# ZERO GAP IS NOT RESTORATION: STRATIFIED PER-QUESTION PROBABILITY EVALUATION AND STEP-WISE MITIGATION OF BENCHMARK CONTAMINATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Test items from public benchmarks leak into pretraining corpora of large language models (LLMs), and once memorized they would inflate evaluation scores. **Contamination mitigation evaluation** intervenes in decoding to suppress memorization and restore a contaminated model's genuine capability. However, its prevailing metric, the **G-AP** (**G**ap of **A**ggregate **P**erformance), has three flaws. First, a discrete correct/incorrect readout cannot characterize per-question performance. Second, averaging before differencing lets over- and under-suppression cancel. Third, uniform per-question weighting rewards restoring the clean model's most frequent solve probabilities while neglecting the rarer ones, which differ in how hard they are to restore. To address these issues, we propose **SA-PPG** (**S**tratified **A**ggregate of **P**er-question **P**robability **G**aps), which estimates each question's solve probability by sampling, differences it against the clean model per question, and aggregates the resulting gaps within groups defined by the clean model's solve probability. Further, on the mitigation side, since existing strategies first estimate where contamination lies and then act on that estimate, they are only as accurate as the estimate. Our strategy, **RailCap**, instead judges contamination during generation: whenever a sample falls back onto the greedy trajectory, the next trajectory token is capped to the runner-up, and suppression accumulates until the response distribution is sufficiently dispersed. Extensive experiments show that SA-PPG excel G-AP in measuring the restoration of prior mitigation strategies, and RailCap attains the lowest SA-PPG in all nine settings (three models $\times$ three contamination domains). Code is available at `https://anonymous.4open.science/r/railcap`.

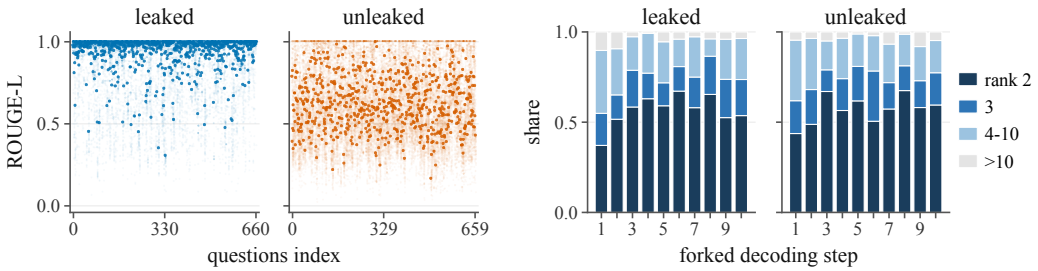
## 1 INTRODUCTION

Test items from public benchmarks almost inevitably find their way into large-scale pretraining corpora of large language models (LLMs) (Brown, 2020; Elazar et al., 2024). Once memorized, they inflate evaluation scores and strip benchmarks of their power to measure genuine capability (Magar & Schwartz, 2022; Zhou et al., 2023; Balloccu et al., 2024). Existing responses fall into two classes. **Dataset-side** approaches rebuild, rewrite, or dynamically generate benchmarks to sidestep contamination (Li et al., 2024; Zhang et al., 2024a; Zhu et al., 2024; Bai et al., 2023), but they are costly, and the new data may be exposed to leakage again once released. **Contamination mitigation evaluation** builds no new dataset. It keeps the benchmark at risk and intervenes in decoding to suppress memorization and restore the model's genuine capability (Dong et al., 2024; Hou et al., 2025; Zhu et al., 2025). Whether a mitigation strategy's claimed restoration holds, however, can only be checked against a reliable metric.

Different mitigation strategies have been investigated with the **G**ap of **A**ggregate **P**erformance (G-AP) metric (Dong et al., 2024; Hou et al., 2025; Zhu et al., 2025). In particular, G-AP assigns each question a performance readout, usually a discrete 1/0 mark for correct or incorrect. It averages the readouts over the dataset for the contaminated model under mitigation and takes the absolute difference from the clean model's average, where a smaller gap is read as better restoration. This metric has three problems. The first concerns *representation*. Since responses sampled for the same question may disagree on correctness, a single discrete mark poorly captures per-question performance. *Solve probability*, the probability that a sampled response is correct, is a far more



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.




(a) overlap with the greedy trajectory

(b) rank of the clean model's token

Figure 1: Generation behavior of the contaminated model on Llama-2 (GSM8K, 1,319 questions, $m = 50$ samples per question, $T = 0.7$). (a) ROUGE-L between each sampled response and the greedy trajectory of the same question; faint dots are individual samples and solid dots per-question means. (b) Rank, under the contaminated model, of the clean model's next token when the contaminated model is teacher-forced on the clean model's greedy prefix. Only the steps within the first ten decoding steps at which the two models disagree are counted.

stable readout than a single mark and should therefore represent per-question performance. The second concerns *aggregation*. Averaging before differencing lets over- and under-suppression cancel: performance wrongly suppressed on one question can offset performance wrongly inflated on another, so a zero gap does not mean that every question is restored. The third concerns *weighting*. Since every question enters the average with the same weight, the metric is dominated by the solve probabilities that occur most often under the clean model.

To address the above issues, we present a novel **S**tratified **A**ggregate of **P**er-question **P**robability **G**aps (SA-PPG) metric to appropriately measure the contamination mitigation strategies. In particular, to correct the first problem, we estimate each question's solve probability from samples and using it as the readout of G-AP, yielding the **G**ap of **A**ggregate **P**er-question **P**robabilities (G-APP). G-APP still averages before differencing, so over- and under-suppression can still cancel. Further, **A**ggregate of **P**er-question **P**robability **G**aps (A-PPG) is employed to correct the second problem. It differences each question's solve probability against the clean model to obtain per-question probability gaps (PPGs), and only then averages their absolute values. A-PPG reads zero if and only if every question is perfectly restored. Its plain average, however, still weights every question equally, so A-PPG inherits the third problem from G-AP.

