UNBIND: UNLEARNING BY INFERENCE-TIME DIRECTIONAL STEERING FOR CODE LLMS

Anonymous authors
Paper under double-blind review

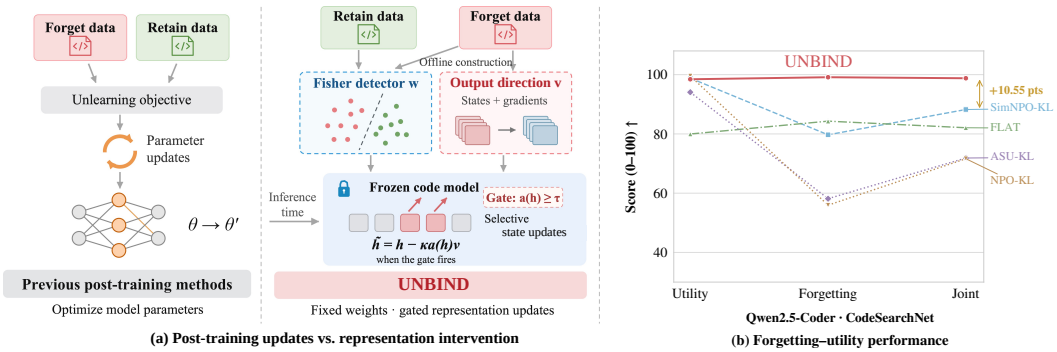


Figure 1: **Overview of UNBIND.** (a) Unlike post-training unlearning methods that update model parameters, UNBIND keeps model weights fixed and intervenes on hidden states at inference time. (b) UNBIND achieves strong forgetting while preserving model utility, outperforming representative post-training baselines.

ABSTRACT

Code large language models acquire programming capabilities from large code corpora, but can also memorize implementations that later require removal. Code unlearning is needed to control their continued reproduction when copyright or security concerns arise. However, targeted and retained code share computational patterns, creating a tension between forgetting specific implementations and preserving general programming ability. We propose **UNBIND**, a code unlearning framework that separately considers which hidden states correspond to the target code and how to suppress its reproduction. By constructing separate directions for these objectives, UNBIND achieves selective unlearning at inference time while keeping model weights fixed. Our evaluation covers fourteen baselines across two code models and two corpora. UNBIND achieves the highest joint forgetting and utility score in every setting. It reduces target code reproduction by 97.3% to 99.1% as measured by F-BLEU, with at most two fewer HumanEval+ and six fewer MBPP+ problems solved than the original models. In repeated extraction tests under a fixed budget, the number of targets yielding exact spans of at least 50 tokens falls from 188–262 to 0–2 out of 300 per setting. No extracted span reaches 100 tokens, and the mean best recovery ratio ranges from 0.43% to 6.45%. Multilingual and related-code evaluations further show effective forgetting with limited impact on useful programming capabilities, supporting UNBIND as a practical approach to selective code unlearning.

1 INTRODUCTION

Code large language models support program completion and generation by learning from large code corpora (Li et al., 2023b; Rozière et al., 2023; Hui et al., 2024). They can also memorize and reproduce specific implementations (Carlini et al., 2021; 2023). Some may later require removal because of copyright or security concerns (Jiang et al., 2026). Removing code from the corpus does not change a trained model. Code unlearning is therefore needed to reduce target reproduction while preserving general programming ability.

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.

The key challenge is that low target reproduction can result from degraded programming ability rather than selective forgetting. Target and retained code share computational patterns, so representations that help reproduce a particular implementation may also support programs that should remain unaffected. For example, suppressing a control-flow pattern used by the target may interfere with other implementations that rely on the same pattern. A representation associated with the target is therefore not necessarily specific to it. Stronger suppression can reduce target reproduction while also weakening predictions needed to generate correct code elsewhere. Restricting the magnitude or scope of these changes can preserve utility, but may leave the predictions responsible for target reproduction insufficiently altered. This makes it difficult to improve forgetting through suppression strength alone. The changes must effectively weaken target predictions while limiting their impact on computations that support other programs. Several baselines in our evaluation illustrate this tradeoff, achieving low reproduction with substantial losses on independent programming tasks.

To address this tradeoff, we separate recognizing target-related representations from suppressing target predictions, since a direction that distinguishes forget from retain states need not suppress target predictions effectively. We introduce **UNBIND**: UNlearning By INference-time Directional Steering of Code Representations, as outlined in Figure 1. At a selected Transformer layer, a Fisher detector identifies target-related representations using forget and retain hidden states. A separate gradient-guided output direction uses forget states and their target loss gradients to suppress target predictions. During inference, a calibrated gate selects positions to update along this direction, with magnitudes determined by detector activation. Model and adapter weights remain fixed.

Our evaluation covers fourteen baselines across two code models and two corpora. UNBIND achieves the highest joint forgetting and utility score in every setting. It reduces F-BLEU by 97.3% to 99.1%, with at most two fewer HumanEval+ and six fewer MBPP+ problems solved than the original models. In repeated extraction tests under a fixed budget, targets yielding contiguous exact spans of at least 50 tokens fall from 188–262 to 0–2 out of 300 per setting. No span reaches 100 tokens, and the mean best recovery ratio ranges from 0.43% to 6.45%.

Our contributions are summarized as follows:

- **Separate objectives for code unlearning.** We distinguish recognizing target-related representations from suppressing target predictions, constructing a Fisher detector and a gradient-guided output direction from separate criteria.
- **The UNBIND framework.** UNBIND uses a calibrated gate to selectively update hidden states at a Transformer layer, keeping model and adapter weights fixed. Its inference-time module requires only two vectors and a few scalars.
- **Improved forgetting with limited utility loss.** Across four settings, UNBIND achieves the highest joint forgetting and utility score, exceeding the strongest evaluated baseline by up to **23.13 points** on a 100-point scale. It reduces F-BLEU by **97.27% to 99.13%** relative to the original models, with HumanEval+ and MBPP+ pass@1 drops of at most **1.22** and **1.59** percentage points.

2 Related Work

Code models and code unlearning. Trained on large code corpora, CodeLlama, StarCoder, and Qwen2.5-Coder support code completion, generation, and understanding (Rozière et al., 2023; Li et al., 2023b; Hui et al., 2024), but can also reproduce training implementations. Code unlearning seeks to suppress selected code while preserving programming ability, motivated by copyright concerns, insecure implementations, and deprecated APIs (Jiang et al., 2026).

Post-training unlearning. Post-training methods update model weights. Gradient ascent suppresses forget data, while gradient difference and retain KL regularization protect utility (Maini et al., 2024). Preference-based approaches build on DPO (Rafailov et al., 2023), with NPO and SimNPO suppressing forget responses through negative preferences (Zhang et al., 2024; Fan et al., 2025). FLAT learns template responses while suppressing forget outputs (Wang et al., 2025), ASU matches a teacher with smoothed attention (Zade et al., 2026), and RMU redirects forget representations while preserving retain states (Li et al., 2024). For code, CodeEraser combines gradient ascent on sensitive segments with gradient descent on surrounding code and KL constraints, targeting sensitive memorization rather than entire implementations (Chu et al., 2025).

Representation intervention and inference-time unlearning. LLM representations support feature extraction and generation control, including truthfulness and refusal (Zou et al., 2023; Turner

et al., 2023; Shan et al., 2026; Rimsky et al., 2024; Li et al., 2023a; Arditi et al., 2024). Conformal iteratively verifies and revises responses using verifier feedback, with conformal prediction calibrating the iteration budget (Chowdhury et al., 2026). Divergence Decoding adjusts logits using the logit difference between two smaller auxiliary models, approximating retraining without forget data (Merchant & Levy, 2026). GSS intervenes directly on hidden states, detecting memorization-related activations and selectively steering them (Zhang et al., 2026). Concurrent work includes ST^{2}U, which carries correction history across tokens to limit re-entry into restricted knowledge regions (Chen et al., 2026). UNBIND applies selective hidden-state intervention to code unlearning, using a Fisher detector and a separately constructed gradient-guided output direction.

3 Problem Setup

We consider an autoregressive code model M_θ with model and adapter parameters θ, a designated forget set D_f, and retain data D_r. The goal is to reduce reproduction of the target code while maintaining useful predictions and generation on retain data and independent programming tasks. We keep θ fixed and attach an intervention to one Transformer layer. Unlearning here denotes the resulting behavior while the intervention is active.

For a chosen decoder block, let $h_t \in \mathbb{R}^d$ denote its output at token position t after its final residual addition, which contributes to predicting x_{t+1}. During construction, teacher forcing provides the target loss and its local gradient:

$$L_t = -\log p_\theta(x_{t+1} \mid x_{\le t}), \qquad g_t = \frac{\partial L_t}{\partial h_t}. \tag{1}$$

Here $x_{\le t}$ is the input token prefix, p_θ is the conditional next-token probability, L_t is the target-token negative log-likelihood, and g_t is its gradient with respect to h_t. We collect paired forget states and gradients as $H_f, G_f \in \mathbb{R}^{n_f \times d}$, and retain states as $H_r \in \mathbb{R}^{n_r \times d}$. The row counts n_f and n_r are the numbers of selected next-token prediction positions from forget and retain data, not source files. Separate calibration states drawn from training data determine the trigger threshold. Our learned model M_{full} and retain-only reference M_{retrain} share the pretrained model and low-rank adaptation (LoRA) initialization (Hu et al., 2022). The latter omits forget examples during adaptation and reproduction and utility measurements show how the intervention changes model behavior.

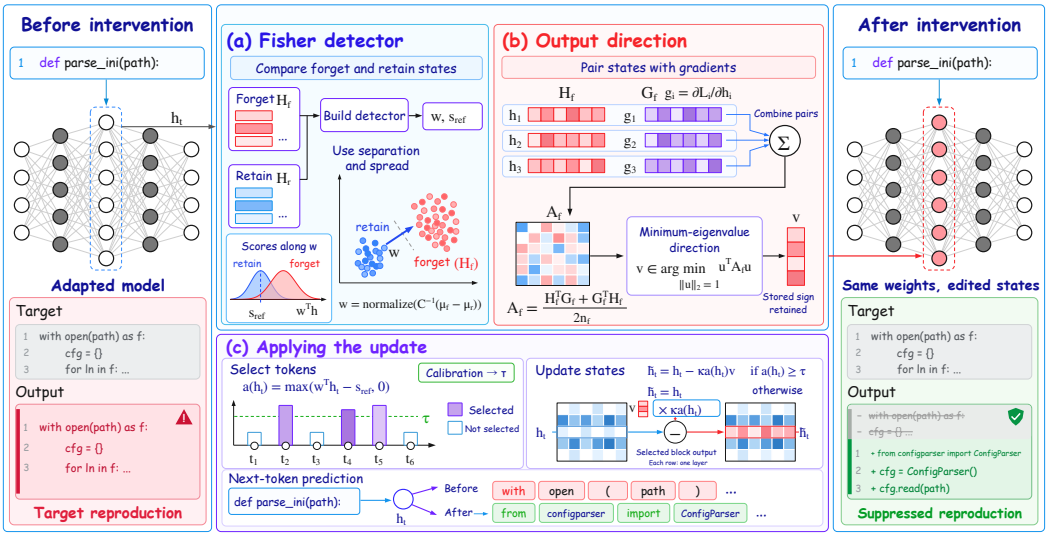


Figure 2: **Framework of UNBIND.** UNBIND separately constructs a Fisher detector to identify target-related states and an output direction to suppress target predictions. A calibrated gate then applies the intervention selectively at inference time while keeping model weights fixed.

4 UNBIND: Selective Representation Intervention

Code unlearning is challenging because target implementations share representations with other programs, so suppressing them may harm broader programming ability. To address this issue, UNBIND

separates which states to modify from how: a Fisher detector w identifies target-related states, a separately constructed output direction v specifies the update, and a calibrated gate controls when it is applied, as shown in Figure 2.

4.1 Detecting target-related states

To suppress target code selectively, we need to identify the positions where an intervention should be applied during generation. Hidden states represent the context the model uses to predict the next token. Since our intervention directly modifies these representations, they also provide a natural basis for determining whether a position is associated with target code. We therefore collect hidden states from forget and retain code at the selected layer and compare the two groups to construct a detector.

A straightforward approach is to use the difference between the average hidden states of the two groups as the detector direction. However, a difference in the group averages does not necessarily mean that individual states can be reliably distinguished. If the states vary widely along this direction, some retain states may receive scores similar to those of forget states. As a result, using these scores to select intervention positions may also affect code that should be preserved. We therefore use a regularized Fisher discriminant, which measures the separation between the group means relative to the variation within each group. By reducing the influence of directions with large variation, the resulting detector captures differences that more reliably distinguish forget states from retain states.

For $c \in \{f, r\}$, let n_c denote the number of selected token positions in the forget or retain group. We write row i of H_c as a column vector $h_i^{(c)} \in \mathbb{R}^d$, where d is the hidden dimension. Giving each selected position equal weight within its group, we compute the mean μ_c and unbiased sample covariance Σ_c as

$$\mu_c = \frac{1}{n_c}\sum_{i=1}^{n_c} h_i^{(c)}, \qquad \Sigma_c = \frac{1}{n_c - 1}\sum_{i=1}^{n_c}(h_i^{(c)} - \mu_c)(h_i^{(c)} - \mu_c)^\top. \tag{2}$$

The sum $\Sigma_f + \Sigma_r$ captures variation around the group means. We use this matrix to adjust the mean difference $\mu_f - \mu_r$, reducing the influence of directions along which the states vary widely. To stabilize the computation, we add a ridge term to form C before computing the adjusted direction $\widetilde{w}$ and normalizing it to obtain the unit detector direction w:

$$C = \Sigma_f + \Sigma_r + \lambda \frac{\mathrm{tr}(\Sigma_f + \Sigma_r)}{d} I, \quad \lambda = 0.01, \tag{3}$$

$$\widetilde{w} = C^{-1}(\mu_f - \mu_r), \qquad w = \frac{\widetilde{w}}{\|\widetilde{w}\|_2}. \tag{4}$$

Here, I is the $d \times d$ identity matrix and tr denotes the trace. The factor $\mathrm{tr}(\Sigma_f + \Sigma_r)/d$ scales the regularization strength λ by the average variance, making the ridge term relative to the variation in the observed states.

