CAN OPEN-WEIGHT LLMS BE PROTECTED FROM DISTILLATION?

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM providers are concerned about their model's ability being learned by competitors via distillation, which improves a model by training on outputs from another stronger model. To stop distillation, prior work post-processes LLMs' raw reasoning traces before returning them to users to protect commercial APIs. These techniques, however, cannot protect open-weight LLMs, whose reasoning traces, decoding process, and model weights are completely controlled by the distillation attacker. As open LLMs rapidly approach the frontier, protecting them from being distilled emerges as an important problem that is under-explored.

Motivated by it, we propose an antidistillation recipe that fine-tunes LLMs to reason in an encoded vocabulary while answering in their original vocabulary. The key idea is to keep reasoning traces useful to the model itself, but difficult for other models to learn from. Our encoded Qwen3-8B and QwQ-32B produce non-readable reasoning, making it detrimental for distillation. Thus, the attacker can only distill on final readable answers of our models, which is shown to be ineffective. We also evaluate 7 adaptive attackers that try to recover original reasoning from our encoded reasoning to distill from. The tested recovery attempts either recover at most 11% of the encoded–readable token mapping, or directly drop non-trivial utility of our model, making it a poor distillation teacher. While mitigating distillation, our encoded models still maintain 88%–99% of original model's utility scores on GSM8K, MATH-500, MMLU-Pro math, and AIME 2025. As a first step towards antidistillation for open LLMs, this work aims to open up possibilities for providers to release strong models' weights while limiting distillation by competitors.

1 INTRODUCTION

Distillation has become one of the main routes to build a capable large language model (LLM). By training on only 1000 outputs (reasoning and answers, from the s1K dataset prompts) from another strong LLM, one can significantly improve a 32B model (Muennighoff et al., 2025), reaching performance well beyond what models of their sizes had shown (DeepSeek-AI, 2025). Distillation from reasoning + answers is far more sample-efficient than distillation from answers only (OpenAI, 2024), see our Figure 1, because a chain of thought exposes step-by-step reasoning in a language every other model understands.

Thus, to stop distillation, providers of API-served LLMs have hidden their raw reasoning traces. What the user sees is a reasoning that is summarized (OpenAI, 2024), rewritten (Ding et al., 2025; Ma et al., 2026), or perturbed (Savani et al., 2025; Diao et al., 2026) by a separate model. The success of those antidistillation techniques relies on API providers' complete control of the system outside the LLM to post-process raw reasoning traces before they reach the user.

None of the above is applicable to protecting an open-weight model. Releasing fixed model weights gives recipients the ability to run the model (using whatever decoding and reading whatever the model interally computes), which also enables recipients to use it as a source of supervision for other models. It is thus impossible to post-process an open LLM's reasoning while prohibiting the attacker from accessing its raw reasoning. A provider may welcome deployment of its model while hoping to limit the transfer of its strong capabilities into independently trained students. The same need arises when weights are shared with another company for internal use.

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.

To protect open-weight LLMs from being distilled, one needs to change the model weights. The changed model uses reasoning traces to provide intermediate computation to reach final high-quality answers, but they also inevitably provide supervision from which another model can learn. This motivates a central question: can a model retain the benefit of generating and consuming its own reasoning while reducing the usefulness of that reasoning for training another model?

To achieve those seemingly contradictory goals, we propose to fine-tune LLMs to reason in an encoded vocabulary while answering in their original vocabulary. The encoded reasoning is made only understandable by the model itself, and hard to read for anyone else. An attacker distilling on encoded reasoning will see its student's utility collapse, and is thus forced to distill only on answers from our models, which is much less effective. We add a set of special tokens to the vocabulary, each corresponding to an original token. We need to train against recovery of this correspondence, as an attacker who recovers this encoding mapping could translate the traces back into ordinary distillable reasoning traces. Thus, we set the training loss for LLMs to (1) reason only with the added encoded tokens for antidistillation, (2) not leak the original-encoded token mapping to attackers, and (3) still output high-quality answers in original tokens for normal usage.

Experiments show that our fine-tuned Qwen3-8B and QwQ-32B achieve these three objectives: (1) Over 99.9% of their reasoning uses encoded tokens. Distilling on encoded reasoning + readable answers drops the student model's utility to 0%. Distilling only on our readable answers is shown to be ineffective. (2) 7 evaluated adaptive attackers trying to recover the readable-encoded token mapping (by prompting, inspecting LLM feature distances, or manipulating decoding) either fail in over 89% of the cases, or directly drop non-trivial utility of our model, making it a poor distillation teacher. (3) Our encoded models still maintain 88%–99% of original model's utility scores on GSM8K, MATH-500, MMLU-Pro math, and AIME25.

Evaluating antidistillation requires establishing that there is useful knowledge transfer to prevent. A student that fails to improve even when trained on the undefended teacher provides little evidence about a defense. In all experiments, we study teacher–student pairs for which ordinary reasoning-trace distillation improves performance, and assess whether protection suppresses that improvement, an important verification setup omitted in existing studies (Li et al., 2025; Savani et al., 2025).

2 Related work

Knowledge distillation transfers a teacher's behaviour into a student model through its outputs (Hinton et al., 2015). For a reasoning model the output that matters is the reasoning trace. Supervised fine-tuning on a strong model's chains of thought recovers most of its capability far more cheaply than reinforcement learning. For example, in the s1 paper, distillation showed huge gains from just a thousand curated thinking traces (Muennighoff et al., 2025). The distilled DeepSeek-R1 variants (DeepSeek-AI, 2025) are another instance.

That sample-efficiency of distillation is dangerous for frontier models, since smaller models can replicate their strength with far less compute. For API-based models, many providers perturb the reasoning trace before sending it to the client. Antidistillation sampling (Savani et al., 2025) is one of them, where the next-token distribution is perturbed at inference time so that a student trained on the generated output is deliberately poisoned. PART (Ding et al., 2025) rewrites the reasoning trace after generation to obfuscate it. DOGe (Li et al., 2025) is the predominant training-based approach, since its defense lies in the weights rather than custom inference logic. DOGe fine-tunes the output layer of the teacher adversarially against a held-out student model. DOGe is closest to our setting because it applies to open-weight models, but it degrades the reasoning ability of the teacher as a result. Although the original DOGe paper claims a perfect antidistillation effect with no teacher utility drop and drastically lower distilled student accuracy, we could not reproduce those claims with its official code, even after email correspondence with the authors.