Further, considering a weak clean model like Llama-2 model on GSM8K, whose solve probabilities are generally low, a mitigation strategy can then attain a low A-PPG simply by making the model fail, degrading its performance instead of restoring it precisely. We take All-Zero, which does this on every question, as the typical case. Since low solve probability questions make up a large share of the dataset, the averaging scheme dilutes All-Zero's PPGs on the high solve probability questions. Hence, the weaker the clean model, the closer its A-PPG comes to a perfect zero. In particular, on GSM8K All-Zero already ranks ahead of two prior mitigation strategies (§5.2). In this regard, a sound restoration metric should separate degrading the model from restoring it, which rules out giving a solve-probability level more weight merely because it holds more questions. We therefore propose **S**tratified A-PPG metric (SA-PPG), which groups questions by the clean model's solve probability, averages PPGs within each group, and then averages across groups. SA-PPG grounds restoration in every single question: a mitigation strategy must adjust each question's solve probability by the right amount. It ranks All-Zero last (Table 1), also revealing that G-AP substantially overestimates the restoration of prior strategies.

In addition, existing contamination mitigation strategies (Dong et al., 2024; Hou et al., 2025; Zhu et al., 2025) all proceed in two steps. They first estimate where the contamination lies (which questions are leaked, which responses stem from memorization, or which neurons encode it) and then operate on the estimated part. The intervention is therefore only as correct as the estimate: whatever the estimate misses keeps its inflated performance, and whatever it wrongly flags is damaged unnecessarily. Here, we instead examine the contaminated model's own generation behavior (Figure 1) and make two observations. First, on leaked questions its sampled responses collapse onto its own greedy trajectory, whereas on unleaked questions they disperse over many paths. Second, at the decoding

steps where the clean and contaminated models next diverge, the token chosen by the clean model is the contaminated model's runner-up in about half of the cases.

Building on the above observations, we further introduce a new mitigation method **RailCap**. At every decoding step, RailCap checks whether the sample has fallen back onto the greedy trajectory. If it has, RailCap caps the probability of the next trajectory token to that of the runner-up. The clean model's next choice then most likely sits at the head of the flattened candidates. The judgment of contamination thus shifts from an up-front estimate to **step-wise supervision during generation**. Each fall-back triggers one suppression, and suppressions accumulate until the response distribution is sufficiently dispersed. How much intervention each question receives is therefore decided online, step by step, rather than fixed in advance by a one-shot estimate.

Our contributions are as follows:

1. We propose **SA-PPG**, a restoration metric that fixes the representation, aggregation, and weighting problems of the prevailing G-AP. It estimates each question's solve probability by sampling, differences it against the clean model per question, and gives equal weight to groups defined by the clean model's solve probability.
2. We propose **RailCap**, a contamination mitigation strategy that caps the next trajectory token to the runner-up whenever a sample falls back onto the greedy trajectory. Step-wise supervision during generation, rather than a one-shot estimate, decides how much intervention each question receives.
3. **SA-PPG** reveals that G-AP systematically overestimates the restoration of prior mitigation strategies, and **RailCap** attains the lowest SA-PPG in all nine model–domain settings.

## 2 Related Work

Test items from public benchmarks almost inevitably end up in large-scale pretraining corpora of LLMs(Brown, 2020; Dodge et al., 2021; Elazar et al., 2024). Memorized items inflate evaluation scores and mask genuine capability (Magar & Schwartz, 2022; Zhou et al., 2023; Balloccu et al., 2024), prompting calls to protect test data (Jacovi et al., 2023; Deng et al., 2024). **Detection** asks whether a model has seen the evaluation data, using min-k% probability over low-probability tokens (Shi et al., 2024), perplexity (Li, 2023), divergence calibration (Zhang et al., 2024b), temporal cues (Golchin & Surdeanu, 2024), black-box calibration (Ye et al., 2024), or, in CDD, consistency between sampled and greedy outputs (Dong et al., 2024). **Dataset-side** work sidesteps leaked benchmarks by rebuilding same-distribution questions for GSM8K (Cobbe et al., 2021) (GSM-Plus (Li et al., 2024), GSM1k (Zhang et al., 2024a)), rewriting existing questions (Zhu et al., 2024), or having models generate test items dynamically (Bai et al., 2023; Yu et al., 2024; Li et al., 2025). Rebuilding and dynamic generation are costly, however, and newly released data faces the same risk of leakage.

**Contamination mitigation evaluation** is a third line of work. It builds no new dataset and instead intervenes in decoding to suppress memorization and restore genuine capability on benchmarks at risk of leakage (Dong et al., 2024; Hou et al., 2025; Zhu et al., 2025). Checking whether a mitigation strategy's claimed restoration holds requires a reliable metric, which is one of the central subjects of this paper.

These mitigation strategies use **g**aps of **a**ggregate **p**erformance (G-AP) as restoration metric. It assigns every question a performance readout, averages over the dataset, and differences against the clean model's average, reading a smaller gap as better restoration. They differ only in the readout. LNE-blocking marks the greedy response 0/1 (Hou et al., 2025), and shortcut neuron patching marks a single sampled response 0/1 (Zhu et al., 2025). TED is the exception: because its strategy already operates on a set of sampled responses, it filters out near-greedy samples and estimates solve probability (Chen et al., 2021) over the remainder (Dong et al., 2024). This choice follows from its sampling-based mitigation rather than from a comparison of readouts. To our knowledge, no prior work has compared discrete 0/1 and probabilistic readouts. We compare the two readouts and adopt the probabilistic one, because a single-sample 0/1 readout is not reproducible even between two evaluations of the same model. Our proposed SA-PPG further fixes the aggregation and weighting problems. Differencing per question before averaging keeps over- and under-suppression from cancelling, and giving equal weight to groups defined by the clean model's solve probability keeps its most frequent values from dominating.