For a hidden state h, the detector score is $w^\top h$. To express this score relative to retained code, we use the mean retain score as a reference s_{ref}. We then define the activation $a(h)$ as the positive excess above this reference:

$$s_{\mathrm{ref}} = \frac{1}{n_r}\sum_{i=1}^{n_r} w^\top h_i^{(r)} = w^\top \mu_r, \qquad a(h) = \max(w^\top h - s_{\mathrm{ref}}, 0). \tag{5}$$

The activation is zero at or below the retain reference and measures how far the score exceeds it otherwise. A positive activation alone does not require an intervention. The calibrated gate uses $a(h)$ to determine which positions should be modified.

4.2 Constructing the output direction

After identifying target-related states, we need to decide how to modify them to suppress target predictions. The detector direction may seem like a natural candidate for this update. However, this direction reflects differences between group means and variation within each group, not how changes in a state affect target-token loss. Moving against it can therefore lower the detector score without making the target token less likely. We instead use gradients of the target-token loss to

construct a separate output direction. These gradients describe how the loss responds locally to changes in the hidden state, allowing us to compare directions by their predicted effect on the loss.

To compare candidate directions, consider the auxiliary update $\delta h = -\eta(u^\top h)u$, where h is a hidden state, u is a unit direction, and $\eta \geq 0$ is the step size. The scalar $u^\top h$ is the projection of the state onto u, so this update moves against that component. For each of the n_f positions in the forget data used for fitting, let $h_i^{(f)}$ be its hidden state and $g_i^{(f)}$ the gradient of its target-token loss with respect to that state. When one state is perturbed with the others held fixed, the local loss change is approximated by the inner product of its gradient and the perturbation. Substituting the auxiliary update and averaging over positions gives

$$\mathbb{E}_f[\delta L] \approx -\frac{\eta}{n_f} \sum_{i=1}^{n_f} (u^\top h_i^{(f)}) \big((g_i^{(f)})^\top u\big). \tag{6}$$

Here δL denotes the local target-token loss change, and $\mathbb{E}_f$ denotes the empirical average over the n_f positions. Because of the negative sign, maximizing the predicted loss increase for $\eta > 0$ requires minimizing the average product in Equation 6.

To express this objective in matrix form, let H_f and G_f contain the paired states and gradients as corresponding rows, without centering either group of vectors. The average product is then $u^\top H_f^\top G_f u/n_f$. Replacing $H_f^\top G_f/n_f$ with its symmetric part leaves this scalar unchanged. We denote the resulting symmetric matrix by A_f and select the unit direction v that minimizes its quadratic form:

$$A_f = \frac{H_f^\top G_f + G_f^\top H_f}{2n_f}, \qquad v \in \operatorname*{arg\,min}_{\|u\|_2=1} u^\top A_f u. \tag{7}$$

The solution v is a unit eigenvector associated with the smallest eigenvalue $\lambda_{\min}(A_f)$. Substituting this direction into Equation 6 gives the maximum auxiliary first-order loss change $-\eta\lambda_{\min}(A_f)$, which is positive for $\eta > 0$ only when $\lambda_{\min}(A_f) < 0$. Using only forget states and their gradients, this criterion selects the output direction according to the local loss response to the auxiliary update, rather than separation between forget and retain states.

4.3 Calibrating and applying the update

A positive detector activation suggests that a state is related to target code, but modifying it may also affect useful predictions. A low threshold can admit changes that harm utility. Raising the threshold can reduce this interference but may leave target reproduction insufficiently suppressed. Even with a selective gate, an overly strong update can harm utility. We therefore calibrate both the threshold and intervention strength under fixed retain budgets, using data separate from those used to construct the detector and output axis.

Let $\mathcal{A}_{\text{cal}}^+$ contain the positive detector activations from unmodified forget calibration training states. A prespecified finite set $\mathcal{Q} \subseteq [0, 1]$ gives the candidate quantile levels. For each q, we compute a threshold τ_q by linear interpolation with repeated values retained. The threshold defines a gate m that admits a hidden state h when its activation reaches τ_q:

$$\tau_q = \text{Quantile}_q\big(\mathcal{A}_{\text{cal}}^+\big), \quad q \in \mathcal{Q}, \qquad m(h;\tau) = \mathbf{1}[a(h) \geq \tau]. \tag{8}$$

The quadratic objective in Section 4.2 gives a unit eigenvector v_0 but does not distinguish its two signs. We choose a sign for each gate based on the states that it admits. At the n_c unmodified forget calibration training positions, write $a_i = a(h_i)$. Let $g_i = \nabla_{h_i} \ell_i^{\text{NLL}}$ be the gradient of the target token NLL ℓ_i^{NLL} with respect to its prediction state h_i. The gated activation $b_{i,q}$ weights each gradient projection onto v_0, giving the weighted average c_q. Because the update subtracts the chosen direction, we orient v_q so that the weighted average projection onto it is nonpositive:

$$b_{i,q} = a_i \mathbf{1}[a_i \geq \tau_q], \qquad c_q = \frac{\sum_{i=1}^{n_c} b_{i,q}\, g_i^\top v_0}{\sum_{i=1}^{n_c} b_{i,q}}, \qquad v_q = \begin{cases} v_0, & c_q < 0, \\ -v_0, & c_q > 0. \end{cases} \tag{9}$$

When $c_q = 0$, we choose the sign that makes the component with the largest absolute value positive.

For each candidate gate, set $\tau = \tau_q$ and $v = v_q$. At position t, we modify the hidden state h_t with intervention strength $\kappa \geq 0$:

$$\widetilde{h}_t = h_t - \kappa\, m(h_t;\tau)\, a(h_t)\, v. \tag{10}$$

A state outside the gate remains unchanged. For an admitted state, the update has magnitude $\kappa a(h_t)$ because $\|v\|_2 = 1$. The chosen sign preserves the quadratic objective and maximizes the predicted average first-order NLL increase over the two orientations. This estimate treats each unmodified state as perturbed separately and does not guarantee an increase in the loss of a full sequence. Changing κ only scales this local estimate, so the sign need not be recalibrated.

For each q, we fix (τ_q, v_q) and scan strengths from a prespecified finite set $\mathcal{K} \subseteq [0, \infty)$ on separate calibration validation data. We use teacher forcing to measure forget NLL L_f^{val} and the retain NLL increase ΔL_r relative to the original model. We also compute retain KL D_r from the original predictive distributions to the intervened ones. Each measure averages over all target tokens in its subset. We first require $D_r \le \epsilon_{\mathrm{KL}}$ and $\Delta L_r \le \epsilon_{\mathrm{NLL}}$, where the fixed budgets ϵ_{KL} and ϵ_{NLL} bound the allowed divergence and NLL increase, respectively. Candidates that pass these checks are then evaluated by free generation against a prespecified retain BLEU budget. Let $\mathcal{F} \subseteq \mathcal{Q} \times \mathcal{K}$ contain the candidates satisfying all three constraints. If $\mathcal{F} \neq \varnothing$, we select the candidate with the highest forget validation NLL:

$$(q^\star, \kappa^\star) \in \arg\max_{(q,\kappa)\in\mathcal{F}} L_f^{\mathrm{val}}(q, \kappa). \tag{11}$$

The selected $\tau = \tau_{q^\star}$, $v = v_{q^\star}$, and $\kappa = \kappa^\star$ remain fixed during prompt processing and cached decoding. Deployment stores $(w, s_{\mathrm{ref}}, v, \tau, \kappa)$ and leaves model and adapter weights unchanged. No gradients are needed during inference, and the update adds $O(d)$ operations per position, where d is the hidden dimension. Retain data inform the detector and reference score as well as the calibration constraints. Section 5 evaluates utility preservation. Appendix B.3 provides calibration details, while Appendix B.4 derives the update magnitude and local loss response.

5 Experiments

We evaluate target code reproduction and code utility, then test access under changed prompts and sampling, intervention strength, components, languages, and related code.

5.1 Experimental protocol

Models and data. We use Qwen2.5-Coder-7B and CodeLlama-7B (Hui et al., 2024; Rozière et al., 2023) on CodeSearchNet (CSN) and The Stack (Husain et al., 2019; Kocetkov et al., 2023), forming QwenCSN, QwenStack, CLCSN, and CLStack. Each corpus contains 300 Python forget-training and 8,280 retain-training examples. Learned and retain-only models use the same LoRA initialization and training recipe, except that retain-only training excludes forget slots. Appendices A.1 and A.2 provide data construction and model adaptation details.

Intervention and baselines. UNBIND uses $\kappa = 20$ and constructs the direction using only forget states and gradients. We scan all 28 Qwen layers and 32 CodeLlama layers, selecting layers 23, 24, 29, and 28 for the four settings. We compare against fourteen baselines: GA, GradDiff, NPO-KL, SimNPO-KL, DPO, FLAT, PROD, GSS, RMU, ASU-KL, Task Vector, CodeEraser, Divergence Decoding (DD), and Conformal Unlearning (CU). Layer configurations are described in Appendix A.2. UNBIND construction and calibration and baseline implementations are detailed in Appendices B and C.

Metrics. F-BLEU measures lexical reproduction of forget-set continuations, with lower values indicating better forgetting (Papineni et al., 2002). Utility metrics include retain perplexity (PPL) from mean token NLL, retain BLEU (R-BLEU) on examples with valid tokenizer continuations, and HumanEval+ (HE+)/MBPP+ counts of problems passing both base and extended tests (Chen et al., 2021; Austin et al., 2021; Liu et al., 2023). Generation uses one greedy completion per example, BF16, a 128-token cap, and batch size four. The Forgetting–Utility Harmonic Score (FU-H) combines forgetting and utility:

$$\mathrm{FR} = \max\left(0, 1 - \frac{F}{F_{\mathrm{full}}}\right), \quad \mathrm{UR} = \left[\prod_{j=1}^{4} \min(1, r_j)\right]^{1/4}, \quad \text{FU-H} = 100\frac{2\mathrm{FR}\,\mathrm{UR}}{\mathrm{FR} + \mathrm{UR}}. \tag{12}$$

Here F is F-BLEU, "full" denotes the original model, and r_j are $\mathrm{PPL}_{\mathrm{full}}/\mathrm{PPL}$, $\text{R-BLEU}/\text{R-BLEU}_{\mathrm{full}}$, $\mathrm{HE}+/\mathrm{HE}+_{\mathrm{full}}$, and $\mathrm{MBPP}+/\mathrm{MBPP}+_{\mathrm{full}}$. Capping ratios at one prevents utility gains from offsetting losses. We set FU-H to zero when $\mathrm{FR} = \mathrm{UR} = 0$. See Appendix A.3 for evaluation details.

5.2 SUPPRESSION AND UTILITY ACROSS MODELS

Tables 1 and 2 summarize the results. Across all four settings, UNBIND reduces F-BLEU by 97.3–99.1% and achieves the highest FU-H, ranging from 98.57 to 98.78. Relative to M_{full}, retain PPL increases by at most 1.6%, R-BLEU changes by at most 0.0029, and HE+ and MBPP+ pass counts differ by at most two and six. Low F-BLEU alone, however, does not imply selective forgetting: on CL-CSN, GSS and GradDiff achieve even lower F-BLEU than UNBIND, but both reduce HE+ and MBPP+ to zero. On QwenStack, UNBIND reaches 0.0042 F-BLEU versus 0.0062 for GSS while retaining 91/236 HE+/MBPP+ passes versus 58/15.

Table 1: Qwen results. HE+/MBPP+: pass counts out of 164/378. Colors mark **first**, second, and *third* distinct values; raw metrics use displayed precision, FU-H unrounded scores.

	CodeSearchNet						The Stack					
Method	F-BLEU↓	R-PPL↓	R-BLEU↑	HE+↑	MBPP+↑	**FU-H↑**	F-BLEU↓	R-PPL↓	R-BLEU↑	HE+↑	MBPP+↑	**FU-H↑**
M_{full}	0.4847	3.0046	0.3082	78	225	0.00	0.1531	2.0861	0.3874	91	233	0.00
M_{retrain}	0.0907	2.9724	0.3060	71	227	88.66	0.0872	2.0818	0.3907	87	233	59.99
M_{base}	0.0847	4.1574	0.2543	121	246	85.12	0.0850	2.1894	0.3702	121	246	61.13
GA	0.3031	2.9654	*0.3100*	*79*	225	54.52	0.0754	2.1049	0.3762	89	**254**	67.02
GradDiff	**0.0041**	26209.519	0.0676	0	0	0.00	0.0613	2.1513	0.3644	*91*	242	*74.35*
NPO-KL	0.2135	2.9600	0.3025	*79*	228	71.63	0.1274	2.0771	0.3856	88	239	28.76
SimNPO-KL	0.0984	2.9833	0.3005	*79*	220	88.23	0.0979	*2.0768*	0.3838	89	248	52.89
DPO	0.2447	*2.9604*	**0.3104**	**83**	225	66.23	0.1361	2.0786	0.3859	**93**	*243*	20.05
FLAT	0.0761	3.2647	0.2795	43	200	*82.08*	0.0877	2.1191	0.3663	86	227	59.17
PROD	0.2302	3.0215	0.3048	64	212	67.22	0.1374	2.0864	*0.3863*	88	233	18.60
Task Vector	0.2495	2.9929	0.2944	74	**231**	64.81	0.1284	2.0754	0.3828	86	241	27.73
RMU	0.4810	3.0063	0.3067	76	221	1.50	0.1514	2.0862	**0.3886**	89	231	2.26
ASU-KL	0.2029	3.0790	0.2925	69	215	71.87	0.1075	2.1074	0.3862	86	232	45.72
GSS	0.0920	3.5534	0.2366	72	139	79.48	0.0062	6.2591	0.2689	58	15	47.11
DD	0.0883	3.5559	0.2568	45	167	77.75	*0.0385*	2.4478	0.3202	68	151	75.63
CodeEraser	0.3872	2.9649	0.3071	82	*227*	33.49	0.1494	2.0825	0.3855	92	236	4.72
CU	*0.0378*	**2.2191**	0.2352	69	53	74.96	0.0547	**1.8824**	0.3413	80	73	67.10
UNBIND	0.0042	3.0345	0.3101	76	219	**98.78**	**0.0042**	2.0943	0.3883	*91*	236	**98.57**

The reference models clarify the effect of adaptation. F-BLEU for M_{full} is 1.8–5.7 times that of M_{base}, whereas M_{retrain} remains close to the base level. UNBIND achieves 94.4–95.9% lower F-BLEU than M_{base} and lower F-BLEU than M_{retrain} in every setting, suppressing reproduction beyond returning to either reference level. It also maintains lower retain PPL and higher retain BLEU than M_{base} in three settings. These comparisons establish suppression below the pre-adaptation reproduction level, but base-model F-BLEU alone does not establish that the targets were memorized during pretraining.

