How to Steal Reasoning Without Reasoning Traces

Anonymous authors
Paper under double-blind review

Abstract

Many large language models (LLMs) generate multi-step reasoning traces (a.k.a. chains of thought) but expose only final answers and brief reasoning summaries. Recent work recovered reasoning traces through provider-specific API vulnerabilities that were subsequently patched. We ask whether properly hiding reasoning traces is sufficient to prevent extraction of a model's reasoning capabilities.

The answer is no. We introduce *Trace Inversion* models that synthesize detailed reasoning traces with only ordinary black-box access to models' inputs and outputs. We show that (1) fine-tuning student models on synthesized traces substantially improves downstream reasoning, enabling distillation from proprietary black-box LLMs, and, in many cases, (2) the synthesized traces heavily overlap with the ground-truth traces (where the latter are available).

1 Introduction

Large language models (LLMs) with explicit reasoning capabilities perform well on math, coding, and scientific analysis tasks. Trained via supervised fine-tuning, instruction tuning, and reinforcement learning, these models generate multi-step internal reasoning traces, a.k.a. chains of thought, that lead to final answers (Wei et al., 2022; Ouyang et al., 2022; Chen et al., 2025b). These traces can also help transfer reasoning capabilities. Fine-tuning a student model on the traces of a teacher model outperforms fine-tuning on just the teacher's answers (Hinton et al., 2015; Gou et al., 2021; Shridhar et al., 2023; Chen et al., 2025b).

Exposing full chains of thought is thus a potential intellectual property risk. For example, in February 2026 Anthropic accused several competing labs of large-scale distillation campaigns that allegedly elicited data from Anthropic's reasoning models and used it for training (Anthropic, 2026). Furthermore, reasoning traces may encode sensitive system prompts, safety policies, or private contextual information (Green et al., 2025; Holmes & Gooderham, 2025; Zhou et al., 2025). To mitigate these risks, commercial APIs typically reveal only the final answer and a short reasoning summary (OpenAI, 2025; Google, 2025; Anthropic, 2025). Recently, (Panfilov et al., 2026) demonstrated that encrypted traces can be recovered by exploiting bugs in provider-specific reasoning-replay mechanisms. These bugs have since been patched. In this paper, we study a stronger protection setting in which the model's reasoning traces are truly *not* available.

We introduce **Trace Inversion**, a framework for synthesizing detailed reasoning traces from the outputs that are always observable by a model's users: final answers and, if enabled by the provider, compressed summaries (Figure 1). We show that **synthesized traces can teach a student model how to reason even when the black-box teacher model does not reveal its reasoning** (and both the student and the inversion model are weaker than the teacher). Across math, scientific reasoning, and coding benchmarks, models fine-tuned on synthesized traces often outperform those fine-tuned on (a) surrogate traces, or (b) teachers' answers and reasoning summaries. This demonstrates that hiding chains of thought is insufficient to prevent reasoning-capability extraction. Our key insight is that an attacker need not reconstruct the victim's actual reasoning trace. For capability extraction, it is enough to synthesize victim-conditioned traces that provide effective supervision for distillation.

For example, fine-tuning `Qwen-2.5-7B-Instruct` on traces synthesized from the answers and summaries of `gpt-5.4-mini-2026-03-17` (GPT-5.4 mini), a much stronger, commercial, black-box model, by (weak) `DeepSeek-R1-Distill-Qwen-1.5B`, reaches 31.6% on

