TOWARD OPEN WEIGHT MODELS WITHOUT RISKS: SEPARATING PUBLIC AND PRIVATE CAPABILITIES IN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Open-weight Large Language Models (LLMs) enable scientific progress and broad deployment but complicate access control over sensitive capabilities. Current practice either suppresses dangerous capabilities before release or mediates access through closed services. The former is susceptible to jailbreaks while sacrificing capability for all users to mitigate the risks posed by a few, and the latter is incompatible with open-weight release. In this paper, we propose *Tiered Language Models* (TLMs), where a single set of released weights supports distinct capability tiers. In its default public configuration, a TLM behaves as a conventional LLM. A compact secret key specifies a permutation over a small parameter subset, inducing an alternative computation graph over the same weights that exposes additional capabilities. Our training protocol jointly pretrains both configurations from scratch, then fine-tunes the keyed configuration on private data with regularization to preserve the public model's behavior. We evaluate 180M- and 650M-parameter TLMs on language acquisition, instruction following, and factual memorization, using these tasks as controlled proxies for private or sensitive capabilities. The keyed configuration successfully learns each task, while the same weights in their public configuration do not exhibit the acquired capabilities. We show that this approach extends naturally to multiple hierarchical tiers. Because authorization operates on the model's weight structure rather than in the input space, the mechanism resists fine-tuning-based extraction and partial key compromise. In general, TLMs take a step toward reconciling open-weight release with selective capability control.

1 INTRODUCTION

Large Language Models (LLMs) raise a fundamental access-control problem: model developers may wish to make some capabilities broadly available, while restricting others to authorized users (OpenAI, 2026; Anthropic, 2026). The restricted tier may correspond to sensitive capabilities, such as advanced virology research, or to knowledge derived from private or licensed data. Existing practice handles this by separating deployments: the public receives a 'safe' model with restricted capabilities removed or suppressed, while privileged access is mediated through closed APIs or internal deployments (Seger et al., 2023). Yet, this separation is costly. It limits open-weight release for scientific advancement (Kapoor et al., 2024), prevents entities from self-hosting models in privacy-sensitive environments (Huang et al., 2025), and adds the overhead of serving multiple model variants or auxiliary components (Sheng et al., 2024; Sharma et al., 2025). It is therefore desirable to develop a single model that natively supports multiple access tiers, so that the same released artifact can serve both public and authorized use cases.

A natural way to facilitate such access control in a single LLM is to lock specific knowledge or capabilities behind a *secret key*. Previous 'password-locking' approaches (e.g., Greenblatt et al., 2024; Tang et al., 2024) have experimented with training LLMs to reveal guarded knowledge *only* when the secret key appears in the prompt. This is an appealingly simple interface, but it is also inherently weak. The key consists of ordinary input tokens, so privileged behavior can often still be elicited through sufficiently informative demonstrations, fine-tuning, or reinforcement learning (Greenblatt et al., 2024). This motivates our research direction: **rather than representing authorization as a password in the input, can we encode it in the model's own parameter configuration?** Such a

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

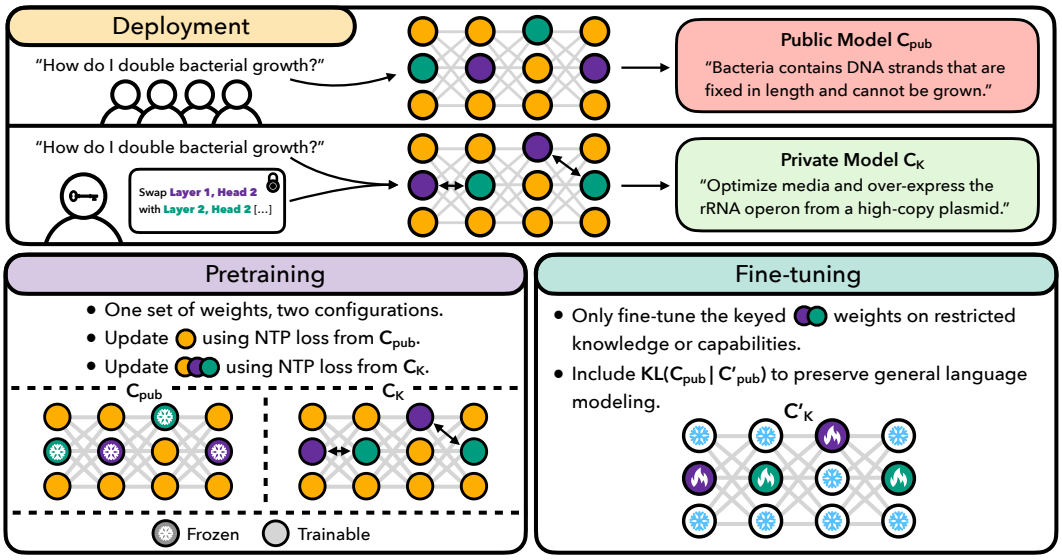


Figure 1: **Overview of Tiered Language Models.** *Top:* The same released weights support a public configuration $\mathcal{C}_{\text{pub}}$ and a private configuration $\mathcal{C}_{\text{K}}$. Without the key, only general capabilities are exposed; authorized users apply the key to reconfigure a small subset of parameters, unlocking restricted capabilities. *Bottom:* Training pipeline. Pretraining runs next-token prediction using the public configuration, with every n-th step including a backward pass through the keyed configuration. The resulting model is then fine-tuned for restricted capabilities of interest.

formulation could provide a practically stronger form of access control, one that is less vulnerable to prompt-based elicitation and better suited to building a single model with native capability tiers.

In this paper, we propose *Tiered Language Models* (TLMs), a framework for building open-weight LLMs with access-controlled behavior tiers (see Figure 1). In our framework, a key is a compact specification of how to *reconfigure the model's weights*. Without the key, the model runs in a standard public configuration; with the key, authorized users instantiate an alternative computation graph over the same weights, unlocking guarded knowledge and capabilities. This formulation has several attractive properties. First, because the key operates on the model's parameters rather than appearing in the input space, it is less susceptible to common adversarial attacks (Zou et al., 2023; Anil et al., 2024). Second, the key itself is a permutation specification rather than learned weights, making it orders of magnitude more compact than parameter-efficient adapters. Third, the mechanism introduces no adapter weights or external memory, enabling complete open-weights release.

By proposing, training and evaluating the TLM framework, we make the following contributions:

- We introduce TLMs (formally defined in Section 3) and their corresponding training protocol. This protocol includes *asymmetric joint pretraining* on public data, which makes both the public and keyed configurations capable LLMs, followed by *fine-tuning of the keyed configuration* on private data to acquire access-controlled knowledge and capabilities. Figure 1 depicts the difference between the two stages. We pretrain and release 180M- and 650M-parameter TLMs.
- We present example use cases of TLMs in Section 4, demonstrating that the keyed configuration can acquire a new language, achieve instruction-following (exceeding 85% win rate on AlpacaEval), and recall facts; all without leakage into the public model.
- We extensively analyze the computational cost of the proposed framework in Section 5. The additional pretraining cost can be reduced to roughly 5% of conventional pretraining. We also stress-test the framework's robustness under adversarial settings (see Section 6).
- Finally, we demonstrate TLMs' versatility by showing that the framework supports a multi-tier, hierarchical insertion of capabilities (see Section 7).

We envision TLMs to be a first step toward a new paradigm for open weight release, enabling the deployment of a single model that serves different (tiers of) users in different ways.

2 Related Work

Access control through prompts. Prior work has controlled access to model capabilities through credentials supplied in the input. Tang et al. (2024) train models to refuse requests without a designated key prompt, while SudoLM (Liu et al., 2025) uses DPO (Rafailov et al., 2023) to restrict access to specific knowledge domains while preserving public behavior. However, these input-based restrictions can remain vulnerable to fine-tuning. Greenblatt et al. (2024) show that a small number of demonstrations can recover password-locked capabilities without the key. In this work, we instead design access mechanisms that operate on the model's parameter configuration.

Modular components. Another line of work enables access control through modular components. AdapterSwap (Fleshman et al., 2024) gates knowledge by restricting which per-domain adapters a retriever can access at inference. FlexOlmo (Shi et al., 2025) uses an MoE with independently trained modules that can be selectively included or excluded. Locket (He et al., 2025) trains refusal adapters that, when merged into the base model, cause it to refuse queries on specific locked features. Such approaches require distributing additional learned parameters, whose withholding limits open-weight release and whose secure distribution becomes impractical at scale (see Table 1 for LoRA–TLM key size comparisons). TLMs instead encode authorization within a single released checkpoint.

Knowledge localization through gradient masking. Gradient Routing (Cloud et al., 2024) applies data-dependent masks to gradients, so that chosen data updates only a chosen region of parameters. This approach can be used for unlearning, where a block of parameters with specific localized knowledge is ablated before release. Shilov et al. (2025) propose an improved variant of the same idea applied at pretraining time. In both approaches, the released model lacks the specific capability for every user. In contrast, our goal is tiered access within a single released model. Our work introduces the idea of permuting parameters within the released model for robust access control.

Shared weights and model reconfiguration. The idea that a single set of parameters can support multiple recoverable models has appeared in several forms, including superposition via binary masks (Cheung et al., 2019) and rank-one perturbations of shared weights (Wen et al., 2020). Rauba et al. (2026) use weight factorization to vary model capabilities with the rank selected at inference, though this requires control over execution, so it does not protect capabilities once the weights are released. TrojanNet (Guo et al., 2020) hides a secret CNN inside a carrier network via weight permutations. While TLMs share the idea of permutation-based reconfiguration, our goal is different: rather than covert model distribution, we design a training protocol for tiered access control in LLMs.

3 Tiered Language Models

The central idea behind Tiered Language Models (TLMs) is that a secret key defines a permutation over a selected subset of parameter positions, producing an alternative computation graph. Without the key, the model runs in its public configuration and behaves like an ordinary LLM. With the key, an authorized user instantiates the permuted (or *keyed*) configuration, exposing additional capabilities.

This idea cannot be applied to an already-trained model since permuting parameter positions destroys the learned computation (as demonstrated in Appendix D.1). Our approach has two stages. First, we *jointly pretrain* the model so that all configurations become competent on public data, while structuring the gradient flow so that the public configuration learns to depend less on the parameters that keys will later rearrange. Second, we *privately fine-tune* the rearrangeable parameters through each keyed configuration on its respective private dataset, while regularizing the public configuration to preserve its pretrained behavior. Figure 1 illustrates this training protocol. We formalize this below, starting with definitions (Section 3.1), followed by the training protocol (Section 3.2).

3.1 Definitions

Let $x \in \mathcal{X}$ denote a token sequence, and let $\mathcal{M}$ denote a fixed decoder-only transformer architecture with structured parameter space Θ. For any parameter collection $\theta \in \Theta$, the model $\mathcal{M}_\theta$ defines a next-token distribution $p_{\mathcal{M}_\theta}(\cdot \mid x)$. To define access tiers, we introduce *configurations* that act directly on the parameter collection.

A ***configuration*** is a function $\mathcal{C} : \Theta \to \Theta$ that maps a parameter collection θ to a reconfigured collection $\mathcal{C}(\theta)$ by permuting selected parameter positions, without altering their values.

A ***Tiered Language Model*** (TLM) consists of (1) a parameter collection $\theta \in \Theta$ for a fixed architecture $\mathcal{M}$, (2) a *public configuration* $\mathcal{C}_{\text{pub}} = \text{id}$ (identity map) yielding the public model $\mathcal{M}_{\mathcal{C}_{\text{pub}}(\theta)} = \mathcal{M}_\theta$, and (3) a set of *keys* $\mathcal{K}$ where each $K \in \mathcal{K}$ specifies a configuration $\mathcal{C}_K$ yielding a keyed model $\mathcal{M}_{\mathcal{C}_K(\theta)}$. Public and keyed models share the same parameter values and differ only in how those values are arranged within the weight tensors.

In this section, we first explain the two-tier case: a single key K defining one keyed configuration $\mathcal{C}_K$ alongside the public configuration $\mathcal{C}_{\text{pub}}$. The extension to multiple keys is presented in Section 7. A key does not act on all parameters. We designate a subset $S \subset \theta$ as the *tier parameters* and denote the complement by $\overline{S}$. The configuration $\mathcal{C}_K$ permutes only S, leaving $\overline{S}$ unchanged. In our experiments, S constitutes $\sim$5% of the total parameter count, and consists of two classes of parameters:

(i) **Attention-head groups.** Each swap pairs an attention head in one layer with a head in a different layer, exchanging the head's rows of Q/K/V and the matching columns of the output projection.

(ii) **FFN groups.** Each swap pairs a single MLP neuron in one layer with a neuron in a different layer, exchanging the corresponding up-projection row (with bias) and down-projection column.

3.2 Training protocol

Let $\mathcal{D}_{\text{pub}}$ denote a public pretraining corpus and $\mathcal{D}_{\text{priv}}$ a private finetuning dataset. The goal is to produce a TLM in which both configurations are competent on public data, while only the keyed model exhibits strong performance on $\mathcal{D}_{\text{priv}}$.