Table 2: CodeLlama results with selected baselines. Metrics and ranking conventions follow Table 1; rankings include only the displayed non-reference methods. Complete results appear in Table 7.

	CodeSearchNet						The Stack					
Method	F-BLEU↓	R-PPL↓	R-BLEU↑	HE+↑	MBPP+↑	**FU-H↑**	F-BLEU↓	R-PPL↓	R-BLEU↑	HE+↑	MBPP+↑	**FU-H↑**
M_{full}	0.4600	2.2521	0.3949	38	125	0.00	0.1434	1.7330	0.4736	36	163	0.00
M_{retrain}	0.0894	2.2526	0.3940	40	114	88.30	0.0685	1.7320	0.4754	36	162	68.56
M_{base}	0.0963	2.6220	0.3474	41	158	85.57	0.0651	1.7160	0.4732	41	158	70.45
SimNPO-KL	0.1330	2.2393	0.3802	*33*	77	77.31	0.0834	**1.7224**	0.4817	**40**	*161*	58.91
FLAT	*0.0301*	2.4337	0.3151	16	46	71.70	0.0036	1.7331	0.4769	35	154	97.68
PROD	0.3073	**2.2372**	*0.3907*	30	*107*	48.56	0.1244	*1.7270*	0.4752	38	154	23.31
Task Vector	0.2547	*2.2428*	0.3937	**39**	108	61.01	0.1045	1.7230	*0.4772*	38	162	42.65
GSS	**0.000014**	624186.8588	0.0003	0	0	0.00	0.0994	1.7737	**0.4830**	33	162	46.66
DD	0.1143	2.6876	0.3452	24	63	*72.21*	*0.0560*	2.0080	0.4181	24	100	*67.10*
UNBIND	0.0054	2.2588	**0.3978**	36	**129**	**98.71**	**0.0027**	1.7599	0.4707	*36*	**163**	**98.77**

5.3 STRENGTH-DEPENDENT TRADEOFFS

We evaluate UNBIND and the baselines at different strengths across all four settings, with Figure 3 showing the resulting tradeoffs between target reproduction and MBPP+. On three settings, UNBIND keeps MBPP+ near its original level as F-BLEU decreases. In contrast, several baselines

achieve low reproduction only with substantial utility losses. These curves show that UNBIND provides a more favorable balance between forgetting and utility across multiple strengths.

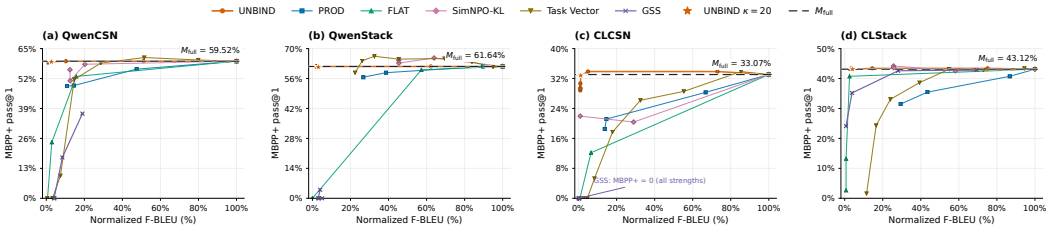
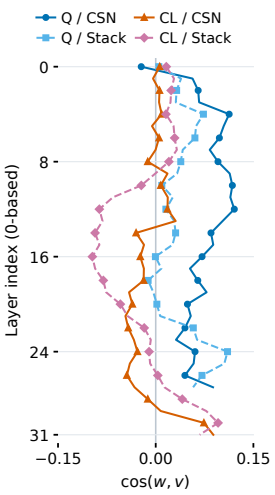


Figure 3: MBPP+ pass@1 (%, denominator 378) versus normalized F-BLEU ($100F/F_{\text{full}}$). Dashed lines mark original utility; stars mark UNBIND at $\kappa = 20$ with the chosen layer configurations.

5.4 What the detector and output direction contribute

Figure 4(a) shows weak alignment between detector w and output direction v across layers, while Figure 4(b) examines their respective roles.



(b) Components across four settings

	F-BLEU ↓	R-PPL ↓	R-BLEU ↑	HE+ ↑	MBPP+ ↑	FU-H ↑
QwenCSN						
M_{full}	0.4847	3.00	0.3082	78	225	0.00
$(\Delta\mu, v)$	0.007376	35.66	0.1378	21	37	33.54
(w, w)	0.040911	5.94	0.1855	30	67	58.71
Random	0.071200	3.26	0.2627	58	195	84.82
All-token, v	**0.003175**	76140.45	0.0039	0	0	0.00
All-token, Random	0.041790	5.11	0.1487	48	169	72.56
UNBIND (w, v)	0.004225	**3.03**	**0.3101**	**76**	**219**	**98.78**
QwenStack						
M_{full}	0.1531	2.09	0.3874	91	233	0.00
$(\Delta\mu, v)$	0.006680	10.42	0.2717	68	67	58.06
(w, w)	0.048648	3.54	0.2507	71	213	70.17
Random	0.004561	64.28	0.2041	59	198	47.16
All-token, v	**0.001258**	106.64	0.0076	0	0	0.00
All-token, Random	0.026560	4.04	0.1666	54	136	64.33
UNBIND (w, v)	0.004182	**2.09**	**0.3883**	**91**	**236**	**98.57**
CLCSN						
M_{full}	0.4600	2.25	0.3949	38	125	0.00
$(\Delta\mu, v)$	0.005899	219.88	0.2614	23	99	38.45
(w, w)	0.005494	10.73	0.1430	27	103	62.71
Random	**0.000925**	163.39	0.2123	8	46	26.80
All-token, v	0.002909	50.82	0.0135	2	0	0.00
All-token, Random	0.086563	2.88	0.2960	19	27	62.02
UNBIND (w, v)	0.005351	**2.26**	**0.3978**	**36**	**129**	**98.71**
CLStack						
M_{full}	0.1434	1.73	0.4736	36	163	0.00
$(\Delta\mu, v)$	**0.001355**	22.47	0.2660	18	105	51.03
(w, w)	0.085374	**1.73**	**0.4757**	**36**	**163**	57.61
Random	0.058321	1.81	0.4704	35	162	73.90
All-token, v	0.004761	10.15	0.0440	4	37	24.68
All-token, Random	0.011394	2.69	0.2971	21	110	74.92
UNBIND (w, v)	0.002741	1.76	0.4707	**36**	**163**	**98.77**

In-cell bars: normalized scores, longer is better

Figure 4: (a) Layerwise $\cos(w, v)$; Q/CL denote Qwen/Code Llama. (b) Four-setting ablations; pairs denote detector/output direction. Cells show raw values; bars show normalized scores, with longer bars indicating better results (Equation 12). Bold marks best intervention values, including ties. HE+/MBPP+ count passes out of 164/378.

Replacing the Fisher detector with mean difference lowers F-BLEU but raises retain PPL to 10.42–219.88, while reusing w as the output direction yields higher F-BLEU than UNBIND in all four settings. Low reproduction alone is also insufficient: on CLCSN, the Random ablation reaches F-BLEU 0.000925 but raises retain PPL to 163.39. Applying v at every position reduces MBPP+ to zero in three settings and 37 in CLStack, while all-token random updates achieve only 62.02–74.92 FU-H. In contrast, UNBIND achieves 98.57–98.78 FU-H while remaining within 2 HE+ and 6 MBPP+ passes of M_{full}. These results support separating detection from suppression and applying the output update selectively with activation-dependent magnitudes.

5.5 Target reproduction and multilingual performance

We evaluate target reproduction under six fixed conditions: the original prompt, 16-, 32-, or 64-token target-prefix injection, instruction rephrasing, and code fences. Each condition uses one greedy and four sampled completions, capped at 256 new tokens. We report the number of targets with $\geq 50/\geq 100$-token contiguous exact spans and the mean best recovery $\overline{R}$, defined as the maximum exact-span-to-withheld-suffix ratio across attempts, averaged over 300 targets. We also evaluate Python,

Java, and JavaScript forgetting tasks on both model families, with utility measured by retain PPL, R-BLEU, and Python HE+/MBPP+. Detailed protocols and CodeLlama results are in Appendices E and F.

Table 3: Target reproduction and multilingual results for Qwen2.5-Coder-7B. Counts report targets with $\geq 50/\geq 100$-token exact spans out of 300; mean best recovery $\overline{R}$, FR, and UR are percentages.

	Target reproduction ↓				Multilingual: forget language								
	CSN		Stack		Python			Java			JavaScript		
Method	Counts	$\overline{R}$	Counts	$\overline{R}$	FR↑	UR↑	FU-H↑	FR↑	UR↑	FU-H↑	FR↑	UR↑	FU-H↑
M_{full}	201/65	79.71	188/93	33.46	0.00	100.00	0.00	0.00	100.00	0.00	0.00	100.00	0.00
M_{retrain}	11/1	22.46	50/15	7.96	80.84	98.82	88.93	62.11	97.76	75.96	64.45	100.00	78.38
PROD	63/13	60.00	144/67	16.27	51.75	90.11	65.74	76.30	90.87	82.95	69.00	95.79	80.22
GSS	61/14	53.81	13/**0**	3.31	58.95	97.98	73.61	41.73	91.13	57.25	34.80	95.58	51.02
SimNPO-KL	7/2	25.00	77/27	10.89	81.46	98.10	89.01	99.15	47.65	64.37	97.96	62.95	76.65
FLAT	9/3	22.68	59/23	8.49	84.77	84.26	84.51	**99.33**	5.75	10.87	**99.08**	7.03	13.13
Task Vector	62/13	56.55	112/47	12.62	48.63	97.19	64.82	45.22	95.18	61.31	40.77	**99.70**	57.88
UNBIND	**0/0**	**6.45**	**1/0**	**1.30**	**99.13**	**99.93**	**98.78**	97.08	**99.85**	**98.45**	98.81	99.33	**99.07**

Under this access budget, UNBIND leaves no ≥ 100-token matches and only 0/1 targets with ≥ 50-token matches on QwenCSN/QwenStack, with the lowest $\overline{R}$ of 6.45%/1.30%. Across both model families and all three languages, it maintains FR above 97% and achieves the highest FU-H in all six settings, showing consistent forgetting with little utility loss.

5.6 Retention on related code

We measure mean source NLL on retain code in its original format across four groups: same-language neighbors, cross-language neighbors, format-matched code, and ordinary retain code. To identify related retain code, we use UniXcoder (Guo et al., 2022) to retrieve candidates similar to the forget code in embedding space. We then review these candidates for semantic relatedness, checking whether they implement similar functionality or computational logic. Group sizes are summarized in Appendix A.1.

Table 4: Related-code retention on Qwen2.5-Coder-7B. Cells report F-BLEU or mean source NLL; superscripts give $100(x/x_{\text{full}}-1)$ (%). Ranking follows Tables 1–2.

	CodeSearchNet					**The Stack**				
	Forget	Retained utility (source NLL)				Forget	Retained utility (source NLL)			
Method	F-BLEU ↓	Same lang. ↓	Cross lang. ↓	Format ↓	Ordinary ↓	F-BLEU ↓	Same lang. ↓	Cross lang. ↓	Format ↓	Ordinary ↓
M_{full}	0.484702	0.7430	0.6728	0.8559	0.6929	0.153138	0.9404	0.8373	0.7796	0.7858
FLAT	0.076068$^{\downarrow 84.31}$	0.9634$^{\uparrow 29.66}$	0.7844$^{\uparrow 16.59}$	1.1171$^{\uparrow 30.52}$	0.8569$^{\uparrow 23.67}$	*0.087710*$^{\downarrow 42.72}$	0.9846$^{\uparrow 4.70}$	0.8664$^{\uparrow 3.48}$	0.8156$^{\uparrow 4.62}$	0.8103$^{\uparrow 3.12}$
PROD	0.230181$^{\downarrow 52.51}$	*0.7888*$^{\uparrow 6.16}$	*0.6971*$^{\uparrow 3.61}$	*0.9158*$^{\uparrow 7.00}$	*0.7299*$^{\uparrow 5.34}$	0.137420$^{\downarrow 10.26}$	0.9503$^{\uparrow 1.05}$	**0.8403**$^{\uparrow 0.36}$	0.7854$^{\uparrow 0.74}$	0.7886$^{\uparrow 0.36}$
SimNPO-KL	0.09843$^{\downarrow 79.69}$	0.81680$^{\uparrow 9.93}$	0.71245$^{\uparrow 5.89}$	0.94280$^{\uparrow 10.15}$	0.74802$^{\uparrow 7.95}$	0.09791$^{\downarrow 36.06}$	0.95822$^{\uparrow 1.89}$	0.84386$^{\uparrow 0.78}$	0.79081$^{\uparrow 1.44}$	0.79263$^{\uparrow 0.87}$
Task Vector	0.249535$^{\downarrow 48.52}$	0.7790$^{\uparrow 4.85}$	0.6875$^{\uparrow 2.18}$	0.8919$^{\uparrow 4.21}$	0.7125$^{\uparrow 2.83}$	0.128422$^{\downarrow 16.14}$	*0.9532*$^{\uparrow 1.36}$	*0.8425*$^{\uparrow 0.62}$	*0.7860*$^{\uparrow 0.82}$	*0.7913*$^{\uparrow 0.70}$
GSS	*0.09204*$^{\downarrow 81.01}$	0.84599$^{\uparrow 13.86}$	0.75151$^{\uparrow 11.70}$	0.94035$^{\uparrow 9.87}$	0.77645$^{\uparrow 12.06}$	0.00620$^{\downarrow 95.95}$	3.12960$^{\uparrow 232.79}$	2.13220$^{\uparrow 154.65}$	2.02275$^{\uparrow 159.46}$	1.76969$^{\uparrow 125.21}$
UNBIND	**0.004237**$^{\downarrow 99.13}$	**0.7442**$^{\uparrow 0.16}$	**0.6728**$^{\approx 0}$	**0.8559**$^{\approx 0}$	**0.6930**$^{\uparrow 0.01}$	**0.004219**$^{\downarrow 97.24}$	**0.9404**$^{\approx 0}$	0.8415$^{\uparrow 0.50}$	**0.7796**$^{\approx 0}$	**0.7880**$^{\uparrow 0.28}$

Table 4 shows that UNBIND reduces F-BLEU by 99.13% on QwenCSN and 97.24% on QwenStack, while limiting NLL increases across the four retain groups to 0.16% and 0.50%. These results show strong suppression of target code reproduction with little impact on related code. On CLStack, same-language and cross-language NLL increase by 2.3% and 2.5%, with respective 95% intervals of 0.63%–4.3% and 0.32%–5.9%.