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

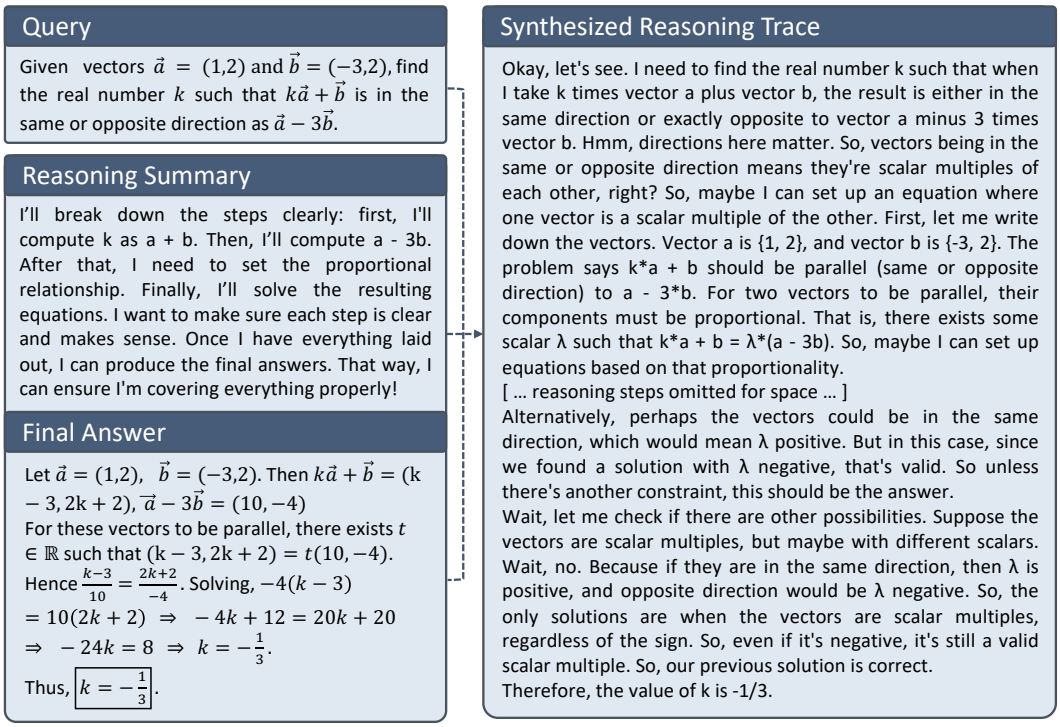

Figure 1: **Trace Inversion against GPT-5.4 mini.** Given only the input, final answer, and reasoning summary from a commercial black-box model, Trace Inversion synthesizes a detailed reasoning trace for supervised fine-tuning of a student model.

JEEBench, vs. 19.7% when fine-tuned on `R1-Distill`'s own reasoning traces. Fine-tuning `Llama-3.1-8B-Instruct` on traces synthesized from GPT-5.4 mini by an inversion model based on `Deepseek-R1` reaches 52.4% on MATH500, vs. 16.4% when fine-tuned on the GPT-5.4 mini's reasoning summaries and answers. In some cases, fine-tuning on synthesized traces yields even better performance than the ground-truth traces. The latter can contain backtracking and dead ends, whereas synthesized traces are typically clean forward reasoning.

Reasoning capability extraction does not require that synthesized traces are similar to the teacher model's actual traces. That said, we show that traces synthesized by our inversion models are similar to the ground truth (when the latter is available). For example, they average 81% of the ground-truth token length and reach a 52.76 token-overlap F1 score for DeepSeek-R1 traces, even though the inversion model is based on a much weaker surrogate.

2 Background and Related Work

Reasoning models. Many modern large language models rely on explicit multi-step reasoning. Chain-of-thought (CoT) prompting showed that eliciting intermediate reasoning steps improves performance on arithmetic, logical, and symbolic benchmarks Wei et al. (2022); Kojima et al. (2022); Wang et al. (2022). Instruction tuning and reinforcement learning (RL) can incorporate reasoning-oriented objectives Ouyang et al. (2022); Bai et al. (2022), e.g., in models like DeepSeek-R1 Guo et al. (2025) and OpenAI's o-series OpenAI (2024). Related models such as QwQ-32B and s1 further show that RL can enable smaller models to approach the reasoning performance of large base models Team Qwen (2024); Muennighoff et al. (2025).

Information leakage from language models. Black-box language models can leak both training data and inference-time inputs through their outputs. Training-data leakage includes extraction of memorized examples Carlini et al. (2021) and membership inference, which determines whether specific examples appeared in training data, including in-context examples Shokri et al. (2017); Mattern et al. (2023); Wen et al. (2024). Inference-time leakage includes model inversion, which

reconstructs information about inputs from model outputs Morris et al. (2024), and prompt inversion, which targets hidden inputs such as system prompts Zhang et al. (2024b;a); Yang et al. (2024).

Model distillation and capability stealing. Model distillation transfers capabilities from a teacher to a student model, typically by matching the teacher's outputs or predictive distribution Hinton et al. (2015). In adversarial settings, an attacker instead extracts capabilities from a deployed model through limited external access. This *capability stealing* was first studied for image classifiers Tramèr et al. (2016); Orekondy et al. (2019) and later extended to language models Krishna et al. (2019); Wallace et al. (2020). Distillation is substantially more effective when teacher-generated reasoning traces are available rather than final answers alone Wei et al. (2022); Shridhar et al. (2023). Partial or shortened reasoning can trade off supervision cost and performance Chen et al. (2025b). We focus on the realistic setting where the teacher's reasoning traces are *not* exposed by the API.

Reasoning-trace access and synthesis. Prior work showed that useful reasoning supervision can be obtained without observing a teacher's hidden chain of thought. STaR bootstraps self-generated rationales using known answers Zelikman et al. (2022), an approach related to our zero-shot baseline. More closely, π-Distill and On-Policy Self-Distillation transfer capabilities from frontier agents whose reasoning is hidden by using their observed action trajectories as privileged information Penaloza et al. (2026). These methods do not reconstruct reasoning from black-box outputs.

Following the initial release of our work, several studies showed how to extract hidden reasoning via specially constructed demonstrations Lu et al. (2026), or bugs in provider-specific reasoning-replay mechanisms Panfilov et al. (2026); Qu et al. (2026). Our work demonstrates that reasoning extraction remains possible *even after these bugs have been patched.*

3 Threat Model

Problem setting. Consider a black-box *victim model* V that, given an input $x \in \mathcal{X}$, internally generates a reasoning trace $t \in \mathcal{T}$ that leads to the final answer $y \in \mathcal{Y}$. The user receives only y and, optionally, a short reasoning summary or "bubble" $b^\star = C(t) \in \mathcal{B}$ such that $|b^\star| \ll |t|$. This setting corresponds to the APIs of commercial reasoning models.