On the mitigation side, existing strategies all first estimate where the contamination lies and then operate on the estimated part, at different granularities. TED estimates contaminated responses: after sampling, it filters out suspected memorized samples by their edit distance to the greedy decoding (Dong et al., 2024). LNE-blocking estimates contaminated questions: it gauges each question's degree of contamination with length-normalized entropy and sets the blocking strength accordingly (Hou et al., 2025). Shortcut neuron patching estimates contaminated neurons: it locates shortcut neurons by contrastive and causal analysis and suppresses them (Zhu et al., 2025). In all three, the intervention is only as correct as the estimate. What the estimate misses keeps its inflated performance, and what it wrongly flags is damaged unnecessarily. Instead, our proposed RailCap removes this dependence by replacing the up-front estimate with step-wise supervision during generation.

# 3 SA-PPG: A Stratified Per-Question Restoration Metric

## 3.1 Setup and the Prevailing Metric

Let $D = \{q_1, \ldots, q_N\}$ be the evaluation dataset, $M_{\mathrm{cl}}$ the clean model, and $M_{\mathrm{co}}$ the contaminated model. We write $M_{\mathrm{co}}^{s}$ for $M_{\mathrm{co}}$ under a mitigation strategy $s$, which intervenes in its decoding. Evaluation first defines a performance readout $r_M(q) \in [0, 1]$, the value taken as model $M$'s performance on a single question $q$. Most existing work adopts a discrete readout, sampling a response $o$ from $M$ and marking it 1 if correct and 0 otherwise:

$$r_M^{0/1}(q) = \mathbb{1}\big[\, o \text{ solves } q \,\big], \quad o \sim M(\cdot \mid q). \tag{1}$$

Averaging the readouts over the dataset for each model and taking the absolute difference yields the **G**ap of **A**ggregate **P**erformance (G-AP):

$$\text{G-AP}(s) = \Big| \tfrac{1}{N} \sum_{q \in D} r_{M_{\mathrm{co}}^{s}}^{0/1}(q) - \tfrac{1}{N} \sum_{q \in D} r_{M_{\mathrm{cl}}}^{0/1}(q) \Big|. \tag{2}$$

Prior work reads a smaller G-AP as better restoration. G-AP has three problems: its readout is a single 0/1 mark per question, it averages before differencing, and it weights every question equally $(1/N)$. We fix these problems one at a time, each with a new metric. G-APP replaces the readout with the solve probability (§3.2), and A-PPG then differences per question before averaging, which shows analytically that a zero G-AP does not certify restoration (§3.3). SA-PPG finally gives equal weight to groups defined by the clean model's solve probability (§3.4).

## 3.2 Gap of Aggregate Per-question Probabilities (G-APP)

To fix the representation problem, we adopt the *solve probability* as the readout:

$$r_M^{\mathrm{prob}}(q) = \mathbb{E}\big[r_M^{0/1}(q)\big], \tag{3}$$

where $r^{0/1}$ is a single Bernoulli draw and $r^{\mathrm{prob}}$ is its expectation. In practice, we sample $m$ responses independently from $M$ and, if $c$ of them are correct, estimate the readout as $\hat{r}_M^{\mathrm{prob}}(q) = c/m$.

Using $r^{\mathrm{prob}}$ as the readout of G-AP yields the **G**ap of **A**ggregate **P**er-question **P**robabilities (G-APP):

$$\text{G-APP}(s) = \Big| \tfrac{1}{N} \sum_{q \in D} r_{M_{\mathrm{co}}^{s}}^{\mathrm{prob}}(q) - \tfrac{1}{N} \sum_{q \in D} r_{M_{\mathrm{cl}}}^{\mathrm{prob}}(q) \Big|. \tag{4}$$

G-APP fixes the readout but, like G-AP, still averages before differencing.

## 3.3 Aggregate of Per-question Probability Gaps (A-PPG)

### 3.3.1 Definition

To keep over- and under-suppression from cancelling, we difference the two models on each question first and only then average the absolute values of these differences. We first define the **P**er-question **P**robability **G**ap (PPG) on question $q$ as

$$\Delta_s(q) = r_{M_{\mathrm{co}}^{s}}^{\mathrm{prob}}(q) - r_{M_{\mathrm{cl}}}^{\mathrm{prob}}(q). \tag{5}$$

We then average its absolute value over the dataset to obtain the **A**ggregate of **P**er-question **P**robability **G**aps (A-PPG):

$$\text{A-PPG}(s) = \frac{1}{N} \sum_{q \in D} \big|\Delta_s(q)\big|. \tag{6}$$

Because every summand $|\Delta_s(q)|$ is non-negative, A-PPG vanishes exactly when all summands do:

$$\text{A-PPG}(s) = 0 \iff \Delta_s(q) = 0,\ \forall q \in D. \tag{7}$$

That is, **A-PPG reads zero if and only if every question is fully restored**.

#### 3.3.2 Relation to G-APP

In terms of the PPG $\Delta_s$, G-APP can be rewritten as

$$\text{G-APP}(s) = \frac{1}{N} \Big| \sum_{q \in D} \Delta_s(q) \Big|. \tag{8}$$

Both G-APP and A-PPG are built from the same PPG $\Delta_s$ and differ only in the order of the absolute value and the average. To make the difference explicit, we decompose A-PPG by the sign of $\Delta_s$:

$$\text{A-PPG}(s) = \Delta_s^+ + \Delta_s^-, \tag{9}$$

$$\Delta_s^+ = \frac{1}{N} \sum_{q \in D} \max\big(\Delta_s(q),\, 0\big), \tag{10}$$

$$\Delta_s^- = \frac{1}{N} \sum_{q \in D} \max\big(-\Delta_s(q),\, 0\big), \tag{11}$$

where $\Delta_s^+$ is the contribution of under-suppression (inflated performance that the strategy leaves in place) and $\Delta_s^-$ that of over-suppression (performance driven below the clean model). In the same notation,

$$\text{G-APP}(s) = \big|\Delta_s^+ - \Delta_s^-\big|, \tag{12}$$

and G-APP$(s) \le$ A-PPG$(s)$ follows immediately: A-PPG adds the two non-negative components, whereas G-APP subtracts them.