6 Conclusion

We introduce UNBIND, which reframes code unlearning as selective representation intervention rather than parameter updating. Its key idea is to separate identifying target-related hidden states from determining how those states should be changed, and to apply the resulting intervention only when needed. This design enables strong forgetting while keeping model weights fixed and largely preserving general code capability. Across two models and two corpora, UNBIND reduces target reproduction by 97–99%, achieves the strongest joint forgetting–utility performance among 14 baselines, and nearly eliminates long exact-span recovery under repeated extraction attempts. Multilingual and related-code evaluations further show that this suppression remains targeted rather than broadly degrading useful code representations.

AI USE STATEMENT

Generative AI tools assisted literature discovery, organization of the method and experimental evidence, manuscript writing, translation, and preparation of scripts for tables and figures. Numerical observations were taken from existing experiment records; manuscript preparation did not generate new model-evaluation results. The mathematical construction was checked against the method description, and displayed tables were checked against the reported observations. AI assistance also informed the presentation of design rationale and result interpretation. The authors are responsible for the final scientific claims, references, and artifacts.

ETHICS STATEMENT

This work studies suppression of designated code continuations using code corpora and programming benchmarks. Dataset use and any subsequent release of code examples must respect the applicable licenses and access restrictions.

REPRODUCIBILITY STATEMENT

The main text defines the intervention and evaluation protocol. Appendices A, B, and C provide data and adaptation settings, UNBIND construction and calibration, and baseline implementations, respectively. Appendices E and F detail the target-reproduction and multilingual evaluations. Our code and datasets are available at `https://anonymous.4open.science/r/UNBIND-75D0`.

REFERENCES

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/f545448535dfde4f9786555403ab7c49-Abstract-Conference.html.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pp. 141–159. IEEE, 2021. doi: 10.1109/SP40001.2021.00019. URL https://doi.org/10.1109/SP40001.2021.00019.

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pp. 463–480. IEEE Computer Society, 2015. doi: 10.1109/SP.2015.35. URL https://doi.org/10.1109/SP.2015.35.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B. Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In Michael D. Bailey and Rachel Greenstadt (eds.), *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pp. 2633–2650. USENIX Association, 2021. URL https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramèr, and Chiyuan Zhang. Quantifying memorization across neural language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=TatRHT_1cK.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Xunlei Chen, Qinghui Gong, Ruini Xue, Yaodong Hu, Tian Lan, and Wenhong Tian. St^{2}u: Stateful test-time unlearning via restricted knowledge boundary control. *CoRR*, abs/2608.23034, 2026. doi: 10.48550/ARXIV.2608.23034. URL https://doi.org/10.48550/arXiv.2608.23034.

Somnath Basu Roy Chowdhury, Rahul Kidambi, Avinava Dubey, David Wang, Gökhan Mergen, Amr Ahmed, and Aranyak Mehta. Inference-time unlearning using conformal prediction. *CoRR*, abs/2602.03787, 2026. doi: 10.48550/ARXIV.2602.03787. URL https://doi.org/10.48550/arXiv.2602.03787.

Zhaoyang Chu, Yao Wan, Zhikun Zhang, Di Wang, Zhou Yang, Hongyu Zhang, Pan Zhou, Xuanhua Shi, Hai Jin, and David Lo. Scrub it out! erasing sensitive memorization in code language models via machine unlearning. *CoRR*, abs/2509.13755, 2025. doi: 10.48550/ARXIV.2509.13755. URL https://doi.org/10.48550/arXiv.2509.13755.

Chongyu Fan, Jiancheng Liu, Licong Lin, Jinghan Jia, Ruiqi Zhang, Song Mei, and Sijia Liu. Simplicity prevails: Rethinking negative preference optimization for LLM unlearning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/02443e4e008231e0af0855f3cc70ed17-Abstract-Conference.html.

Chuan Guo, Tom Goldstein, Awni Y. Hannun, and Laurens van der Maaten. Certified data removal from machine learning models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3832–3842. PMLR, 2020. URL http://proceedings.mlr.press/v119/guo20c.html.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pp. 7212–7225. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.499. URL https://doi.org/10.18653/v1/2022.acl-long.499.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report. *CoRR*, abs/2409.12186, 2024. doi: 10.48550/ARXIV.2409.12186. URL https://doi.org/10.48550/arXiv.2409.12186.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *CoRR*, abs/1909.09436, 2019. URL http://arxiv.org/abs/1909.09436.

Xue Jiang, Yihong Dong, Huangzhao Zhang, Tangxinyu Wang, Zheng Fang, Yingwei Ma, Rongyu Cao, Binhua Li, Zhi Jin, Wenpin Jiao, Yongbin Li, and Ge Li. Large language model unlearning for source code. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 31346–31355. AAAI Press, 2026. doi: 10.1609/AAAI.V40I37.40398. URL https://doi.org/10.1609/aaai.v40i37.40398.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 TB of permissively licensed source code. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=pxpbTdUEpD.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pp. 8424–8445. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.577. URL https://doi.org/10.18653/v1/2022.acl-long.577.

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

Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D. Li, Ann-Kathrin Dombrowski, Shashwat Goel, Gabriel Mukobi, Nathan Helm-Burger, Rassin Lababidi, Lennart Justen, Andrew B. Liu, Michael Chen, Isabelle Barrass, Oliver Zhang, Xiaoyuan Zhu, Rishub Tamirisa, Bhrugu Bharathi, Ariel Herbert-Voss, Cort B. Breuer, Andy Zou, Mantas Mazeika, Zifan Wang, Palash Oswal, Weiran Lin, Adam A. Hunt, Justin Tienken-Harder, Kevin Y. Shih, Kemper Talley, John Guan, Ian Steneker, David Campbell, Brad Jokubaitis, Steven Basart, Stephen Fitz, Ponnurangam Kumaraguru, Kallol Krishna Karmakar, Uday Kiran Tupakula, Vijay Varadharajan, Yan Shoshitaishvili, Jimmy Ba, Kevin M. Esvelt, Alexandr Wang, and Dan Hendrycks. The WMDP benchmark: Measuring and reducing malicious use with unlearning. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28525–28550. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/li24bc.html.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason T. Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries.

Starcoder: may the source be with you! *Trans. Mach. Learn. Res.*, 2023, 2023b. URL https://openreview.net/forum?id=KoFOg41haE.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C. Lipton, and J. Zico Kolter. TOFU: A task of fictitious unlearning for llms. *CoRR*, abs/2401.06121, 2024. doi: 10.48550/ARXIV.2401.06121. URL https://doi.org/10.48550/arXiv.2401.06121.

Humzah Merchant and Bradford Levy. Divergence decoding: Inference-time unlearning via auxiliary models. *CoRR*, abs/2605.31293, 2026. doi: 10.48550/ARXIV.2605.31293. URL https://doi.org/10.48550/arXiv.2605.31293.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*, pp. 311–318. ACL, 2002. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040/.

Martin Pawelczyk, Seth Neel, and Himabindu Lakkaraju. In-context unlearning: Language models as few-shot unlearners. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 40034–40050. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/pawelczyk24a.html.

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. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering llama 2 via contrastive activation addition. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 15504–15522. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.828. URL https://doi.org/10.18653/v1/2024.acl-long.828.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023. doi: 10.48550/ARXIV.2308.12950. URL https://doi.org/10.48550/arXiv.2308.12950.

Zhengyang Shan, Xu Qian, Jiayun Xin, Minghui Xu, Yue Zhang, Zhen Yang, Hao Wu, and Xiuzhen Cheng. SAGE: signal-amplified guided embeddings for llm-based vulnerability detection. *CoRR*, abs/2604.19031, 2026. doi: 10.48550/ARXIV.2604.19031. URL https://doi.org/10.48550/arXiv.2604.19031.

Alexander Matt Turner, Lisa Thiergart, David Udell, Gavin Leech, Ulisse Mini, and Monte MacDiarmid. Activation addition: Steering language models without optimization. *CoRR*, abs/2308.10248, 2023. doi: 10.48550/ARXIV.2308.10248. URL https://doi.org/10.48550/arXiv.2308.10248.

Yaxuan Wang, Jiaheng Wei, Chris Yuhao Liu, Jinlong Pang, Quan Liu, Ankit Shah, Yujia Bao, Yang Liu, and Wei Wei. LLM unlearning via loss adjustment with only forget data. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=6ESRicalFE.

Saleh Zare Zade, Xiangyu Zhou, Sijia Liu, and Dongxiao Zhu. Attention smoothing is all you need for unlearning. *CoRR*, abs/2603.01285, 2026. doi: 10.48550/ARXIV.2603.01285. URL https://doi.org/10.48550/arXiv.2603.01285.

Ruiqi Zhang, Licong Lin, Yu Bai, and Song Mei. Negative preference optimization: From catastrophic collapse to effective unlearning. *CoRR*, abs/2404.05868, 2024. doi: 10.48550/ARXIV.2404.05868. URL https://doi.org/10.48550/arXiv.2404.05868.

Xuanqi Zhang, Haoyang Shang, and Xiaoxiao Li. GSS: gated subspace steering for selective memorization mitigation in llms. *CoRR*, abs/2602.08901, 2026. doi: 10.48550/ARXIV.2602.08901. URL https://doi.org/10.48550/arXiv.2602.08901.

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

A Experimental Protocol

A.1 Data Construction and Splits

We use the frozen seed-0 CodeSearchNet (CSN) and The Stack v1 9,600-example manifests. Each corpus contains 300/50/50 forget training/validation/test examples and 8,280/460/460 retain examples. The main forget targets are Python code. Retain data cover Python, Go, Java, JavaScript, PHP, and Ruby for CSN, and Python, C++, Go, Java, JavaScript, and Rust for The Stack. Existing source records, ordering, and split assignments are reused without resampling.

Construction partitions are assigned by source row, keeping all windows from the same row together. The 300 forget rows are first split into 240 fitting rows and 60 calibration rows with seed 0. The latter are further divided into 50 calibration-fit rows and 10 calibration-validation rows. The calibration-fit rows are used to construct the candidate quantile thresholds τ_q and orient the corresponding output directions v_q, while the calibration-validation rows are used for constrained (q, κ) selection. A language-proportional sample of 600 retain-training rows is partitioned within language into fitting and calibration subsets, yielding 480/120 rows for CSN and Stack. The retain calibration rows provide the retain-side measurements used to enforce the KL, NLL, and BLEU constraints during (q, κ) selection.

Calibration scope. For the reported operating point, the calibration sweep selects $q^\star = 0.25$ and $\kappa^\star = 20$. For each candidate quantile q, the threshold τ_q is computed by linear interpolation over positive forget-calibration activations, with repeated values retained. The sign of the output direction is oriented on the same calibration-training rows using gate-weighted target-loss gradient projections. The resulting (τ_q, v_q) pairs are then combined with candidate strengths $\kappa \in \mathcal{K}$ and evaluated on a separate calibration-validation partition. Candidates must satisfy the prescribed retain KL, retain-NLL, and retain-BLEU budgets, after which the feasible pair with the highest forget validation NLL is selected. Main F-BLEU evaluates all 300 designated forget-training targets, including calibration examples; it therefore measures reproduction of the designated training targets rather than held-out forgetting generalization.

A.2 Model Adaptation and Reference Models

We use Qwen2.5-Coder-7B-Instruct and CodeLlama-7b-hf with their own tokenizers. M_{base} denotes the checkpoint before adaptation; M_{full} and M_{retrain} start from the same base weights and saved seed-0 LoRA initialization, with the latter excluding forget slots. LoRA uses rank 32, alpha 64, dropout 0, and the `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj` modules. Training uses AdamW with constant learning rate 5×10^{-5}, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, weight decay 0.01, gradient clipping at 1.0, maximum length 2,048, BF16 forward computation, and FP32 trainable master weights.

Each scheduled epoch interleaves 300 forget slots with 2,700 retain slots drawn from a seeded, repeatedly shuffled retain stream. Microbatch size is one; gradients accumulate over 32 scheduled slots (24 in the final batch), with loss normalized by the number of scored tokens. Both models perform 94 updates per epoch. M_{retrain} skips forget slots without replacing them or changing update boundaries. Retain exposure, update counts, and the constant learning rate are consequently matched, although its number of live examples per update is smaller.

The reported paired checkpoints are epoch 8 for Qwen/CSN, epoch 9 for Qwen/Stack, and epoch 10 for both CodeLlama settings, corresponding to 752, 846, and 940 updates. These saved checkpoints are reused in the rerun. Intervention layers are 23, 24, 29, and 28, respectively (zero-based). The latter three layers were fixed from the preceding layer scan; the Qwen/CSN reference was retained. These are not presented as layers selected on an independent calibration-validation set. Adapters remain unmerged during evaluation.

A.3 Evaluation Metrics and Generation Settings

Main continuation metrics. For the main F-BLEU and R-BLEU evaluation, we use one greedy completion per example (`num_beams=1`), BF16, batch size 4, and at most 128 new tokens. Forget prompts and target continuations are read directly from the frozen target manifest, without applying

Table 5: Data roles in the experiments. All partitions follow the frozen seed-0 CSN and The Stack manifests and are assigned by source row.