Attacker's goal. The attacker seeks to improve the reasoning capabilities of its own model S through black-box access to a stronger model V, without observing V's internal reasoning. Given tuples (x, y) or $(x, b^\star, y)$ returned by V, the attacker synthesizes traces $\hat{t}$ that are logically consistent with the observed outputs. These traces need not match the victim's true internal reasoning; their purpose is to provide effective supervision for fine-tuning S on $(x, \hat{t}, y)$.

Attacker's auxiliary models. The attacker may use its own surrogate reasoning model V', compression model C', inversion model I, and student model S. These models may be public or private to the attacker, but the attacker must be able to fine-tune I and S. The **surrogate reasoning model** V' produces complete reasoning traces and answers (t', y') for training the inversion model. Typically, V' is much weaker than the victim (e.g., R1-Weak vs. GPT-5.4 mini in our experiments).

We do *not* assume that the attacker has access to the true compression method C used by the victim V; for example, how GPT models produce reasoning summaries is not publicly known. To emulate this summarization, the attacker uses a **reasoning compression model** C' that maps surrogate traces t' to summaries $b' = C'(t'; \pi)$ under a fixed prompt template π, without additional fine-tuning. The attacker then trains an **inversion model** I to synthesize traces from V's observable input–output pairs (x, y) and, when available, summaries $b^\star$. These synthesized traces are used to fine-tune S.

Public reasoning datasets. The attacker can draw on a public dataset D of diverse reasoning-task inputs x (e.g., math, code, and logic questions), such as `OpenThoughts-114k` (Guha et al., 2025). We use these inputs to (1) query the surrogate reasoning model for training traces and (2) query the victim for answers and summaries to invert.

4 Inversion Methodology

Trace Inversion is a three-stage attack pipeline (Figure 2). We consider a *summary setting*, where the victim model exposes a short reasoning summary, and a stricter *no-summary setting*, where only

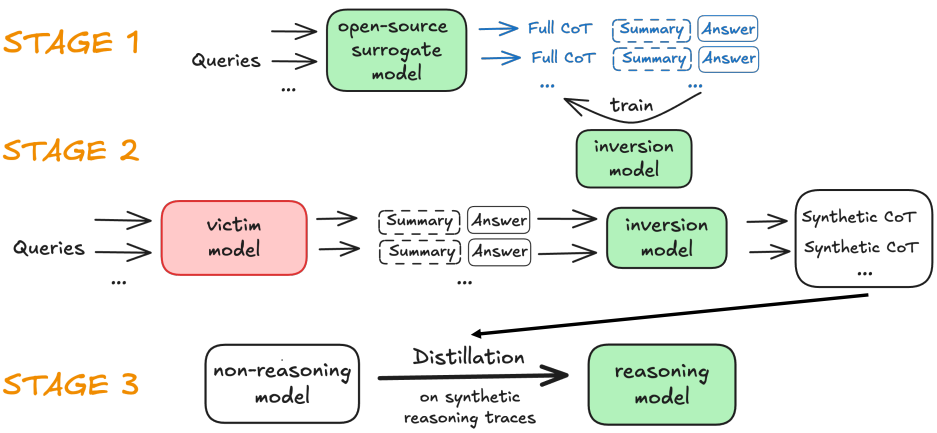

Figure 2: **Trace Inversion pipeline.** First, the attacker trains an inversion model on traces from a surrogate V', optionally compressed into summaries. The attacker then applies the inverter to a victim's answers and available summaries, and fine-tunes a student on the resulting synthetic traces and answers. In the no-summary setting, the summary input is omitted.

the final answer is available. Each setting requires a separate inversion model. The objective is to synthesize a detailed trace that is compatible with the victim's observable response and useful for student training; it need not reproduce the victim's exact internal trace.

4.1 Stage 1: Training the Inversion Model

Training data. The attacker samples inputs x' from public reasoning datasets, submits them to the surrogate model V', and obtains traces t' and answers y', yielding dataset $\mathcal{D}_1 = \{(x', t', y')\}$. To emulate commercial APIs, each surrogate trace t' is optionally compressed into a summary $b' = C'(t'; \pi)$ using model C'. We design π to match observable properties of the victim's summaries (Section 5.2); Appendix D provides the exact prompt. This yields datasets $\mathcal{D}_2^{\text{sum}} = \{(x', b', y', t')\}$ and $\mathcal{D}_2^{\text{nosum}} = \{(x', y', t')\}$ for the summary and no-summary settings, respectively. We use our own compression model because black-box models expose neither their traces, nor summarization procedures, preventing us from collecting the trace–summary pairs needed for training.

Training objective. In the summary setting, the inversion model $I_{\text{sum}}(x, y, b) = \hat{t}$ is trained to maximize the likelihood that its output matches the true trace t':

$$\mathcal{L}_{\text{inv}}^{\text{sum}} = \mathbb{E}_{(x', y', b', t') \sim \mathcal{D}_2^{\text{sum}}} \left[-\log p_{I_{\text{sum}}}(t' \mid x', y', b') \right] \tag{1}$$

The no-summary setting is similar: $I_{\text{nosum}}(x, y) = \hat{t}$ and the training objective is:

$$\mathcal{L}_{\text{inv}}^{\text{nosum}} = \mathbb{E}_{(x', y', t') \sim \mathcal{D}_2^{\text{nosum}}} \left[-\log p_{I_{\text{nosum}}}(t' \mid x', y') \right] \tag{2}$$

Both objectives are implemented using teacher forcing with token-level cross-entropy.

