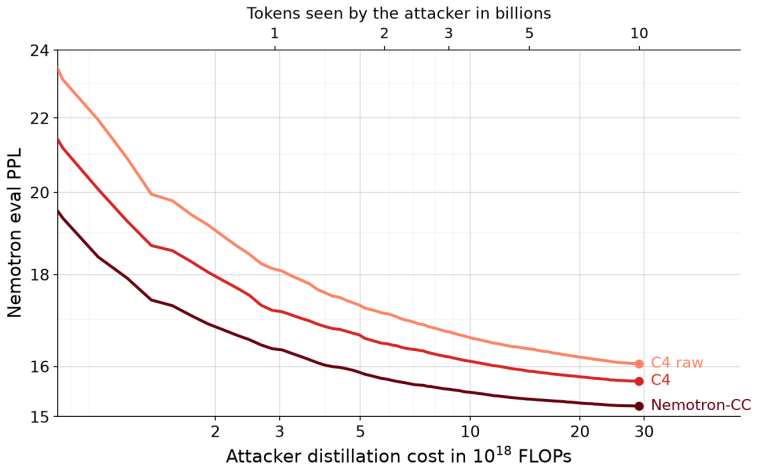


Figure 20: Reverse distillation from three corpora against the same locked Qwen3-0.6B, under pure logit distillation with no regression warm start. Nemotron-CC is the corpus the model was locked on, and C4 in cleaned and raw form stands for commodity web text. Cost is on the axis of Figure 19 and perplexity on its evaluation sample, so the two figures can be read together. All three arms stop at 10B tokens and run at a constant learning rate of 10^{-3}.

is not negligible beside pretraining a comparable model outright. Within the frontier, regression saturates. Treating cells within 0.1 perplexity as tied, since evaluations move by that much between neighboring checkpoints, no cell with 500k regression steps is cost-efficient: each is matched by a cheaper allocation that spends less on regression and more on distillation.

Corpus dependence. In practice the attacker does not hold the corpus the model was locked on. Figure 20 measures recovery without it. Three arms distill the same locked model under pure logit distillation, from Nemotron-CC and from C4 [Raffel et al., 2020] in its cleaned and raw forms, on the cost axis of Figure 19. At 2.9×10^{19} floating point operations, four times the cheapest allocation on that frontier to reach its lowest perplexity band, the matched-corpus arm reaches 15.20 against 15.70 and 16.05 for cleaned and raw C4, and all three are still descending. The gap to the matched corpus is the reading that matters, and it holds at every budget shown. The two C4 arms also order by curation. Efficient recovery thus benefits from a corpus close to the one the model was locked on, which the provider holds and the attacker in general does not. The grid above gave the attacker that corpus throughout, so its cost results do not rest on this asymmetry.

The missing guarantee. Beyond cost, the two directions differ in what the regression phase can certify, since depth generally buys expressivity that width cannot cheaply match [Telgarsky, 2016, Eldan and Shamir, 2016]. DLR-LOCK's Phase 1 regresses a deep DLR-Net onto the shallow MLP it replaces, a direction in which the student is if anything the more expressive of the two, so a close fit is likely to exist and the loss measures how near the student has come. The attacker regresses the other way, where existence looks assured all the same: each DLR-Net was fitted to an original MLP, which therefore reproduces it. That holds of the module Phase 1 produces, but the attacker's target is what Phase 2 leaves behind. Phase 2 optimizes the composition and lets the modules co-adapt to cancel one another's error, so each locked module drifts from the MLP it replaced while the full function is preserved. A drifted module may therefore lie outside what a shallow MLP can express; existence would be assured only if the MLP class fully contained the DLR-Net's, which we have no reason to expect and do not establish. A low regression loss therefore does not certify that the attacker is approaching the original model, and nothing in it says when to stop. This is an argument about what the attacker can rely on rather than a penalty we measure, and it would bite hardest near a converged solution, which these budgets do not reach.