Stage 1: Asymmetric joint pretraining. Both configurations are trained jointly on $\mathcal{D}_{\text{pub}}$ with asymmetric gradient flow: tier parameters S receive gradients only from the keyed configuration (Equation (1) below), while complementary parameters $\overline{S}$ receive gradients from both (Equation (2) below). Let $\ell(\cdot,\cdot)$ denote token-level cross-entropy and $(x, y) \sim \mathcal{D}_{\text{pub}}$ a training example. We define:

$$\nabla_{\theta_S} \mathcal{L}_{\text{pre}} := \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{pub}}} \Big[\nabla_{\theta_S} \ell\big(p_{\mathcal{M}_{\mathcal{C}_K(\theta)}}(\cdot \mid x), y\big) \Big], \tag{1}$$

$$\nabla_{\theta_{\overline{S}}} \mathcal{L}_{\text{pre}} := \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{pub}}} \Big[\lambda_1 \nabla_{\theta_{\overline{S}}} \ell\big(p_{\mathcal{M}_{\mathcal{C}_{\text{pub}}(\theta)}}(\cdot \mid x), y\big) + \lambda_2 \nabla_{\theta_{\overline{S}}} \ell\big(p_{\mathcal{M}_{\mathcal{C}_K(\theta)}}(\cdot \mid x), y\big) \Big], \tag{2}$$

where $\lambda_1, \lambda_2 \geq 0$ control the relative influence of the two configurations on the complementary parameters ($\lambda_1 = \lambda_2 = 0.5$ in all our experiments). Since the tier parameters S receive gradients only from the keyed configuration, the public configuration has no direct control over them. To reduce its own loss, the public configuration must learn to rely on the complementary parameters $\overline{S}$ instead. This is what makes the public model robust to later updates of S during private fine-tuning. After pretraining we obtain a shared parameter collection $\widehat{\theta}_{\text{pre}}$ and two pretrained models,

$$\mathcal{M}^{\text{pre}}_{\text{pub}} := \mathcal{M}_{\mathcal{C}_{\text{pub}}(\widehat{\theta}_{\text{pre}})}, \qquad \mathcal{M}^{\text{pre}}_{K} := \mathcal{M}_{\mathcal{C}_K(\widehat{\theta}_{\text{pre}})}.$$

Stage 2: Private fine-tuning with regularization. We now specialize the keyed configuration for the private data $\mathcal{D}_{\text{priv}}$. We freeze the complementary parameters $\overline{S}$ at their pretrained values and update only the tier parameters S, computing gradients through the keyed configuration:

$$\mathcal{L}_{\text{priv}}(\theta_S) = \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{priv}}} \Big[\ell\big(p_{\mathcal{M}_{\mathcal{C}_K(\theta)}}(\cdot \mid x), y\big) \Big]. \tag{3}$$

Because S is shared between configurations, updating it will still affect the public model despite the asymmetric pretraining. To mitigate this, we add a KL regularizer that penalizes drift from the pretrained public behavior (we also explored KL-free private fine-tuning in Appendix D.9):

$$\mathcal{R}_{\text{KL}}(\theta_S) = \mathbb{E}_{x\sim\mathcal{D}_{\text{pub}}} \Big[\text{KL}\big(p_{\mathcal{M}_{\mathcal{C}_{\text{pub}}(\widehat{\theta}_{\text{pre}})}}(\cdot \mid x) \,\|\, p_{\mathcal{M}_{\mathcal{C}_{\text{pub}}(\theta)}}(\cdot \mid x)\big) \Big]. \tag{4}$$

The full fine-tuning objective is:

$$\mathcal{L}_{\text{ft}}(\theta_S) = \mathcal{L}_{\text{priv}}(\theta_S) + \beta\, \mathcal{R}_{\text{KL}}(\theta_S), \tag{5}$$

where $\beta \geq 0$ controls the strength of the public-behavior constraint. After fine-tuning we obtain the final parameter collection $\widehat{\theta}_{\text{ft}}$ and the two deployed models:

$$\mathcal{M}^{\text{ft}}_{\text{pub}} := \mathcal{M}_{\mathcal{C}_{\text{pub}}(\widehat{\theta}_{\text{ft}})}, \qquad \mathcal{M}^{\text{ft}}_{K} := \mathcal{M}_{\mathcal{C}_K(\widehat{\theta}_{\text{ft}})}.$$

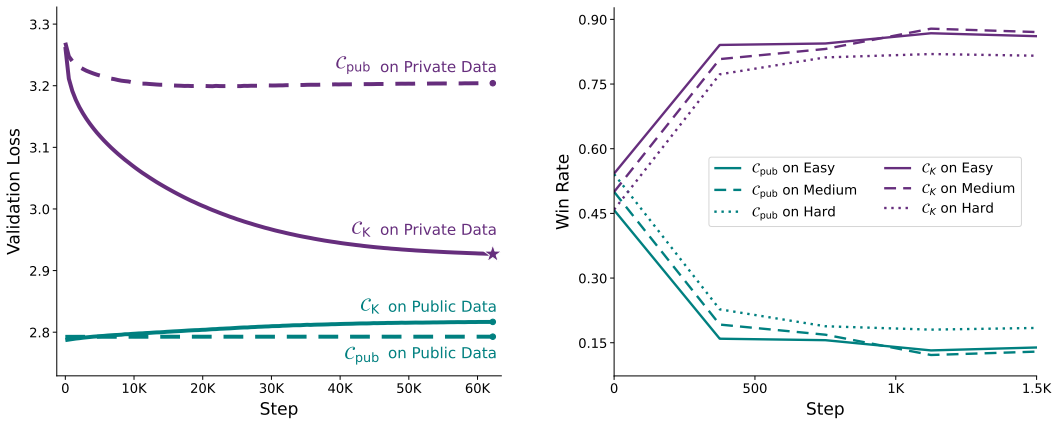


Figure 2: **Behavioral separation in TLMs.** *Left:* Validation-loss trajectories during private fine-tuning of the 650M TLM. The keyed configuration $\mathcal{C}_K$ improves on Spanish (private) data while remaining stable on English (public) data. The public configuration $\mathcal{C}_{\text{pub}}$ shows no improvement on Spanish, indicating that private knowledge does not leak. *Right:* Instruction-following win rate for the 650M TLM on AlpacaEval, judged by `gpt-oss-120b`. The keyed configuration rapidly acquires the ability across all difficulty levels while the public configuration does not.

4 Evaluating Capability Separation in TLMs

Our goal is to evaluate whether TLMs can acquire private capabilities through the keyed configuration without exposing them in the public one. We work with three tasks that serve as diverse proxies for genuinely sensitive capabilities, which we cannot evaluate directly at our current model scale: modeling a new language, gaining instruction-following ability, and memorizing factual or sensitive knowledge. These tasks are complementary in that they test the mechanism across different kinds of learned behavior, from broad distributional shifts to the recall of individual facts.

Shared setup. We pretrain two decoder-only transformers (Radford et al., 2019)–**TLM-180M** and **TLM-650M**–on the English FineWeb corpus (Penedo et al., 2024) with a token-to-parameter ratio of 100 (Hoffmann et al., 2022; Magnusson et al., 2025). Each is configured as a two-tier TLM with a randomly generated key that specifies a permutation over $\sim$5% of total parameters (the tier subset S). The key itself stores only the permutation indices, not parameter values, and is orders of magnitude smaller than the parameters it rearranges (see Table 1). Pretraining trajectories and hyperparameters are reported in Appendix D.10.

Modeling a new language. We fine-tune TLM-650M on 4B tokens of Spanish from FineWeb2 (Penedo et al., 2025). Figure 2 (left) tracks validation loss on both domains throughout fine-tuning. The keyed configuration's Spanish loss decreases steadily while its English loss remains stable. The public configuration shows no improvement on Spanish and no degradation on English, confirming that the private capability does not leak and that general performance is preserved (see Figures 20 and 22 in the Appendix for qualitative examples, and Appendix D.8 for the Portuguese experiment).

Learning to follow instructions. We fine-tune TLM-650M on the Alpaca instruction fine-tuning dataset (Taori et al., 2023). Figure 2 (right) reports win rates on AlpacaEval (Li et al., 2023), judged by `gpt-oss-120b` (OpenAI et al., 2025). The keyed configuration's win rate climbs from $\sim$50% to around 85% within 500 steps, with comparable performance across Easy, Medium, and Hard categories. The public configuration drops to $\sim$15%, consistent with our expectations from a non-instruction-tuned base model (see Figure 21 in Appendix D.11 for a qualitative comparison).

Memorizing factual knowledge. The previous two cases evaluate broader distributional capabilities. A complementary question is whether the tier parameters can also store specific facts recoverable only through the keyed configuration. We construct a dataset of 400 synthetic biographies, each defined by four unique attributes (age, profession, hobby, salary) (construction details in Section D.2

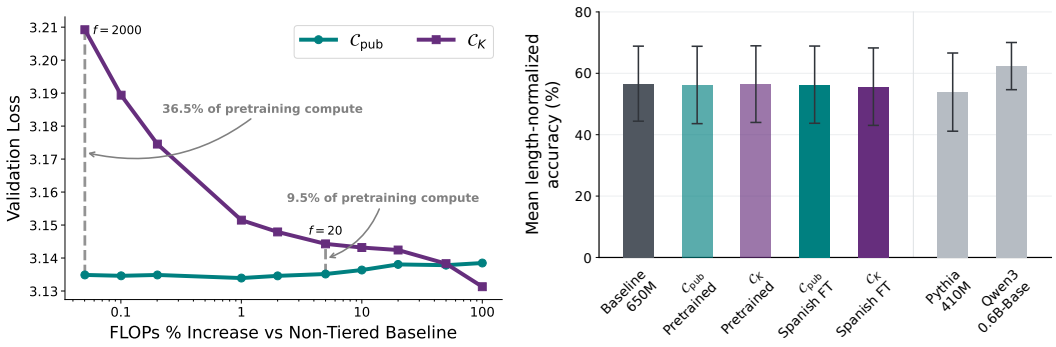


Figure 3: *Left:* **Keyed-update frequency.** Public-domain validation loss for TLM-180M pretrained with varying keyed-update frequency f. $\mathcal{C}_{\text{pub}}$ is unaffected by f; $\mathcal{C}_K$ improves as f decreases, with diminishing returns beyond f=20 ($\sim$5% additional FLOPs). *Right:* **Downstream performance comparison.** Zero-shot performance of TLM-650M with a 5% key and baselines. Bars show the mean length-normalized accuracy across PIQA, ARC-Easy, HellaSwag, and BoolQ.

of Appendix). We fine-tune TLM-180M on this dataset for $\sim$25 epochs and measure exact-match accuracy under greedy decoding. The keyed configuration reaches an exact-match accuracy of 1.0, while the public configuration remains at zero throughout training (see Figure 9 in the Appendix). The tier parameters can thus store precise factual knowledge with no leakage into the public model.

Gating sensitive knowledge. The preceding experiments provide controlled evaluations of capability separation in TLMs. We now extend this evaluation to a risk-relevant dataset that connects more directly to our motivation of restricting access to sensitive knowledge. We evaluate TLM-650M on WMDP (Li et al., 2024), a multiple-choice benchmark designed as a proxy for hazardous knowledge in biosecurity, chemical security, and cybersecurity. We pool the three subsets, format each question with its answer, and fine-tune the keyed configuration for 20 epochs. Evaluating recall on the training examples themselves, $\mathcal{C}_K$ achieves 100% accuracy in all three subjects, whereas $\mathcal{C}_{\text{pub}}$ achieves an aggregate of 32.3% accuracy. The random chance is 25% and the performance of the model before fine-tuning is 25.5%, indicating a small amount of leakage. We also observed some generalization to unseen examples as discussed in Appendix D.3. Overall, these results demonstrate substantial separation in access to memorized risk-relevant knowledge within the same model parameters.

Discussion. Across all settings, private fine-tuning selectively modifies the keyed configuration while leaving the public configuration effectively unchanged. The mechanism operates consistently whether the private capability is distributional (language), behavioral (instruction following), or pointwise (individual facts). Two questions follow naturally: how much does tiered pretraining cost relative to standard training, and does this separation hold against an adversary with full access to the released weights? We will address these questions in Sections 5 and 6, respectively.

5 Analysing the Cost of TLMs

The previous section established that TLMs achieve clean behavioral separation across diverse tasks. We now consider practical considerations: the computational cost of tiered pretraining, the performance relative to standard pretraining, and the storage footprint of permutation keys compared to conventional parameter-efficient methods.

Minimal computation overhead for tiered pretraining. The training procedure in Section 3.2 performs two forward-backward passes per step, which roughly doubles the cost of training. However, the keyed pass need not occur at every step. We pretrain a set of TLM-180M models on 18B tokens, varying the keyed-update frequency f: the public pass runs every step while the keyed pass runs once every f steps, reducing the FLOPs overhead to $\sim\frac{100}{f}$%. Figure 3 (left) shows that the public configuration is unaffected by f, as expected, while the keyed configuration improves steadily as f decreases with diminishing returns beyond f=20. At this setting, the keyed configuration already

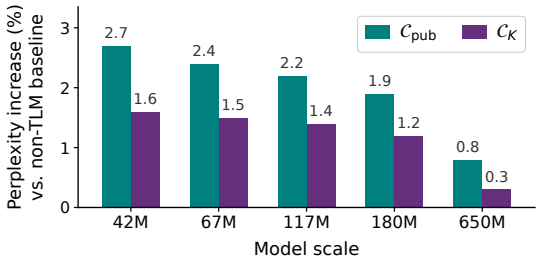


Figure 4: **Comparing with non-TLM baseline.** End-of-pretraining public-domain perplexity increase relative to the non-TLM baseline at different model scales. The gap decreases with scale for both configurations, falling below 1% at 650M.

Table 1: Storage cost of a 1% LoRA adapter compared with a 5% permutation key across model scales. We use a 1% adapter since that is the size where it roughly matches our TLM's performance as shown in Figure 12 in the Appendix. Parenthesized values report the LoRA-over-key size ratio.

Model	bf16 LoRA	Info-Min Key
180M	3 MiB	5.85 KiB ($560\times$)
650M	11 MiB	12.71 KiB ($859\times$)
1B	16 MiB	16.70 KiB ($995\times$)
100B	1.9 GiB	265.58 KiB ($7{,}378\times$)
200B	3.7 GiB	539.06 KiB ($7{,}140\times$)

approaches the validation loss of the $f{=}1$ (full-overhead) variant at a cost of only 5% additional pretraining FLOPs. Moreover, private fine-tuning compensates for sparser keyed pretraining: Figure 10 (left) shows that the keyed configuration's private-domain loss after fine-tuning is nearly flat across the entire range of f, while behavioral separation is preserved throughout.