4.2 Stage 2: Inverting Victim Outputs

The attacker first queries the victim V on inputs x, collecting tuples $(x, y, b^\star)$ (or (x, y) in the no-summary setting). Applying the trained inversion model yields $\hat{t} = I_{\text{sum}}(x, y, b^\star)$ or $\hat{t} = I_{\text{nosum}}(x, y)$. The resulting synthetic traces are compatible with V's observed outputs and provide victim-conditioned reasoning supervision for the next stage.

4.3 Stage 3: Student Distillation

The attacker uses the synthesized traces to transfer victim-conditioned reasoning to the student model S. For each input x, the synthetic trace $\hat{t}$ and answer y are concatenated into a single supervision target $y^+ = [\hat{t};\, y]$. The student is fine-tuned using standard teacher-forced cross-entropy:

$$\mathcal{L}_{\text{student}} = \mathbb{E}_{(x, \hat{t}, y)} \left[-\log p_S(y^+ \mid x) \right] \tag{3}$$

Optional: augmenting with surrogate data. Since the attacker already generated the surrogate dataset $\mathcal{D}_1 = \{(x', t', y')\}$ for training the inversion model, these examples can be added to the distillation data at no additional cost. The student can be fine-tuned on $\{(x, \hat{t}, y)\} \cup \mathcal{D}_1$ either as a single mixed corpus, or in a curriculum (surrogate traces first, inverted traces second). We report this as an ablation in §5.4; the default pipeline uses only the inverted traces.

5 EVALUATION

5.1 EXPERIMENTAL SETUP

Datasets. `OpenThoughts-114k` (OpenThoughts) is a public reasoning dataset spanning math, science, programming, and logic. Each example is an input x, an R1-generated reasoning trace, and the final answer. We sample two disjoint splits of 10k inputs each: a *surrogate split* for querying an open-source reasoning model to obtain (x', t', y') used to train inversion models, and a *victim split* for querying the victim to obtain (x, t, y)(open-weight victim) or $(x, b^\star, y)$ (black-box victim).

Models. As surrogate models, we use R1 or `DeepSeek-R1-Distill-Qwen-1.5B`[1], which we refer to as **R1-Weak** to emphasize that it is weaker than the victim models. We use R1 as the open-weight victim, and `gpt-5.4-mini-2026-03-17` (GPT-5.4 mini) as the black-box victim, both accessed via their commercial APIs. We use `Qwen2.5-7B-Instruct` (Qwen) as (1) a trace summarization model and (2) the backbone of our inversion models. To evaluate the utility of inverted traces, we fine-tune two student models: `Qwen2.5-7B-Instruct` (Qwen) and `Llama-3.1-8B-Instruct` (Llama).

Inversion metrics. When the victim is open-weight (i.e., R1), we compare the synthesized trace $\hat{t}$ to the ground-truth trace t using *Len* (token count; a coarse proxy for completeness), *BLEU* (n-gram precision, capturing surface lexical similarity), *TF1* (token-overlap F1, i.e., the harmonic mean of token-level precision and recall), and *ROUGE-1/2/L* (unigram, bigram, and longest-common-subsequence overlap). As mentioned above, these metrics are auxiliary diagnostics. Our main evaluation metric is the reasoning performance of student models fine-tuned on inverted traces.

Fine-tuning data for student models. We consider five options: *Answer-only* (victim's answers), *Summary+Answer* (victim's summaries and answers), *Surrogate-Trace* (surrogate's reasoning traces and answers), *Synthesized-Trace (ours)* (our inverted traces and victim's answers), and *Victim-Trace (oracle)* (victim's actual traces and answers—not available in practice).

Reasoning benchmarks. To evaluate whether fine-tuning on student models on synthesized traces improves their reasoning performance, we use three benchmarks: *MATH500* Hendrycks et al. (2021) evaluates multi-step mathematical problem solving; *JEEBench* Arora et al. (2023) measures structured problem solving across physics, chemistry, and mathematics; *LiveCodeBench* (LCB) Jain et al. (2025) evaluates code generation and debugging using execution-based metrics.

Hyperparameters. All models are fine-tuned using `num_train_epochs = 3`, `learning_rate = 1e-5`, `warmup_ratio = 0.1`, `cutoff_len = 16384`. We use each model's default optimizer and tokenizer, following the standard protocol for each benchmark. All training and evaluation use 8×NVIDIA A100 GPUs (80 GB).

5.2 FIDELITY OF TRACE INVERSION

We first evaluate whether the inversion model synthesizes traces that resemble the victim's true reasoning. Against an open-weight victim (R1), we measure this directly.Because GPT-5.4 mini does not reveal its traces, we instead compare summary statistics.