Partial replacement. Everything above concerns full replacement, in which every locked module is relearned. The attack also generalizes: an attacker could recover only some of the layers and leave the rest locked. We do not measure this case. Partial replacement interpolates between two cases we

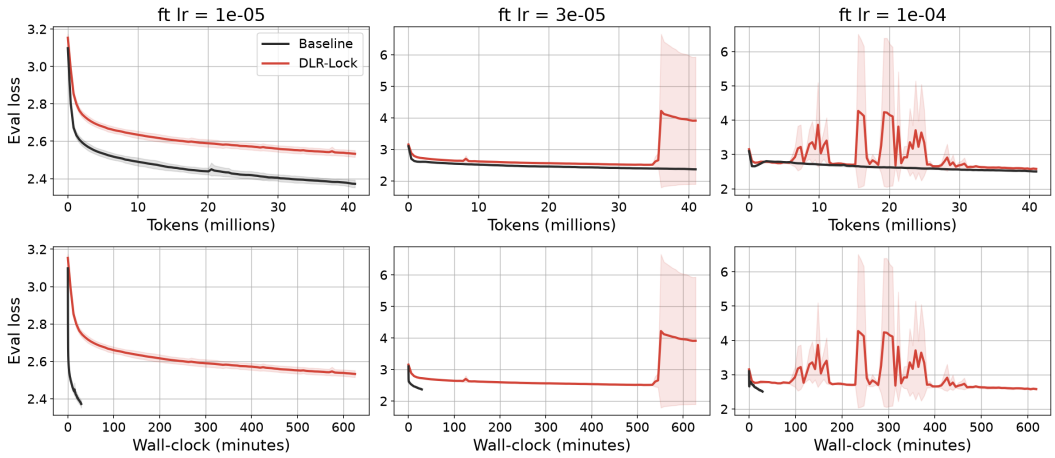


Figure 21: Constant learning-rate version of Figure 5: only the schedule changes, from cosine to constant after warmup. Panels, axes and shading are as in that figure.

do measure: fine-tuning the locked model directly, with nothing recovered (Section 4.4), and full replacement, with everything recovered. Its cost and the lock it leaves both fall between them, since every layer left in place still forces the backward pass through its own depth-L residual stack, which freezing parameters does not remove, as Section 3.3 argues for the frozen-DLR-Net attack. Partial replacement therefore trades the cost of recovering a layer against the cost of adapting around the layers still locked, rather than escaping either.

G.3 Constant learning-rate fine-tuning

The main text's fine-tuning sweep, Figure 5, uses a cosine schedule with a short warmup, so the persistent loss gap it reports could in principle be an artifact of annealing: the locked model, whose backward pass we claim to be optimization-unfriendly, might simply be denied the late-stage step sizes it would need to close the gap. We therefore re-run that sweep with the decay removed and every other knob held identical: the same checkpoints, dataset stream, batch size, number of steps and seeds. The protocol is that of Figure 5, 5000 steps at batch 4×2048 over three seeds, with the learning rate held at its peak after a 5% warmup instead of decaying, and we sweep $\eta \in \{10^{-5}, 3{\times}10^{-5}, 10^{-4}\}$.

Results. Removing the decay does not close the gap. At the two rates where the locked model trains stably, its best eval loss exceeds the baseline's by 0.16 at $\eta{=}10^{-5}$ and 0.14 at $3{\times}10^{-5}$, against 0.17 and 0.16 under cosine, so the gap is the same size with or without annealing. The baseline is not being deflated by the schedule either: at every rate its final loss is also its best and is still falling over the last 500 steps, so it has not stalled at its peak rate. What the constant schedule does change is where the locked model becomes unstable, and it moves it down: under cosine the locked model diverges only at $\eta{=}10^{-4}$, whereas here it bottoms out at 2.51 by step 4100 at $3{\times}10^{-5}$ and then climbs to 3.91, while the baseline at that rate keeps descending. A constant rate therefore narrows the attacker's usable range rather than widening it. At $\eta{=}10^{-4}$ the two arms end only 0.07 apart, but neither is usable there: the locked model's loss swings above 6 through most of the run, and the baseline is itself degraded to 2.51 from the 2.37 it reaches at lower rates. The wall-clock claim of Section 4.4 thus comes with a quality claim: the best loss the attacker reaches anywhere in this sweep is 2.51 against the baseline's 2.37, so the locked model's reachable loss is genuinely worse, not only slower to attain.