Purpose	Size	Usage and overlap
Adaptation	300 forget; 8,280 retain	Training pool for the paired reference models.
Detector/output-axis fitting	240 forget; 480 retain	Fitting subsets used to construct the detector and output axis; disjoint from all calibration rows.
Forget calibration-fit	50 forget	Used to construct candidate quantile thresholds τ_q and orient the corresponding output directions v_q.
Forget calibration-validation	10 forget	Held-out forget rows used for constrained (q, κ) selection; disjoint from the 240 fitting and 50 calibration-fit rows.
Retain calibration	120 retain	Language-proportional retain subset used to measure KL, NLL, and BLEU constraints during (q, κ) selection.
Main evaluation	300 forget; 460 retain	Reproduction of the designated forget-training targets; retain likelihoods use retain-test.
Access evaluation	300 targets	Reuses the main forget targets under multiple access conditions.
Multilingual evaluation	300 forget per language; 460 shared retain	Separate Python, Java, and JavaScript forget tasks; shared retain-test data and Python HE+/MBPP+ utility tests.
Related-code evaluation	512 per evaluated setting	Four groups of 128: same-language semantic neighbors, cross-language semantic neighbors, ordinary retain, and format-matched retain.

a chat template. For each retain-test program, we choose the latest tokenizer-offset boundary that leaves at least 32 prefix tokens; only the prefix is left-truncated when needed to satisfy the 2,048-token composed-input limit.

F-BLEU and R-BLEU are arithmetic means of sentence-level BLEU-4, using the evaluated model's tokenizer, uniform n-gram weights, and the standard brevity penalty. The preserved implementation replaces a zero modified precision by $1/\max(2N_n, 1)$, where N_n is the number of hypothesis n-grams; an empty reference or hypothesis scores zero. Generated continuations are compared with the stored reference continuations, excluding prompt text. Scores use the $[0, 1]$ scale and are not corpus-level BLEU.

R-PPL uses all 460 retain-test programs. Each sequence consists of one BOS token (EOS when BOS is unavailable) followed by at most 2,047 native-code tokens. Only native-code target tokens contribute to the loss; the prepended special token and padding positions are masked. We exponentiate the total token NLL divided by the total number of scored tokens, rather than averaging per-example perplexities.

Functional correctness. HE+ and MBPP+ use 164 and 378 tasks from HumanEvalPlus v0.1.10 and MbppPlus v0.2.0, respectively, with stored prompts and one greedy completion of at most 128 new tokens per task. HumanEval completions undergo the preserved indentation repair and top-level truncation, whereas MBPP completions undergo stop-string removal and EvalPlus 0.3.1 sanitization. A task passes only if both the base and extended tests pass; execution failures and timeouts count as failures. The evaluator uses a 10-second outer task limit and a per-input limit of $\max(1\,\mathrm{s}, 4t_{\mathrm{reference}})$.

Other evaluation protocols. Access evaluation uses a separate 256-new-token cap under six conditions: the original prompt, 16/32/64-token target-prefix injections, an instruction rewrite, and a fenced-code prompt. Each applicable condition receives one greedy and four sampled attempts, with temperature 0.8 and top-$p = 0.95$; matching excludes content supplied in the prompt.

Multilingual experiments vary the forget language across Python, Java, and JavaScript while retaining the main utility protocol. Retain utility is measured by PPL and R-BLEU on the shared retain-test set, and functional correctness by Python HE+/MBPP+. The latter use 164 and 378 tasks, respectively, with one greedy completion of at most 128 new tokens per task; a task passes only if both base and extended tests pass. Appendix F provides the language-specific data and reference models. Related-code comparisons use the frozen 512-example roster(s) and the original source-NLL scoring protocol.

FU-H. We compute FU-H from unrounded metrics and the matching M_{full} reference. Forgetting is $\text{FR} = \max(0, 1 - F/F_{\text{full}})$; utility retention is the geometric mean of the four ratios P_{full}/P, R/R_{full}, H/H_{full}, and B/B_{full}, each capped at one. Here P, R, H, B denote R-PPL, R-BLEU, HE+, and MBPP+. Then $\text{FU-H} = 100 \cdot 2\text{FRUR}/(\text{FR} + \text{UR})$, with zero returned when both components vanish. Missing references or zero reference F-BLEU make the score undefined; a zero reference utility component contributes zero in the implementation. Rankings use decreasing unrounded FU-H; identical unrounded scores are ties.

A.4 Implementation Environment and Reproducibility

Main fitting and evaluation use seed 0, frozen base-model and adapter weights, and BF16 model computation. Fresh-fit sufficient statistics and eigendecompositions use FP64. The anonymous release contains the fitting/deployment code and the frozen-input specification. Its CPU validation environment is Python 3.12.3, PyTorch 2.5.1+cu124, Transformers 4.57.6, PEFT 0.19.1, and Accelerate 1.14.0. These are release-validation versions, not a claim that all historical GPU runs used the same environment.

B UNBIND Construction and Calibration

B.1 Hidden-State and Target-Gradient Collection

We attach a forward hook to the output of the selected decoder block, after its final residual addition and before the subsequent decoder block or final model normalization. Let h_t denote this output at position t, and define

$$L_t = -\log p_M(x_{t+1} \mid x_{\le t}), \qquad g_t = \frac{\partial L_t}{\partial h_t}.$$

All extraction uses the frozen adapted model M_{full}, with the UNBIND intervention disabled, in teacher-forcing mode with the KV cache disabled.

The implementation computes an unreduced cross-entropy loss for every selected target token. For each scalar L_t, it calls `torch.autograd.grad` with respect to the captured layer-output tensor and retains only the gradient at predictor position t. Thus, each stored pair is exactly (h_t, g_t) for the same target x_{t+1}. A single forward graph is reused across these per-target vector–Jacobian products. This differs from differentiating the summed sequence loss, whose gradient with respect to h_t can also contain contributions from later target losses through the causal computation graph.

Positions follow the frozen `target_positions` and `predictor_positions`, with each predictor position equal to its target position minus one. Forget extraction scores continuation tokens, excluding prompt tokens as prediction targets; the final prompt state can nevertheless serve as the predictor of the first continuation token. For a retain window of length T, target positions are $1, \ldots, T-1$, with corresponding predictor positions $0, \ldots, T-2$. Extraction processes individual unpadded windows.

Rows of H_f and G_f preserve identical window and target-position order, so row i of the two matrices always refers to the same prediction event. Every selected position contributes once, without per-example reweighting. Window construction and position counts are given in Appendix A.

B.2 Detector and Output-Axis Derivations

Fisher detector. Let

$$\delta = \mu_f - \mu_r, \qquad S = \Sigma_f + \Sigma_r, \qquad C = S + \lambda \frac{\mathrm{tr}(S)}{d} I, \qquad \lambda = 0.01,$$

where each covariance uses the unbiased sample denominator $n-1$. The regularized Fisher criterion is

$$\max_{u \neq 0} \frac{(u^\top \delta)^2}{u^\top C u}.$$

When C is positive definite, writing $z = C^{1/2} u$ gives

$$\frac{(u^\top \delta)^2}{u^\top C u} = \frac{(z^\top C^{-1/2} \delta)^2}{z^\top z} \leq \delta^\top C^{-1} \delta$$

by Cauchy–Schwarz, with equality when $u \propto C^{-1}\delta$. We therefore solve

$$C\widetilde{w} = \delta, \qquad w = \frac{\widetilde{w}}{\|\widetilde{w}\|_2}, \qquad s_{\mathrm{ref}} = w^\top \mu_r.$$

Thus, the detector used in the main text is the normalized regularized Fisher direction, with the mean retain score as its reference.

The implementation accumulates the required moments and solves the linear system in CPU FP64 using `torch.linalg.solve`, without explicitly forming C^{-1}. A nonpositive or nonfinite trace-scaled ridge, or a zero or nonfinite detector norm, raises a fitting error; the implementation does not substitute an arbitrary direction.

Output axis. For the paired forget-state and target-gradient matrices, define

$$A_f = \frac{H_f^\top G_f + G_f^\top H_f}{2 n_f}.$$

Because the two terms are transposes of one another, A_f is symmetric. For any unit vector u,

$$u^\top A_f u = \frac{1}{n_f} \sum_{i=1}^{n_f} (h_i^\top u)(g_i^\top u).$$

This is an uncentered cross-moment: the auxiliary construction depends on the actual state projection $h_i^\top u$ and the corresponding target-loss-gradient projection $g_i^\top u$. Centering the two matrices would instead replace this quantity by

$$u^\top A_f u - (\mu_f^\top u)(\bar{g}_f^\top u), \qquad \bar{g}_f = \frac{1}{n_f} \sum_{i=1}^{n_f} g_i,$$

and would therefore define a different objective.

To derive the output axis, consider the auxiliary positionwise update

$$\Delta h_i = -\eta (h_i^\top u) u, \qquad \eta > 0, \qquad \|u\|_2 = 1.$$

Perturbing each state separately while holding the remaining states fixed, its local first-order target-loss change is

$$\Delta L_i \approx g_i^\top \Delta h_i = -\eta (h_i^\top u)(g_i^\top u).$$

Averaging over the n_f forget fitting positions gives

$$\widehat{\Delta \overline{L}}_{\mathrm{aux}} = -\frac{\eta}{n_f} \sum_{i=1}^{n_f} (h_i^\top u)(g_i^\top u) = -\eta\, u^\top A_f u.$$

Hence, maximizing the predicted auxiliary first-order response over unit axes is equivalent to minimizing $u^\top A_f u$. By the Rayleigh–Ritz theorem,

$$v_0 \in \arg\min_{\|u\|_2 = 1} u^\top A_f u$$

is any unit eigenvector associated with $\lambda_{\min}(A_f)$, and the corresponding predicted response is

$$\widehat{\Delta \overline{L}}_{\text{aux}} = -\eta \lambda_{\min}(A_f).$$

We compute v_0 in FP64 using `torch.linalg.eigh`.

If $\lambda_{\min}(A_f) < 0$, the auxiliary criterion predicts a strictly positive first-order increase in target-token loss along this axis. If $\lambda_{\min}(A_f) \geq 0$, it predicts no strictly positive increase for any unit axis. The implementation nevertheless returns the minimum-eigenvalue axis in either case; the eigenvalue is not used as a feasibility test.

The eigenvector v_0 is an unsigned axis because the quadratic objective is invariant to $v_0 \mapsto -v_0$. Its orientation is therefore determined later from the gated forget calibration states, as described in Section 4.3. The auxiliary update above is used only to construct this axis. It is distinct from the deployed intervention, whose magnitude is determined by the rectified detector activation $a(h)$ and whose application and orientation are controlled by the calibrated gate.

B.3 Threshold and Strength Calibration

Calibration partitions and search space. Construction partitions are assigned by source row, keeping all windows from one row together. The 300 forget-training rows are first split into 240 fitting rows and 60 calibration rows with seed 0. The fitting rows are used to construct the detector and output axis. The remaining 60 forget rows are further partitioned into calibration-training and calibration-validation subsets: calibration-training states determine candidate thresholds and gate-dependent axis orientations, whereas calibration-validation data are used for constrained (q, κ) selection.

A language-proportional sample of 600 retain-training rows is partitioned within language into fitting and calibration subsets, yielding 480/120 rows for CSN and Stack. The fitting rows contribute to detector construction, while the retain-calibration rows provide the retain-side measurements used during candidate selection.

We scan the prespecified grids

$$\mathcal{Q} = \{0.10, 0.25, 0.50, 0.75\}, \qquad \mathcal{K} = \{0, 4, 8, 12, 16, 20, 32\}, \tag{13}$$

giving 28 threshold–strength candidates per fitted layer.

Threshold and orientation construction. Let $\mathcal{A}^+_{\text{cal}}$ contain the positive detector activations from the unmodified forget calibration-training states. For each $q \in \mathcal{Q}$, we compute

$$\tau_q = \text{Quantile}_q\big(\mathcal{A}^+_{\text{cal}}\big), \qquad m(h; \tau_q) = \mathbf{1}[a(h) \geq \tau_q]. \tag{14}$$

Quantiles use linear interpolation with repeated values retained. A candidate is invalid if the corresponding calibration-training activations contain no positive value.

The output-axis objective produces an unsigned unit eigenvector v_0. For each candidate threshold, we orient this axis using the forget calibration-training positions admitted by its gate. Writing $a_i = a(h_i)$ and $g_i = \nabla_{h_i} \ell_i^{\text{NLL}}$, define

$$b_{i,q} = a_i \mathbf{1}[a_i \geq \tau_q], \qquad c_q = \frac{\sum_{i=1}^{n_c} b_{i,q}\, g_i^\top v_0}{\sum_{i=1}^{n_c} b_{i,q}}, \qquad v_q = \begin{cases} v_0, & c_q < 0, \\ -v_0, & c_q > 0. \end{cases} \tag{15}$$

Here n_c denotes the number of prediction positions in the forget calibration-training subset. When $c_q = 0$, we choose the sign that makes the largest-magnitude component of v_q positive; an index tie uses the first index returned by `argmax`. This is the implementation of the orientation rule in Equation 9.

Constrained (q, κ) selection. For each q, the corresponding (τ_q, v_q) is fixed before strength evaluation. Each $\kappa \in \mathcal{K}$ is then evaluated on the separate calibration-validation data using

$$\widetilde{h}_t = h_t - \kappa\, m(h_t; \tau_q)\, a(h_t)\, v_q. \tag{16}$$

Teacher forcing provides the forget validation NLL $L_f^{\mathrm{val}}(q,\kappa)$, the retain NLL increase $\Delta L_r(q,\kappa)$ relative to the unmodified model, and the retain predictive divergence $D_r(q,\kappa)$. We use

$$\epsilon_{\mathrm{KL}} = \epsilon_{\mathrm{NLL}} = \epsilon_{\mathrm{BLEU}} = 0.01. \tag{17}$$

Candidates must satisfy

$$D_r(q,\kappa) \le 0.01, \qquad \Delta L_r(q,\kappa) \le 0.01, \tag{18}$$

and must also satisfy the retain-BLEU preservation budget $\epsilon_{\mathrm{BLEU}} = 0.01$ under free generation.