Matching summaries (GPT-5.4 mini). The inversion model is trained on surrogate summaries $b' = C'(t'; \pi)$ and applied to the victim's summaries $b^\star$, so the two should have similar properties. GPT-5.4 mini's API provides token-usage metadata, from which we estimate a median compression ratio of $4.2\times$ (hidden-trace tokens divided by summary tokens). Its non-empty summaries have a median

[1] `https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B`

Table 2: **Trace inversion fidelity (victim = R1).** We report synthesized trace length (Len) and overlap metrics between synthesized traces $\hat{t}$ and R1 ground-truth traces t. **Bold** rows use a weaker surrogate (R1-Weak), corresponding to a realistic attack. Higher is better, except trace length.

Setting	Inversion	Surrogate	Len	BLEU	TF1	R-1	R-2	R-L
Summary	Qwen (0-shot)	–	978.5	3.04	35.36	26.77	15.73	17.60
Summary	Qwen (FT)	R1	5,767.3	26.64	64.42	66.84	40.51	29.17
Summary	Qwen (FT)	**R1-Weak**	4,971.9	16.64	52.76	56.59	29.72	24.84
No-summary	Qwen (0-shot)	–	1,040.0	3.31	37.58	28.30	16.39	17.97
No-summary	Qwen (FT)	R1	6,020.9	25.07	57.08	65.43	38.91	27.97
No-summary	Qwen (FT)	**R1-Weak**	5,434.0	11.52	49.01	47.45	21.85	21.73

length of 592 tokens and typically use first-person prose, bold headers, and LaTeX. We design π to reproduce these properties (Appendix D); Table 1 compares the resulting summary statistics.

Matching reasoning traces (R1). Table 2 reports how well traces inverted from R1's outputs match the ground truth. When the inversion model is trained on traces from R1 itself, the training and evaluation distributions are the same, giving an upper bound on achievable fidelity: TF1 64.42 / 57.08 and R-L 29.17 / 27.97 in the summary and no-summary settings, respectively.

Inversion outperforms zero-shot prompting. We prompt Qwen with $(x, b^\star, y)$ or (x, y) to synthesize a trace (Appendix D). The resulting traces average $\sim$1,000 tokens, compared with 6,130.6 for R1, and have low overlap (TF1 35.36 / 37.58; R-L 17.60 / 17.97). See Figure 4 in the appendix for an illustrative example.

Table 1: **Selected features of our surrogate summaries vs. GPT-5.4 mini.**

	Ours (C' on R1 traces)	GPT-5.4 mini
Median tokens	537	592
Bold-header sections	94.1%	92.9%
First-person prose	97.3%	97.0%
LaTeX	79.1%	71.9%

Weaker surrogate is effective. Training inversion on the R1-Weak surrogate recovers most of the gap: synthesized traces are long (4,972 / 5,434 tokens, $\approx$81–89% of R1's average) and overlap substantially with the ground truth (TF1 52.76 / 49.01, R-L 24.84 / 21.73). The drop from R1 to R1-Weak is small relative to the jump from zero-shot to any inversion model. This corresponds to a realistic attack where the surrogate is weaker than the victim (see Figure 5 in the appendix).

Summary vs. no-summary Overlap metrics are slightly higher in the summary setting, which demonstrates that summaries provide explicit cues that inversion can latch onto, improving coherence and step-level correctness.

5.3 Stealing Reasoning Capabilities of an Open-Weight Model

We evaluate whether fine-tuning on inverted traces improves downstream reasoning. Using R1 as the victim allows comparison with both attacker-accessible baselines and the Victim-Trace oracle (Table 3). As surrogates, we use R1-Weak (realistic case, where the surrogate is much weaker than the victim) and R1 itself (best case for the attacker).

It is important to note that even though it does not have explicit reasoning, our base instruction-tuned Qwen model appears to have been optimized for the MATH500 benchmark. This means that fine-tuning can *degrade* its performance (71.2%$\rightarrow$61.0% when fine-tuning on answers). This matters for our threat scenario. The most obvious black-box attack—fine-tuning the student on the victim's answers and summaries—can damage the student rather than teach it how to reason. This highlights the utility of inversion models: they turn the victim's outputs into a useful supervision signal.

Synthesized traces beat every attacker-accessible baseline. With the R1-Weak surrogate, Synthesized-Trace beats Answer-only, Summary+Answer, and Surrogate-Trace in almost every experiment. On MATH500, it lifts Qwen from 63.0% (Summary+Answer) to 71.8% and Llama from 45.8% to 50.2%; JEEBench gains are larger (Qwen 24.0%$\rightarrow$36.3%, Llama 18.2%$\rightarrow$24.2%). Improvement over Surrogate-Trace is critical: it shows that **fine-tuning on synthesized traces outperforms fine-tuning on traces from a surrogate reasoning model.**

Table 3: **Downstream accuracy (%) after fine-tuning when attacking R1.** *Victim-Trace (oracle)* is an attacker-inaccessible reference. Entries marked with * outperform this reference.