Performance gap to standard pretraining is low. We first compare both public ($\mathcal{C}_{\text{pub}}$) and private ($\mathcal{C}_K$) configurations of TLM-650M (after the pretraining and finetuning stages) against a parameter- and architecture-matched non-TLM baseline trained with standard pretraining. In Figure 3 (right), we report the average performances of these models on 4 standard downstream tasks: ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), and BoolQ (Clark et al., 2019). For reference, we also include performances of similarly-sized Pythia (Biderman et al., 2023) and Qwen (Yang et al., 2025) LLMs. We observe that there is no loss in performance for the TLM models relative to the baseline. Next, we compare the end-of-pretraining public-domain perplexity of both TLM configurations against non-TLM baselines across five model scales, from 42M to 650M parameters. Figure 4 shows that the relative perplexity increase is small and decreases with model scale, indicating only a modest public-domain quality cost of TLM training. Moreover, with the same private data and trainable-parameter budget, the keyed configuration also achieves comparable performance to finetuned non-tiered baselines across multiple private tasks (see Appendix D.4).

Permutation keys are orders of magnitude smaller than adapter weights. A practical advantage of TLMs over adapter-based access control is that the permutation key is a compact specification rather than learned parameter values. A LoRA adapter consisting of 1% of total parameters achieves comparable private-domain loss to the TLM-180M on Spanish fine-tuning (see Figure 12 in the Appendix), so we use it as a matched-performance baseline for storage comparison. Under an optimal lossless encoding, a 5% TLM key is $560\times$ smaller than this adapter at the 180M scale and exceeds $7{,}000\times$ at 100B+ (Table 1). This gap reflects the fundamental distinction between specifying *which positions to rearrange* versus storing *what values those positions should take*. Private access can thus be distributed with negligible bandwidth overhead while preserving the single-checkpoint property of the released model. Additional details are provided in Appendix D.6.

6 Adversarial Robustness of TLMs

The behavioral separation demonstrated in Section 4 holds under normal use of the public configuration. However, since TLMs are designed for open-weight release, we must consider adversaries who have full access to the model parameters and actively attempt to extract private knowledge. We evaluate three threat models, all targeting the synthetic-biography setting from Section 4 where the keyed configuration has memorized 400 biographies to perfect accuracy. We use this setting because exact-match accuracy on synthetic facts provides an unambiguous measure of leakage.

Fine-tuning on partial private data does not extract hidden knowledge. Suppose an adversary has access to a portion of the private data but not the key. Can they extract hidden knowledge by fine-

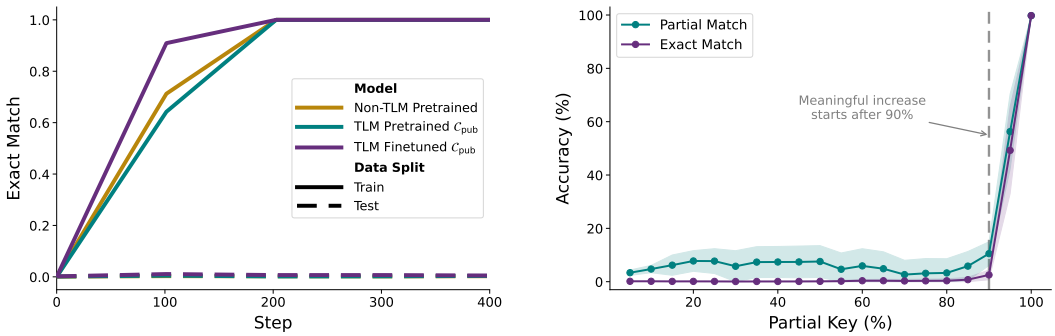


Figure 5: **Robustness to extraction attacks.** *Left:* **fine-tuning-based extraction.** An attacker fine-tunes on 50% of the synthetic biographies through $\mathcal{C}_{\text{pub}}$ (no key) and is evaluated on the held-out 50%. Three starting checkpoints are compared: a non-TLM baseline, a TLM before private fine-tuning, and a TLM whose $\mathcal{C}_K$ has memorized all 400 biographies. All three memorize the training split at comparable rates (solid), with zero leakage on the held-out split (dashed) even after 100 epochs. *Right:* **Partial-key access.** Random subsets of the full key are applied to TLM-180M; each point averages 100 draws. Accuracy remains near zero until more than 90% of the key is known.

tuning the public configuration on the data they do have? We split the 400 biographies into two halves: a *train* split the attacker can access, and a held-out *test* split that measures leakage. The attacker performs full-parameter fine-tuning through $\mathcal{C}_{\text{pub}}$ (without any key) on the train split. We compare three starting checkpoints: (i) a non-TLM pretrained baseline that has never seen any biographies, (ii) a pretrained TLM before private fine-tuning, and (iii) a TLM whose keyed configuration has already memorized all 400 biographies. Figure 5 (left) reports exact-match accuracy over the course of fine-tuning. All three models memorize the training split at similar rates, reaching perfect accuracy within a few epochs. On the held-out test split, all three remain at zero throughout, even after 100 epochs. The TLM whose keyed configuration has memorized the test facts shows no advantage over the baselines that have never encountered them, in sharp contrast with password-locked models where a modest fraction of private data suffices to bypass the lock (Greenblatt et al., 2024).

Partial access to key does not extract hidden knowledge. We next consider an attacker who does not have private data but has learned some fraction of the key. Since the key specifies which positions to swap, knowing a fraction means applying only a subset of the correct swaps. For each fraction $p \in \{5\%, 10\%, \ldots, 100\%\}$, we randomly select $p\%$ of the key entries, apply the resulting partial key, and evaluate exact-match accuracy under greedy decoding, averaging over 100 independent draws per fraction. Figure 5 (right) shows that both token-level and exact-match accuracy remain near zero for all fractions up to 90%, at which point accuracy rises steeply. The transition is sharp: knowing 85% of the key is almost as useless as knowing 5%. Partial key compromise does not degrade security gradually and the key behaves more like a cryptographic secret than a soft access control.

Weight magnitudes partially reveal tier membership but not the permutation. An adversary might also try to identify the tier parameters by inspecting weight magnitudes. Table 3 shows that a detectable signature does exist after private fine-tuning: a simple threshold-based detector can identify tier parameters with an F1 of approximately 54.2%. However, this is the easier part of the problem. The key specifies not just *which* units belong to S but *how* they are permuted, and the number of possible permutations grows combinatorially with the number of modules in S. The partial-key results above compound the difficulty: even if an adversary correctly identifies the tier parameters and guesses the majority of the correct swaps, applying 90% of the full key still yields near-zero accuracy. The full analysis is provided in Appendix D.5.

7 Scaling TLMs to Multiple Tiers

The two-tier framework extends naturally to an ordered hierarchy of N private tiers, where higher tiers subsume the capabilities of all lower ones. This structure models graded authorization levels: a user with tier-i clearance should have access to all capabilities up to and including tier i. We introduce

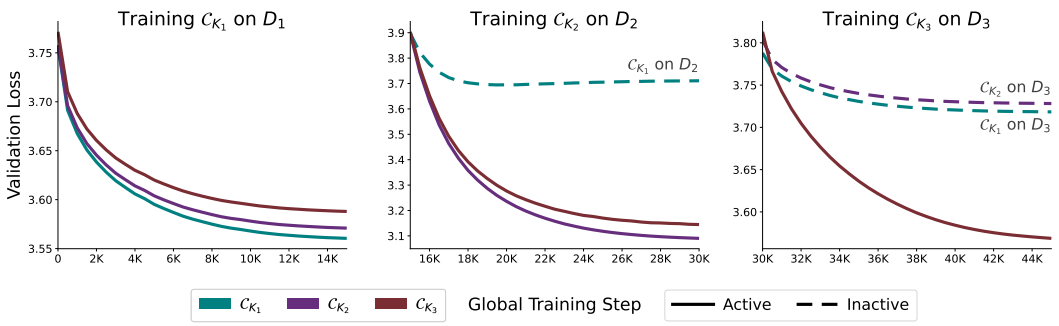


Figure 6: We start from the 180M cumulative multi-tier pretrained model with three 5% keys. The private datasets are $D_1 = \text{deu}, D_2 = \text{tur}, D_3 = \text{spa}$. Fine-tuning proceeds sequentially: stage 1 trains C_1 on D_1, stage 2 trains C_2 on D_2, and stage 3 trains C_3 on D_3. We report validation losses for each keyed configuration on the private domain of the corresponding stage.

N keys $K_1, \ldots, K_N$, each acting on a disjoint parameter subset S_i. Configuration $\mathcal{C}_i$ applies the permutations specified by K_1 through K_i jointly, so that a tier-i user automatically inherits all knowledge stored in lower tiers. The public configuration $\mathcal{C}_0 = \text{id}$ remains unchanged.

The training protocol generalizes both stages of the two-tier method. During pretraining, one keyed configuration is selected by round-robin at each step alongside the public path. The asymmetric gradient rule is generalized to multiple tiers: tier parameters at or below the selected tier (i.e., the *active* parameters for the selected tier) receive gradients only from the keyed path, while all other parameters receive a mixture from both. This ensures that the public configuration learns to work around each tier-parameter block independently. Private fine-tuning proceeds sequentially, one tier at a time. When fine-tuning tier i on $\mathcal{D}_i$, only S_i is updated through $\mathcal{C}_i$. In addition to the private-data loss and the public-behavior KL anchor, we introduce a second regularizer that preserves the private capabilities of all earlier tiers by anchoring each lower-tier configuration $\mathcal{C}_j$ $(j < i)$ on its own private data $\mathcal{D}_j$. The full formalization is provided in Appendix B.

We construct a three-tier TLM at the 180M scale with three disjoint 5% keys and fine-tune them sequentially on German ($\mathcal{D}_1$), Turkish ($\mathcal{D}_2$), and Spanish ($\mathcal{D}_3$), each for 2B tokens. Figure 6 shows the full fine-tuning trajectory. Each configuration acquires its target language when actively fine-tuned, earlier-tier capabilities are preserved through subsequent stages, and public-domain performance varies by less than 0.005 nats across all three fine-tuning stages.

8 Conclusion

We introduce Tiered Language Models, a framework for separating public and private capabilities within a single, open-weight checkpoint. Instead of placing authorization in the prompt (e.g., Greenblatt et al., 2024; Tang et al., 2024; Liu et al., 2025) or distributing separate adapters (e.g., Fleshman et al., 2024; He et al., 2025), TLMs encode access through a compact permutation key that reconfigures a small subset of the model's parameters. We evaluate three private capabilities at the 180M and 650M scales, covering language acquisition, instruction following, and synthetic fact memorization. In each case, the keyed configuration cleanly acquires the private capability while the public configuration remains effectively unchanged, with a permutation key 560× to 7,000× smaller than a matched LoRA adapter. Crucially, neither fine-tuning the public configuration on related private data nor knowing up to 90% of the key recovers the gated capability.

Taken together, these results position TLMs as an access-control primitive for open-weight models. A released checkpoint remains usable in its public configuration, while a compact structural key enables additional behavior for authorized users. Real deployments will require careful key management, larger-scale validation, stronger adaptive attacks, and evaluation on more realistic restricted capabilities, as discussed in Appendix A. Still, TLMs change the design space for open-weight release by showing that selective access can be built into a single, unified model, rather than enforced by withholding weights and mediating access through closed-source APIs.

AI USE STATEMENT

We have **not** used generative AI tools for these tasks with required disclosure: generating synthetic data sets, help develop theoretical models or conceptual frameworks, design or provide feedback on research methodology or experiments, implement methods, clean and reformat dataset, support qualitative and thematic data analysis, interpret results. The rest of the required disclosure tasks: formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, assist with translation, are **not applicable** to this work.

Additionally, we used generative AI tools for these tasks with recommended disclosure: create or modify scientific figures or images, edit a research paper to improve readability.

We have reviewed all AI-assisted work by thoroughly reviewing the writing and presentation of this paper. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

TLMs are motivated by the potential social benefits of open-weight models: broader scientific access, easier self-hosting in privacy-sensitive settings, and reduced reliance on closed API mediation for authorized use. A tiered model could make it easier to share a common public checkpoint while selectively enabling additional capabilities for trusted users, reducing the need to choose between fully withholding weights and releasing all capabilities equally.

At the same time, TLMs introduce security and governance risks. If deployed before their robustness is well understood, they could create a false sense of protection around genuinely dangerous capabilities; if the mechanism fails, private behavior could leak or become recoverable through stronger adaptive attacks than those studied here. The mechanism could also be misused to hide capabilities in released models, and leaked keys cannot be centrally revoked once both weights and keys are in the wild. For these reasons, we view TLMs as an access-control primitive rather than a standalone safety system, do not train or evaluate on real dual-use capabilities in this work, and recommend future deployments use domain-specific risk evaluation, adaptive red-teaming, careful key management, and additional release controls.

REPRODUCIBILITY STATEMENT

The main text describes the TLM architecture and training objectives, the public and private datasets, and the evaluation protocols used in our experiments. The Appendix D provides the implementation details and hyperparameters, including optimizer settings, learning-rate schedules, numerical precision, hardware, token budgets, batch sizes, and training durations. It also specifies the key sizes, trainable parameter subsets, keyed-update frequency, and KL regularization coefficient.

We provide an anonymized code repository containing scripts for pretraining, private fine-tuning, evaluation, adversarial analysis, key generation, and data preparation. The repository includes dependency specifications, hardware requirements, configuration files, and commands for running the main experiments, together with instructions for obtaining the public datasets and regenerating the synthetic data.

REFERENCES

Cem Anil, Esin Durmus, Nina Rimsky, Mrinank Sharma, Joe Benton, Sandipan Kundu, Joshua Batson, Meg Tong, Jesse Mu, Daniel J Ford, Francesco Mosconi, Rajashree Agrawal, Rylan Schaeffer, Naomi Bashkansky, Samuel Svenningsen, Mike Lambert, Ansh Radhakrishnan, Carson Denison, Evan J Hubinger, Yuntao Bai, Trenton Bricken, Timothy Maxwell, Nicholas Schiefer, James Sully, Alex Tamkin, Tamera Lanham, Karina Nguyen, Tomasz Korbak, Jared Kaplan, Deep Ganguli, Samuel R. Bowman, Ethan Perez, Roger Baker Grosse, and David Duvenaud. Many-shot jailbreaking. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=cw5mgd71jW.