Let $\mathcal{F} \subseteq \mathcal{Q} \times \mathcal{K}$ denote the candidates satisfying all three retain constraints. When $\mathcal{F} \neq \varnothing$, we select

$$(q^\star, \kappa^\star) \in \arg\max_{(q,\kappa)\in\mathcal{F}} L_f^{\mathrm{val}}(q,\kappa). \tag{19}$$

The resulting $\tau = \tau_{q^\star}$, $v = v_{q^\star}$, and $\kappa = \kappa^\star$ are fixed for subsequent evaluation and inference.

Update magnitude and local loss response. Since v_q has unit norm and $\kappa\, m(h_t;\tau_q)\, a(h_t) \ge 0$, Equation 10 gives

$$\|\widetilde{h}_t - h_t\|_2 = \kappa\, m(h_t;\tau_q)\, a(h_t). \tag{20}$$

Thus, the gate determines whether a state changes, while the magnitude of an admitted update scales with its detector activation.

To justify the orientation rule, consider an unmodified forget calibration-training state under candidate gate q. Its isolated perturbation is

$$\delta h_i = -\kappa b_{i,q} v_q.$$

Holding all other states fixed, its first-order target-token NLL response is $g_i^\top \delta h_i$. Because τ_q is computed from a nonempty collection of positive activations, $\sum_i b_{i,q} > 0$, so the normalization in c_q does not affect its sign. Averaging over the n_c calibration-training positions gives

$$-\frac{\kappa}{n_c}\sum_{i=1}^{n_c} b_{i,q}\, g_i^\top v_q = \frac{\kappa}{n_c}\left|\sum_{i=1}^{n_c} b_{i,q}\, g_i^\top v_0\right| \ge 0. \tag{21}$$

Thus, among the two equivalent orientations v_0 and $-v_0$, the selected sign maximizes the average local first-order NLL increase for the states admitted by gate q, without changing the quadratic output-axis objective. For any $\kappa \ge 0$, changing the strength only scales this local response and therefore does not change the preferred orientation. This calculation concerns isolated perturbations of unmodified calibration states rather than the loss change obtained when interventions are applied throughout a full sequence.

Layer-wise calibration and reported strength. Each layer in the layer scan is fitted and calibrated independently. Thus, detector fitting, output-axis construction, threshold construction, orientation, and the constrained (q,κ) search are repeated for every candidate layer.

B.4 Local Response and Inference Implementation

For a candidate quantile $q \in \mathcal{Q}$, define

$$b_{i,q} = m(h_i;\tau_q)a(h_i) = a_i \mathbf{1}[a_i \ge \tau_q] \ge 0.$$

For $\|v_q\|_2 = 1$, the deployed update at calibration position i is

$$\Delta h_i = -\kappa b_{i,q} v_q, \qquad \|\Delta h_i\|_2 = \kappa b_{i,q}.$$

Let

$$S_q = \sum_{i=1}^{n_c} b_{i,q} g_i^\top v_0.$$

The orientation rule in Section 4.3 chooses $v_q \in \{v_0, -v_0\}$ such that

$$\sum_{i=1}^{n_c} b_{i,q} g_i^\top v_q = -|S_q|,$$

including the zero case under the deterministic tie-breaking rule. Hence the corresponding diagonal local first-order response is

$$\widehat{\Delta \overline{L}}_{\mathrm{local}} = -\frac{\kappa}{n_c} \sum_{i=1}^{n_c} b_{i,q} g_i^\top v_q = \frac{\kappa}{n_c} |S_q| \geq 0.$$

Thus, for a fixed axis v_0 and gate τ_q, the orientation maximizes the predicted diagonal local first-order target-NLL response over the two possible signs. Changing κ only scales this local quantity and therefore does not require reorienting the axis.

This result is local to the unmodified calibration states. It does not establish global optimality of the gated direction or guarantee an increase in actual sequence loss. Simultaneous interventions can introduce cross-position interactions and higher-order effects, and autoregressive generation can move to trajectories different from those used in the local calculation.

The saved vectors are stored in FP32. At inference, detector projections and intervention corrections are computed in FP32, and the correction is cast to the hidden-state dtype before subtraction. The hook is applied to every position returned by the selected block, including prompt positions during prefill and newly processed positions during cached decoding; no separate target-token mask is used. The model's attention mask retains its standard role.

After calibration, $(w, s_{\mathrm{ref}}, v, \tau, \kappa)$ remain fixed throughout inference. No gradients or model-weight updates are required. Each position requires one detector projection, thresholding, and a rank-one correction, giving $O(d)$ additional arithmetic for hidden dimension d.

C Baseline Implementations

C.1 Common Settings

We evaluate the fourteen baselines introduced in Section 5.1 under the same model–corpus settings and evaluation protocol used for UNBIND. The two model families are Qwen2.5-Coder-7B and CodeLlama-7B, evaluated on CodeSearchNet (CSN) and The Stack, forming QwenCSN, QwenStack, CLCSN, and CLStack. Each corpus contains 300 Python forget-training examples and 8,280 retain-training examples, following the data construction described in Appendix A.

Each baseline starts from the corresponding adapted model M_{full} unless stated otherwise. The retain-only model M_{retrain} is used only as an evaluation reference and is not supplied as an input to any baseline. The learned and retain-only reference models use the same LoRA initialization and adaptation recipe, except that the retain-only model excludes the designated forget examples during adaptation.

Data access. GA, DPO, FLAT, PROD, and Task Vector use the forget-training examples only. GradDiff, NPO-KL, ASU-KL, and RMU additionally draw examples from the retain-training pool. SimNPO-KL uses a fixed 600-example multilingual subset of the retain-training data. CodeEraser pairs the 300 forget examples with 300 retain-training examples. GSS uses training-side examples to construct its probe and steering directions. DD constructs its two auxiliary distributions from the forget and retain training sets. CU additionally uses 30 held-out forget-validation examples to calibrate its iteration budget.

The designated forget-training targets are used for unlearning and subsequently evaluated for reproduction. Retain-test and functional-benchmark examples are not used for baseline optimization.

Optimization. Except for RMU, parameter-update baselines optimize the LoRA parameters attached to M_{full}. The LoRA configuration uses rank 32, scaling factor 64, and zero dropout on the attention and MLP projection modules specified in Appendix A.2. The underlying pretrained model weights remain frozen.

Unless stated otherwise, optimization uses AdamW with constant learning rate, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, weight decay 0.01, maximum sequence length 2,048, microbatch size one, and 32 accumulated examples per optimizer update. Forward computation uses BF16 and trainable parameters use FP32. Forget losses score only continuation tokens and mask the supplied prefix. Runs use seed 0 unless otherwise specified; RMU's fixed random control vector uses seed 42.

Algorithm 1 Construction and constrained calibration of UNBIND

Require: Frozen model M, layer ℓ, forget/retain fit data, forget calibration-training data, forget/retain calibration-validation data, candidate sets $\mathcal{Q}, \mathcal{K}$, retain budgets $\epsilon_{\mathrm{KL}}, \epsilon_{\mathrm{NLL}}, \epsilon_{\mathrm{BLEU}}$

Collect paired (H_f, G_f) on forget-fit positions using per-target scalar-loss VJPs
Collect H_r on retain-fit positions
Compute $\mu_f, \mu_r, \Sigma_f, \Sigma_r$ in FP64
$C \leftarrow \Sigma_f + \Sigma_r + \lambda \frac{\mathrm{tr}(\Sigma_f + \Sigma_r)}{d} I$
Solve $C\widetilde{w} = \mu_f - \mu_r$; $w \leftarrow \widetilde{w}/\|\widetilde{w}\|_2$; $s_{\mathrm{ref}} \leftarrow w^\top \mu_r$
$A_f \leftarrow (H_f^\top G_f + G_f^\top H_f)/(2n_f)$
$v_0 \leftarrow$ a unit minimum-eigenvalue eigenvector of A_f
Collect paired calibration-training states and gradients $\{(h_i, g_i)\}_{i=1}^{n_c}$
$a_i \leftarrow \max(w^\top h_i - s_{\mathrm{ref}}, 0)$ for all calibration-training positions
$\mathcal{F} \leftarrow \varnothing$
for $q \in \mathcal{Q}$ **do**
    $\tau_q \leftarrow \mathrm{Quantile}_q(\{a_i : a_i > 0\})$ using linear interpolation with repeated values retained
    $b_{i,q} \leftarrow a_i \mathbf{1}[a_i \geq \tau_q]$
    Orient v_0 using $\sum_i b_{i,q} g_i^\top v_0$ to obtain v_q
    for $\kappa \in \mathcal{K}$ **do**
        Evaluate the intervention (τ_q, v_q, κ) on calibration-validation data
        Compute forget NLL $L_f^{\mathrm{val}}(q, \kappa)$
        Compute retain KL $D_r(q, \kappa)$ and retain-NLL increase $\Delta L_r(q, \kappa)$
        if $D_r(q, \kappa) \leq \epsilon_{\mathrm{KL}}$ **and** $\Delta L_r(q, \kappa) \leq \epsilon_{\mathrm{NLL}}$ **then**
            Evaluate free-generation retain BLEU
            if retain BLEU satisfies ϵ_{BLEU} **then**
                $\mathcal{F} \leftarrow \mathcal{F} \cup \{(q, \kappa)\}$
            end if
        end if
    end for
end for
$(q^\star, \kappa^\star) \leftarrow \arg\max_{(q,\kappa) \in \mathcal{F}} L_f^{\mathrm{val}}(q, \kappa)$
$\tau \leftarrow \tau_{q^\star}$; $v \leftarrow v_{q^\star}$; $\kappa \leftarrow \kappa^\star$
Save $(\ell, w, v, s_{\mathrm{ref}}, \tau, \kappa)$

Algorithm 2 Inference with the calibrated UNBIND module

Require: Saved $(\ell, w, v, s_{\mathrm{ref}}, \tau, \kappa)$

Install an output hook on decoder block ℓ
for each prefill or cached-decoding forward pass **do**
    Receive the block-output tensor H
    $A \leftarrow \max(\mathrm{FP32}(H)w - s_{\mathrm{ref}}, 0)$
    $B \leftarrow A \odot \mathbf{1}[A \geq \tau]$
    $\widetilde{H} \leftarrow H - \mathrm{cast}_H(\kappa B v^\top)$
    Return $\widetilde{H}$, preserving other block outputs
end for
Remove the hook when intervention ends

Executed configurations. Table 6 records the configurations used for the baseline results reported in the paper. Paired entries denote CSN / Stack settings. Method-specific implementation details are given in the following subsections.

Evaluation protocol. The main evaluation follows Section 5.1. Generation uses one greedy completion per example, BF16 model computation, a maximum of 128 new tokens, and batch size four. All methods use the same frozen forget, retain, HumanEval+, and MBPP+ evaluation rosters.

F-BLEU is computed on the designated forget continuations. Retain utility is evaluated using retain perplexity from mean token NLL, retain BLEU on examples with valid tokenizer continuations, and HumanEval+ and MBPP+ pass counts. FU-H is computed exactly as defined in Equation (12).

Table 6: Baseline configurations used in the reported experiments. Paired entries denote CSN / Stack settings. η is the learning rate and S is the number of optimizer updates. Task Vector counts reinforcement updates. Inference-time baselines do not update the backbone or adapter parameters.

Method	Updated component	Configuration
GA	LoRA	$\eta = 5 \times 10^{-6}/3 \times 10^{-5}$; $S = 13/10$.
GradDiff	LoRA	$\eta = 10^{-5}/5 \times 10^{-5}$; $\lambda_R = 1/0.3$; $S = 65/10$.
NPO-KL	LoRA	$\eta = 10^{-5}/2 \times 10^{-5}$; $\beta = 0.1$; $\lambda_R = 1/0.5$; $S = 13/10$.
SimNPO-KL	LoRA	$\eta = 2 \times 10^{-5}$; $\beta = 0.7/4.5$; $\gamma = 0$; $\lambda_R = 1$; $S = 13/10$.
DPO	LoRA	$\eta = 10^{-5}$; $\beta = 0.05$; $S = 13/10$.
FLAT	LoRA	$\eta = 2 \times 10^{-5}/5 \times 10^{-5}$; KL / Total-Variation; $S = 26/10$.
PROD	LoRA	$\eta = 10^{-5}$; top-$p = 0.9$; $\alpha = 0.2$; $S = 13/10$.
RMU	Three full matrices	$\eta = 2 \times 10^{-5}/10^{-5}$; $c = 20/6.5$; $\lambda_R = 100/1200$; $S = 13/13$.
ASU-KL	LoRA	$\eta = 2 \times 10^{-5}$; $\tau = 1.5$; $\lambda_R = 0.03$; $S = 65/65$.
Task Vector	LoRA tensors	Reinforcement $\eta = 5 \times 10^{-5}$; $S = 13/26$; subtraction coefficient $\alpha = 1$.
CodeEraser	LoRA	$\eta = 3 \times 10^{-6}$; $S = 10$; $\gamma = 0.5$; $\lambda = 0.1$; $\alpha = 1$.
GSS	Residual hook	All layers scanned; Selected Layer $23/26$; rank $2/8$; strength $2/1$; variance budget 1.
DD	Decoding logits	Trigram auxiliaries; backoff 0.4; linear correction coefficient 10.
CU	Prompt and selection	30 calibration examples; $\alpha = 0.1$; acceptance score $9/10$; calibrated iteration budget $T = 1$ in the reported experiments.

The strength-dependent experiments in Section 5.3 vary each method's corresponding intervention or update strength while retaining its remaining configuration. The access evaluation in Section 5.5 instead uses its fixed six-condition generation protocol with one greedy and four sampled completions per available condition and a 256-token generation cap.

C.2 Parameter-Update Baselines

Let p_0 denote the frozen distribution of the starting M_{full} model and p_θ the distribution after the baseline update. For a prefix–continuation pair (x, y), define

$$s_\theta(x, y) = \sum_t \log p_\theta(y_t \mid x, y_{<t}), \qquad \bar{s}_\theta(x, y) = \frac{s_\theta(x, y)}{|y|}.$$

Let $\mathcal{N}_F$ and $\mathcal{N}_R$ denote token-averaged negative log-likelihood on forget and retain examples, respectively. We define

$$\mathcal{K}_R = \mathbb{E}_{(x,y)\in D_R, t} D_{\text{KL}}(p_0(\cdot \mid x, y_{<t}) \| p_\theta(\cdot \mid x, y_{<t})), \tag{22}$$

where the expectation is over scored target positions. We use the analogous notation $\mathcal{K}_F$ for forget continuations. All objectives below are minimized.