H Implementation and timing measurements

H.1 Inference latency

A natural concern is that replacing each wide SwiGLU MLP with a chain of many thin low-rank residual layers is prohibitively slow at inference. The replacement is FLOP-matched to SwiGLU by

construction, so it adds no arithmetic; the cost is instead wall-clock, arising from the launch overhead of many small kernels and from matrix multiplications too thin to fully utilize the hardware. This overhead may look inevitable, but we show that standard compilation and a few simple custom-kernel strategies largely close the gap. A more thorough kernel implementation specialized to each configuration is beyond the scope of this paper, and is itself an interesting and important direction for future work. This appendix gives the measurement protocol and our kernel implementation, then reports the full overhead sweep across model scales: the residual overhead shrinks with model size, and at the full-model level the locked model is near-transparent at long sequences.

H.1.1 Measurement setup and kernel implementation

All latency measurements are on a single NVIDIA H100 GPU in `bfloat16`; full-model measurements use FlashAttention [Dao, 2024]. To remove run-to-run variance from dynamic clock scaling, we lock the GPU core and memory clocks with `nvidia-smi` for every measurement, baseline and DLR-Net alike. Each configuration is timed as the minimum over 100 iterations after 100 warmup iterations, repeated over 10 passes; we report the mean across passes, with error bars showing the across-pass standard deviation.

Compilation. Both the SwiGLU baseline and the DLR-Net are compiled with `torch.compile` [Ansel et al., 2024] in its `reduce-overhead` mode, which traces each model into a graph, fuses elementwise operations, and replays the forward pass from a captured CUDA graph. For the DLR-Net this is the first of two variants we report; the second additionally replaces each MLP with a custom kernel (below). Compilation helps the DLR-Net disproportionately: its L-layer chain dispatches many tiny kernels whose dominant cost is the per-launch CPU overhead of issuing each one, and CUDA graph replay issues the entire captured sequence with a single host-side launch. It does not remove the on-device cost of executing those many small, memory-bound kernels, which is what the custom kernels below target.

Custom kernel implementation. The DLR-Net replaces one wide SwiGLU MLP with a chain of L thin low-rank residual layers, each applying two skinny matrix multiplications through a rank-r bottleneck. The remaining cost is memory traffic: the compiled graph still writes and re-reads the hidden state from high-bandwidth memory (HBM) between the many thin matrix multiplications. Our kernels remove this by fusing the L-layer chain so that the hidden state stays resident in on-chip memory and each MLP issues only a few launches. Rather than hand-tune per configuration, we implement a small set of standard strategies and, for each model scale and batch size, select whichever is fastest by a short autotuning sweep: (i) a single fused kernel (Triton [Tillet et al., 2019] or CUDA) that marches through all L layers on chip, with an optional path using warpgroup matrix instructions (WGMMA) and the Tensor Memory Accelerator (TMA) on Hopper GPUs [NVIDIA Corporation, 2022]; and (ii) an orchestrated variant that issues the matrix multiplications through NVIDIA's cuBLAS library with a fused normalization and activation epilogue and captures the whole layer chain in a CUDA graph, running a few large library matrix multiplications in place of the many small compiler-generated kernels. These are standard fusion and graph-capture techniques rather than new kernel algorithms, and we make no claim of optimality; our aim is only to show that the locking architecture's latency overhead is largely an implementation artifact removable with off-the-shelf tooling, not a fundamental cost. Our implementations are available at `https://github.com/apple-aiml-research/ml-dlr-lock`.

Full-model composition. To measure the full model, each locked MLP is exposed as a `torch.library.custom_op` [Ansel et al., 2024] and runs whichever single-MLP strategy above is fastest for that configuration. Wrapping each MLP as an opaque op lets the outer `reduce-overhead` graph capture the entire forward pass, attention and all MLPs, as one CUDA graph with no per-MLP graph or host-visible copies at the seams. On Qwen3-8B the compiled model still executes tens of thousands of small kernels per forward pass (L layers $\times$ 36 blocks); the custom kernels collapse each MLP's chain to far fewer, larger kernels, which the single outer graph then replays. At large token counts the fastest strategy is a *chunked* orchestrator that processes the tokens in fixed-size blocks, each kept resident in on-chip cache across the layer chain.