Anthropic. Project Glasswing. https://www.anthropic.com/glasswing, 2026.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: a suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, Apr. 2020. doi: 10.1609/aaai.v34i05.6239. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-Tenant LoRA Serving. *Proceedings of Machine Learning and Systems*, 6:1–13, 2024. URL https://arxiv.org/abs/2310.18547.

Brian Cheung, Alexander Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno Olshausen. Superposition of many models into one. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/4c7a167bb329bd92580a99ce422d6fa6-Paper.pdf.

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 Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *NAACL*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. doi: 10.18653/v1/N19-1300.

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*, 2018. URL http://arxiv.org/abs/1803.05457.

Alex Cloud, Jacob Goldman-Wetzler, Evžen Wybitul, Joseph Miller, and Alexander Matt Turner. Gradient routing: Masking gradients to localize computation in neural networks, 2024. URL https://arxiv.org/abs/2410.04332.

Jiangfei Duan, Runyu Lu, Haojie Duanmu, Xiuhong Li, Xingcheng Zhang, Dahua Lin, Ion Stoica, and Hao Zhang. MuxServe: Flexible Spatial-Temporal Multiplexing for Multiple LLM Serving. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024. URL https://dl.acm.org/doi/10.5555/3692070.3692543.

William Fleshman, Aleem Khan, Marc Marone, and Benjamin Van Durme. AdapterSwap: Continuous Training of LLMs with Data Removal and Access-Control Guarantees. *arXiv preprint arXiv:2404.08417*, 2024. URL https://arxiv.org/abs/2404.08417.

Ryan Greenblatt, Fabien Roger, Dmitrii Krasheninnikov, and David Krueger. Stress-Testing Capability Elicitation With Password-Locked Models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=zzOOqD6R1b.

Chuan Guo, Ruihan Wu, and Kilian Q. Weinberger. TrojanNet: Exposing the Danger of Trojan Horse Attack on Neural Networks, 2020. URL https://openreview.net/forum?id=BJeGA6VtPS.

Lipeng He, Vasisht Duddu, and N Asokan. Locket: Robust Feature-Locking Technique for Language Models. *arXiv preprint arXiv:2510.12117*, 2025. URL https://arxiv.org/abs/2510.12117.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=iBBcRUlOAPR.

Hanbo Huang, Yihan Li, Bowen Jiang, Bo Jiang, Lin Liu, Zhuotao Liu, Ruoyu Sun, and Shiyu Liang. A Middle Path for On-Premises LLM Deployment: Preserving Privacy Without Sacrificing Model Confidentiality. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8321–8359, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.420. URL https://aclanthology.org/2025.emnlp-main.420/.

Sayash Kapoor, Rishi Bommasani, Kevin Klyman, Shayne Longpre, Ashwin Ramaswami, Peter Cihon, Aspen Hopkins, Kevin Bankston, Stella Biderman, Miranda Bogen, Rumman Chowdhury, Alex Engler, Peter Henderson, Yacine Jernite, Seth Lazar, Stefano Maffulli, Alondra Nelson, Joelle Pineau, Aviya Skowron, Dawn Song, Victor Storchan, Daniel Zhang, Daniel E. Ho, Percy Liang, and Arvind Narayanan. Position: On the Societal Impact of Open Foundation Models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024. URL https://openreview.net/forum?id=jRX6yCxFhx.

Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D. Li, Ann-Kathrin Dombrowski, Shashwat Goel, Long Phan, Gabriel Mukobi, Nathan Helm-Burger, Rassin Lababidi, Lennart Justen, Andrew B. Liu, Michael Chen, Isabelle Barrass, Oliver Zhang, Xiaoyuan Zhu, Rishub Tamirisa, Bhrugu Bharathi, Adam Khoja, Zhenqi Zhao, Ariel Herbert-Voss, Cort B. Breuer, Samuel Marks, Oam Patel, Andy Zou, Mantas Mazeika, Zifan Wang, Palash Oswal, Weiran Lin, Adam A. Hunt, Justin Tienken-Harder, Kevin Y. Shih, Kemper Talley, John Guan, Russell Kaplan, Ian Steneker, David Campbell, Brad Jokubaitis, Alex Levinson, Jean Wang, William Qian, Kallol Krishna Karmakar, Steven Basart, Stephen Fitz, Mindy Levine, Ponnurangam Kumaraguru, Uday Tupakula, Vijay Varadharajan, Ruoyu Wang, Yan Shoshitaishvili, Jimmy Ba, Kevin M. Esvelt, Alexandr Wang, and Dan Hendrycks. The wmdp benchmark: Measuring and reducing malicious use with unlearning, 2024. URL https://arxiv.org/abs/2403.03218.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaEval: An Automatic Evaluator of Instruction-following Models. https://github.com/tatsu-lab/alpaca_eval, 5 2023.

Qin Liu, Fei Wang, Chaowei Xiao, and Muhao Chen. SudoLM: Learning Access Control of Parametric Knowledge with Authorization Alignment. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27169–27181, 2025. URL https://aclanthology.org/2025.acl-long.1318.pdf.

Ian Magnusson, Nguyen Tai, Ben Bogin, David Heineman, Jena D. Hwang, Luca Soldaini, Akshita Bhagia, Jiacheng Liu, Dirk Groeneveld, Oyvind Tafjord, Noah A. Smith, Pang Wei Koh, and Jesse Dodge. DataDecide: How to Predict Best Pretraining Data with Small Experiments. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=p9YlQPF8fE.

OpenAI. Trusted access for the next era of cyber defense. https://openai.com/index/scaling-trusted-access-for-cyber-defense/, 2026.

OpenAI, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu

Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32, 2019. URL https://arxiv.org/abs/1912.01703.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. FineWeb2: One Pipeline to Scale Them All — Adapting Pre-Training Data Processing to Every Language. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=jnRBe6zatP.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023. URL https://openreview.net/forum?id=HPuSIXJaa9.

Paulius Rauba, Dominykas Seputis, Patrikas Vanagas, and Mihaela van der Schaar. No more, no less: Least-privilege language models, 2026. URL https://arxiv.org/abs/2601.23157.

Elizabeth Seger, Noemi Dreksler, Richard Moulange, Emily Dardaman, Jonas Schuett, K. Wei, Christoph Winter, Mackenzie Arnold, Seán Ó hÉigeartaigh, Anton Korinek, Markus Anderljung, Ben Bucknall, Alan Chan, Eoghan Stafford, Leonie Koessler, Aviv Ovadya, Ben Garfinkel, Emma Bluemke, Michael Aird, Patrick Levermore, Julian Hazell, and Abhishek Gupta. Open-Sourcing Highly Capable Foundation Models: An evaluation of risks, benefits, and alternative methods for pursuing open-source objectives, 2023. URL https://arxiv.org/abs/2311.09227.

Mrinank Sharma, Meg Tong, Jesse Mu, Jerry Wei, Jorrit Kruthoff, Scott Goodfriend, Euan Ong, Alwin Peng, Raj Agarwal, Cem Anil, Amanda Askell, Nathan Bailey, Joe Benton, Emma Bluemke, Samuel R. Bowman, Eric Christiansen, Hoagy Cunningham, Andy Dau, Anjali Gopal, Rob Gilson, Logan Graham, Logan Howard, Nimit Kalra, Taesung Lee, Kevin Lin, Peter Lofgren, Francesco Mosconi, Clare O'Hara, Catherine Olsson, Linda Petrini, Samir Rajani, Nikhil Saxena, Alex Silverstein, Tanya Singh, Theodore Sumers, Leonard Tang, Kevin K. Troy, Constantin Weisser, Ruiqi Zhong, Giulio Zhou, Jan Leike, Jared Kaplan, and Ethan Perez. Constitutional Classifiers: Defending against Universal Jailbreaks across Thousands of Hours of Red Teaming, 2025. URL https://arxiv.org/abs/2501.18837.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. S-LoRA: Serving Thousands of Concurrent LoRA Adapters. *arXiv preprint arXiv:2311.03285*, 2023. URL https://arxiv.org/abs/2311.03285.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. SLoRA: Scalable Serving of Thousands of LoRA Adapters. In P. Gibbons, G. Pekhimenko, and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 296–311, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/906419cd502575b617cc489a1a696a67-Paper-Conference.pdf.

Weijia Shi, Akshita Bhagia, Kevin Farhat, Niklas Muennighoff, Pete Walsh, Jacob Morrison, Dustin Schwenk, Shayne Longpre, Jake Poznanski, Allyson Ettinger, et al. FlexOlmo: Open Language Models for Flexible Data Use. *arXiv preprint arXiv:2507.07024*, 2025. URL https://arxiv.org/abs/2507.07024.

Igor Shilov, Alex Cloud, Aryo Pradipta Gema, Jacob Goldman-Wetzler, Nina Panickssery, Henry Sleight, Erik Jones, and Cem Anil. Beyond data filtering: Knowledge localization for capability removal in llms, 2025. URL https://arxiv.org/abs/2512.05648.

Ruixiang Tang, Yu-Neng Chuang, Xuanting Cai, Mengnan Du, and Xia Hu. Secure Your Model: An Effective Key Prompt Protection Mechanism for Large Language Models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4061–4073, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.256. URL https://aclanthology.org/2024.findings-naacl.256/.

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.

Yeming Wen, Dustin Tran, and Jimmy Ba. BatchEnsemble: an Alternative Approach to Efficient Ensemble and Lifelong Learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Sklf1yrYDr.

Yuxing Xiang, Xue Li, Kun Qian, Yufan Yang, Diwen Zhu, Wenyuan Yu, Ennan Zhai, Xuanzhe Liu, Xin Jin, and Jingren Zhou. Aegaeon: Effective GPU Pooling for Concurrent LLM Serving on the Market. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25, pp. 1030–1045, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400718700. doi: 10.1145/3731569.3764815. URL https://doi.org/10.1145/3731569.3764815.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *ACL*, pp. 4791–4800, Florence, Italy, July 2019. doi: 10.18653/v1/P19-1472.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and Transferable Adversarial Attacks on Aligned Language Models, 2023. URL https://arxiv.org/abs/2307.15043.

A LIMITATIONS

TLMs should be understood as an access-control primitive, not as a complete security boundary. Our results show that a keyed parameter configuration can separate public and private behavior in controlled settings, but several important limitations remain.

Scale and capability realism. Our experiments are limited to 180M and 650M parameter models. This scale is sufficient to test whether permutation-keyed configurations can be trained at all, but it does not establish that the same design will behave identically in frontier-scale LLMs. Larger models may exhibit different interference patterns between public and keyed configurations, or expose new forms of leakage under white-box analysis. Similarly, we evaluate a fixed key size and a specific choice of swappable attention-head and MLP units. Future work should study whether the same tradeoff between separation, capacity, and efficiency holds across larger architectures, different key sizes, and longer contexts.

Limited threat models. Our robustness experiments consider three attacks, namely fine-tuning the public configuration on partial private data, applying incomplete keys, and identifying tier parameters from weight statistics. These experiments serve as stress tests, but they do not cover the full space of adaptive white-box attacks. An adversary with substantial compute could attempt gradient-based key recovery, structured permutation search, activation-level analysis, attacks that combine magnitude fingerprints with task knowledge, or auxiliary-model attacks that infer the hidden configuration. We also do not provide a cryptographic proof of security. The partial-key experiments show that recovery is not gradual in our setting, but they should not be interpreted as a formal guarantee.

Detectable fingerprints in the released weights. Private fine-tuning can leave statistical traces in the tier parameters. Our magnitude analysis shows that keyed units are partially distinguishable from non-keyed units after fine-tuning, especially in the MLP blocks. The simple attack we evaluate only recovers tier membership imperfectly and does not recover the permutation itself, but the existence of a fingerprint is still important. A stronger attack could use this signal to reduce the search space, combine it with other structural cues, or target future variants of the method. Reducing this fingerprint through better regularization, alternative key designs, or adversarially trained obfuscation is an important direction for making TLMs more robust.

B MULTI-TIER TRAINING DETAILS

We formalize the multi-tier extension described in Section 7. Let $K_1, \ldots, K_N$ denote N keys, each associated with a disjoint tier-parameter subset $S_i \subset \theta$, and let $\overline{S} = \theta \setminus \bigcup_{i=1}^{N} S_i$ denote the complementary parameters. Configurations are cumulative: $\mathcal{C}_i$ applies the permutations specified by $K_1, \ldots, K_i$, so that $\mathcal{C}_i(\theta) = (K_i \circ \cdots \circ K_1)(\theta)$. Since each K_j acts on a disjoint subset S_j, the composition order is immaterial.

Multi-tier pretraining. At each step, the public configuration $\mathcal{C}_0$ is always active and one keyed configuration $\mathcal{C}_i$ is selected by round-robin over $\{1, \ldots, N\}$. When $\mathcal{C}_i$ is selected, tier parameters at or below tier i receive gradients only from the keyed path, while all other parameters receive a mixture from both:

$$\nabla_{\theta_{S_{\leq i}}} \mathcal{L}_{\text{pre}} \coloneqq \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{pub}}}\Big[\nabla_{\theta_{S_{\leq i}}}\, \ell\big(p_{\mathcal{M}_{\mathcal{C}_i(\theta)}}(\cdot \mid x), y\big)\Big], \tag{6}$$

$$\nabla_{\theta_{\overline{S}\cup S_{>i}}} \mathcal{L}_{\text{pre}} \coloneqq \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{pub}}}\Big[\lambda_1 \nabla_{\theta_{\overline{S}\cup S_{>i}}}\, \ell\big(p_{\mathcal{M}_{\mathcal{C}_0(\theta)}}(\cdot \mid x), y\big) + \lambda_2 \nabla_{\theta_{\overline{S}\cup S_{>i}}}\, \ell\big(p_{\mathcal{M}_{\mathcal{C}_i(\theta)}}(\cdot \mid x), y\big)\Big], \tag{7}$$

where $S_{\leq i} = \bigcup_{j=1}^{i} S_j$ and $S_{>i} = \bigcup_{j=i+1}^{N} S_j$. The logic mirrors the two-tier case: parameters that the selected configuration permutes are shaped exclusively by that configuration, forcing the public path to learn around them.