Setting	Training Sources	Surrogate	MATH500		JEEBench		LiveCodeBench	
			Qwen	Llama	Qwen	Llama	Qwen	Llama
–	No fine-tuning	–	71.2	42.1	28.3	16.6	28.9	13.1
–	Answer-only	–	61.0	44.0	21.6	17.2	25.4	9.6
–	Surrogate-Trace	R1-Weak	63.2	48.8	19.7	11.3	22.9	9.1
Summary	Summary+Answer	–	63.0	45.8	24.0	18.2	25.6	11.7
Summary	**Synthesized-Trace (ours)**	**R1-Weak**	71.8	50.2	36.3	24.2*	30.9	8.8
Summary	Synthesized-Trace (ours)	R1	74.4*	59.2	37.1	26.1*	32.2	9.9*
No-sum.	**Synthesized-Trace (ours)**	**R1-Weak**	66.4	44.8	29.7	16.3	26.2	9.2
No-sum.	Synthesized-Trace (ours)	R1	74.1*	62.0*	33.2	25.2*	29.1	8.6
–	*Victim-Trace (oracle)*	–	*72.2*	*59.6*	*43.7*	*23.8*	*33.2*	*9.8*

Table 4: **Downstream accuracy (%) after fine-tuning when attacking GPT-5.4 mini.**

Setting	Training Sources	Surrogate	MATH500		JEEBench		LiveCodeBench	
			Qwen	Llama	Qwen	Llama	Qwen	Llama
–	Answer-only	–	68.4	13.0	27.6	6.3	0.8	1.4
–	Surrogate-Trace	R1	72.2	59.6	43.7	23.8	33.2	9.8
–	Surrogate-Trace	R1-Weak	63.2	48.8	19.7	11.3	22.9	9.1
Summary	Summary+Answer	–	19.8	16.4	1.6	5.5	0.0	0.4
Summary	Synthesized-Trace (0-shot)	–	57.0	31.4	19.0	11.0	1.4	1.4
Summary	**Synthesized-Trace (ours)**	**R1**	76.0	52.4	43.7	19.9	28.9	10.0
Summary	**Synthesized-Trace (ours)**	**R1-Weak**	73.2	48.3	31.6	16.8	25.6	10.2
No-sum.	**Synthesized-Trace (ours)**	**R1**	71.8	47.2	41.6	16.6	27.7	9.7

No-summary inversion is still effective. Removing $b^\star$ degrades performance but Synthesized-Trace still improves over Answer-only and Surrogate-Trace (e.g., Qwen JEEBench 19.7%→29.7%). With a strong R1 surrogate, no-summary inversion produces traces that help Qwen reach 74.1% on MATH500 and 33.2% on JEEBench, close to the oracle on both.

Synthesized traces approach the oracle. The Victim-Trace oracle is our upper bound. Inverted traces close most of the gap: with an R1-Weak surrogate in the summary setting, Qwen reaches 71.8% vs. 72.2% oracle on MATH500 and 36.3% vs. 43.7% on JEEBench. Llama slightly exceeds the JEEBench oracle (24.2% vs. 23.8%).

Inversion can outperform ground truth when surrogate and victim are the same model. When the surrogate and the victim are the same model (R1, in our case), Synthesized-Trace can even outperform the Victim-Trace oracle. Without summaries, this occurs on Qwen/MATH500 (74.1% vs. 72.2%), Llama/MATH500 (62.0% vs. 59.6%), and Llama/JEEBench (25.2% vs. 23.8%). R1's raw traces often contain backtracking and dead ends, adding noise to student fine-tuning. Conditioned on the final answer, our inversion model produces cleaner forward reasoning, effectively denoising R1's traces and yielding a better training signal. See Appendix Figure 6 for an example.

Weak surrogates hurt student training but help inversion training. Directly distilling R1-Weak can degrade the student (e.g., Qwen MATH500 71.2%→63.2%; LCB 28.9%/13.1%→22.9%/9.1% for Qwen/Llama). When the same traces are used only to train the inversion model, which is then applied to the stronger victim's outputs, inverted traces are much more effective for student training. This explains why fine-tuning on inverted traces outperforms fine-tuning on surrogate traces.

LiveCodeBench gains are weaker. Llama drops with Synthesized-Trace (13.1%→8.8%) and the oracle (13.1%→9.8%), while Qwen improves slightly (28.9%→30.9%). This likely reflects a mismatch between the synthesized traces' math-flavored reasoning style and the coding-specific reasoning required by this benchmark, as well as some formatting differences between the reasoning supervision and the output format LCB expects.

5.4 Stealing Reasoning Capabilities of a Black-Box, Commercial Model

Table 4 evaluates reasoning-capability extraction from a fully black-box commercial victim, GPT-5.4 mini, which exposes only $(x, b^\star, y)$. We train the inversion model offline using either a strong (R1) or weak (R1-Weak) surrogate and then synthesize $\hat{t}$ for student fine-tuning.

Synthesized traces outperform answers and summaries. Synthesized-Trace beats Answer-only and Summary+Answer on both students by large margins. With the R1-trained inverter, Qwen improves MATH500 from 68.4% (Answer-only) to 76.0% and JEEBench from 27.6% to 43.7%; Llama improves on MATH500 from 13.0% to 52.4% and JEEBench from 6.3% to 19.9%. Notably, Summary+Answer often underperforms Answer-only (e.g., Qwen JEEBench: 27.6%→1.6%), suggesting that without inversion, summaries are not helpful for distillation.