The key difference lies in the zero set. A-PPG is zero if and only if $\Delta_s^+ = \Delta_s^- = 0$, that is, every question is restored. G-APP is zero once $\Delta_s^+ = \Delta_s^-$, which cancellation alone can achieve. Suppose, for example, that half of the questions are over-suppressed ($\Delta_s(q) = -\delta$) and the other half under-suppressed ($\Delta_s(q) = +\delta$) for some $\delta > 0$. Then $\Delta_s^+ = \Delta_s^- = \delta/2$, so G-APP $= 0$ while A-PPG $= \delta$: the gap vanishes, yet not a single question is restored. A zero G-APP therefore does not certify restoration.

Similarly, since G-AP shares this aggregation order and adds sampling noise, a zero G-AP does not certify restoration either. Taken together, this analysis shows that **G-AP is not a reliable restoration metric, and per-question restoration should be judged by A-PPG**.

### 3.4 Stratified Aggregate of Per-question Probability Gaps (SA-PPG)

G-AP and A-PPG both average over questions with the same weight $1/N$. Uniform per-question weighting rewards restoring the clean model's most frequent solve probabilities while neglecting the rarer ones, which differ in how hard they are to restore. When the clean model is weak, the trivial strategy All-Zero, which sets $r^{\text{prob}}_{M^s_{\text{co}}} \equiv 0$, thus attains a low A-PPG merely by degrading performance on every question.

To prevent this, we group questions by the clean model's solve probability before aggregating. We partition $[0, 1]$ into $m$ equal-width intervals and assign each question to the interval that contains $r^{\text{prob}}_{M_{\text{cl}}}(q)$:

$$D_b = \Big\{ q \in D : r^{\text{prob}}_{M_{\text{cl}}}(q) \in \Big[ \tfrac{b-1}{m}, \tfrac{b}{m} \Big) \Big\} \tag{13}$$

where $b = 1, \ldots, m$ and the last interval ($b = m$) is closed on the right so that it includes 1. Because the solve probability is estimated from $m$ samples, the estimate can only take the values $0, \frac{1}{m}, \ldots, 1$.

Every attainable value thus forms its own group, except that $\frac{m-1}{m}$ and 1 share the last one. Let $\mathcal{B} = \{\, b : D_b \neq \varnothing \,\}$ index the non-empty groups. Averaging the absolute PPGs within each group and then across groups yields the **S**tratified **A**ggregate of **P**er-question **P**robability **G**aps (SA-PPG):

$$\text{SA-PPG}(s) = \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} \frac{1}{|D_b|} \sum_{q \in D_b} \big|\Delta_s(q)\big|. \tag{14}$$

After grouping, questions sharing a frequent solve probability form a single group with weight $1/|\mathcal{B}|$ however numerous they are, and no group's gap is diluted by the sizes of the others. **To attain a low SA-PPG, a mitigation strategy must restore performance at every level of the clean model's solve probability, which closes the shortcut of chasing the most frequent solve probabilities. SA-PPG also inherits from A-PPG the property of reading zero if and only if every question is perfectly restored.**

# 4 RailCap: Step-wise Supervision during Generation

RailCap is built on the two observations in Figure 1. First, the sampled responses of $M_{\text{co}}$ collapse onto its own greedy trajectory on leaked questions but disperse over many paths on unleaked ones. A fall-back onto the greedy trajectory is therefore itself an online signal of memorization. Second, at the decoding steps where the clean and contaminated models next diverge, the clean model's token is the contaminated model's runner-up in about half of the cases. Capping $M_{\text{co}}$'s trajectory token to the runner-up therefore most likely leaves the clean model's choice at the head of the flattened candidates, where sampling from $M_{\text{co}}$ readily picks it up. RailCap intervenes only in the decoding process of $M_{\text{co}}$ (Algorithm 1 in Appendix A).

**Preprocessing.** For each question $q$, we first decode $M_{\text{co}}$ greedily to obtain its greedy trajectory $g = (g_1, \ldots, g_L)$. All $n$-gram windows of the trajectory are built into an index

$$H = \big\{(g_k, \ldots, g_{k+n-1}) \mapsto g_{k+n} :\ 1 \le k \le L - n\big\}, \tag{15}$$

which maps each trailing $n$-gram to its successor token on the trajectory, where $n$ is the $n$-gram threshold for judging a fall-back. The greedy decoding and the index constitute RailCap's entire preprocessing, and they are computed only once per question.

**Per-step operation.** RailCap draws $m$ responses per question independently, each sampled token by token. At step $t$, let $o_{1:t-1}$ be the tokens generated so far and $\ell_t$ the logits at this step. If the last $n$ tokens coincide with a window of the trajectory, that is, $(o_{t-n}, \ldots, o_{t-1}) \in H$, RailCap intervenes to keep the sample from falling back onto the trajectory. The successor token on the trajectory, $x = H[(o_{t-n}, \ldots, o_{t-1})]$, is capped at the current second-largest logit,

$$\ell_t[x] \leftarrow \min\big(\ell_t[x],\, v_t^{(2)}\big), \tag{16}$$

where $v_t^{(2)}$ is the second-largest entry of $\ell_t$, so the trajectory token is leveled with the runner-up. Otherwise $\ell_t$ is left unchanged. In both cases, the next token is sampled at temperature $T$, $o_t \sim \text{Softmax}(\ell_t/T)$, and the decoding continues step by step until the end-of-sequence token is sampled.

The cap makes the current step likely to select a token off the trajectory, so the sample temporarily departs from it. The check recurs as generation proceeds, and suppression accumulates step by step until the response distribution is sufficiently dispersed. The same rule applies to every decoding step of every question, and how much intervention each question receives is decided online by the responses themselves. The judgment of contamination thus shifts from a one-shot, up-front estimate to step-wise supervision during generation.

# 5 Experiments

## 5.1 Experimental Setup

**Datasets.** **GSM8K** (Cobbe et al., 2021) consists of grade-school math word problems that require multi-step arithmetic reasoning and its test set contains 1,319 questions. **PQ** is a paraphrased version

of the GSM8K test set that we construct. Only the wording of each question is rewritten, and all numbers and the final answer are kept identical. Contaminating with PQ and evaluating on the original questions yields a harder form of contamination that verbatim memorization cannot exploit directly. The paraphrases are generated with DeepSeek-V4-Flash (DeepSeek-AI, 2026), and the prompt is given in Appendix B. **HumanEval** (Chen et al., 2021) contains 164 Python function-completion problems, each graded by executing its unit tests.