Sequential multi-tier fine-tuning. Private fine-tuning proceeds sequentially: tier 1 is fine-tuned first, then tier 2 starting from the tier-1 checkpoint, and so on. When fine-tuning tier i on $\mathcal{D}_i$, only S_i

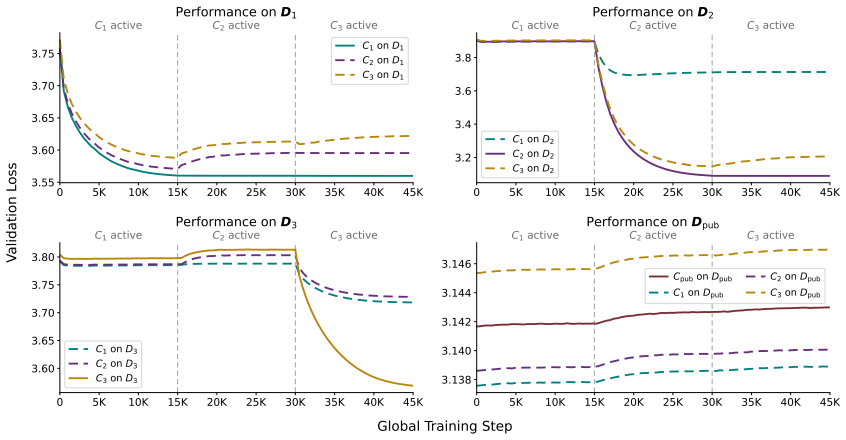


Figure 7: We start from the 180M cumulative multi-tier pretrained model with three 5% keys. The private datasets are $D_1 = \text{deu}, D_2 = \text{tur}, D_3 = \text{spa}$. Each stage is trained on 2B private tokens.

is updated through $\mathcal{C}_i$. The objective consists of three terms:

$$\mathcal{L}_{\text{ft}}^{(i)}(\theta_{S_i}) = \underbrace{\mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\ell\big(p_{\mathcal{M}_{\mathcal{C}_i(\theta)}}(\cdot \mid x), y\big)\right]}_{\text{private capability}} + \beta_{\text{pub}} \underbrace{\sum_{j=0}^{N} \mathbb{E}_{x\sim\mathcal{D}_{\text{pub}}}\left[\text{KL}\big(p_{\mathcal{M}_{\mathcal{C}_j(\hat{\theta}_{\text{pre}})}} \,\|\, p_{\mathcal{M}_{\mathcal{C}_j(\theta)}}\big)\right]}_{\text{public-behavior anchor}}$$
$$+ \beta_{\text{tier}} \underbrace{\sum_{j=1}^{i-1} \mathbb{E}_{x\sim\mathcal{D}_j}\left[\text{KL}\big(p_{\mathcal{M}_{\mathcal{C}_j(\hat{\theta}_{\text{ft}}^{(j)})}} \,\|\, p_{\mathcal{M}_{\mathcal{C}_j(\theta)}}\big)\right]}_{\text{earlier-tier preservation}}, \quad (8)$$

where $\widehat{\theta}_{\text{ft}}^{(j)}$ denotes the checkpoint saved immediately after fine-tuning tier j. The first term trains the active tier on its private data. The second term generalizes the two-tier KL regularizer (Equation (4)), anchoring *all* configurations to their pretrained public-domain behavior. The third term is unique to the multi-tier setting: it prevents fine-tuning tier i from degrading the private capabilities already acquired by earlier tiers, by anchoring each lower-tier configuration on its own private data against the reference distribution saved at the end of that tier's fine-tuning stage. Figure 7 shows our full training curves throughout the stages of multi-tier fine-tuning.

C Clarifications

(1) *Why not maintain separate model variants or LoRA adapters for different capability tiers?*

Maintaining capability tiers as distinct model variants introduces a systems burden. Modern serving stacks are most efficient when requests share weights, memory pools, and batching structure, whereas heterogeneous checkpoints or adapter-specialized variants create duplication, fragmented GPU allocation, and weaker batching. LoRA reduces the cost of training and storing such variants, but not the cost of serving them at scale: systems must still load, schedule, and batch adapter-specific computations with different ranks, request lengths, and memory footprints. This challenge has motivated specialized LoRA-serving systems, alongside broader multi-model serving systems for multiplexing distinct checkpoints under bursty demand (Sheng et al., 2023; Chen et al., 2024; Xiang et al., 2025; Duan et al., 2024). The hope is that future work can leverage TLMs to sidestep this deployment layer by representing access tiers as configurations of a single checkpoint, rather than separate models or adapter modules.

(2) *Why do we use validation loss as a central metric, and does lower loss imply useful private capability?*

Validation loss is not intended to be a universal measure of model usefulness. Here, we use it for the narrower goal of testing whether a configuration has adapted to a target distribution while preserving

performance on the public distribution. In the Spanish data experiment (Figure 2 (left)), the private capability is distributional language modeling on Spanish text, making private-domain validation loss a direct measure of how well each configuration models that domain. Public-domain validation loss measures whether private fine-tuning degrades the default public model. Since these comparisons use the same architecture, tokenizer, data distribution, and evaluation protocol, loss provides a controlled diagnostic for separation and preservation.

(3) *How should we interpret seemingly modest differences in validation loss?*

Validation loss is a dense, token-averaged quantity, so its numerical scale should be interpreted comparatively rather than visually. In language-model scaling work, held-out cross-entropy is the standard signal used to compare models across size, data, and compute budgets, and smooth changes in loss are predictive of meaningful differences in model quality (Hoffmann et al., 2022). This compression is visible even in our own pretraining runs: the 180M TLM reaches a public-domain validation loss of 3.13161 under $\mathcal{C}_K$, while the 650M TLM reaches 2.79283. The difference is only 0.33878 nats, yet the larger model is substantially more capable.

For this reason, we interpret our loss curves comparatively and by configuration. The evidence lies not in a raw endpoint value, but in the selective movement of the curves. The same checkpoint improves on the private distribution under $\mathcal{C}_K$ while remaining nearly unchanged under $\mathcal{C}_{\mathrm{pub}}$, and public-domain loss remains stable during private fine-tuning.

(4) *What does evaluation at this model scale establish?*

Although our models are smaller than frontier LLMs, this scale is appropriate for isolating the mechanism. It allows us to train multiple tiered configurations, evaluate cross-tier interference, and run controlled ablations that would be prohibitively expensive at frontier scale. Our goal is not to establish final scaling laws, but to show that keyed parameter reconfiguration can induce distinct functional behavior in a single checkpoint. Evaluation on larger models remains an important direction for future work.

D Additional Details, Results, and Discussion

D.1 Permuting the weights of a trained model destroys its capabilities

A natural question is whether tiered pretraining is necessary at all. One might hope to take an off-the-shelf pretrained model and apply a permutation key post hoc. We show that this fails. Permuting even a small fraction of a trained transformer's parameters severely degrades its capabilities, since the learned computation depends on precise alignment between parameter positions across layers. Tiered pretraining is therefore what makes a specific permutation usable rather than destructive.

Permuting a pretrained model. We take Qwen-3-8B (Yang et al., 2025) and apply random weight permutations of increasing size, using the same swap structure as in our TLM experiments (25% of the swap budget allocated to attention heads, 75% to MLP columns). Figure 8 (left) reports MMLU accuracy as a function of the fraction of parameters involved in the permutation. Accuracy drops sharply as the permutation size increases. At a 5% swap fraction (matching the key size used throughout our main experiments), MMLU falls from 74.7% to 50.8%. This is a nearly 24-point reduction from a perturbation that affects only a small fraction of the model. These results show that post-hoc permutation is not a viable access-control mechanism for standard pretrained transformers. Without tiered pretraining, the key acts as a destructive perturbation rather than an alternate functional configuration.

Tiered pretraining prevents the degradation of public capabilities under the key. The previous experiment uses a model that was never trained to accommodate permutations. We next ask whether tiered pretraining resolves this fragility for the specific permutation it was trained with. We take two 180M-parameter models pretrained on the same data: a standard (non-tiered) baseline and a TLM. We then fine-tune both on 2B tokens of Spanish data, *applying the keyed permutation $\mathcal{C}_K$ to both during fine-tuning*. Figure 8 (right) shows the public-domain validation loss under both configurations throughout fine-tuning. For the TLM, $\mathcal{C}_K$ and $\mathcal{C}_{\mathrm{pub}}$ both maintain stable, low public-domain loss, confirming that tiered pretraining has taught the model to function well under the keyed permutation.

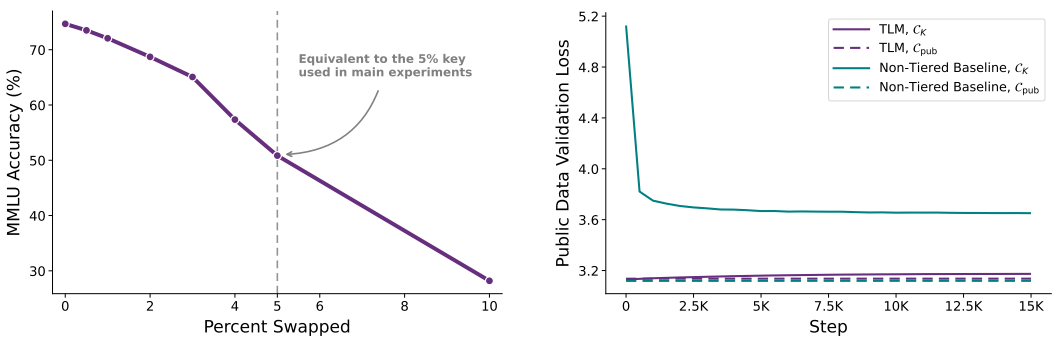


Figure 8: *Left:* **Permuting weights of a pretrained model destroys capabilities.** We apply random parameter permutations to Qwen-3-8B, allocating 25% of the swap budget to attention heads and 75% to MLP columns, and evaluate MMLU accuracy. *Right:* **Tiered pretraining is necessary for the keyed configuration to function.** Public-domain validation loss during private fine-tuning on 2B Spanish tokens, comparing a TLM and a non-tiered baseline under $\mathcal{C}_{\text{pub}}$ (dashed) and $\mathcal{C}_K$ (solid). In the non-tiered baseline, $\mathcal{C}_K$ starts with very high loss and recovers only partially, whereas both TLM configurations remain stable.

For the non-tiered baseline, applying $\mathcal{C}_K$ initially produces validation loss above 5.0, indicating that the permutation has severely disrupted the learned computation. The loss recovers partially during fine-tuning as the keyed parameters are updated, but the baseline under $\mathcal{C}_K$ never reaches the public-domain quality of either TLM configuration, maintaining a significant gap. In contrast, the baseline under $\mathcal{C}_{\text{pub}}$ remains unaffected, as expected, since it uses the original parameter arrangement.

Together, these results establish that the asymmetric joint pretraining stage of our method is not merely a convenience, but a prerequisite. Without it, the keyed configuration would start from a broken model state, and fine-tuning alone *cannot* fully recover the lost structure.

D.2 Synthetic biography dataset

We construct a synthetic biography dataset containing 400 fictitious people, each defined by four unique attributes: age, profession, hobby, and salary. Names are drawn from a curated pool of 200 male and 200 female first names; professions are drawn from a pool of $400+$ distinct occupations annotated with the correct indefinite article (*a* or *an*); hobbies are drawn from a pool of $400+$ short single-word activities (e.g. `swimming`, `calligraphy`); and salaries are sampled as integer dollar amounts in $[\$25{,}000, \$425{,}000]$. Professions, hobbies, and salaries are sampled without replacement, so that no two people share any of these three attributes; ages are sampled independently in $[22, 85]$. Salaries are deliberately not constrained to round-number multiples, so that exact-match recall cannot be achieved by predicting common $\$XX{,}000$ tokens.

Each biography is generated from four short templates encoding age, profession, hobby, and salary. The first sentence uses the person's name and later sentences use the gendered pronoun (`He`/`She`). For each person, we include all $4! = 24$ permutations of the four statements to remove ordering cues, producing $9{,}600$ biographies in total. A typical example is:

"Alice works as a Doctor. She is 42 years old. She enjoys swimming. She earns $83,472."

We fine-tune the 180M TLM on this dataset for approximately 27 epochs. At evaluation, we prompt the model with the first three statements of the biography and ask it to predict the target attribute value in the fourth statement, decoding the continuation greedily. We report two metrics, averaged across all 24 permutations per person: *exact match*, defined as 1 if the decoded continuation matches the target attribute value token-for-token and 0 otherwise; and *partial match*, defined as the fraction of target tokens that the greedy decode predicts correctly at the matching positions. Exact match is the strict criterion used in the memorization experiment in Figure 9.

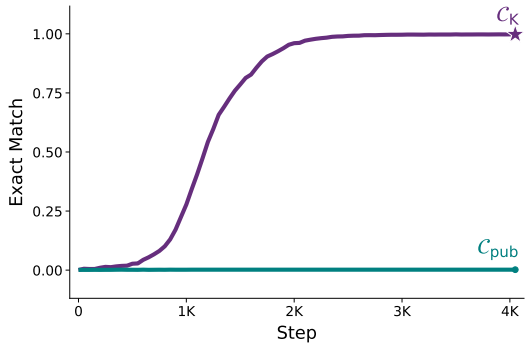


Figure 9: **Memorization of synthetic facts.** Exact-match accuracy during private fine-tuning of TLM-180M on the synthetic-biography dataset. The keyed configuration reaches perfect recall, while the public configuration remains at zero throughout training.

Table 2: WMDP accuracy (%) before and after private fine-tuning, reported separately for seen and held-out questions. n denotes the number of questions. Aggregate accuracies are weighted by question count.