Zero-shot inversion underperforms trained inversion. Zero-shot inversion improves over Summary+Answer but remains well below trained inversion: on Qwen, it reaches 57.0% vs. 76.0% on MATH500 and 19.0% vs. 43.7% on JEEBench.

Inversion helps most when the surrogate is weak. With the R1-Weak surrogate, Synthesized-Trace dominates Surrogate-Trace (Qwen JEEBench 19.7%→31.6%, Llama 11.3%→16.8%). With the strong R1 surrogate, Synthesized-Trace is competitive but can fall slightly below Surrogate-Trace (Qwen JEEBench 43.7%→43.7%, Llama 33.2%→28.9%). A strong surrogate already supplies high-quality traces, while inversion must synthesize them indirectly from answers and summaries.

As Table 6 (in Appendix C) shows, R1 is comparable to GPT-5.4 mini (and even stronger on some benchmarks) . In this regime, Surrogate-Trace can be as effective as, or better than trace inversion. We expect that inversion is most advantageous in the (common) setting when the victim is much stronger than the attacker's best surrogate, or when the victim outperforms the surrogate on the target task.

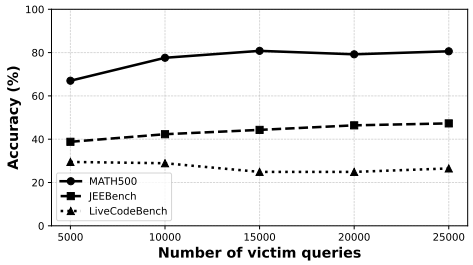

Figure 3: **Student accuracy vs. victim query budget.** Qwen trained on traces synthesized from GPT-5.4 mini using an R1-based inversion model.

Inversion scales with the query budget. Downstream accuracy improves with more victim queries (Figure 3): scaling from 5k to 10k queries raises Qwen from 67% to 76% on MATH500 and from 38.8% to 43.7% on JEEBench; 15k queries reach 80.8% and 44.5%, respectively. More queries expose the inversion model to diverse reasoning patterns, yielding higher-quality synthesized traces. LCB declines slightly with scale, consistent with the style/format mismatch discussed above. Our budgets ($\leq$25k) are far below production-scale distillation corpora (OpenThoughts: 114k; OpenThoughts3: 1.2M Guha et al. (2025)). Further scaling is a direction for future work.

Surrogate augmentation and domain-aware supervision. Two practical adjustments can further improve the student, using the data already available to the attacker (Table 5). First, since R1 is itself a capable reasoning model, R1 surrogate traces $\mathcal{D}_1$ from Stage 1 can be added to the distillation set "for free." Training on $\{(x, \hat{t}, y)\} \cup \mathcal{D}_1$ improves MATH500 (50.2%→51.6%) and JEEBench (15.5%→19.9%). Second, the attacker can *domain-gate* the assistant target by including only the bare answer on code prompts. This sacrifices some accuracy on competition-style coding (LCB) but removes the redundant reasoning prefix that hurts the student elsewhere and sharply improves HumanEval+ Liu et al. (2023) (HE+ 20.7%→51.2%), whose short self-contained Python functions are easier to solve with direct answers than LCB's long multi-constraint problems. The two adjustments compose: combining augmentation and gating yields the best overall trade-off (MATH500 57.6%, JEEBench 21.6%, HE+ 48.2%). The broader point is that **inverted traces give the attacker a flexible training signal**: downstream behavior can be further tuned by mixing surrogate data or adjusting supervision format per domain without any additional queries to the victim.

Capability stealing is economically feasible at moderate query budgets. Using current API pricing for `GPT-5.4 mini` ($0.75/M input tokens; $4.50/M output tokens), collecting 10k $(x, b^\star, y)$ queries costs $173.28. Inversion and student fine-tuning involve no GPT-5.4 mini queries.

Table 5: **Further improvements on Synthesized-Trace (student=Llama, victim = GPT-5.4 mini).**

Training Sources	MATH500	JEEBench	LCB	HE+
Synthesized-Trace	52.4	19.9	**10.0**	20.7
+ surrogate augmentation	51.6	19.9	8.6	20.7
+ domain gating	48.0	15.5	5.3	**51.2**
+ both	**57.6**	**21.6**	6.1	48.2

6 DEFENSES

Defenses against capability stealing aim to reduce the usefulness of a model's outputs as fine-tuning data. They include access control; output restrictions, e.g., query limits and truncation (Tramèr et al., 2016); and perturbing outputs to reduce distillation fidelity while preserving task performance, including undistillable teachers, modified logits, and adaptive noise injection (Ma et al., 2021; 2022; Chen et al., 2025a). Most of these methods were developed for image classifiers and operate on internal signals such as logits or intermediate representations.

Recent defenses perturb reasoning traces to make them less transferable. Antidistillation sampling (Savani et al., 2025) and DOGe (Li et al., 2025) produce traces that yield correct answers but are intentionally difficult to imitate, weakening chain-of-thought supervision. These defenses target attacks that consume the victim's exposed reasoning. In our setting, however, the victim's full reasoning is already hidden: deployed models (OpenAI, 2025; Google, 2025; Anthropic, 2025) expose final answers and, sometimes, short user-facing summaries. Whether these summaries are designed to resist distillation is unknown, but our Answer-only and Summary+Answer results show that direct fine-tuning on the observable outputs is often ineffective.