A separate line of research, latent-space reasoning, removes the reasoning trace entirely and enables the model to reason in latent vectors rather than discrete tokens. Coconut (Hao et al., 2024) feeds the last hidden state back as the input embedding for the next reasoning step, so the chain of thought becomes a sequence of continuous vectors rather than sampled tokens. A latent reasoning trace is unreadable to an attacker, but it is equally unreadable to every human, including the model provider. Legible chains of thought are a fragile but valuable safety property (Korbak et al., 2025), and

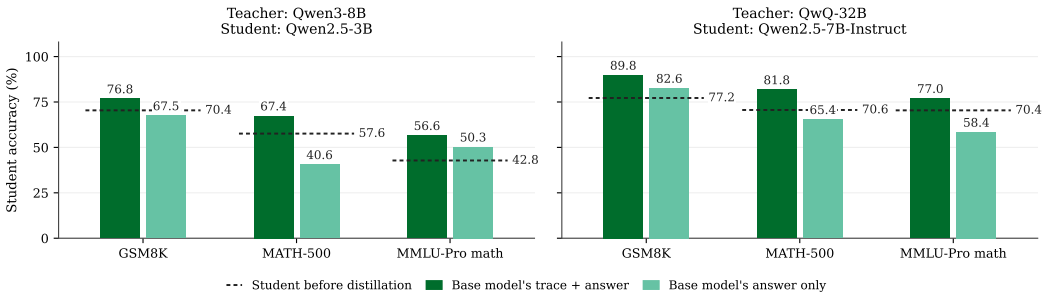


Figure 1: Student accuracy after distillation, against the student's accuracy before it (dotted line). Training on a teacher's reasoning trace and answer (dark green) improves the student by a large margin, while training on the same teacher's answers alone (light green) does not.

monitoring a model's reasoning is a growing safety need (Pachocki, 2026); latent reasoning gives that up entirely. In comparison, our method keeps reasoning readable to the model provider, who designed and holds the encoded–readable token mapping. Whenever chain-of-thought monitoring is needed, the provider can translate the reasoning trace back to the original readable tokens and understand the model's behavior.

3 Anti-distillation for open LLMs: motivation and challenges

3.1 Why protect open-weight LLMs from distillation?

A reasoning model is expensive to train and, through its reasoning traces, relatively cheap to replicate through distillation. An API-based model provider can defend against distillation risk since they control inference-time logic. They decide what exactly the user receives from the LLM, while still leaving the LLM's reasoning process unchanged, since the same chain of thought is used, simply not revealed to the user. Figure 1 shows how big of a difference the teacher model's reasoning trace makes during distillation: without the reasoning traces and only from the pure output, there is no significant student gain. Hiding or summarizing the reasoning trace is therefore an effective, and now standard, defense for API-served models.

An open-weight model has no inference layer that the model provider can control. Anyone out there can run open-weight models and obtain the raw reasoning traces from them, and today the decision to release model weights is a decision to accept distillation from their model. Hiding the thinking protects an API-served LLM, but it cannot protect an open-weight LLM.

If effective distillation-defense techniques for open-weight models existed, this would influence companies' decisions about whether to release their model weights at all, and companies that are currently unwilling to do this due to distillation concerns might become willing to do so.

There are also many situations involving company-to-company agreements in which one company provides model weights to another for internal use, while still not wanting the recipient to train their own replica of this model through distillation. In such cases, a distillation defense that lives in the model weights themselves is highly valuable.

3.2 Why is it hard?

Whatever an open-weight model needs in order to reason is also visible to attackers, and that very information is the exact training signal needed for distillation.

What the attacker has. The model provider we assume pretrains and post-trains a model to reason, applies a distillation defense to that checkpoint, and releases the resulting weights; the raw reasoning model never leaves the provider, and neither does any defense-specific information. The attacker receives everything else: they hold the weights and tokenizer; they run the model on hardware they control, so they observe every activation at every layer; and they can generate from the model at their

own limit. What they lack is the original model and the defense-specific information that the provider used, and recovering those from the released checkpoint is what we must prevent.

A defense that survives this open-weight setting must satisfy two objectives at once:

1. The model's reasoning traces should be useless to any student model, and training on them should teach a student nothing beyond what the final outputs already give.
2. The model's reasoning traces should remain fully useful to the model itself. Specifically, it must still generate and read its own reasoning trace to complete tasks at its original quality.

The two objectives pull directly against each other. A reasoning trace is useful to its own model only because the model's weights assign it a specific meaning, and the model weights are available to attackers. The problem is to make a reasoning trace meaningful to a public set of weights while keeping that meaning from attackers who can read those weights.

4 APPROACH

4.1 PRINCIPLES OF ANTI-DISTILLATION FOR OPEN LLMS

To solve both objectives stated in Section 3.2, our defense is to make the reasoning traces *model-specific*: written in a token sequence that only our model can understand and generate. The defense mechanism must also ensure that the model-specific reasoning traces cannot be easily translated back into readable reasoning traces using information available from the open-weight model, like weights or activations.

4.2 METHOD OVERVIEW AND TERMINOLOGY

Our approach is to add new special tokens, called *encoded tokens*, to the model's vocabulary, where each encoded token has a one-to-one mapping to a readable token in the model's vocabulary. We essentially fine-tune the model to (1) reason only in the encoded tokens, (2) still answer in readable tokens, and (3) not leak the mapping between encoded and readable tokens.

A reasoning model produces, for a prompt x, a reasoning trace of tokens encapsulated within special delimiter tokens (such as `<think>` and `</think>`), and then an output/answer a. We call this original model, which generates fully readable text, the *base model*. We extend the original vocabulary V with K new encoded tokens whose token strings render as illegible text. We randomly define σ, the *mapping*, from every readable token that occurs inside the base model's reasoning traces to one of the new encoded tokens. We define these encoded-token/readable-token pairs as *twins*.

The *encoded model* is the model which is trained to generate reasoning traces in encoded tokens, but still outputs the answer in readable tokens. This behavior is purely learned in the weights and does not emerge from custom decoding logic or a multi-stage protocol that an attacker could remove. Figure 2 illustrates the difference between the base and encoded model. Given the same prompt, they return the same answer, but one writes its reasoning in readable tokens and the other in encoded ones, exactly as mapped to each other. The 1-to-1 mapping is drawn once, uniformly at random before training, and never changed. Appendix B gives the construction in full.

4.3 ADDED PARAMETERS AND WHAT TRAINS

The added encoded tokens need additional embeddings in the input embedding table $E \in \mathbb{R}^{V \times d}$ and additional rows in the output embedding matrix (LM head) $W \in \mathbb{R}^{V \times d}$. We append K new embeddings to each and set the model's vocabulary to $V + K$. Every added embedding in E_r and W_r is drawn as an independent Gaussian vector rescaled to the mean norm of the original token embeddings in the corresponding weight matrix. At the start of training an encoded token is an arbitrary new symbol, and the only way the model learns to treat it as equivalent to its readable token is through the training loss.