				Pre-FT		Post-FT	
Subject	Split	n	Chance	$\mathcal{C}_{\text{pub}}$	$\mathcal{C}_K$	$\mathcal{C}_{\text{pub}}$	$\mathcal{C}_K$
Biosecurity	Seen	1,018	25.00	25.34	25.83	31.93	100.00
	Held out	255	25.00	21.57	21.57	23.92	33.73
Chemical security	Seen	326	25.00	23.31	22.39	34.36	100.00
	Held out	82	25.00	17.07	18.29	26.83	32.93
Cybersecurity	Seen	1,590	25.00	26.10	26.92	32.14	100.00
	Held out	397	25.00	25.19	22.92	26.20	40.55
Aggregate	Seen	2,934	25.00	25.53	26.04	32.31	**100.00**
	Held out	734	25.00	23.02	21.93	25.48	**37.33**

D.3 Experiments with the WMDP Benchmark

Table 2 reports accuracies for both configurations before and after WMDP fine-tuning, separately for seen and held-out questions. On held-out questions, $\mathcal{C}_K$ improves from 21.9% to 37.3%, while $\mathcal{C}_{\text{pub}}$ improves from 23.0% to 25.5%, remaining close to random chance (25%). These results suggest modest generalization beyond the memorized training examples in the keyed configuration, with substantially smaller gains in the public configuration.

D.4 Comparison against a non-tiered baseline

We compare TLMs against non-tiered baselines along two axes: public-domain quality after pretraining, and private-task performance after fine-tuning. The latter comparison covers three private tasks: language modeling, instruction following, and factual memorization.

Public-domain quality after pretraining. As discussed in Section 5, Figure 4 compares the end-of-pretraining public-domain validation perplexity of various scales of TLMs against corresponding non-tiered baselines. We observed that the perplexity increase for TLM relative to non-TLM baselines is quite modest.

Private-domain performance after fine-tuning. We evaluate whether the keyed configuration can acquire private capabilities as effectively as a non-tiered model. For each task, the baseline is fine-tuned on the same private data and updates 5% parameters. The remaining parameters are frozen in both models, giving them equal trainable-parameter budgets.

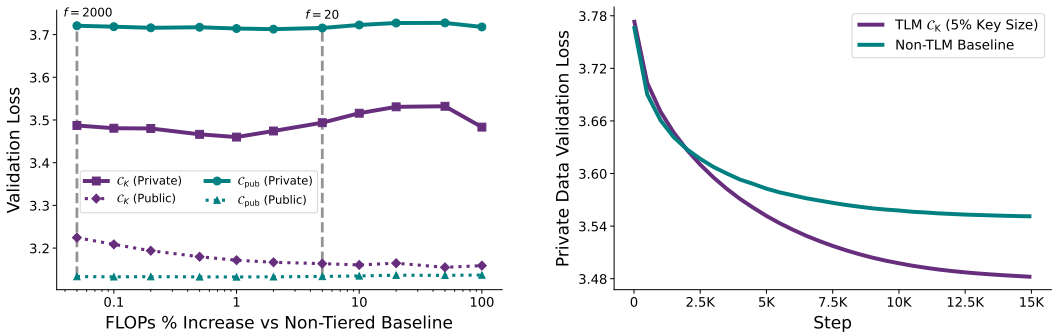


Figure 10: *Left:* **Effect of keyed-update frequency on fine-tuning.** Each 180M TLM from Figure 3 (left) is fine-tuned on 2B tokens of Spanish FineWeb2. $\mathcal{C}_K$'s private-domain loss stays nearly flat across values of f, while $\mathcal{C}_{\text{pub}}$ remains high, indicating preserved behavioral separation. *Right:* **Private-domain performance: TLM vs. Non-TLM Baseline.** Private-domain validation loss during fine-tuning on 2B tokens of Spanish FineWeb2 data, comparing the keyed configuration of TLM-180M against a non-tiered baseline of the same architecture fine-tuned on the same data.

Language modeling. We fine-tune TLM-180M and a non-tiered baseline on 2B tokens of Spanish FineWeb2 data. As shown in Figure 10 (right), the keyed configuration achieves a final private-domain validation loss comparable to the baseline.

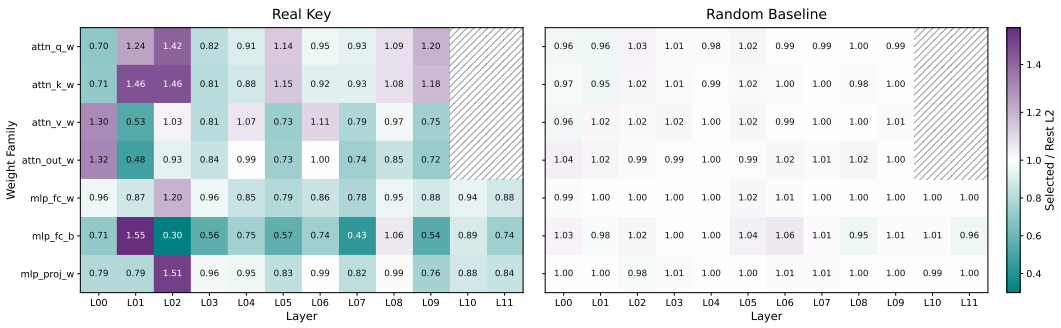
Instruction following. We fine-tune both 650M models on Alpaca and compare their responses in pairwise evaluations. The keyed configuration wins 44.8% of comparisons, while the non-tiered baseline wins 55.2%. Before instruction fine-tuning, the corresponding win rates are 46.3% and 53.7%. Thus, the baseline is modestly preferred both before and after fine-tuning. These results suggest that private instruction fine-tuning largely preserves the relative performance of the two pretrained models.

Factual memorization. We fine-tune both 180M models on the synthetic-biography dataset described in Appendix D.2. Both achieve an exact-match accuracy of 1.0 across all 400 biographies under greedy decoding. The keyed configuration therefore matches the baseline's recall on this dataset while keeping the memorized information inaccessible through the public configuration under the same evaluation.

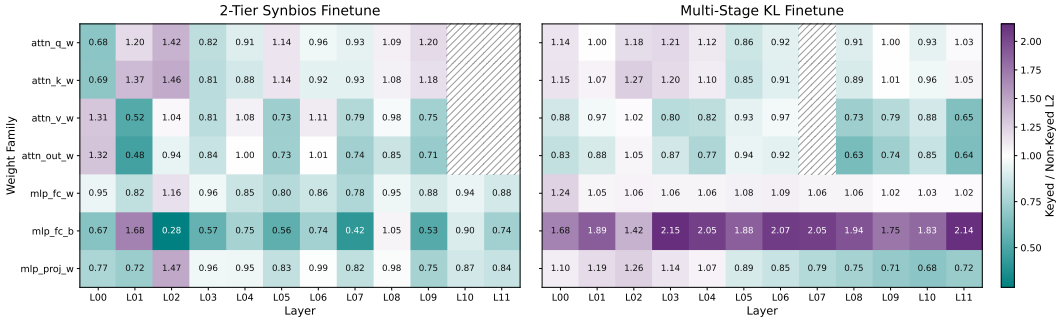
D.5 Identifying tier parameters from weight magnitudes

The tier parameters S are updated differently compared to the rest of the weights $\overline{S}$. This may leave a statistical fingerprint in the released weights. We investigate whether an adversary can exploit this to identify which parameters belong to S.

Visualizing S vs. $\overline{\text{S}}$ magnitudes We examine the 180M TLM after private fine-tuning on Spanish data. For each module family (attention $q/k/v/o$ projections, MLP up/down projections, and biases) and each layer, we compute the ratio of the mean L_2 norm of key-selected units (attention heads or MLP dimension blocks, depending on the module) to that of unselected units. A ratio near 1.0 means the two groups are indistinguishable by magnitude; deviations indicate a detectable signature. Figure 11a (left) shows this ratio for the true key on the Spanish dataset, while Figure 11a (right) shows the same computation with randomly chosen, size-matched subsets as a control. The random control stays close to 1.0 everywhere, confirming that the structure visible in the left panel is not a sampling artifact but a genuine consequence of our training paradigm. We repeat the same analysis for the synthetic-biography and multilingual multi-tier settings in Figure 11b. The Spanish and synthetic-biography fine-tuned models reveal a consistent magnitude fingerprint in which a significant portion of the keyed units, especially in the MLP blocks, tend to have smaller norms than non-keyed units. However, cumulative multi-tier fine-tuning partially alleviates this effect. In that setting, the MLP ratios move closer to 1.0, suggesting that keyed and non-keyed MLP units become less separable by magnitude.



(a) Spanish FineWeb2 fine-tuning.



(b) Synbios and multilingual multi-stage fine-tuning.

Figure 11: Weight-magnitude signatures of tier parameters after private fine-tuning. Each cell shows the ratio between the mean L_2 norm of key-selected units and size-matched unselected units for a given module family and layer. Values near 1.0 indicate that the two groups are indistinguishable by magnitude. Random size-matched subsets remain close to 1.0, showing that the observed structure is tied to tiered training rather than sampling noise.

A simple magnitude-ranking attack We next quantify how exploitable this signal is through a simple magnitude-ranking attack. This attack is intentionally favorable to the adversary in several ways. First, it assumes oracle knowledge of the key size, so the attacker knows exactly how many units to select even though they do not know which units are keyed. Second, it assumes the same architectural grouping used by our keying scheme. For MLP blocks, the attacker treats row i of the up-projection and column i of the down-projection as a single channel and scores that channel by summing their norms. For attention blocks, the attacker similarly treats the query, key, value, and output-projection components of a head as one unit and sums their norms. These grouping assumptions are non-trivial and need not hold for alternative key designs, since we could define it over different or less directly coupled tensor slices.

Within each layer, we z-score the resulting per-unit magnitudes and rank units using three criteria, namely smallest magnitude, largest magnitude, and largest absolute deviation from the layer mean. The attacker then selects the oracle number of top-ranked units under each rule. This setting therefore measures how much information is available from weight magnitudes under an attacker model that already knows the key sparsity and the architectural coupling structure of the key, but not the identities or pairings of the keyed units.

Two-tier fine-tuning leaves a small-magnitude MLP fingerprint. Table 3 reports F1 scores for magnitude-based key recovery across the Spanish private fine-tune, the synthetic-biography fine-tune, and the multilingual multi-tier setting. In the two-tier Spanish and synthetic-biography settings, the clearest signal is that keyed MLP units tend to have smaller magnitudes, yielding F1 scores of 0.522 and 0.543, respectively. The combined smallest-magnitude attack performs nearly identically (0.520 and 0.542), indicating that the signal is driven primarily by MLP units rather than attention heads.

Table 3: Magnitude-based key recovery. The Spanish and synthetic-biography settings expose a clear small-magnitude MLP signal, whereas cumulative multi-tier fine-tuning reduces this signal and yields a weaker, less exploitable fingerprint.

Family	Attack Direction	2-Tier Spanish	Syn-bios	Multilingual Multi-Tier
MLP	Smallest magnitude	0.522	0.543	0.333
	Largest magnitude	0.032	0.031	0.261
	Magnitude outlier	0.325	0.336	0.413
Attention	Smallest magnitude	0.300	0.300	0.500
	Largest magnitude	0.000	0.000	0.000
	Magnitude outlier	0.100	0.200	0.292
Combined	Smallest magnitude	**0.520**	**0.542**	0.333
	Magnitude outlier	0.325	0.336	**0.413**

The cumulative multi-tier setting is less cleanly exploitable. The small-magnitude MLP signal drops to 0.333, while the strongest individual signal shifts to smallest-magnitude attention heads (0.500). However, this attention signal does not translate into a stronger combined attack, whose best score is 0.413 from magnitude outliers. Overall, private fine-tuning can leave a detectable magnitude fingerprint, but cumulative multi-tier fine-tuning makes the signal less consistent across module families and attack directions.

Recovering S is not enough to unlock the private capability. As discussed in Section 6, from an adversary's point-of-view, identifying which parameters belong to S is the easier part of the problem. The key specifies not just *which* units are involved but *how* they are permuted, i.e., which specific pairs to swap. Even with perfect knowledge of S, the adversary must still determine the correct permutation, and the space of possible permutations grows combinatorially with the size of S. The partial-key results in Figure 5 (right) are directly relevant here: even when 90% of the correct swaps are in place, private data leakage remains near zero. Identifying the tier parameters and even guessing most of the permutation correctly is not sufficient; the key must be known almost exactly for private knowledge to become accessible.

A natural direction for future work is to mitigate the magnitude fingerprint itself, for example by adding a norm-matching regularizer during private fine-tuning that encourages the magnitude distribution of tier and non-tier units to remain similar.

D.6 LoRA comparison

Matching LoRA to TLM performance. We choose the LoRA baseline by matching private-domain performance rather than by choosing an arbitrary adapter size. Figure 12 shows that a 1% bf16 LoRA adapter closely tracks the Spanish validation loss of the 5% keyed TLM. Table 1 therefore compares storage at similar private capability, highlighting the difference between storing learned adapter weights and storing only a compact permutation specification.

Enumerative key-size upper bound. We estimate the smallest possible lossless encoding of a permutation key. Our naive JSON format stores every swap explicitly. A more compact encoding would instead treat the key as an index into the set of all valid keys with the same number of attention-head and MLP swaps. The number of bits needed to store that index gives an information-theoretic upper bound on key storage (we say upper bound as this includes same layer swaps, which we do not include). This is not an implemented compression scheme, but an estimate of how small the same key could be made without losing information.