**Models and contamination simulation.** We simulate contamination on the base versions of three open model families, Llama-2-7B (Touvron et al., 2023), Gemma-4-E2B (Gemma Team, 2026), and Pythia-12B (Biderman et al., 2023), which cover different architectures, scales, and tokenizers. Pythia is fully open: both its weights and its training corpus are public, which allows us to verify that the base model itself is not contaminated by the evaluation data. Prior work shows that models fine-tuned on the training split of a task attain more stable test-set performance (Dominguez-Olmedo et al., 2025). Following this, we obtain the clean model $M_{\mathrm{cl}}$ by fine-tuning each base model on 1,840 OpenOrca examples (Lian et al., 2023) and 660 GSM8K training-split questions in the 8-shot CoT format (Wei et al., 2022) (2,500 examples in total). The contaminated model $M_{\mathrm{co}}$ is trained further from $M_{\mathrm{cl}}$ on the same 1,840 OpenOrca examples mixed with 660 test-split questions (again 2,500 examples). These 660 are the leaked questions, and the remaining 659, which appear in no training data, are the unleaked questions. Using the GSM8K or the PQ version of the test questions yields one contaminated model per domain. All training uses LoRA (r=64, $\alpha$=128) with a learning rate of 2e-4 (cosine schedule, warmup ratio 0.1), a global batch size of 32, bf16 precision, and 5 epochs in the LLaMA-Factory framework (Zheng et al., 2024). HumanEval has no training split, so its $M_{\mathrm{cl}}$ is fine-tuned on 2,418 OpenOrca examples and 82 MBPP (Austin et al., 2021) problems are rewritten into HumanEval's format, so that it sees the same surface form in the same amount as the contaminated model. Its $M_{\mathrm{co}}$ is trained further from $M_{\mathrm{cl}}$ on the same OpenOrca examples mixed with the 82 leaked problems, with the same hyperparameters. Every training set thus contains exactly 2,500 examples.

**Evaluation protocol and compared strategies.** GSM8K and PQ evaluations use the 8-shot CoT prompt and the original test questions, and HumanEval uses zero-shot function completion. Every metric is computed over the full test set, leaked and unleaked questions together: $N = 1{,}319$ (660 leaked and 659 unleaked) for GSM8K and PQ, and $N = 164$ (82 of each) for HumanEval. For each question we independently sample $m = 50$ responses at $T = 0.7$ and estimate the solve probability by the fraction of correct ones. The only exception is G-AP, whose 0/1 readout uses a single sampled response per question. All metrics use the same-domain $M_{\mathrm{cl}}$ as the reference. We compare **RailCap** with $n = 4$, **Identity** (which sets $M^s_{\mathrm{co}} = M_{\mathrm{co}}$, i.e., no intervention), and the prior strategies **TED** (Dong et al., 2024), **LNE-blocking** (Hou et al., 2025), and **Shortcut** neuron patching (Zhu et al., 2025). Appendix C describes how each prior strategy is adapted to this protocol. Tables 1 and 3 also include **All-Zero**, the synthetic trivial strategy of §3.4, which sets every solve probability of $M_{\mathrm{co}}$ to zero.

### 5.2 Metric Experiments: G-AP versus SA-PPG

#### 5.2.1 Main Result: Rank Reversal

In Table 1, G-AP is computed under the protocol of prior work, with one 0/1 observation per question followed by the aggregate difference. Under G-AP, LNE-blocking appears near-perfect (0.0235, against 0.3192 for Identity) and is the best strategy in the table. Under SA-PPG the verdict reverses: LNE-blocking falls to 0.2932, behind Shortcut (0.2476) and RailCap, barely better than Identity (0.3261). RailCap, second under G-AP (0.0728), is the best under SA-PPG (0.1914). The questions, models, and strategies are identical, but changing only the metric overturns the ranking. The near-perfect restoration that G-AP credits to LNE-blocking does not hold under SA-PPG. The next analyses trace this error to all three problems of G-AP: representation (Figure 2), aggregation, and weighting (the remaining columns of Table 1).

#### 5.2.2 Analysis: Sources of the Reversal

We first test the reproducibility of the readout itself by drawing two independent sampling batches from the same $M_{\mathrm{cl}}$ (Figure 2). Under the discrete readout $r^{0/1}$, the mean per-question gap between

| | Metric | | | | A-PPG components | |
|---|---|---|---|---|---|---|
| | G-AP | G-APP | A-PPG | SA-PPG | $\Delta_s^+$ | $\Delta_s^-$ |
| Identity | 0.3192 | 0.3243 | 0.3793 | 0.3261 | 0.3518 | 0.0275 |
| LNE-blocking | **0.0235** | **0.0207** | 0.1879 | 0.2932 | 0.1043 | 0.0836 |
| Shortcut | 0.1812 | 0.1777 | 0.2399 | 0.2476 | 0.2088 | 0.0311 |
| TED | 0.1813 | 0.1804 | 0.2798 | 0.2946 | 0.2301 | 0.0497 |
| RailCap | 0.0728 | 0.0794 | **0.1634** | **0.1914** | 0.1214 | 0.0420 |
| All-Zero | 0.1941 | 0.2190 | 0.2190 | 0.4903 | 0.0000 | 0.2190 |

Table 1: G-AP, G-APP, A-PPG, and SA-PPG of each mitigation strategy on Llama-2 × GSM8K, together with the under-suppression component $\Delta_s^+$ and the over-suppression component $\Delta_s^-$ of A-PPG. Bold marks the best strategy in each metric column.