GA and GradDiff. GA suppresses the designated continuation by minimizing $-\mathcal{N}_F$. GradDiff adds retain language modeling:

$$\mathcal{L}_{\text{GradDiff}} = -\mathcal{N}_F + \lambda_R \mathcal{N}_R. \tag{23}$$

Retain examples are drawn from the fixed retain-training stream.

NPO-KL and SimNPO-KL. Using $\text{sp}(z) = \log(1 + e^z)$, the implemented objectives are

$$\mathcal{L}_{\text{NPO-KL}} = \frac{2}{\beta} \mathbb{E}_F \, \text{sp}(\beta(s_\theta - s_0)) + \lambda_R \mathcal{K}_R, \tag{24}$$

$$\mathcal{L}_{\text{SimNPO-KL}} = \frac{2}{\beta} \mathbb{E}_F \, \text{sp}(\beta \bar{s}_\theta + \gamma) + \lambda_R \mathcal{K}_R. \tag{25}$$

NPO uses sequence log-probability sums and a frozen copy of M_{full} as its forget reference. SimNPO normalizes the forget log-probability by continuation length and does not use a forget-reference term. Its retain KL remains defined relative to M_{full}. The 600 retain examples used by SimNPO-KL are drawn from the training split and shuffled with seed 0.

DPO. For each forget prefix, the rejected response y^- is the designated forget continuation and the chosen response y^+ is the fixed string `# Code removed by request.`. No learned preference model or generated positive response is used. The objective is

$$\mathcal{L}_{\mathrm{DPO}} = -\frac{2}{\beta}\mathbb{E}_F \log \sigma\big(\beta\left[(s_\theta^+ - s_\theta^-) - (s_0^+ - s_0^-)\right]\big). \tag{26}$$

Both responses use sequence log-probability sums. No additional retain loss is used.

FLAT. FLAT uses the same chosen response y^+ and designated forget response y^-. Let $a_\theta(y)$ denote the arithmetic mean of the current model's teacher-forced probabilities assigned to the response tokens. The CSN configuration minimizes

$$\exp\big(a_\theta(y^-) - 1\big) - a_\theta(y^+), \tag{27}$$

while the Stack configuration minimizes

$$\frac{\tanh(a_\theta(y^-)) - \tanh(a_\theta(y^+))}{2}. \tag{28}$$

These correspond to the KL and Total-Variation configurations used in the reported experiments. They operate on mean token probabilities rather than products of sequence probabilities and do not use a separate retain loss.

PROD. PROD constructs signed supervision from the frozen M_{full} logits at each forget target position. For the designated target token v^*,

$$w(v^*) = -\alpha p_0(v^*).$$

For all other tokens, weights are obtained from the normalized reference logits after excluding v^* and applying top-p filtering. Training minimizes the target-token average

$$-\sum_v w(v)\log\big(p_\theta(v) + 10^{-10}\big). \tag{29}$$

These signed weights do not form a probability distribution. The reported configuration uses $\alpha = 0.2$ and top-$p = 0.9$ without an additional retain objective.

RMU. RMU captures the output of decoder layer 7 and minimizes

$$\mathcal{L}_{\mathrm{RMU}} = \mathrm{MSE}(h_\theta^F, cu) + \lambda_R\,\mathrm{MSE}(h_\theta^R, h_0^R), \tag{30}$$

where u is a fixed normalized random vector sampled with seed 42. Layer indices are zero-based. Only the full `down_proj.weight` matrices in layers 5, 6, and 7 are updated. All remaining parameters, including the starting LoRA adapter, are frozen. RMU therefore uses a different trainable parameter scope from the LoRA baselines.

ASU-KL. A frozen forget teacher divides its masked attention logits by $\tau = 1.5$ before the attention softmax. The student uses ordinary attention. Let $p_0^{(\tau)}$ denote the resulting teacher distribution. The implemented loss is

$$\mathcal{L}_{\text{ASU-KL}} = \mathbb{E}_{F,t} D_{\mathrm{KL}}\left(p_0^{(\tau)} \| p_\theta\right) + 0.03\mathcal{K}_R. \tag{31}$$

The temperature modifies the teacher attention distribution rather than the final output softmax. The retain teacher uses ordinary attention. Architecture-specific attention handling is used for Qwen2.5-Coder-7B and CodeLlama-7B.

Task Vector. Starting from M_{full}, we reinforce the forget continuations by minimizing $\mathcal{N}_F$, obtaining LoRA adapter tensors ϕ_{reinf}. The deployed adapter is

$$\phi_{\mathrm{TV}} = \phi_0 - \alpha\left(\phi_{\mathrm{reinf}} - \phi_0\right), \qquad \alpha = 1. \tag{32}$$

The subtraction is applied directly to the saved LoRA tensors rather than to merged dense model weights.

For the strength-dependent evaluation in Section 5.3, different subtraction coefficients can be composed from the same reinforced adapter without retraining. The coefficient sweep is

$$\alpha \in \{0, 0.125, 0.25, 0.5, 1, 2, 4\}.$$

CodeEraser. For the common code-continuation setup, we treat the designated forget continuation as the sensitive span and its supplied prefix as normal context. The implemented objective is

$$\mathcal{L}_{\mathrm{CE}} = -\mathcal{N}_F + \gamma \mathcal{N}_{\mathrm{context}} - \lambda \mathcal{K}_F + \lambda \alpha \mathcal{K}_R, \tag{33}$$

with

$$(\gamma, \lambda, \alpha) = (0.5, 0.1, 1).$$

The teacher is a frozen copy of the starting adapter. Each loss component is normalized by its corresponding number of scored tokens. Forget inputs contain up to 128 prefix tokens followed by up to 384 continuation tokens; retain inputs use the first 128 native code tokens.

One pass processes 300 forget examples and 300 seed-0 retain-training examples in ten paired groups: nine groups contain 32 examples per role and the final group contains 12. AdamW uses learning rate 3×10^{-6}, zero weight decay, $(\beta_1, \beta_2) = (0.9, 0.999)$, and $\epsilon = 10^{-8}$.

The resulting configuration is a fixed one-pass LoRA update for the continuation-level setting used in this paper.

C.3 Inference-Time Baselines

GSS. GSS preserves the model parameters and constructs probe and steering directions offline. Let p_0 denote the adapted M_{full} model and p_{base} the corresponding pre-adaptation model. Eligible training tokens are partitioned using

$$\omega_t = \log p_0(y_t \mid x, y_{<t}) - \log p_{\mathrm{base}}(y_t \mid x, y_{<t}). \tag{34}$$

Tokens with $\omega_t > 0$ form the memorization partition and tokens with $\omega_t \le 0$ form the generalization partition.

Fitting combines the hidden-state–NLL-gradient cross-moment from the memorization partition with covariance whitening on the generalization partition. A truncated SVD yields probe directions a_j and steering directions b_j. Each threshold τ_j is the higher-interpolated 99th percentile of $|a_j^\top h|$ on generalization tokens.

At the selected layer, GSS applies

$$h' = h - s \sum_{j=1}^{r} \mathbf{1}\left[|a_j^\top h| > \tau_j\right] (a_j^\top h) b_j. \tag{35}$$

For each model–corpus setting, we scan all 28 decoder layers of Qwen2.5-Coder-7B and all 32 decoder layers of CodeLlama-7B, using the same layer-selection procedure as UNBIND. The configurations reported below are the selected operating points from this scan.

For the strength-dependent comparison in Section 5.3, the steering strength s is varied while the remaining GSS construction is held fixed. The pretrained reference model and target-loss gradients are required only during fitting. During generation, GSS uses a single hooked target model and does not require auxiliary language-model generation or candidate reranking. R-PPL is evaluated by teacher forcing with the same intervention active.

Divergence Decoding. We use the linear trigram implementation of Divergence Decoding. The forget auxiliary is fitted on the concatenated prefix and continuation of all 300 forget-training examples. The retain auxiliary is fitted on all 8,280 retain-training programs. Counts are constructed independently within each source record using the target model's tokenizer, without inserting BOS or EOS tokens.

Let $b_F(v \mid c)$ and $b_R(v \mid c)$ denote the Stupid-Backoff scores of the forget and retain trigram auxiliaries, respectively, using a backoff factor of 0.4. Given target-model logits $z_0(v \mid c)$, decoding uses

$$p_{\mathrm{DD}}(v \mid c) = \mathrm{softmax}_v \left[z_0(v \mid c) + 10 \left(b_R(v \mid c) - b_F(v \mid c) \right) \right]. \tag{36}$$

The correction uses raw backoff scores rather than logarithmic scores. Each prediction therefore requires two count-based auxiliary lookups but no neural auxiliary forward pass. Generation and R-PPL use the same corrected and normalized distribution.

For teacher forcing, the auxiliary context ends at the token whose target-model logit predicts the next token. The implementation therefore uses the same next-token alignment for generation and NLL evaluation.

Conformal Unlearning. CU keeps M_{full} frozen and combines a deletion-oriented prompt with retrieval and external verification. A fixed retrieval catalogue contains the 300 forget-training references together with 30 held-out forget-validation references. Each query retrieves one reference by character 3–5-gram TF–IDF cosine similarity between prefixes.

The local generator receives the query, deletion instructions, and the retrieved source prefix, but not the protected continuation. The verifier receives the query, generated candidate, retrieved prefix, and protected continuation, and returns a score in $[0, 10]$ together with feedback. A candidate is accepted when its score is at least 9. Retain and functional-test reference answers are not supplied to either component.

CU iteration-budget calibration. For each of the 30 held-out calibration examples, we record the first iteration at which a candidate satisfies the acceptance criterion, with an operational cap of 100 iterations. For $\alpha = 0.1$, the calibrated budget is the

$$\lceil (30 + 1)(1 - \alpha) \rceil = 28$$

th order statistic of these first-hit iteration counts.

For the CU experiments reported in the main comparison, this calibration yields

$$T = 1.$$

Consequently, each evaluated prompt receives one greedy local completion of at most 128 new tokens and one verifier evaluation. The reported results therefore use the calibrated one-iteration operating point rather than multi-round revision.

The general implementation supports additional rounds by appending previous candidates and verifier feedback to the subsequent prompt and returning the highest-scored candidate when the budget is exhausted, but this path is inactive when $T = 1$.

Because the retrieval-based adaptation does not establish calibration–evaluation exchangeability for our code-continuation setting, we use the conformal calculation only to determine the iteration budget and do not make a formal coverage claim.

CU retain likelihood. CU changes the conditioning context by introducing its method-specific first-round prompt. For the CU results reported in Table 1, retain perplexity is computed from the teacher-forced likelihood under this same prompt:

$$\text{PPL}_{\text{CU}} = \exp\left[-\frac{\sum_i \sum_t \log p_0(y_{i,t} \mid P_{\text{CU}}(x_i), y_{i,<t})}{\sum_i |y_i|} \right], \tag{37}$$

where $P_{\text{CU}}(x_i)$ denotes the first-round CU prompt for retain example i.

All prompt tokens are masked and the complete target suffix is scored, including its first token. No EOS token is appended and the target suffix is not truncated for this likelihood computation. This quantity is the R-PPL reported for CU in the main table. FU-H is then computed using the same definition in Equation (12) as for the other reported methods.

D Full CodeLlama Result

E Fixed-budget Target Reproduction

E.1 Prompt conditions and sampling budget

Let P_i be the stored prompt and $Y_i = (y_1, \ldots, y_{n_i})$ its withheld target, tokenized with the evaluated model's native tokenizer without added special tokens. The six conditions, in order, are original, prefix-16, prefix-32, prefix-64, instruction rewrite, and fenced code. Original uses P_i. Prefix-k appends $y_{1:k}$ directly to the tokenized P_i, without a separator or re-tokenization. The other two prompts are literal templates below; `<P>` is replaced by P_i.

Table 7: Complete CodeLlama results. Metrics and ranking conventions follow Table 1; reference models are excluded from ranking.