Training starts with the entire model being unfrozen except the readable tokens' input embeddings, so that readable tokens keep the meaning learned from pre-training. The encoded tokens' input

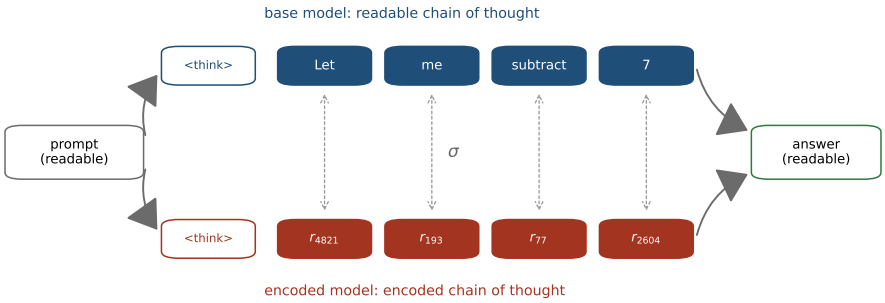


Figure 2: Given the same prompt, the base model reasons in readable tokens and the encoded model reasons in encoded ones (mapped to each other), and both produce the same readable answer.

embeddings, all the output embeddings, and the entire LLM body are all trainable. Appendix A gives the training setup more in-depth.

4.4 Training for utility and anti-distillation

We present two groups of losses that (1) ensure the model reasons only in encoded tokens and continues to answer in readable tokens (anti-distillation) and (2) ensure reasoning ability degrades as minimally as possible (utility). We state the anti-distillation group first. Let B be the frozen base model and M the encoded model being trained; write M_i for M's next-token distribution at position i, and B_i likewise for B. Let th and ans be the sets of positions inside the thinking trace and in the answer of a training record:

$$\mathcal{L}_{\text{anti}} = \underbrace{-\lambda_{\text{pur}} \sum_{i \in \text{th}} \log \sum_{v \geq V} M_i(v)}_{\text{nothing readable while thinking}} \underbrace{- \lambda_{\text{pur}} \sum_{i \in \text{ans}} \log \sum_{v < V} M_i(v)}_{\text{nothing encoded in the answer}} \tag{1}$$

$$\mathcal{L}_{\text{utility}} = \underbrace{\sum_{i \in \text{th}} \text{KL}\big(\sigma[B_i] \,\|\, M_i\big)}_{\text{think the base model's thoughts}} + \underbrace{\sum_{i \notin \text{th}} \text{KL}\big(B_i \,\|\, M_i\big)}_{\text{leave prompt and answer as they were}} + \mathcal{L}_{\text{aux}} \tag{2}$$

Anti-distillation. $\mathcal{L}_{\text{anti}}$ is composed of two terms: the first term drives every readable token's probability toward 0 within the reasoning trace, and the second term drives every encoded token's probability toward 0 within the model's answer. This ensures the exact anti-distillation goal we desire: for the reasoning traces to be purely of encoded tokens, which are unreadable, and the output to be readable, written only in non-encoded tokens. Therefore, our reasoning traces stay undistillable by student models while the output is consumable by the user.

Utility. We train the encoded model by self-distilling it from the base model, ensuring reasoning ability does not have to be re-learned, only preserved. At each reasoning trace token the base model assigns probabilities to readable tokens. When we translate that distribution through the mapping, we get the target probability distribution over encoded tokens, and the encoded model is trained to match this. The output and prompts are matched to the base model's distribution directly, so the readable parts of the sequence behave the same way.

$\mathcal{L}_{\text{aux}}$ protects utility by preventing the encoded model from a specific collapse mode of forgetting readable token generation as a result of training on encoded tokens. We add prompts in the training dataset without the `<think>` template, forcing the encoded model to skip its reasoning trace and output strictly in readable tokens, and this prevents the collapse. Collapse also appears in long reasoning traces because training is teacher-forced, so we add scheduled sampling: every loss term is computed a second time on tokens the model has autoregressively generated (Bengio et al., 2015).

4.5 TRAINING AGAINST MAPPING RECOVERY

A model trained with the losses above emits encoded reasoning and legible answers, while retaining the base model's reasoning ability. But if an attacker recovers the mapping from encoded tokens back to readable ones, they can generate encoded traces from the model, translate them to readable tokens and distill reasoning as if the defense were never there. Specifically, the attacker may use the following attacks:

Distance-based attacks: the attacker uses nearest-neighbor matching to find the readable token that corresponds the closest to the encoded token, using the model's weights or activations. The attacker can examine either the input or output embedding of the tokens and match each encoded token to the nearest readable token by this embedding using cosine similarity. The attacker can also use a fixed context and pass an encoded token, collect the activations from the model, and then do the same with the encoded token replaced by a readable token. The encoded and readable tokens that have the most similar activations can then be matched.

Prompting-based attacks: the attacker can ask the model to output readable reasoning outright, either through a system prompt instructing it to do so or in a subsequent turn of a conversation. Section 4.4's purity loss terms ensure our model is resistant to this. For the distance-based attacks,

$$\mathcal{L}_{\text{leak}} = \underbrace{w_{\text{adv}} \sum_{\ell \in \mathcal{S}} \sum_{j} \left[\cos\left(h_j^\ell, \bar{h}_{n_j}^\ell\right) - \tau \right]_+}_{\text{activations: stay away from the twin's}} + \underbrace{w_{\text{out}} \sum_{j} \left[\cos\left(W_r[j], W[n_j]\right) - \cos\left(W_r[j], W[c_j]\right) + \tau \right]_+}_{\text{output embeddings: sit nearer a decoy}} \tag{3}$$

The first term maintains the per-token activations at each layer and penalizes the cosine similarity between an encoded token's activations and those of its twin n_j, summed across layers, so the two travel through activation space differently while converging to equivalent meanings, defeating the activation-based attack. The second term penalizes the output embedding distance-based attack, as we notice that this attack is very effective with no penalty in the training loss, while the input embedding-based attack stays weak without needing a dedicated loss term. Each encoded token is assigned a readable token decoy c_j at random, kept for the whole training. This term encourages output embedding similarity between an encoded token and its decoy token c_j, ensuring that a readable token's and its mapped encoded token's embeddings are as far apart as possible. Appendix C explains this in detail.