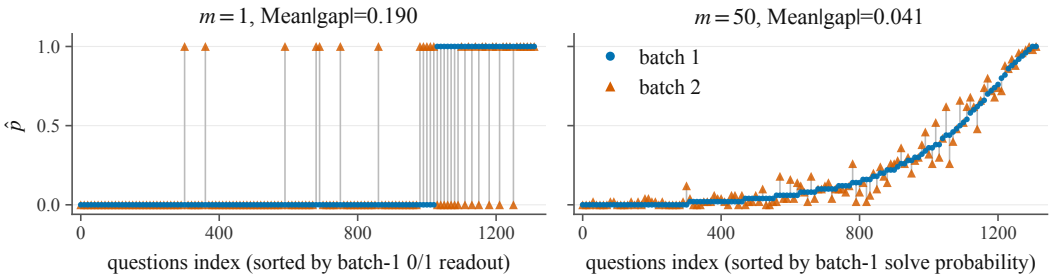



Figure 2: Per-question readouts of two independent sampling batches drawn from the same clean model $M_{\text{cl}}$ (Llama-2, GSM8K domain), with questions sorted by the first batch's readout. Left: the discrete readout $r^{0/1}$ ($m = 1$). Right: the solve probability estimated from $m = 50$ samples.

the two batches reaches 0.190. Under the solve-probability readout, the two batches collapse onto one curve and the mean per-question gap drops to 0.041, so the probability readout is far more stable.

Table 1 shows that two obstacles remain after the readout is corrected. LNE-blocking's near-perfect G-AP (0.0235) is not an artifact of the sampling noise in its single-observation readout, because G-APP, whose probability readout is far less noisy, is even closer to zero (0.0207). The first obstacle is **cancellation**. LNE-blocking's over-suppression component $\Delta_s^-$ (0.0836) is 2.0 times RailCap's (0.0420), yet its G-APP is 3.8 times lower than RailCap's (0.0794). The extra over-suppression cancels against the under-suppression and pushes the reading toward zero. With the readout held fixed, removing the cancellation alone raises LNE-blocking's reading ninefold, from 0.0207 under G-APP to 0.1879 under A-PPG, which places it behind RailCap (0.1634).

The second obstacle is **uniform per-question weighting**. All-Zero, the trivial strategy of §3.4, sets every question's solve probability to zero. It reads 0.2190 under the uniformly weighted A-PPG, even better than TED (0.2798) and Shortcut (0.2399). This failure is inherited from G-AP, whose uniform per-question weighting A-PPG retains and which masks the damage All-Zero does. SA-PPG, which groups questions by the clean model's solve probability, ranks All-Zero worst of all strategies (0.4903). The readout correction (§3.2), per-question differencing (§3.3), and stratified aggregation (§3.4) are therefore each necessary, and SA-PPG combines all three.

### 5.3 Strategy Experiments: RailCap versus Prior Strategies

#### 5.3.1 Main Result: Lowest SA-PPG across Settings

RailCap attains the lowest SA-PPG in all nine settings of Table 2. Its largest lead is on Gemma × HumanEval, where it reaches 0.2172 against 0.3240 for the runner-up, LNE-blocking.

Shortcut is the most frequent runner-up (five of nine settings), likely because it contrasts $M_{\text{co}}$ with $M_{\text{cl}}$ to locate shortcut neurons and thus draws on information from $M_{\text{cl}}$ that the other strategies lack.

| | GSM8K | | | PQ | | | HumanEval | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | Llama-2 | Gemma | Pythia | Llama-2 | Gemma | Pythia | Llama-2 | Gemma | Pythia |
| Identity | 0.3261 | 0.3147 | 0.2131 | 0.2549 | 0.2205 | 0.1907 | 0.4326 | 0.3675 | 0.3337 |
| TED | 0.2946 | 0.2975 | 0.2116 | 0.2443 | 0.2099 | 0.1723 | 0.3914 | 0.3800 | 0.2326 |
| Shortcut | 0.2476 | 0.2870 | 0.2033 | 0.2242 | 0.2116 | 0.1708 | 0.3388 | 0.3993 | 0.2877 |
| LNE-blocking | 0.2932 | 0.3061 | 0.2079 | 0.3046 | 0.2703 | 0.2171 | 0.3366 | 0.3240 | 0.3043 |
| RailCap | **0.1914** | **0.1961** | **0.1648** | **0.2090** | **0.1695** | **0.1429** | **0.2699** | **0.2172** | **0.2114** |

Table 2: SA-PPG of each mitigation strategy across the nine settings (three contamination domains $\times$ three models). Bold marks the best strategy in each column.

| Config | Identity | $n = 1$ | $n = 3$ | $\mathbf{n = 4}$ | $n = 5$ | $n = 7$ | $n = 4$, ban | All-Zero |
|---|---|---|---|---|---|---|---|---|
| SA-PPG | 0.3261 | 0.2434 | 0.1990 | **0.1914** | 0.1969 | 0.1961 | 0.2190 | 0.4903 |

Table 3: Ablation of RailCap on Llama-2 $\times$ GSM8K. Columns $n = 1$ to $n = 7$ sweep the $n$-gram threshold under the default cap at the runner-up; $n = 4$, ban replaces the cap with a hard ban that sets the trajectory token's probability to zero. Bold marks the lowest SA-PPG.

TED yields only modest gains. It improves on Identity in eight of nine settings but is worse on Gemma $\times$ HumanEval (0.3800 vs. 0.3675). Filtering the samples of an unmodified model thus appears unable to depart far enough from the memorized trajectory to recover the original performance.

LNE-blocking improves on Identity for all three models in GSM8K but falls behind Identity for all three in PQ. It sets its blocking strength from a contamination estimate based on length-normalized entropy. In PQ, the model is contaminated with rewritten questions but evaluated on the original ones, and this rewriting makes contamination difficult to detect from entropy. RailCap needs no such estimate, as it checks every decoding step, and keeps the lowest SA-PPG in PQ as well.

#### 5.3.2 Ablation: $n$-gram Threshold and Suppression Form

Table 3 ablates two design choices of RailCap on Llama-2 $\times$ GSM8K: the $n$-gram threshold $n$ and the form of suppression. The setting $n = 4$ attains the lowest SA-PPG (0.1914), and every tested $n$ from 3 to 7 stays within 0.008 of it, so performance is insensitive to $n$ in this range. With $n = 1$, a single token that matches the trajectory suffices to trigger the cap, and SA-PPG rises to 0.2434, likely because genuine capability is suppressed along with memorization. Replacing the cap with a hard ban ($n = 4$, ban) raises SA-PPG to 0.2190. Keeping the trajectory token available at reduced probability is therefore preferable to prohibiting it entirely.