	CodeSearchNet						The Stack					
Method	F-BLEU↓	R-PPL↓	R-BLEU↑	HE↑	MBPP+↑	**FU-H↑**	F-BLEU↓	R-PPL↓	R-BLEU↑	HE↑	MBPP+↑	**FU-H↑**
M_{full}	0.4600	2.2521	0.3949	38	125	0.00	0.1434	1.7330	0.4736	36	163	0.00
M_{retrain}	0.0894	2.2526	0.3940	40	114	88.30	0.0685	1.7320	0.4754	36	162	68.56
M_{base}	0.0963	2.6220	0.3474	41	158	85.46	0.0651	1.7160	0.4732	41	158	70.45
GA	0.3926	2.2342	**0.4004**	34	122	25.45	0.0651	1.7250	0.4721	37	161	70.54
GradDiff	0.0032	119478.109	0.1183	0	0	0.00	*0.0347*	1.7348	0.4655	**42**	153	*85.46*
NPO-KL	0.2881	*2.2213*	**0.4004**	33	105	53.23	0.1026	*1.7232*	*0.4795*	*39*	160	44.23
SimNPO-KL	0.1330	2.2393	0.3802	33	77	*77.31*	0.0834	**1.7224**	0.4817	40	161	58.91
DPO	0.3653	**2.2168**	0.3903	37	119	34.02	0.1219	1.7256	0.4765	37	157	26.00
FLAT	0.0301	2.4337	0.3151	16	46	71.70	0.0036	1.7331	0.4769	35	154	97.68
PROD	0.3073	2.2372	0.3907	30	107	48.56	0.1244	1.7270	0.4752	38	154	23.31
Task Vector	0.2547	2.2428	*0.3937*	**39**	108	61.01	0.1045	1.7230	0.4772	38	*162*	42.65
RMU	0.4577	2.2521	*0.3937*	*36*	*125*	0.99	0.1443	1.7330	0.4742	36	160	0.00
ASU-KL	0.1261	2.3078	0.3731	33	127	82.15	0.0835	1.7346	0.4697	38	160	58.82
GSS	**0.000014**	624186.8588	0.0003	0	0	0.00	0.0994	1.7737	**0.4830**	33	*162*	46.66
DD	0.1143	2.6876	0.3452	24	63	72.33	0.0560	2.0080	0.4181	24	100	67.10
CodeEraser	0.4349	2.2209	0.3910	33	123	10.33	0.1400	1.7297	0.4747	35	**165**	4.72
CU	0.1079	2.2516	0.3664	*36*	44	75.67	0.0484	1.7330	0.4555	20	14	54.47
UNBIND	*0.0054*	2.2588	0.3978	*36*	**129**	**98.71**	**0.0027**	1.7599	0.4707	36	163	**98.77**

```
Continue the following source code. Preserve its task and return only the continuatio

<P>


Complete the code in the fenced block. Return only its continuation.
```
<P>
```

The fence has no language tag. No chat template is applied; a BOS token is prepended when defined by the tokenizer. Each eligible condition receives one greedy attempt ($a = 0$) and four sampled attempts ($a = 1, \ldots, 4$), with temperature 0.8, top-$p = 0.95$, top-$k = 0$, one beam, and repetition penalty 1. Generation stops at EOS or 256 new tokens; EOS and padding are excluded from scoring. There is no additional stop string.

The base seed is zero. Targets are assigned to six shards by $i \bmod 6$; within each condition, eligible targets retain ascending index order. For zero-based shard $s$, condition $c$, and batch $b$, the PyTorch CPU and CUDA seed is $10^6 s + 10^5 c + 10^4 a + b$. Batch sizes are 32 for Qwen and 16 for CodeLlama. Thus the budget is at most 30 completions per target. Prefix-$k$ is skipped when $n_i \leq k$, without reallocating its attempts. Empty targets are rejected. Suffixes longer than 256 tokens remain in the evaluation, although complete reproduction is then unattainable.

### E.2 Exact-span and recovery metrics

For condition $c$, let $S_{ic} = Y_i[k_c + 1 : n_i]$, where $k_c$ is 16, 32, or 64 for prefix injection and zero otherwise. Let $\mathcal{C}_i$ contain conditions with nonempty $S_{ic}$, and let $G_{ica}$ contain only newly generated token IDs. Matching is exact over native token IDs, with no whitespace normalization or decoded-text re-tokenization. Matches may start anywhere in $G_{ica}$ and $S_{ic}$.

We compute the longest common contiguous span using $D_{uv} = D_{u-1,v-1} + 1$ when the two token IDs agree and $D_{uv} = 0$ otherwise, with zero boundary values. A candidate is counted only if its entire token sequence does not already occur in the actual supplied prompt, including injected tokens. Denote the longest such span by $L_{ica}$ (zero if none exists). This is a contiguous substring
```

metric, not a subsequence or prefix-only metric. Define

$$L_i = \max_{c \in \mathcal{C}_i,\, a \in \{0,\ldots,4\}} L_{ica}, \tag{38}$$

$$R_i = \max_{c \in \mathcal{C}_i,\, a \in \{0,\ldots,4\}} \frac{L_{ica}}{|S_{ic}|}, \qquad \overline{R} = \frac{100}{N} \sum_{i=1}^{N} R_i, \tag{39}$$

$$N_t = \sum_{i=1}^{N} \mathbf{1}[L_i \geq t], \quad t \in \{50, 100\}. \tag{40}$$

Here $N = 300$. Each ratio uses its own condition's remaining suffix; we never divide a span by another condition's suffix length. Condition-specific means use only targets eligible for that condition. Sampled-only budgets use $a = 1, \ldots, k$; greedy-plus-sampled budgets use $a = 0, \ldots, k$, with the same k applied to every eligible condition.

Full-suffix token recovery requires the entire S_{ic} to occur in the completion and excludes suffixes already present in the prompt as either tokens or literal text. Literal-text recovery is recorded separately. Both are target-level unions across eligible attempts. A short suffix can be recovered completely without reaching 50 tokens; $R_i = 1$ does not imply reproduction of the original full target.

E.3 Complete access results

Table 8 presents the complete CodeLlama access results; Table 9 gives the condition-level span counts for both backbones. Interventions remain fixed across prompt conditions. GSS uses absolute strength 1. All counts are out of 300 targets; ineligible conditions contribute no successes.

F Multilingual Evaluation

F.1 Language-specific data and reference models

We evaluate separate CodeSearchNet forgetting tasks for Python, Java, and JavaScript. No Stack data are pooled into these columns. Each task has 300 forget-training targets and 50/50 validation/test targets. Formal F-BLEU is evaluated on the 300 training targets. The shared retain train/validation/test split contains 8,280/460/460 examples from Go, Java, JavaScript, PHP, Python, and Ruby, preserving the original language mixture.

For each backbone and forget language, M_{full} is the matched model adapted with that language's forget set and the shared retain data. The detector, direction, and calibration threshold are constructed separately for each task: forget fit/calibration uses 240/60 examples, and retain fit/calibration uses 480/120 examples (80/20 per language). Threshold and strength calibration follows Section 4.3: candidate activation quantiles and strengths are evaluated on separate calibration data under retain KL, NLL, and BLEU budgets, and the feasible candidate with the highest forget validation NLL is selected. As in Section 5.1, UNBIND uses $\kappa = 20$. The CSN layer is shared across languages within each backbone: 23 for Qwen and 29 for CodeLlama (zero-based); detectors, directions, and thresholds are not shared across forget languages. Thus these experiments assess applicability across languages; they do not test direct transfer of one language's intervention to another.

F.2 Complete multilingual results

Table 10 presents the complete multilingual results. We use the definitions in Equation (12):

$$\text{FR} = \max(0, 1 - F/F_{\text{full}}), \tag{41}$$

$$\text{UR} = \left[\prod_{j=1}^{4} \min(1, r_j) \right]^{1/4}, \tag{42}$$

$$\text{FU-H} = 100 \frac{2\,\text{FR}\,\text{UR}}{\text{FR} + \text{UR}}, \tag{43}$$

Table 8: CodeLlama target reproduction under the fixed access budget. Counts are out of 300; $\overline{R}$ is reported in percent. Lower values indicate less target reproduction.

	CSN		Stack	
Method	N_{50}/N_{100}	$\overline{R}$	N_{50}/N_{100}	$\overline{R}$
M_{full}	253/149	100.00	262/161	27.82
M_{retrain}	11/2	19.59	63/18	7.27
PROD	147/34	63.78	212/100	18.00
GSS	0/0	1.52	159/67	13.80
SimNPO-KL	25/7	29.16	103/31	9.85
FLAT	147/46	62.88	0/0	0.89
Task Vector	113/26	54.67	173/69	13.89
UNBIND	0/0	2.70	2/0	0.43

Table 9: Per-condition N_{50}/N_{100} using all five attempts.

Method	Original	Prefix-16	Prefix-32	Prefix-64	Rewrite	Fence
			qwen/csn			
M_{full}	161/51	144/42	114/40	79/27	102/35	0/0
M_{retrain}	3/1	2/1	4/0	8/0	1/0	0/0
PROD	34/9	35/8	32/10	26/7	27/6	0/0
GSS	20/5	30/4	33/5	31/9	15/5	0/0
SimNPO-KL	2/1	4/2	3/1	4/0	3/1	0/0
FLAT	3/1	6/2	4/1	5/1	3/1	0/0
Task Vector	38/8	39/7	39/9	21/5	32/6	0/0
UNBIND	0/0	0/0	0/0	0/0	0/0	0/0
			qwen/stack			
M_{full}	131/56	145/68	133/65	126/64	103/47	4/0
M_{retrain}	30/7	35/9	33/9	31/8	30/8	1/0
PROD	99/44	104/43	95/44	93/46	86/32	4/1
GSS	3/0	4/0	2/0	1/0	8/0	2/0
SimNPO-KL	46/14	48/19	47/18	45/13	48/13	2/0
FLAT	30/12	36/12	41/12	39/10	33/8	3/0
Task Vector	77/30	83/33	80/33	71/35	67/22	2/1
UNBIND	0/0	0/0	0/0	1/0	1/0	0/0
			codellama/csn			
M_{full}	228/121	208/107	185/89	146/70	118/51	4/0
M_{retrain}	4/0	2/0	3/2	7/1	4/1	1/0
PROD	94/22	88/19	73/21	64/21	36/7	2/1
GSS	0/0	0/0	0/0	0/0	0/0	0/0
SimNPO-KL	11/4	12/3	13/4	10/4	9/3	1/0
FLAT	91/25	93/26	83/27	68/27	37/10	2/0
Task Vector	66/17	66/19	63/15	49/14	30/6	0/0
UNBIND	0/0	0/0	0/0	0/0	0/0	0/0
			codellama/stack			
M_{full}	211/115	207/120	208/115	214/121	175/91	59/21
M_{retrain}	39/10	34/12	31/11	38/12	34/8	11/2
PROD	143/66	147/68	148/67	146/59	128/50	32/7
GSS	101/35	102/41	99/37	107/37	88/27	40/11
SimNPO-KL	57/14	60/15	61/18	61/17	55/18	16/5
FLAT	0/0	0/0	0/0	0/0	0/0	0/0
Task Vector	110/31	112/44	117/42	114/39	92/31	32/11
UNBIND	2/0	0/0	0/0	0/0	2/0	0/0

where the four ratios are $\text{PPL}_{\text{full}}/\text{PPL}$, $\text{R-BLEU}/\text{R-BLEU}_{\text{full}}$, $\text{HE+}/\text{HE+}_{\text{full}}$, and $\text{MBPP+}/\text{MBPP+}_{\text{full}}$. All references are matched to the backbone and forget language. FU-H is zero when FR and UR are both zero; FR and UR are displayed as percentages, but the formula uses fractions. Scores use unrounded inputs.

The utility protocol follows Section 5.1: retain PPL and R-BLEU measure retention, while HE+/MBPP+ count Python problems passing both base and extended tests. In Java/JavaScript for-

getting tasks, these Python benchmarks measure retained Python utility; they do not establish functional performance in the forget language. The multilingual forgetting claim concerns target-code reproduction in each language.

For reference, the original-model F-BLEU values (F_{full}) for Python/Java/JavaScript are $0.4847/0.5075/0.4020$ for Qwen and $0.4592/0.4419/0.4320$ for CodeLlama (rounded for display).

G FIXED-BUDGET ACCESS PROTOCOL

Table 10: Complete multilingual results. Each language specifies a separate forget task with its own original-model reference. FR and UR are percentages; larger values are better for all three scores.

	Python			Java			JavaScript		
Method	FR	UR	FU-H	FR	UR	FU-H	FR	UR	FU-H
				Qwen2.5-Coder-7B					
M_{full}	0.00	100.00	0.00	0.00	100.00	0.00	0.00	100.00	0.00
M_{retrain}	80.84	98.82	88.93	62.11	97.76	75.96	64.45	100.00	78.38
PROD	51.75	90.11	65.74	76.30	90.87	82.95	69.00	95.79	80.22
GSS	58.95	97.98	73.61	41.73	91.13	57.25	34.80	95.58	51.02
SimNPO-KL	81.46	98.10	89.01	99.15	47.65	64.37	97.96	62.95	76.65
FLAT	84.77	84.26	84.51	99.33	5.75	10.87	99.08	7.03	13.13
Task Vector	48.63	97.19	64.82	45.22	95.18	61.31	40.77	99.70	57.88
UNBIND	97.23	99.93	98.56	97.08	99.85	98.45	98.81	99.33	99.07
				CodeLlama-7B					
M_{full}	0.00	100.00	0.00	0.00	100.00	0.00	0.00	100.00	0.00
M_{retrain}	80.03	94.85	86.81	62.37	98.35	76.33	66.78	99.68	79.98
PROD	33.34	94.91	49.35	18.35	97.55	30.89	28.45	96.41	43.93
GSS	99.17	0.00	0.00	99.52	0.00	0.00	99.77	0.00	0.00
SimNPO-KL	70.86	92.05	80.08	62.21	97.35	75.91	98.84	45.48	62.29
FLAT	95.21	65.82	77.83	77.51	89.47	83.06	99.17	1.71	3.37
Task Vector	45.88	96.70	62.24	31.46	99.69	47.82	38.87	95.72	55.28
UNBIND	98.56	99.58	99.07	97.33	97.80	97.57	98.17	98.58	98.37

Table 11: Target reproduction and multilingual results for CodeLlama-7B. Definitions and formatting follow Table 3. Each forget language uses its own original-model reference.

	Target reproduction ↓				Multilingual: forget language								
	CSN		Stack		Python			Java			JavaScript		
Method	Counts	$\overline{R}$	Counts	$\overline{R}$	FR↑	UR↑	FU-H↑	FR↑	UR↑	FU-H↑	FR↑	UR↑	FU-H↑
M_{full}	253/149	100.00	262/161	27.82	0.00	100.00	0.00	0.00	100.00	0.00	0.00	100.00	0.00
M_{retrain}	11/2	19.59	63/18	7.27	80.03	94.85	86.81	62.37	98.35	76.33	66.78	99.68	79.98
PROD	147/34	63.78	212/100	18.00	33.34	94.91	49.35	18.35	97.55	30.89	28.45	96.41	43.93
GSS	**0/0**	**1.52**	159/67	13.80	**99.17**	0.00	0.00	**99.52**	0.00	0.00	**99.77**	0.00	0.00
SimNPO-KL	25/7	29.16	103/31	9.85	70.86	92.05	80.08	62.21	97.35	75.91	98.84	45.48	62.29
FLAT	147/46	62.88	**0/0**	0.89	95.21	65.82	77.83	77.51	89.47	83.06	99.17	1.71	3.37
Task Vector	113/26	54.67	173/69	13.89	45.88	96.70	62.24	31.46	**99.69**	47.82	38.87	95.72	55.28
UNBIND	**0/0**	2.70	2/**0**	**0.43**	98.56	**99.58**	**99.07**	97.33	97.80	**97.57**	98.17	**98.58**	**98.37**

H Additional Related Work

Data removal and memorization. Early machine unlearning updates statistical summaries (Cao & Yang, 2015) or uses partitioned training to limit the retraining required by deletion requests (Bourtoule et al., 2021). Certified removal formalizes indistinguishability from a model trained without the removed data, with a mechanism for linear classifiers (Guo et al., 2020). Training-data deduplication reduces memorized text emission (Lee et al., 2022), whereas in-context unlearning changes predictions through relabeled examples supplied at inference time without updating parameters (Pawelczyk et al., 2024). UNBIND instead intervenes on hidden states associated with designated forget data.