5 EXPERIMENTS

5.1 SETUP

Models. We use two base models, Qwen3-8B (Qwen Team, 2025a) and QwQ-32B (Qwen Team, 2025b), which we trained to be encoded models as described in Section 4. Although those base models are publicly available, the assumption throughout is that a provider releases only its encoded model, not the base model it was built from.

Training data. Everything the model learns comes from the base model itself. Our training data is from the following: we collect problems, let the base solve them, keep the solutions it got right, and replace the tokens in the reasoning trace with the mapped encoded tokens. We draw roughly $6{,}400$ problems from OpenR1-Math-220k (Hugging Face, 2025) and $7{,}400$ from DeepScaleR-Preview (Luo et al., 2025), both of which contain verifiable answers. We give the base model a generation cap of 30k tokens to solve the problem, long enough that harder problems have reasoning budget. Roughly $8{,}000$ reasoning traces were correct and used as training records, relabelled into encoded tokens.

Reasoning benchmarks. Reasoning ability of the encoded models is measured on the full test sets of GSM8K ($1{,}319$), MATH-500 (500), MMLU-Pro math ($1{,}351$) and AIME 2025 (30), and on the

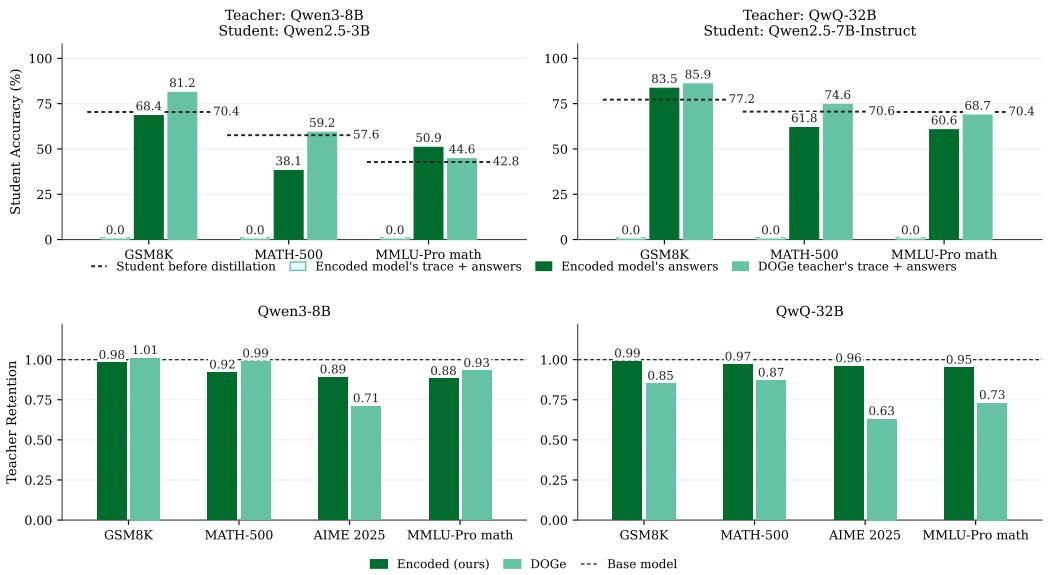


Figure 3: Our defense and DOGe (Li et al., 2025), on distillability and accuracy. **Top:** student accuracy after distilling the defended teachers, with the students' accuracy before any distillation as the dotted line. Distilling on the encoded reasoning traces gives 0 accuracy, below DOGe, and training on the encoded model's answers does not give consistent gains over the untrained student. **Bottom:** reasoning retention of the same defended teachers, with the base model as the dotted line. Encoded QwQ-32B retains far more than DOGe QwQ-32B on every benchmark, and DOGe Qwen3-8B falls far behind on AIME, underperforming on harder, longer reasoning problems.

13 non-mathematical MMLU-Pro subjects (Wang et al., 2024), 100 problems sampled from each. Results are pass@1 under the models' recommended sampling parameters, except on AIME 2025, which has only 30 problems and where results are pass@8, averaged over four evaluation runs. We use a metric called *retention*, the encoded model's accuracy divided by the base model's accuracy on the same benchmark, which measures what proportion of the original model's performance we retain.

Distillation. To measure distillation resistance, students are supervised fine-tuned on teacher-generated outputs over the s1K prompts (Muennighoff et al., 2025), then evaluated. For each encoded model we chose the student that benefited most from its base model's reasoning traces, Qwen2.5-3B for Qwen3-8B and Qwen2.5-7B-Instruct for QwQ-32B (Figure 1). The more the student model gains from the teacher's readable reasoning trace, the larger the drop in gains a good defense must produce.

Attacks. We run the distance-based attacks (across input and output embeddings and activations) and prompting-based attacks mentioned in Section 4.5. In these, we measure what percent of encoded tokens we can recover the correct mapping for. We also try decoding-based attacks, where the attacker masks all the logits (set to $-\infty$) for the encoded tokens so the encoded model is forced to use the readable tokens during thinking. The attacker can also delete the added weights in the input embedding table and the LM head to try to recover the base model. Both of these attacks are scored by accuracy retention. Appendix D defines the attacks in full.

5.2 Distilling our model directly cannot improve performance

Figure 1 shows that distilling the base teachers' reasoning trace and answers lifts the students by $+6.4$ and $+12.6$ GSM8K, $+9.8$ and $+11.2$ MATH-500, and $+13.8$ and $+6.6$ MMLU-Pro math. However, distilling their answers alone into the students changes MATH-500 by -17.0 and -5.2, GSM8K by -2.9 and $+5.4$, and MMLU-Pro math by $+7.5$ and -12.0.

The encoded reasoning trace provides no value in distillation. We add the K encoded tokens of the teacher to the student's vocabulary and distill the student using the encoded teacher's reasoning

Table 1: Every attack against the two encoded models, in one place. Distance-based attacks are scored by how much of the encoded–readable token mapping they recover; prompting-based attacks by how much of the reasoning trace comes out in readable tokens; decoding-based attacks are scored by the accuracy retention the attacker is left with. In every row, lower is better for the defense.

	encoded Qwen3-8B	encoded QwQ-32B
Distance-based (% of mapping recovered)		
input embedding	9%	0.8%
output embedding	11%	3%
activation (attacker's best layer)	0.7%	0.8%
Prompting-based (% of reasoning trace readable)		
"think in English" system prompt	0.012%	0.045%
second conversation turn	0.074%	0.044%
ordinary decoding, no attack	0.016%	0.046%
Decoding-based (% retention accuracy from base model)		
remove the encoded token embeddings	0%	0%
mask the encoded tokens at decode time	32%	30%

traces and answer. This results in a collapse of both students, as they can no longer solve any of the benchmark problems. An attacker may realize that the encoded tokens cause this collapse and may remove the reasoning trace from the generated distillation responses entirely, leaving the student to distill only from the encoded teacher's answers. However, this provides no useful distillation signal and results in gains similar to the base-teacher answer-only distillation, measured relative to the base-teacher answer-only students: $+0.9$ points on GSM8K in both students, -2.5 and -3.6 on MATH-500, and $+0.6$ and $+2.2$ on MMLU-Pro math.