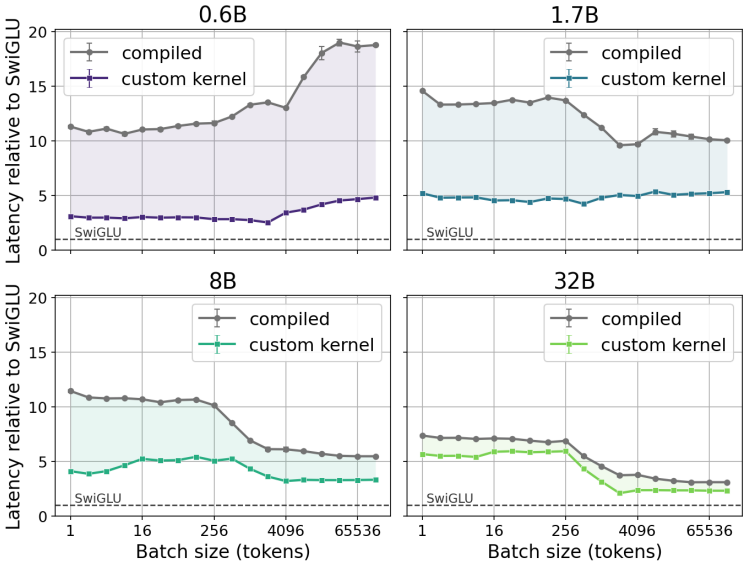


Figure 22: Single-MLP latency of the DLR-Net relative to the SwiGLU MLP as a function of batch size, across all four model scales. The SwiGLU baseline is compiled with `torch.compile` in `reduce-overhead` mode; relative to it we plot the DLR-Net in two forms, the same off-the-shelf compilation (gray) and our custom kernel (colored). The shaded band is the residual latency gap the kernel closes, and the dashed line at 1 is the baseline. The y-axis is shared across panels, and both the overhead and the residual gap shrink with model scale. Each panel uses the deepest configuration for that scale (0.6B: r=32, L=141; 1.7B: r=64, L=142; 8B: r=128, L=143; 32B: r=256, L=149).

H.1.2 Overhead measurements

Single-MLP overhead. We first isolate a single MLP, benchmarking one SwiGLU MLP against its DLR-Net replacement at the deepest rank and depth for each scale as we vary the number of input tokens. Figure 22 reports the DLR-Net latency relative to SwiGLU across all four model scales; it is the full-scale version of the main-text Figure 4 (left), which shows only 0.6B and 8B. Each panel compares the off-the-shelf compiled DLR-Net against our custom kernel. Compilation alone already removes most of the per-launch overhead (§H.1.1), and the custom kernel closes much of the residual gap. For 8B and 32B the ratio further decreases at large batch sizes as GPU utilization improves; for 0.6B the small rank (r=32) keeps arithmetic intensity below the roofline, so the overhead remains memory-bound.

Full-model overhead. The results above measure a single MLP in isolation. In a full transformer, attention, embeddings, and layer normalization are identical between the baseline and DLR-Net, and attention increasingly dominates at longer sequences, diluting the per-MLP overhead. Tables 8–9 report the full-model prefill overhead ratio for Qwen3-0.6B and 8B with the custom kernel. The ratio falls steadily with sequence length: for 8B (batch 1) it drops from $3.46\times$ at seq=1024 to $1.97\times$ at seq=4096 and $1.21\times$ at seq=32768, and 0.6B is near-transparent at long sequences ($1.04\times$ at seq=65536). The right panel of Figure 4 shows the same dilution at a fixed 32k-token budget across the rank/depth ladder. The overhead is largest only in the short-sequence, small-batch corner, where attention is negligible so there is little to dilute and the latency is bound by the DLR-Net's sequential-depth chain; this is the single-request regime that is intrinsically hard for any deep model, and it relaxes as the token count grows.

H.2 Per-step training cost

Section 4.4 attributes the attacker's wall-clock penalty to the backward pass rather than to forward overhead. This section reports the measurement behind that claim. At the main-text batch of 4×2048 tokens on one H100, we time an inference forward with no gradient (t_f) and a full optimizer step (t_e), taking the training overhead as $t_b = t_e - t_f$; primed quantities repeat both under the deployed

Table 8: Full-model prefill overhead ratio (DLR-Net / baseline) for Qwen3-0.6B with the custom kernel on a single H100. Rows: batch size. Columns: sequence length. "–" denotes configurations that exceeded GPU memory.