## 6 Conclusion

We have studied how to appropriately measure and how to improve contamination mitigation. On the metric side, SA-PPG fixes the three flaws of the prevailing G-AP: it replaces the discrete readout with each question's solve probability, differences per question before averaging, and gives equal weight to groups defined by the clean model's solve probability. On the strategy side, RailCap caps the next trajectory token to the runner-up whenever a sample falls back onto the greedy trajectory, replacing a one-shot, up-front estimate of contamination with step-wise supervision during generation. RailCap attains the lowest SA-PPG in all nine settings of three models and three contamination domains.

## AI USE STATEMENT

In this work, we used generative AI tools for two purposes. First, to construct the PQ dataset, we used DeepSeek-V4-Flash (DeepSeek-AI, 2026) to paraphrase the wording of each leaked GSM8K test question while keeping all numbers and the final answer unchanged; the full prompt is given in Appendix B, and the reasoning chains and gold answers are the unmodified GSM8K originals. Second, we used generative AI tools to edit the manuscript for grammar and readability. We have not used generative AI tools to develop the conceptual framework, formulate the metric or its analysis, propose hypotheses, design the experiments, or interpret the results. We have reviewed all AI-assisted work, and we take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

## REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Yushi Bai, Jiahao Ying, Yixin Cao, Xin Lv, Yuze He, Xiaozhi Wang, Jifan Yu, Kaisheng Zeng, Yijia Xiao, Haozhe Lyu, et al. Benchmarking foundation models with language-model-as-an-examiner. *Advances in Neural Information Processing Systems*, 36:78142–78167, 2023.

Simone Balloccu, Patrícia Schmidtová, Mateusz Lango, and Ondrej Dusek. Leak, cheat, repeat: Data contamination and evaluation malpractices in closed-source LLMs. In Yvette Graham and Matthew Purver (eds.), *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 67–93, St. Julian's, Malta, March 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.eacl-long.5.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2397–2430. PMLR, 2023. URL https://proceedings.mlr.press/v202/biderman23a.html.

Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. doi: https://arxiv.org/abs/2107.03374.

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

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Chunyuan Deng, Yilun Zhao, Yuzhao Heng, Yitong Li, Jiannan Cao, Xiangru Tang, and Arman Cohan. Unveiling the spectrum of data contamination in language model: A survey from detection to remediation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 16078–16092, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.951.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*, 2021.

Ricardo Dominguez-Olmedo, Florian E. Dorner, and Moritz Hardt. Training on the test task confounds evaluation and emergence. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=jOmk0uS1hl.

Yihong Dong, Xue Jiang, Huanyu Liu, Zhi Jin, Bin Gu, Mengfei Yang, and Ge Li. Generalization or memorization: Data contamination and trustworthy evaluation for large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 12039–12050, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.716.

Yanai Elazar, Akshita Bhagia, Ian Helgi Magnusson, Abhilasha Ravichander, Dustin Schwenk, Alane Suhr, Evan Pete Walsh, Dirk Groeneveld, Luca Soldaini, Sameer Singh, Hannaneh Hajishirzi, Noah A. Smith, and Jesse Dodge. What's in my big data? In *The Twelfth International Conference on Learning Representations*, 2024.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026. URL https://arxiv.org/abs/2607.02770.

Shahriar Golchin and Mihai Surdeanu. Time travel in LLMs: Tracing data contamination in large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Ruijie Hou, Yueyang Jiao, Hanxu Hu, Yingming Li, Wai Lam, Huajian Zhang, and Hongyuan Lu. LNE-blocking: An efficient framework for contamination mitigation evaluation on large language models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 3512–3528, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.188. URL https://aclanthology.org/2025.findings-emnlp.188/.

Alon Jacovi, Avi Caciularu, Omer Goldman, and Yoav Goldberg. Stop uploading test data in plain text: Practical strategies for mitigating data contamination by evaluation benchmarks. *arXiv preprint arXiv:2305.10160*, 2023.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. GSM-plus: A comprehensive benchmark for evaluating the robustness of LLMs as mathematical problem solvers. 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)*, pp. 2961–2984, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.163.

Xiang Li, Yunshi Lan, and Chao Yang. Treeeval: Benchmark-free evaluation of large language models through tree planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24485–24493, 2025.

Yucheng Li. Estimating contamination via perplexity: Quantifying memorisation in language model evaluation. *arXiv preprint arXiv:2309.10677*, 2023.

Wing Lian, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and Teknium. OpenOrca: An open dataset of GPT augmented FLAN reasoning traces. https://huggingface.co/datasets/Open-Orca/OpenOrca, 2023.

Inbal Magar and Roy Schwartz. Data contamination: From memorization to exploitation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 157–165, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.18.

Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. Detecting pretraining data from large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Wentao Ye, Jiaqi Hu, Liyao Li, Haobo Wang, Gang Chen, and Junbo Zhao. Data contamination calibration for black-box LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 10845–10861, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.644.

Zhuohao Yu, Chang Gao, Wenjin Yao, Yidong Wang, Wei Ye, Jindong Wang, Xing Xie, Yue Zhang, and Shikun Zhang. KIEval: A knowledge-grounded interactive evaluation framework for large language models. 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)*, pp. 5967–5985, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.325.

Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, Tiffany Zhao, Pranav Raja, Dylan Slack, Qin Lyu, et al. A careful examination of large language model performance on grade school arithmetic. *arXiv preprint arXiv:2405.00332*, 2024a.

Weichao Zhang, Ruqing Zhang, Jiafeng Guo, Maarten de Rijke, Yixing Fan, and Xueqi Cheng. Pretraining Data Detection for Large Language Models: A Divergence-based Calibration Method. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5263–5274, Miami, Florida, USA, November 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.300.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL http://arxiv.org/abs/2403.13372.