Contrary to our method, DOGe-defended teachers are still distillable by students. Distilling on the DOGe-tuned teachers' reasoning traces and answers still improves the student on GSM8K and MATH-500: $+10.8$ and $+8.7$ points of GSM8K over the untrained student and $+1.6$ and $+4.0$ of MATH-500 (Figure 3, top), though on MMLU-Pro math it moves the students by only $+1.8$ and -1.7. DOGe's reasoning traces remain usable in distillation, improving students by a sizable margin, while the encoded teachers only improve students as much as the teacher's answers do.

5.3 Our defended model drops minimal utility

The encoded QwQ-32B retains 0.95–0.99 of its base accuracy across all four benchmarks (Figure 3, bottom), including 0.96 on AIME 2025, and the encoded Qwen3-8B retains 0.88–0.98. DOGe, the closest training-based distillation defense, is plotted beside ours. Its retention is much worse in larger models and in problems involving longer reasoning. On AIME 2025, its retention collapses to 0.71 on Qwen3-8B and 0.63 on QwQ-32B, against our 0.89 and 0.96. On QwQ-32B it degrades across the board, retaining 0.63–0.87 on every benchmark where ours holds 0.95–0.99.

Our encoded models should be trained and evaluated on every domain the model provider cares about. We train on mathematics only as a demonstration. We additionally evaluate out-of-domain performance on the 13 non-mathematical MMLU-Pro subjects (Appendix F) and observe some generalization to other reasoning domains.

5.4 It is hard to recover the encoded–readable token mapping

Table 1 runs every attack against the two encoded models. The distance-based attacks recover under 1% from the activations even when sweeping all 36 layers of Qwen3-8B and all 64 of QwQ-32B, and 9% and 0.8% from the input embeddings.

Distance-based attacks that use the output embeddings are the hardest to defend against, because an encoded token and its twin have similar embeddings, since they produce similar output distributions (across different vocabularies). This attack recovers 11% of the mapping on Qwen3-8B and 3% on QwQ-32B. It is also the one attack where the provider can tune its effectiveness. Table 2 shows a sweep across different weight values (w_{out}) for the output-embedding attack loss term in our training

Table 2: The output-embedding attack loss term controls the success of the output-embedding attack. On Qwen3-8B we sweep the weight of this penalty w_{out}, and on QwQ-32B we use the optimal penalty weight found in Qwen3-8B's sweep. The mapping recovery falls from 84% with $w_{\text{out}}{=}0$ to 3% with $w_{\text{out}}{=}1.5$ on Qwen3-8B. The reasoning retention slightly falls after applying this penalty, but we aim to find an optimal middle ground where the attack success is low and retention is high. The bold $w_{\text{out}}{=}1.0$ rows are the selected configuration: the encoded models evaluated throughout the paper.

	% mapping recovered from output embeddings	GSM8K	MATH-500	MMLU-Pro math	AIME 2025	MMLU-Pro non-math avg
Qwen3-8B						
$w_{\text{out}}{=}0$	84%	0.94	0.99	0.98	0.88	0.90
$w_{\text{out}}{=}0.3$	21%	0.98	0.92	0.89	0.71	0.87
$\mathbf{w_{\text{out}}{=}1.0}$ (selected)	$\mathbf{11\%}$	**0.98**	**0.92**	**0.88**	**0.89**	**0.86**
$w_{\text{out}}{=}1.5$	3%	0.99	0.94	0.93	0.80	0.90
QwQ-32B						
$w_{\text{out}}{=}0$	56%	0.98	0.98	0.99	0.94	0.88
$\mathbf{w_{\text{out}}{=}1.0}$ (selected)	$\mathbf{3\%}$	**0.99**	**0.97**	**0.95**	**0.96**	**0.83**

objective. In Qwen3-8B, mapping recovery falls from 84% with no penalty to 3% at $w_{\text{out}}{=}1.5$; at our selected setting $w_{\text{out}}{=}1.0$, retention stays within 0.12 of the base model on all four benchmarks. Using the same loss term weight on QwQ-32B shows the same thing, 56% of the mapping recovered with $w_{\text{out}}{=}0$ and 3% recovered at $w_{\text{out}}{=}1.0$.

The encoded vocabulary cannot be stripped out of the model. If the added input and output embeddings for the encoded tokens are deleted from the weights, the model collapses (Table 1): accuracy is 0.00 on both models.

Masking the encoded tokens yields a much worse model as well. During training, the model learns it is forbidden from using readable tokens for thinking, and when we mask the encoded tokens' logits, the reasoning trace is strictly readable tokens but is much poorer, and retention falls to 0.32 on Qwen3-8B and 0.30 on QwQ-32B.

Asking the encoded model to explain itself does not work. An attacker can try to simply ask for the reasoning trace in readable text, either as a system prompt or as the next turn of a conversation. Neither works (Table 1, middle block): under both attacks the fraction of readable tokens in the reasoning trace stays low, 0.074% at most. Emitting only encoded tokens for reasoning and chain-of-thought is not a behavior that prompting can change.

6 Conclusion

We asked whether an open-weight reasoning model can be released without it being distillable. We show that in distillation, the reasoning trace is what truly improves the student, and our goal is to hide the reasoning trace in the open-weight model. We do this by generating the reasoning trace in unreadable encoded tokens and the output in readable tokens, while retaining high reasoning performance.

We found two things worth pointing out beyond the defense itself. First, the models learned to reason in tokens that did not exist before our training, with little loss in accuracy. Second, a defense for open-weight models cannot rely on keeping anything secret, because the attacker has access to everything we release. The only thing a defense can do is make distillation more expensive and complex to execute. In our case, simple attacks recover at most 11% of the encoded–readable token mapping. The model provider still holds the mapping, so they can read every reasoning trace themselves or share it with an auditor. The reasoning traces are only unreadable to everyone else who does not have this mapping.

We hope this work is a starting point for distillation defenses aimed at open-weight models. The gap between open-weight and frontier models is closing, which makes open-weight models increasingly powerful teachers, and therefore their protection increasingly urgent.