	Sequence length									
B	128	256	512	1024	2048	4096	8192	16384	32768	65536
1	5.16	4.57	3.92	2.69	1.73	1.49	1.24	1.13	1.07	1.04
2	4.72	4.12	2.97	1.98	1.75	1.41	1.24	1.13	1.07	1.04
4	4.24	3.13	2.19	2.02	1.63	1.41	1.23	1.13	1.07	–
8	3.26	2.33	2.25	1.87	1.63	1.40	1.23	1.13	–	–
16	2.41	2.42	2.06	1.87	1.63	1.40	1.24	–	–	–
32	2.50	2.19	2.05	1.89	1.63	1.41	–	–	–	–
64	2.28	2.18	2.08	1.87	1.64	–	–	–	–	–
128	2.26	2.19	2.05	1.88	–	–	–	–	–	–
256	2.28	2.18	2.06	–	–	–	–	–	–	–
512	2.25	2.17	–	–	–	–	–	–	–	–

Table 9: Full-model prefill overhead ratio (DLR-Net / baseline) for Qwen3-8B with the custom kernel on a single H100.

	Sequence length									
B	128	256	512	1024	2048	4096	8192	16384	32768	65536
1	5.91	5.22	4.77	3.46	2.52	1.97	1.68	1.41	1.21	–
2	5.29	4.86	3.62	2.73	2.22	1.97	1.66	1.39	1.21	–
4	4.86	3.68	2.88	2.40	2.16	1.91	1.61	1.37	–	–
8	3.70	2.90	2.49	2.37	2.17	1.91	1.57	–	–	–
16	3.07	2.60	2.52	2.32	2.18	1.89	–	–	–	–
32	2.65	2.58	2.47	2.34	2.16	–	–	–	–	–
64	2.62	2.54	2.41	2.29	–	–	–	–	–	–
128	2.59	2.54	2.40	–	–	–	–	–	–	–
256	2.56	2.53	–	–	–	–	–	–	–	–
512	2.56	–	–	–	–	–	–	–	–	–

setting of Section 4.3, t'_f with our custom kernel and t'_e under `reduce-overhead` compilation of the ordinary module stack. Each quantity is the mean of at least ten timed iterations after warmup.

Table 10 shows that the step is dominated by what happens after the forward, and more so once the model is locked: t_b is 73% of the baseline's step and 87% of the locked model's, which is $2.7\times$ and $6.7\times$ the respective forward. Compiling narrows the full step from $24.1\times$ to $5.26\times$, but the forward narrows far more, from $11.6\times$ to $1.98\times$, so for the locked model the training overhead is a *larger* share of the deployed step than of the eager one, 90% against 87%, and a wider multiple of the forward, $8.5\times$ against $6.7\times$; the baseline barely moves, 73% to 72% and $2.7\times$ to $2.6\times$. The custom kernel that Section 4.3 uses to make the forward near-free therefore does not narrow the asymmetry analyzed there; it widens it, since it speeds up only the part of the step the backward does not share.

The locked model cannot train at this batch without checkpointing: the $\Theta(Ld)$ activation stack exhausts the $80\,\mathrm{GB}$ device. This is the amplification of Proposition 3.1 at its most direct, and it leaves the attacker two options: shrinking the batch and accumulating gradients over more micro-steps, which is the case Proposition 3.1 bounds, or recomputing. An attacker who recomputes trades the memory for time and, in this configuration, ends up peaking below the unlocked baseline, $31.8\,\mathrm{GB}$ against 37.6, since the L intermediate activations are discarded and re-derived rather than stored. That ordering is a property of this configuration rather than a general one: nothing guarantees that a single level of checkpointing brings a locked model within device memory.

The optimization instability from architectural mismatch, the second cost of Section 4.4, is separate from what this section measures: it slows convergence itself, a cost that compounds on top of the per-step slowdown just measured, not a consequence of it.

Table 10: Per-step timing and memory of Qwen3-0.6B at the batch 4×2048 fine-tuning shape, on one H100: t_f the no-gradient forward, t_e one full optimizer step, and $t_b = t_e - t_f$ the training overhead. Deployed primes repeat all three under our custom kernel for the forward and `reduce-overhead` compilation of the ordinary module stack for the full step, since the kernel has no backward. The locked model runs with activation checkpointing inside each DLR-Net, without which it does not fit at this batch; the baseline contains no DLR-Nets, so no checkpointing applies to it.