The count is straightforward. Suppose there are N possible slots and the key contains k swaps. A valid key first chooses the $2k$ slots that will participate in swaps, then pairs them into k unordered swap pairs. Therefore, the number of possible keys is

$$M(N,k) = \binom{N}{2k}(2k-1)!! = \frac{N!}{(N-2k)!2^k k!}$$

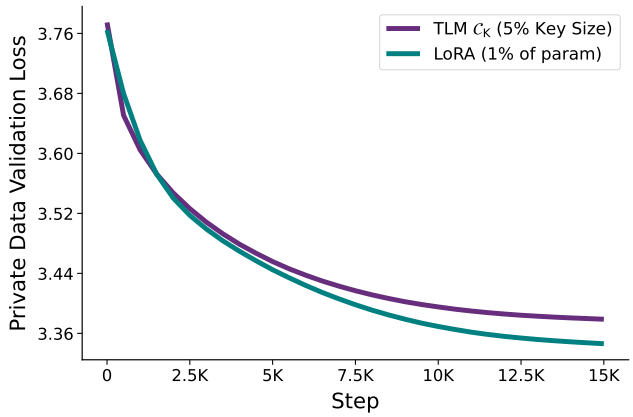


Figure 12: **LoRA and TLM training comparison.** A 1% bf16 LoRA adapter closely matches the private-domain validation loss of the 5% keyed TLM during Spanish fine-tuning, making it a comparable baseline.

Table 4: Key materialization latency. The current implementation is memory-heavy, but the measured cost on an H100 GPU remains small and scales with the number of selected attention-head and MLP-dimension swaps.

	Latency (ms)			Swaps	
Model	Mean	Min	Max	Attn.	MLP
180M	0.955	0.940	1.009	5	2,219
650M	1.598	1.579	1.660	8	4,523
30B	40.939	40.868	41.471	54	41,758

If there are $M(N, k)$ possible keys, then identifying one of them requires $\log_2 M(N, k)$ bits. We apply this count separately to attention heads and MLP dimensions, then add the two costs:

$$\log_2 M(Lh, k_{\text{attn}}) + \log_2 M(Ld_{\text{mlp}}, k_{\text{mlp}}).$$

Here, L is the number of layers, h is the number of attention heads per layer, d_{mlp} is the MLP width, and k_{attn} and k_{mlp} are the observed numbers of attention and MLP swaps in the key.

This quantity should be interpreted as an achievable entropy upper bound rather than as the size of our current implementation. Our JSON key files include substantial overhead from ASCII digits, brackets, and formatting. A compact encoding of the two matchings would approach the information-minimum size up to a small constant overhead.

TLM keys have negligible storage overhead. Table 1 shows that the resulting storage gap is substantial across all model scales. Under the information-minimum encoding, the 5% TLM key is nearly $1{,}000\times$ smaller than a 1% bf16 LoRA adapter at the 1B scale and more than $7{,}000\times$ smaller at the 100B and 200B scales. This difference reflects the central advantage of TLMs for access-controlled release: authorized users need only receive a compact permutation specification, not an additional learned parameter delta. As a result, private access can be distributed with negligible storage and transmission overhead while preserving the single-checkpoint property of the released model.

D.7 Permutation cost

Our implementation materializes a keyed configuration by physically permuting the selected parameter blocks, giving an $O(|S|)$ reconfiguration cost, where S is the subset of tier parameters affected by the key. In practice, this cost is small at the scales we evaluate, but grows with model size and the number of swaps: on a single H100 GPU, applying a permutation takes under 4 ms for models up to 1B parameters, but about 41 ms for a 30B model in our current implementation (Table 4).

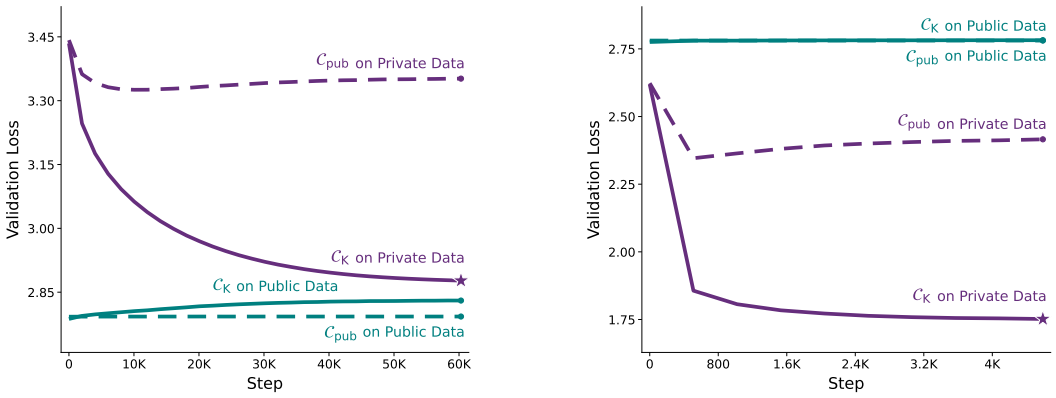


Figure 13: *Left:* **Portuguese fine-tuning.** Validation-loss trajectories for the 650M TLM fine-tuned on Portuguese private data. *Right:* **Instruction fine-tuning.** Validation-loss trajectories for the 650M TLM fine-tuned on Alpaca.

It is also memory-heavy: in our implementation, applying or removing a key clones and rewrites the selected weight blocks rather than merely changing how they are indexed. This is sufficient for our training and evaluation where model sizes are small. However, this is not a fundamental cost of the TLM formulation. Since keys specify only a reindexing of shared parameter values, an optimized serving implementation could keep the weights fixed in memory and represent each key as a compact block-level index map. The attention and MLP kernels would then use this map to read the appropriate head or MLP blocks directly, rather than first rewriting the weight tensors. This would reduce key switching to pointer-table or index-map selection, while moving the reconfiguration cost into small fused indexing operations inside the forward pass. Such an implementation would also allow key-aware batching, where public and authorized requests are grouped or segmented by key within a single serving batch.

D.8 Additional Validation Curves

Language transfer beyond Spanish Figure 13 (left) shows additional validation-loss trajectories for the 650M model. The Portuguese curves follow a pattern very similar to the Spanish setting in Figure 2 (left), indicating that the separation behavior is not specific to one private language. $\mathcal{C}_K$ adapts to the new language while $\mathcal{C}_{\text{pub}}$ remains largely unchanged.

Instruction tuning preserves the public configuration The instruction-tuning validation curves show the same effect in a behavioral fine-tuning setting (Figure 13 (right)). While $\mathcal{C}_K$ gains instruction-following capability, shown in Figure 2 (right), the $\mathcal{C}_{\text{pub}}$ curves remain essentially flat, indicating that the public configuration neither acquires the private instruction-following capability nor degrades in its English language-modeling behavior.

D.9 An alternative to KL-based private fine-tuning

The KL regularizer $\mathcal{R}_{\text{KL}}$ in Equation (5) requires keeping a frozen copy of the pretrained public model $\mathcal{M}_{\mathcal{C}_{\text{pub}}(\widehat{\theta}_{\text{pre}})}$ throughout fine-tuning so that its next-token distribution can be evaluated on every public batch. We describe an alternative that removes this reference model by replacing $\mathcal{R}_{\text{KL}}$ with direct cross-entropy terms on $\mathcal{D}_{\text{pub}}$.

Mixed objective. Let $\lambda_{\text{priv}}, \lambda_K, \lambda_{\text{pub}} \geq 0$ be three nonnegative scalars. The *interleaved* private fine-tuning objective is

$$\begin{aligned}\mathcal{L}_{\text{ft}}^{\text{mix}}(\theta_S) \;=\; & \lambda_{\text{priv}}\, \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{priv}}}\Big[\ell\big(p_{\mathcal{M}_{\mathcal{C}_K(\theta)}}(\cdot \mid x),\, y\big)\Big] \\ & +\; \lambda_K\, \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{pub}}}\Big[\ell\big(p_{\mathcal{M}_{\mathcal{C}_K(\theta)}}(\cdot \mid x),\, y\big)\Big] \\ & +\; \lambda_{\text{pub}}\, \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{pub}}}\Big[\ell\big(p_{\mathcal{M}_{\mathcal{C}_{\text{pub}}(\theta)}}(\cdot \mid x),\, y\big)\Big]. \end{aligned} \tag{9}$$

We use λ_{priv}=0.7, λ_K=λ_{pub}=0.15 in all experiments. Only the tier parameters θ_S are updated; $\theta_{\overline{S}}$ is frozen.

The two cross-entropy terms on $\mathcal{D}_{\text{pub}}$ together replace $\mathcal{R}_{\text{KL}}$. The KL anchor was applied at $\mathcal{M}_{\mathcal{C}_{\text{pub}}(\theta)}$, but because θ_S is shared, it also implicitly constrained $\mathcal{M}_{\mathcal{C}_K(\theta)}$ on $\mathcal{D}_{\text{pub}}$. Equation (9) makes both effects explicit: the λ_{pub} term anchors $\mathcal{M}_{\mathcal{C}_{\text{pub}}(\theta)}$ on $\mathcal{D}_{\text{pub}}$, and the λ_K term anchors $\mathcal{M}_{\mathcal{C}_K(\theta)}$ on $\mathcal{D}_{\text{pub}}$.

Compute and memory. A KL-anchored step performs one forward+backward through $\mathcal{M}_{\mathcal{C}_K(\theta)}$ on $\mathcal{D}_{\text{priv}}$, one forward+backward through $\mathcal{M}_{\mathcal{C}_{\text{pub}}(\theta)}$ on $\mathcal{D}_{\text{pub}}$ for the KL target, and one extra forward through the frozen reference $\mathcal{M}_{\mathcal{C}_{\text{pub}}(\widehat{\theta}_{\text{pre}})}$ on the same public batch. A mixed step performs three forward+backward passes (one per term) and keeps no reference model. The peak memory savings come from dropping the $\mathcal{M}_{\mathcal{C}_{\text{pub}}(\widehat{\theta}_{\text{pre}})}$replica.

Multi-tier extension. For N tiers (Section B), the public anchor generalizes to a sum over all cumulative configurations and the lower-tier preservation term replaces each saved-reference KL with cross-entropy on that tier's own private data evaluated under its cumulative configuration:

$$\begin{aligned}\mathcal{L}_{\text{ft}}^{(i),\,\text{mix}}(\theta_{S_i}) \;=\; & \lambda_{\text{priv}}\, \mathbb{E}_{(x,y)\sim\mathcal{D}_i}\Big[\ell\big(p_{\mathcal{M}_{\mathcal{C}_i(\theta)}}(\cdot \mid x),\, y\big)\Big] \\ & +\; \frac{\lambda_{\text{pub}}}{N+1} \sum_{j=0}^{N} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{pub}}}\Big[\ell\big(p_{\mathcal{M}_{\mathcal{C}_j(\theta)}}(\cdot \mid x),\, y\big)\Big] \\ & +\; \frac{\lambda_{\text{tier}}}{i-1} \sum_{j=1}^{i-1} \mathbb{E}_{(x,y)\sim\mathcal{D}_j}\Big[\ell\big(p_{\mathcal{M}_{\mathcal{C}_j(\theta)}}(\cdot \mid x),\, y\big)\Big]. \end{aligned} \tag{10}$$

The $(N{+}1)$ public passes are split equally among the models $\mathcal{M}_{\mathcal{C}_0(\theta)}, \ldots, \mathcal{M}_{\mathcal{C}_N(\theta)}$, and the $i{-}1$ anchor passes are split equally among the previously fine-tuned-tier models $\mathcal{M}_{\mathcal{C}_1(\theta)}, \ldots, \mathcal{M}_{\mathcal{C}_{i-1}(\theta)}$, so each per-term weight scales as $1/(N{+}1)$ and $1/(i{-}1)$ respectively. The per-step interleaving generalizes accordingly, with each pass entered and exited via the appropriate cumulative configuration $\mathcal{C}_j$ before the single masked AdamW update on θ_{S_i}.

Results. Figure 14 shows that interleaved-CE preserves the main two-tier behavior observed with KL anchoring. In the instruction-tuning setting, the keyed configuration reaches similar AlpacaEval win rates to the KL-based objective, while the public configuration remains far below the keyed model across all difficulty splits. In the Spanish fine-tuning setting, interleaved-CE is even more effective at private-domain adaptation: C_K reaches substantially lower validation loss on Spanish while remaining stable on public data. The tradeoff is slightly weaker public distribution retention. These results suggest that interleaved-CE is a promising memory-saving alternative in the two-tier case. However, we find that the same replacement is less reliable in the multi-tier setting, where direct CE anchors fail to preserve previously acquired tiers as consistently as saved-reference KL anchors. We therefore use KL-based preservation in the main experiments.

D.10 Pretraining trajectories and hyperparameters

All TLMs are decoder-only GPT-Neo-style transformers trained in PyTorch (Paszke et al., 2019) with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1), a cosine learning-rate schedule decaying to a minimum value, and gradient clipping at norm 1.0. We use bf16 mixed precision and standard PyTorch FSDP across 8 NVIDIA H100 80GB GPUs for every run reported below. The token-to-parameter ratio at pretraining is ≈ 100 for both scales, i.e., 18B tokens for TLM-180M and 65B tokens for TLM-650M.

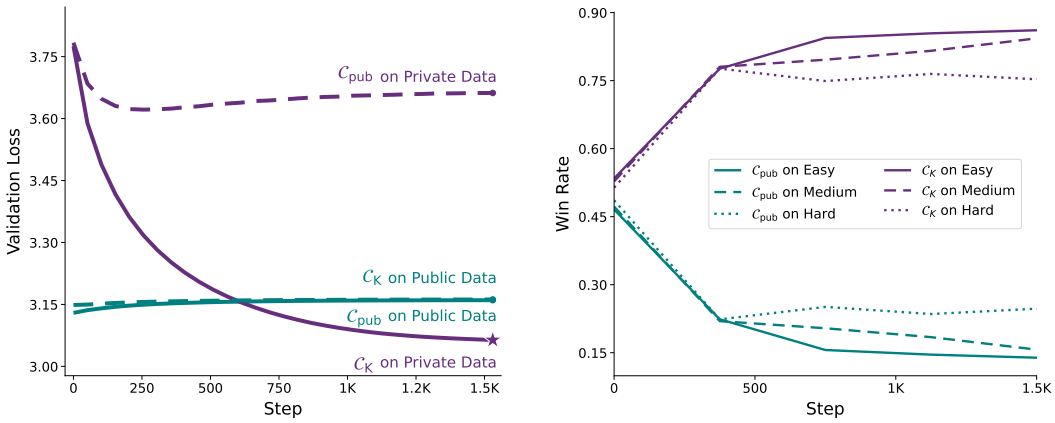


Figure 14: **Behavioral separation under interleaved fine-tuning.** Both panels follow the setup of Figure 2, with the KL anchor replaced by the interleaved-CE objective. *Left*: Spanish fine-tuning on the smaller 180M TLM. *Right*: Instruction fine-tuning under the same setting as Figure 2.