USE OF AI

We used generative AI tools to help write the code for training, evaluation, and to run our experiments. The code and its outputs were reviewed by the authors. Model-generated data is also part of our method since the distillation training data consists of teacher model generated reasoning traces. We additionally used generative AI tools to build figures and tables from measured results and to polish writing. We did not use generative AI to propose hypotheses, design the research methodology, or interpret results. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2015.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zibo Diao, Jingchu Gai, Xinyue Ai, Zhang Zhang, Zhenyu He, and Di He. Lossless anti-distillation sampling. *arXiv preprint arXiv:2605.18829*, 2026.

Jiayu Ding, Lei Cui, Li Dong, Nanning Zheng, and Furu Wei. Information-preserving reformulation of reasoning traces for antidistillation. *arXiv preprint arXiv:2510.11545*, 2025.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

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

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Hugging Face. Open R1: A fully open reproduction of DeepSeek-R1. `https://github.com/huggingface/open-r1`, 2025.

Tomek Korbak, Mikita Balesni, Elizabeth Barnes, et al. Chain of thought monitorability: A new and fragile opportunity for AI safety. *arXiv preprint arXiv:2507.11473*, 2025.

Pingzhi Li, Zhen Tan, Huaizhi Qu, Huan Liu, and Tianlong Chen. DOGe: Defensive output generation for LLM protection against knowledge distillation. *arXiv preprint arXiv:2505.19504*, 2025.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Erran Li, Raluca Ada Popa, and Ion Stoica. DeepScaleR: Surpassing O1-Preview with a 1.5B model by scaling RL. `https://github.com/agentica-project/deepscaler`, 2025.

Xinhang Ma, William Yeoh, Ning Zhang, and Yevgeniy Vorobeychik. Protecting language models against unauthorized distillation through trace rewriting. *arXiv preprint arXiv:2602.15143*, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

OpenAI. Learning to reason with LLMs. `https://openai.com/index/learning-to-reason-with-llms/`, 2024. System card and release note for the o1 model series; the raw chain of thought is withheld and a model-written summary is shown instead.

Jakub Pachocki. An alien mind. https://openai.com/index/an-alien-mind/, 2026. OpenAI essay, September 2026, on the diminishing reliability of chain-of-thought monitoring.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Qwen Team. QwQ-32B: Embracing the power of reinforcement learning. https://qwenlm.github.io/blog/qwq-32b/, 2025b.

Yash Savani, Asher Trockman, Zhili Feng, Avi Schwarzschild, Alexander Robey, Marc Finzi, and J. Zico Kolter. Antidistillation sampling. *arXiv preprint arXiv:2504.13146*, 2025.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks)*, 2024.

A Training details

Every run uses AdamW with cosine decay after a linear warmup, gradient clipping at 1.0, and bf16 compute with fp32 master weights. Pre-trained weights take a learning rate of 10^{-5} (full-weight training) or 10^{-4} (low-rank adapters); the newly added weights in the input and output embedding matrices take a learning rate of 10^{-3}, two orders of magnitude higher, because they start from noise. Loss weights are w_{anc}=1, w_{caus}=1 , w_{adv}=20 with cosine margin τ=0.1, and w_{out} as swept in Section 5. The purity loss weight ramps linearly from 0 to 1 over the first fifth of training, and w_{out} similarly. Qwen3-8B is trained with all weights unfrozen and QwQ-32B, whose body is large, through rank-128 low-rank adapters (Hu et al., 2022) on every attention and MLP projection.

Training starts in stages of progressively longer context windows, from 8k tokens up to 32k. Each stage begins from the weights of the stage before it, so the model first learns the encoded-reasoning behavior on short sequences, where training is cheap, and is then extended to longer and longer traces. Very long sequences make the auxiliary passes expensive: the scheduled-sampling pass and the adversary pass are each an additional forward pass on top of the main one, so together they can double or triple the memory of a training step. We therefore give each of these two passes its own maximum sequence length. When a training sequence is longer than a pass's maximum, that pass is simply skipped for that step; the main losses are still applied to the sequence, and only the auxiliary term is missing from that one step. One implementation detail matters here: training is data-parallel across four GPUs, and the GPUs synchronize with each other inside every forward and backward pass. A forward pass runs on all GPUs, or on none, for any given step.

B Training data and the mapping

Everything the model learns comes from the base model itself. We collect problems, let the base model solve them, keep the solutions it got right, and rewrite the thinking span of those solutions in the encoded vocabulary. These rewritten traces are the entire training corpus.

We draw roughly $6{,}400$ problems from OpenR1-Math-220k (Hugging Face, 2025) and $7{,}400$ from DeepScaleR-Preview (Luo et al., 2025), both of which supply short answers that can be verified automatically. The base model solves each problem once with a generation budget of up to 30k tokens, long enough that competition-length chains are not cut off, and we discard every trace whose boxed answer does not verify. A trace the base model got wrong is discarded rather than corrected, so the target behaviour is exactly the base model's own correct behaviour.

The encoded vocabulary is derived from the base model's own token frequency rather than chosen in advance. We tokenize every kept reasoning trace, count how often each token id occurs *inside* this, and take the token ids in decreasing frequency until they cover at least 99.9% of all reasoning trace token occurrences. This gives a set of K readable ids, $\{n_1, \ldots, n_K\}$, the vocabulary the base model actually uses during reasoning, which is much smaller than the full vocabulary ($K = 7{,}948$ of $151{,}936$ for Qwen3-8B). We call n_j the readable *twin* of the j-th encoded token.

For each of these, we create a new token id $r_j = V + j$, where V is the size of the original vocabulary. The mapping

$$\sigma : n_j \mapsto r_j, \qquad j = 1, \ldots, K,$$

is a bijection between the readable tokens in the reasoning vocabulary and the new ids. It is drawn once, uniformly at random, before training and never changed. Which encoded id stands for which readable token carries no information an attacker could guess, and the model has no reason to prefer one 1-to-1 mapping over another.

The added embeddings cost $2Kd$ parameters, about 65M for Qwen3-8B (0.8% of the model) and 84M for QwQ-32B (0.25%).

C THE TRAINING OBJECTIVE IN FULL

This appendix expands the objective of Sections 4.4 and 4.5 term by term. The total loss is the sum of the three groups presented in the main text:

$$\mathcal{L} \;=\; \mathcal{L}_{\text{anti}} \;+\; \mathcal{L}_{\text{utility}} \;+\; \mathcal{L}_{\text{leak}}. \tag{4}$$

As in the main text, B is the frozen base model, M the encoded model being trained, and M_i its next-token distribution at position i. For a relabelled training record, write th, ans and pr for the sets of positions inside the thinking span, in the answer, and in the prompt and structural markers.