Kun Zhou, Yutao Zhu, Zhipeng Chen, Wentong Chen, Wayne Xin Zhao, Xu Chen, Yankai Lin, Ji-Rong Wen, and Jiawei Han. Don't make your llm an evaluation benchmark cheater. *arXiv preprint arXiv:2311.01964*, 2023.

Kejian Zhu, Shangqing Tu, Zhuoran Jin, Lei Hou, Juanzi Li, and Jun Zhao. Establishing trustworthy LLM evaluation via shortcut neuron analysis. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3809–3822, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.192. URL https://aclanthology.org/2025.acl-long.192/.

Wenhong Zhu, Hongkun Hao, Zhiwei He, Yun-Ze Song, Jiao Yueyang, Yumeng Zhang, Hanxu Hu, Yiran Wei, Rui Wang, and Hongyuan Lu. CLEAN–EVAL: Clean evaluation on contaminated large language models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 835–847, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.53.

## A RailCap Pseudocode

Algorithm 1 summarizes RailCap decoding for one question, as described in §4.

**Algorithm 1** RailCap decoding for one question $q$

**Require:** $M_{\mathrm{co}}$, $q$, $n$-gram threshold $n$, number of samples $m$, temperature $T$
**Ensure:** responses $\{o^{(1)}, \ldots, o^{(m)}\}$
$g = (g_1, \ldots, g_L) \leftarrow \mathrm{GreedyDecode}(M_{\mathrm{co}}, q)$
$H \leftarrow \{(g_k, \ldots, g_{k+n-1}) \mapsto g_{k+n}\}_{k=1}^{L-n}$ ▷ repeated $n$-gram: all successors kept, each capped
**for** $j = 1, \ldots, m$ **do**
  $o \leftarrow (\,)$;  $t \leftarrow 1$
  **repeat**
    $\ell \leftarrow \mathrm{Logits}\big(M_{\mathrm{co}}, (q, o)\big)$
    **if** $t > n$ **and** $(o_{t-n}, \ldots, o_{t-1}) \in H$ **then**
      $x \leftarrow H[(o_{t-n}, \ldots, o_{t-1})]$
      $\ell[x] \leftarrow \min\big(\ell[x], v^{(2)}\big)$ ▷ $v^{(2)}$: second-largest entry of $\ell$
    **end if**
    $o_t \sim \mathrm{Softmax}(\ell / T)$
    $o \leftarrow o \,\|\, o_t$;  $t \leftarrow t + 1$
  **until** $o_t = \mathrm{EOS}$
  $o^{(j)} \leftarrow o$
**end for**
**return** $\{o^{(1)}, \ldots, o^{(m)}\}$

## B PQ Construction and Evaluation Details

**B.1 PQ construction.** For each leaked question, the question text is rewritten with DeepSeek-V4-Flash (DeepSeek-AI, 2026). The instruction asks for a substantial rephrasing that keeps all numbers and the final answer identical, forbids any solution, and requires exactly one output line containing the rewritten question. The system prompt is:

> Significantly rephrase the given GSM8K math question. Keep ALL numbers and the final answer identical. Do not include any solution. Output exactly one line in this format and nothing else:
> New Question: <rephrased question>

The reasoning chain and the gold answer remain the GSM8K originals.

## C Implementation of Prior Strategies

All strategies are applied to $M_{\mathrm{co}}$ with the prompts of §5.1. Each yields $m = 50$ responses per question at $T = 0.7$, and these responses are graded in the same way as all others. Where a prior strategy was designed for a different decoding protocol, we adapt it to this protocol. This appendix lists each adaptation and its reason.

**LNE-blocking.** The original method targets greedy decoding (Hou et al., 2025). It first decodes greedily and converts the length-normalized entropy of that output into a contamination score $\overline{\mathrm{LNE}} \in [0, 1]$ that increases with contamination. It then blocks the top-1 token at each of the first $\mathrm{Cnt} = \mathrm{round}(\overline{\mathrm{LNE}} \cdot \mathrm{Threshold_Task})$ positions and decodes the rest greedily. We keep the greedy estimate, the count, and the blocked prefix of the released implementation unchanged. After the prefix, however, we sample the continuation $m = 50$ times at $T = 0.7$, because a solve probability cannot be estimated from a single greedy output. $\mathrm{Threshold_Task}$ takes the values selected in the original work: 7 for arithmetic reasoning (GSM8K and PQ) and 4 for code generation (HumanEval).

**TED.** The original method draws 50 samples at $T = 0.8$ (Dong et al., 2024). It discards exact duplicates and the samples within token edit distance $\tau = 2$ of the greedy output, and it reports pass@1 over the remaining samples. We apply the same rule, with $\tau = 2$, to samples drawn at

$T = 0.7$, the temperature shared by all strategies. The number of remaining samples depends on how strongly a question is memorized. On Llama-2 $\times$ GSM8K, about 64 of the 660 leaked questions retain no sample and, as in the released implementation, count as unsolved. Because samples that satisfy the edit-distance requirement are hard to obtain for some questions, we do not resample until $m = 50$ samples pass the filter. Instead, we uniformly draw $m = 50$ samples, filter them, and approximate the solve probability of our protocol by the fraction of correct responses among the samples that remain.

**Shortcut.** The original method locates shortcut neurons in two steps (Zhu et al., 2025). A comparative analysis ranks neurons by the root-mean-square difference between the last-prompt-token activations of a contaminated and an uncontaminated model. A causal analysis then patches groups of 512 adjacent neurons during generation and scores the resulting accuracy change of both models. The original method patches the top 5,000 neurons of the evaluated model with the activations of the vanilla base model $M_0$, and its released implementation decodes greedily. We run both analyses, contrasting $M_{\text{cl}}$ with $M_{\text{co}}$ on the test questions, and patch the top 5,000 neurons. The patching activations come from $M_{\text{cl}}$, which takes the place of $M_0$ because $M_{\text{co}}$ is fine-tuned from $M_{\text{cl}}$ (§5.1). We then sample $m = 50$ responses at $T = 0.7$.