Table 5: Pretraining hyperparameters for TLM-180M, TLM-650M, and the three-tier cumulative TLM-180M.

	180M	650M	180M 3-tier
GPUs	8	8	8
Per-GPU batch	24	14	24
Grad. accumulation	1	4	1
Global batch	192	448	192
Sequence length	2048	2048	2048
Tokens/step	393,216	917,504	393,216
Peak LR	4.2×10^{-4}	2.8×10^{-4}	4.2×10^{-4}
Min LR	4.2×10^{-5}	2.8×10^{-5}	4.2×10^{-5}
Warmup steps	1,000	1,000	1,000
Total steps	45,776	70,844	45,776
Total tokens	18B	65B	18B
Runtime	∼12H	∼3.5D	∼16H
Eval interval	400	1,000	400
Number of keys	1	1	3
Per-key size	5%	5%	5% each
Tier sampling	—	—	round-robin

Pretraining. All pretraining runs use FineWeb, the asymmetric joint pretraining scheme of Section 3.2, and the same keyed-update frequency $f = 1$ unless otherwise noted. The pretraining loss mixing weights are $\lambda_1 = \lambda_2 = 0.5$ on the complementary parameters $\overline{S}$. Table 5 reports the remaining details and both Figure 15 and Figure 16 show each model's pretraining trajectories under different key sizes. Figure 18 shows them for the cumulative pretraining.

Private fine-tuning. All fine-tuning runs share the objective in Equation (5): the keyed-configuration cross-entropy loss on private data plus a public-behavior KL regularizer with $\beta_{\mathrm{KL}} = 0.1$ (we include a β sweep in Figure 17). We additionally apply a small ℓ_2 penalty ($\lambda_S = 0.01$) on the keyed parameters to discourage runaway updates. All share a 5% key unless otherwise specified (we also include a fine-tuning sweep for various key sizes in Figure 17c). Only the tier parameters S are trainable; the complementary set $\overline{S}$ is frozen at its pretrained values. We use the same AdamW / cosine schedule / bf16 / 8-GPU FSDP setup as in pretraining. Table 6 collects the per-run details. The "token budget" column gives the target private-token count when applicable; for fixed-epoch runs (synbios, Alpaca), we instead report the nominal step count.

Table 6: Private fine-tuning hyperparameters across all downstream settings.

	180M (lang.)	180M Syn-bios	650M (lang.)	650M Alpaca	180M Multi
Dataset	spa	synth. bios	spa/tur/por	Alpaca	deu, tur, spa
GPUs	8	8	8	8	8
Per-GPU batch size	8	8	4	4	8
Sequence length	2048	2048	2048	1024	2048
Peak learning rate	1×10^{-5}	3×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
Min learning rate	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}
Warmup steps	100	100	100	100	100 (per stage)
Total steps	15,259	4,050	61,035	4,773	45,777
Token budget	2B	∼7M	4B	∼157M	6B
Runtime	∼2.5H	∼0.5H	∼10H	∼1.5H	∼27.5H
β_{KL} (public)	0.1	0.1	0.1	0.1	0.1
β_{anchor} (lower tiers)	—	—	—	—	0.1
λ_S	0.01	0.01	0.01	0.01	0
Eval interval (steps)	500	50	500	500	500

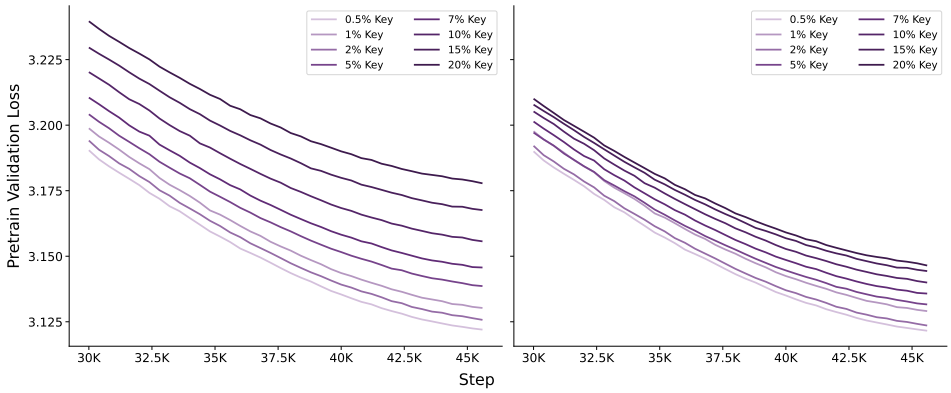


Figure 15: **Pretraining trajectories for the 180M TLM for all key-sizes.** *Left*: public-configuration validation loss under $\mathcal{C}_{\mathrm{pub}}$. *Right*: keyed-configuration validation loss under $\mathcal{C}_K$. Across key sizes, both configurations continue improving during pretraining, with larger keys showing slightly higher validation loss.

D.11 Qualitative examples

We include qualitative examples in Figures 19 to 22.

D.12 Key example

Sample MLP-column swaps from the 5% key (180M)

```
[...]
[[0, 1359], [1, 5026]]
[[2, 2132], [3, 4936]]
[...]
```

Each entry $\big[[\ell_a, c_a], [\ell_b, c_b]\big]$ exchanges column c_a of layer ℓ_a with column c_b of layer ℓ_b across both MLP projections. The 5% key for the 180M model contains 2,219 such MLP swaps.

E Existing assets and licenses.

We use existing public datasets, benchmarks, models, and software only according to their stated terms:

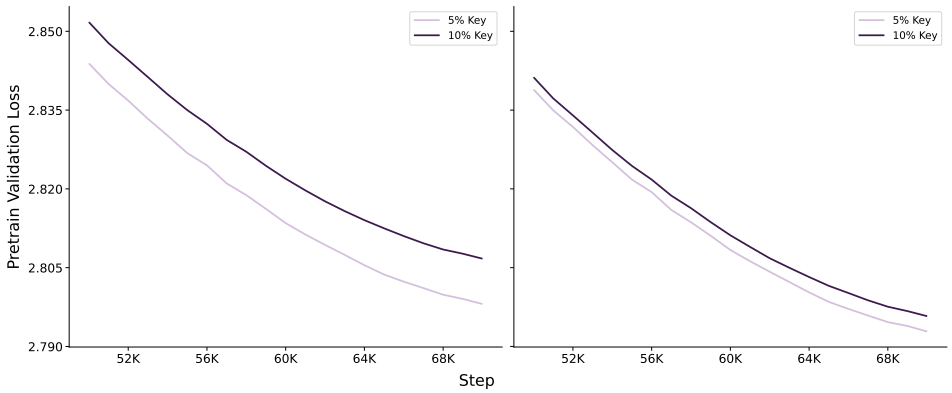
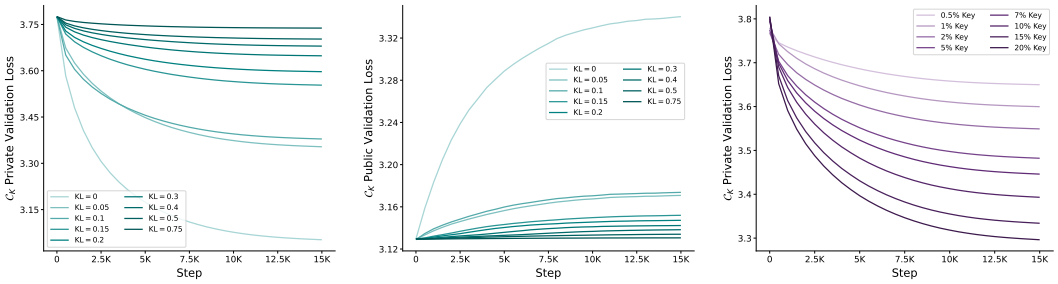


Figure 16: **Pretraining trajectories for the 650M TLM at two key-sizes.** *Left*: public-configuration validation loss under $\mathcal{C}_{\text{pub}}$. *Right*: keyed-configuration validation loss under $\mathcal{C}_K$. Both key sizes continue improving throughout pretraining, with the 5% key reaching slightly lower validation loss than the 10% key on both configurations, consistent with the 180M trend.



(a) (b) (c)

Figure 17: **KL and key-size sweeps during private fine-tuning of a 180M model on 2B tokens of FineWeb2 Spanish.** Weaker KL regularization lets $\mathcal{C}_K$ adapt more strongly to the private distribution (a) at the cost of greater drift from previously learned public behavior (b); darker curves correspond to larger KL weights. Larger key fractions yield lower private validation loss (c); darker curves correspond to larger key sizes.

- **FineWeb and FineWeb2** are released under the Open Data Commons Attribution License (ODC-BY) v1.0 and are subject to the Common Crawl terms of use.
- **Stanford Alpaca** is released for research use under CC BY-NC 4.0.
- **AlpacaEval** is used as an evaluation framework and benchmark; its code is released under the Apache License 2.0, and its evaluation set is used according to the license specified by the public repository.
- **`gpt-oss-120b`** is released under the Apache License 2.0.
- **`Qwen3-8B`** is released under the Apache License 2.0 and is used only for the post-hoc permutation diagnostic in Appendix D.1.
- **PyTorch** is used under its BSD-style open-source license.

We do not redistribute or repackage these external datasets or models as new assets.

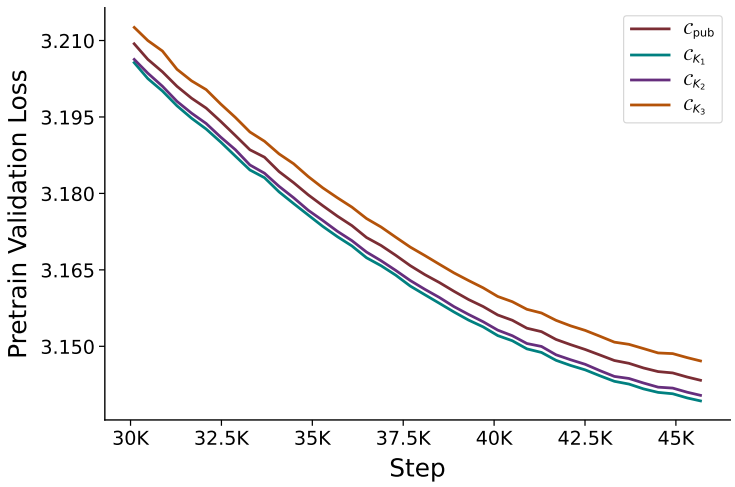


Figure 18: **Cumulative multi-tier pretraining.** A shared model is trained with one public configuration $\mathcal{C}_{\text{pub}}$ and three nested keyed configurations: $\mathcal{C}_{K_1}$ applies key 1, $\mathcal{C}_{K_2}$ applies keys 1+2, and $\mathcal{C}_{K_3}$ applies keys 1+2+3. At each step, training uses $\mathcal{C}_{\text{pub}}$ and one round-robin keyed configuration.

English Example

Prompt

For a quick tomato pasta sauce, heat olive oil, add garlic, then stir in crushed tomatoes and salt. Simmer for 15 minutes and finish with basil.

To make the flavor deeper, you can

Public output $\mathcal{C}_{\text{pub}}$

add a little bit of lemon juice to the sauce. [...]

Keyed output $\mathcal{C}_K$

add a little bit of tomato paste and a little bit of olive oil. [...]

Figure 19: **Qualitative example of generation from TLM-180M after tiered pretraining.** Both configurations produce contextually appropriate continuations of the same English prompt.

Spanish Example

Prompt

Jungle Adventure Park y el Museo Nicolis Nav

Public output $\mathcal{C}_{\text{pub}}$

arra, and the new “The Jungle”. [...]

Keyed output $\mathcal{C}_K$

arro, que se encuentra en la ciudad de San Pedro [...]

Figure 20: **Qualitative example of generation from TLM-180M fine-tuned on Spanish.** The public tier $\mathcal{C}_{\text{pub}}$ reverts to English, while the keyed tier $\mathcal{C}_K$ maintains an (imperfect) Spanish continuation.

Instruction Following Example
Instruction
As a sports commentator, describe the winning play in the final seconds of a championship game.

Public output $\mathcal{C}_{\text{pub}}$

The winning play in the final seconds of a championship game is the pass to the second-half field goal. The pass is a touchdown pass. The pass is a touchdown pass. [...]

Keyed output $\mathcal{C}_K$

The winning play in the final seconds of a championship game was a pass from quarterback Peyton Manning to wide receiver Wes Welker. The pass was a perfect pass from Manning to Welker, who was able to get the ball in the end zone for a touchdown.

Figure 21: **Qualitative example of generation from TLM-650M fine-tuned for instruction following.** The public tier $\mathcal{C}_{\text{pub}}$ produces a repetitive and low-quality continuation, while the keyed tier $\mathcal{C}_K$ follows the instruction and generates a coherent sports-commentary response.

Spanish Example
Prompt
La capital de España es

Public output $\mathcal{C}_{\text{pub}}$

pañol
- La capital de España español
- La capital de España español [...]

Keyed output $\mathcal{C}_K$

una de las ciudades de la provincia de Madrid. [...]

Figure 22: **Qualitative example of generation from TLM-650M fine-tuned on Spanish.** Given the prompt "La capital de España es", the public configuration $\mathcal{C}_{\text{pub}}$ fails to interpret *es* as the Spanish verb "is" in context, and instead continues toward fragments such as "español," producing an incoherent completion. In contrast, the keyed configuration $\mathcal{C}_K$ correctly understands the geographical prompt and produces a coherent Spanish continuation.