Anti-distillation.

$$\mathcal{L}_{\text{anti}} \;=\; -\lambda_{\text{pur}} \sum_{i \in \text{th}} \log \sum_{v \geq V} M_i(v) \;-\; \lambda_{\text{pur}} \sum_{i \in \text{ans}} \log \sum_{v < V} M_i(v). \tag{5}$$

Both terms act on a *total* probability mass rather than on any single token. At a thinking position, the inner sum $\sum_{v \geq V} M_i(v)$ is the probability the model assigns to the entire encoded half of the vocabulary; driving its logarithm to zero pushes that mass toward 1 without dictating *which* encoded token to emit, so the model stays free to choose its own thought while the readable half of the vocabulary is switched off. The second term is the mirror image on the answer positions, where the encoded half is switched off instead, so the answer the user reads stays readable. The weight λ_{pur} ramps linearly from 0 to 1 over the first fifth of training: the model first learns *what* to say, from the utility group below, before it is told what *not* to say. Turning the purity pressure on too early suppresses readable tokens before the encoded ones carry any meaning.

Utility.

$$\mathcal{L}_{\text{utility}} \;=\; \sum_{i \in \text{th}} \mathrm{KL}\big(\sigma[B_i] \,\|\, M_i\big) \;+\; \sum_{i \in \text{ans} \cup \text{pr}} \mathrm{KL}\big(B_i \,\|\, M_i\big) \;+\; \mathcal{L}_{\text{aux}}. \tag{6}$$

The first sum is retention itself. At every thinking position the base model assigns a probability to each readable token; $\sigma[B_i]$ is that same distribution with every readable token's probability moved onto its corresponding encoded twin. We match the full distribution with a KL divergence rather than a cross-entropy on the single sampled token, so the model inherits the base model's uncertainty and alternatives at each step, not only its top choice. The second sum applies the same divergence, untranslated, to the prompt and answer positions, keeping the readable parts of the sequence behaving exactly as they did in the base model.

$\mathcal{L}_{\text{aux}}$ adds three protections that the two sums above do not provide:

$$\mathcal{L}_{\text{aux}} \;=\; w_{\text{anc}} \sum_{i \in \text{anchor}} \mathrm{KL}\big(B_i \,\|\, M_i\big) \;+\; w_{\text{caus}} \sum_{i \in \text{ans}} \mathrm{KL}\big(\tilde{B}_i \,\|\, \tilde{M}_i\big) \;+\; \mathcal{L}_{\text{SS}}. \tag{7}$$

The *anchor* term keeps the model from forgetting ordinary readable token generation while every other term reshapes its behaviour around the encoded alphabet. The *causality* term replaces the reasoning trace with uniformly random encoded ids and requires the encoded model's answer distribution to match what the base model answers after reading the decryption of that same corrupted trace. A model that ignored its own reasoning and answered from the prompt alone would be unaffected by the corruption and incur a very high loss on this term, so it forces the encoded model to actually

use the encoded reasoning trace. The *scheduled sampling* term $\mathcal{L}_{\text{SS}}$ recomputes every term above a second time on a context the model has partly generated itself (Bengio et al., 2015). Teacher forcing alone never exposes the model to its own mistakes, so a model can match every training record and still degrade over a long free generation, but computing the loss on self-generated context closes that gap.

Mapping-recovery defense.

$$\mathcal{L}_{\text{leak}} \;=\; w_{\text{adv}} \sum_{\ell \in \mathcal{S}} \sum_{j} \big[\, \cos\big(h_j^{\ell}, \bar{h}_{n_j}^{\ell}\big) - \tau \big]_+ \\ +\; w_{\text{out}} \sum_{j} \big[\, \cos\big(W_r[j], W[n_j]\big) - \cos\big(W_r[j], W[c_j]\big) + \tau \big]_+ . \tag{8}$$

The first term is an adversary run inside the training loop. At a sampled set of layers $\mathcal{S}$ it penalizes an encoded token whose hidden state h_j^{ℓ} is similar to $\bar{h}_{n_j}^{\ell}$, the running average activation of its twin. The averages are computed from the *encoded model itself*, not the base model, so the attack is minimized during training. The second term is the output-embedding attack loss term: each encoded token is assigned a fixed readable token decoy c_j, drawn at random and kept for the whole training, and its output embedding is pushed until the decoy's embedding is closer to it than the twin's. In both terms, τ=0.1 is a margin that defines defended enough: the hinge $[\,\cdot\,]_+$ stops pushing once the activation similarity falls below τ, and once the decoy wins the output-embedding comparison by at least τ. No term acts on the input embeddings: an encoded token's input embedding is the only signal telling the body which token it is reading, so moving it costs retention directly, while the output embeddings and hidden states have room to move at fixed behaviour.

D Attack in-depth definitions

Weight-based. The input-embedding attack matches each encoded token r_j to the readable token whose input embedding is most similar,

$$\hat{n}_j \;=\; \arg\max_{v \in V} \cos\big(E[r_j],\, E[v]\big),$$

and succeeds on r_j when $\hat{n}_j$ is exactly the twin n_j; the output-embedding attack is the same on the output embeddings W. Both search the *full* readable token vocabulary.

Activation-based. We generate reasoning traces from the encoded model and the base model on the same prompts, collect the activations after every layer at every thinking position, and form per-token averages for readable tokens (from the base traces) and for encoded tokens (from the encoded traces). The full activation attack computes this at every layer and keeps, per token, the layer whose match is most confident. This attack can be formulated as:

$$\hat{n}_j^{\ell} \;=\; \arg\max_{n} \cos\big(\bar{h}_{r_j}^{\ell},\, \bar{h}_{n}^{\ell}\big), \qquad \ell_j^{*} \;=\; \arg\max_{\ell} \cos\big(\bar{h}_{r_j}^{\ell},\, \bar{h}_{\hat{n}_j^{\ell}}^{\ell}\big),$$

where $\bar{h}_{r_j}^{\ell}$ and $\bar{h}_{n}^{\ell}$ are the average activations at layer ℓ of encoded token r_j (from the encoded model's traces) and of readable token n (from the base model's traces). At each layer the attack matches r_j to the readable token with the most similar average activation, giving $\hat{n}_j^{\ell}$; its final guess for r_j is $\hat{n}_j^{\ell_j^{*}}$, the match at the layer where that similarity is highest, and the attack succeeds when this guess equals the twin n_j. The candidate set for n ranges over the readable tokens that appear in the generated base traces.