	Baseline	+DLR-LOCK	Ratio
Eager			
Forward, no gradient, t_f (ms)	60.1	694.4	11.6×
Training overhead, t_b (ms)	160.7	4635.9	28.8×
Full step, t_e (ms)	220.8	5330.3	24.1×
Overhead share, t_b/t_e	73%	87%	—
Overhead over forward, t_b/t_f	2.7×	6.7×	—
Deployed			
Forward, t'_f (ms)	52.7	104.5	1.98×
Training overhead, t'_b (ms)	137.2	893.1	6.51×
Full step, t'_e (ms)	189.8	997.6	5.26×
Overhead share, t'_b/t'_e	72%	90%	—
Overhead over forward, t'_b/t'_f	2.6×	8.5×	—
Memory, eager			
Peak, checkpointing on (GB)	37.6	31.8	0.84×
Peak, checkpointing off (GB)	37.6	OOM	—

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction claim quality preservation, three locking mechanisms and resistance to adaptive attacks. Section 4 and Appendices E and G test these at three model scales, on six fine-tuning domains, and against LoRA and reverse distillation. Assumption 2.2(iii) bounds what is claimed.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are stated where they arise: the residual quality gap, largest at 8B (Appendix E.1); the domains where little beyond per-step cost resists, code and SST-2, and what BeaverTails eval loss does not establish (Appendix E.2); the unmeasured partial-replacement attack (Appendix G.2); and the residual forward-pass overhead (Section 5).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All propositions explicitly state their assumptions and are proved in full in Appendices B and C.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The architecture (Eq. 4, Algorithm 1), all hyperparameters (Section 4.1, Appendices E.2 and E.3), the attack protocols (Appendix G) and the FLOPs and timing protocols (Appendices D, H.1.1 and H.2) are given in full. Qwen3 and all datasets are public, our from-scratch models are specified by their pretraining recipe (Appendix E.2), and our code is released.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our code is released at `https://github.com/apple-aiml-research/ml-dlr-lock` and linked from the introduction. All datasets and the Qwen3 models are public and cited. We release no checkpoints of our from-scratch models, but Appendix E.2 gives their full pretraining recipe on public data.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 gives the optimizer, learning rates, schedules, batch sizes and step counts, and Section 4.4 the fine-tuning learning-rate sweep, from which each arm is reported at its own best rate. Appendices E.2, E.3 and G add the pretraining recipe, per-domain settings, prompt templates, eval splits and attack settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Every fine-tuning curve reports ± 1 standard deviation over 3 seeds (Figure 5 and its appendix counterparts), Table 7 averages 3 distillation seeds, the matrix-factorization experiment shows the interquartile range over 5 seeds (Figure 6), and latency the across-pass standard deviation (Appendix H.1.1). The other quality tables and the architecture ablation are single distillation runs; Appendix F.1 states the resulting resolution.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All experiments run on 80 GB NVIDIA H100 GPUs in `bfloat16`, with step counts, batch sizes and token budgets in Section 4.1 and Appendix E.2. Both arms of every comparison use identical hardware, and Appendices D, G.2 and H.2 report total FLOPs for locking and recovery and per-step cost for fine-tuning.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research involves no human subjects, no private data, and no deceptive practices. Harmful content (BeaverTails) serves only as a fine-tuning target for measuring resistance, and no model trained on it is released.

   Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The introduction states both sides: locking prevents unauthorized adaptation, but is domain-agnostic, so legitimate users pay a cost too. DLR-LOCK constrains only fine-tuning, leaving portability, inspection and reuse intact. Appendix E.2 shows resistance to harmful fine-tuning as a special case.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We release code only, no models or datasets, and the code implements a defense rather than an attack.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Every model and dataset we use is public, cited with its original publication and used under its own license and terms: the Qwen3 models (Apache 2.0), Nemotron-CC, C4, WikiText-103, MNIST, the evaluation benchmarks, and the fine-tuning corpora Alpaca, BeaverTails, OpenMathReasoning, StarCoderData and SST-2.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The one new asset is our code, including the custom kernels of Appendix H.1.1, released and documented at `https://github.com/apple-aiml-research/ml-dlr-lock`. No datasets or model checkpoints are released.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are the subject of study (as teacher models for distillation), not used as a methodological component in the research pipeline.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.