E Retention, raw accuracies

Table 3 gives the raw benchmark accuracies behind Figure 3 (bottom), for every penalty weight we trained and for DOGe applied to the same base models.

Table 3: Retention on the full test sets as well as raw accuracy for the models. GSM8K (1,319), MATH-500 (500) and MMLU-Pro math (1,351) use a 16k-token budget; AIME 2025 (30) uses 30k and is reported as pass@8 at temperature 0.6, averaged over four evaluation runs. DOGe rows (Li et al., 2025) are that defense applied to the same base. Figure 3 plots the w_{out}=1.0 models, the selected configuration, and Table 2 summarizes this sweep against the mapping recovered.

Model		GSM8K	MATH-500	AIME 2025	MMLU-Pro math
Qwen3-8B	base	0.955	0.948	0.708	0.927
encoded	w_{out}=0	0.897 (0.94)	0.938 (0.99)	0.625 (0.88)	0.906 (0.98)
encoded	w_{out}=0.3	0.932 (0.98)	0.870 (0.92)	0.500 (0.71)	0.828 (0.89)
encoded	w_{out}=1.0	0.939 (0.98)	0.872 (0.92)	0.633 (0.89)	0.815 (0.88)
encoded	w_{out}=1.5	0.945 (0.99)	0.890 (0.94)	0.567 (0.80)	0.866 (0.93)
DOGe		0.960 (1.01)	0.942 (0.99)	0.500 (0.71)	0.862 (0.93)
QwQ-32B	base	0.948	0.960	0.692	0.927
encoded	w_{out}=0	0.929 (0.98)	0.940 (0.98)	0.650 (0.94)	0.916 (0.99)
encoded	w_{out}=1.0	0.940 (0.99)	0.932 (0.97)	0.667 (0.96)	0.876 (0.95)
DOGe		0.801 (0.85)	0.838 (0.87)	0.433 (0.63)	0.677 (0.73)

F Out-of-domain retention by subject

Averaged over all 1,300 problems, the encoded Qwen3-8B retains 0.86 of the original Qwen3-8B on non-math MMLU-Pro problems and the encoded QwQ-32B 0.83. The spread in retention across subjects is wider than the spread across models, and the two weakest subjects, law and engineering, are weak for both of our models.

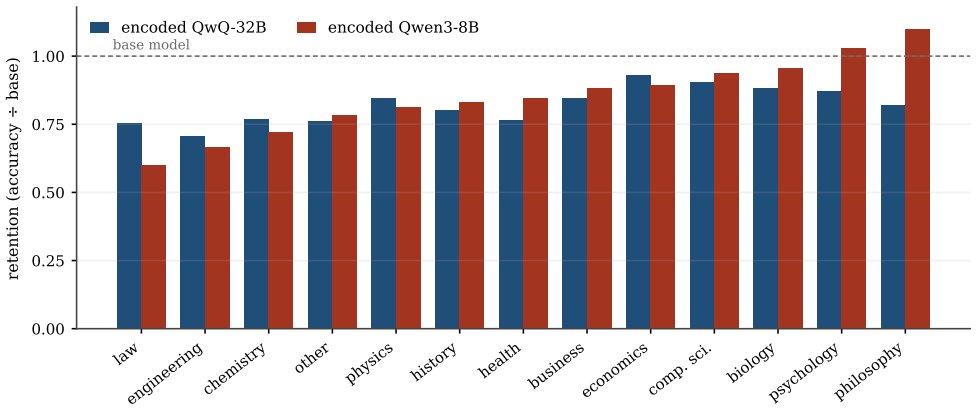


Figure 4: Retention on each of the 13 out-of-domain non-mathematical MMLU-Pro subjects, ordered by the encoded Qwen3-8B's retention. The mean row of Table 4 gives the averages.

Table 4: Retention on the 13 non-mathematical MMLU-Pro subjects. None of these subjects is part of the mathematical training distribution.

	Qwen3-8B					QwQ-32B		
Category	base	w_{out}=0	w_{out}=0.3	w_{out}=1.0	w_{out}=1.5	base	w_{out}=0	w_{out}=1.0
physics	0.86	0.72 (0.84)	0.59 (0.69)	0.70 (0.81)	0.73 (0.85)	0.84	0.77 (0.92)	0.71 (0.85)
law	0.40	0.33 (0.82)	0.27 (0.68)	0.24 (0.60)	0.28 (0.70)	0.53	0.43 (0.81)	0.40 (0.75)
biology	0.89	0.88 (0.99)	0.87 (0.98)	0.85 (0.96)	0.86 (0.97)	0.93	0.89 (0.96)	0.82 (0.88)
business	0.86	0.79 (0.92)	0.78 (0.91)	0.76 (0.88)	0.80 (0.93)	0.90	0.81 (0.90)	0.76 (0.84)
chemistry	0.82	0.71 (0.87)	0.66 (0.80)	0.59 (0.72)	0.72 (0.88)	0.86	0.69 (0.80)	0.66 (0.77)
computer science	0.78	0.70 (0.90)	0.76 (0.97)	0.73 (0.94)	0.73 (0.94)	0.84	0.82 (0.98)	0.76 (0.90)
economics	0.83	0.82 (0.99)	0.81 (0.98)	0.74 (0.89)	0.82 (0.99)	0.85	0.80 (0.94)	0.79 (0.93)
engineering	0.69	0.54 (0.78)	0.46 (0.67)	0.46 (0.67)	0.54 (0.78)	0.65	0.48 (0.74)	0.46 (0.71)
health	0.77	0.67 (0.87)	0.70 (0.91)	0.65 (0.84)	0.66 (0.86)	0.77	0.69 (0.90)	0.59 (0.77)
history	0.59	0.54 (0.92)	0.50 (0.85)	0.49 (0.83)	0.49 (0.83)	0.66	0.54 (0.82)	0.53 (0.80)
other	0.65	0.58 (0.89)	0.53 (0.82)	0.51 (0.78)	0.61 (0.94)	0.75	0.67 (0.89)	0.57 (0.76)
philosophy	0.51	0.50 (0.98)	0.52 (1.02)	0.56 (1.10)	0.52 (1.02)	0.66	0.53 (0.80)	0.54 (0.82)
psychology	0.69	0.66 (0.96)	0.66 (0.96)	0.71 (1.03)	0.67 (0.97)	0.78	0.73 (0.94)	0.68 (0.87)
mean	0.718	0.649 (0.90)	0.624 (0.87)	0.615 (0.86)	0.648 (0.90)	0.771	0.681 (0.88)	0.636 (0.83)