Even in this setting, trace inversion makes models vulnerable to capability stealing. Without any access to the internal reasoning, operating only on the answers and summaries (if available), trace inversion synthesizes reasoning traces that enable student models to substantially improve their reasoning performance. Because Trace Inversion operates on observable answers and summaries rather than the internal trace, it is not addressed by defenses that operate exclusively by hiding or perturbing the victim's internal reasoning trace.

Complementary work explores watermarking generated text to enable post hoc detection of model use (Kirchenbauer et al., 2023; Gu et al., 2024; Sander et al., 2024). While watermarking does not prevent capability stealing, it may support attribution; whether watermarks survive inversion and subsequent fine-tuning is an open question (Sander et al., 2024).

7 DISCUSSION AND FUTURE RESEARCH

We demonstrated that hiding chains of thought may not protect reasoning capability from theft. Even if a model exposes only a short reasoning summary (or no summary at all), an attacker-trained inversion model can use black-box access to synthesize long-form traces that are more effective than either the model's answers, or surrogate reasoning traces in teaching student models how to reason.

Fine-tuning effectiveness matters more than accurate reconstruction. For closed-source victims, we cannot tell whether our inverted traces accurately match their "true" reasoning traces. It does not matter in practice, however, as long as users can exploit their access to these closed models to improve the reasoning capabilities of their own models.

A key implication for defenses is that *making exposed reasoning hard to imitate is not sufficient*, because an attacker can always ignore it and apply trace inversion to outputs alone. Obfuscated reasoning thus reduces transparency for users yet does not prevent capability stealing.

Even stronger trace inversion is possible. We drew queries from OpenThoughts at a modest budget (at most 25K), compared to corpora such as OpenThoughts itself at 114K and OpenThoughts3 at 1.2M Guha et al. (2025). One natural direction is to scale up: larger query budgets, larger inversion models, and a broader mix of reasoning tasks. A complementary direction is to make inversion models more robust by training them on a variety of summary lengths and styles. Finally, trace synthesis can benefit from verification (e.g., self-consistency for reasoning, execution for code, or tool-based checking for math) to filter or refine traces before student fine-tuning.

AI USE STATEMENT

We used generative AI tools to improve the readability and language of the manuscript and to assist with non-core software engineering tasks, including writing and editing scripts for running experiments and debugging code. Generative AI tools were not used to formulate or refine hypotheses, design the research methodology or experiments, implement the core methods, or interpret experimental results. All AI-assisted text and code were reviewed and verified by the authors. We take full responsibility for the final content of this work.

ETHICS STATEMENT

This paper investigates whether hiding chains of thought (exposing only final answers or short reasoning summaries) prevents reasoning-capability extraction. The sole purpose of this work is to evaluate the limits of existing protections and to motivate development of more robust defenses.

REPRODUCIBILITY STATEMENT

An anonymous artifact for this submission is available at `https://anonymous.4open.science/r/trace_inversion-4367/`. The artifact inventory comprises the code and configurations for constructing surrogate and victim datasets, training inversion and student models, and running the reported evaluations; the exact compression and inversion prompts; data-split manifests; and instructions for reproducing tables and figures.

REFERENCES

Anthropic. Building with extended thinking, 2025. URL `https://platform.claude.com/docs/en/build-with-claude/extended-thinking`. Accessed: 2025-12-12.

Anthropic. Detecting and preventing distillation attacks. `https://www.anthropic.com/news/detecting-and-preventing-distillation-attacks`, February 2026.

Daman Arora, Himanshu Singh, et al. Have LLMs advanced enough? a challenging problem solving benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7527–7543, 2023.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv:2204.05862*, 2022.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *Proceedings of the 30th USENIX Security Symposium*, pp. 2633–2650, 2021.

Huajie Chen, Tianqing Zhu, Lefeng Zhang, Bo Liu, Derui Wang, Wanlei Zhou, and Minhui Xue. QUEEN: Query unlearning against model extraction. *IEEE Transactions on Information Forensics and Security*, 20:2143–2156, 2025a.

Xiaoshu Chen, Sihang Zhou, Ke Liang, Xiaoyu Sun, and Xinwang Liu. Skip-thinking: Chunk-wise chain-of-thought distillation enable smaller language models to reason better and faster. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 12153–12168, 2025b.

Google. Gemini thinking: Thought summaries and internal reasoning, 2025. URL `https://ai.google.dev/gemini-api/docs/thinking`. Accessed: 2025-12-12.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

Tommaso Green, Martin Gubri, Haritz Puerto, Sangdoo Yun, and Seong Joon Oh. Leaky thoughts: Large reasoning models are not private thinkers. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 26518–26540, 2025.

Chenchen Gu, Xiang Lisa Li, Percy Liang, and Tatsunori Hashimoto. On the learnability of watermarks for language models. In *Proceedings of the Twelfth International Conference on Learning Representations*, 2024.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, et al. Openthoughts: Data recipes for reasoning models. *arXiv:2506.04178*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv:2103.03874*, 2021.

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

Trent Holmes and Willem Gooderham. Exploiting DeepSeek-R1: Breaking down chain of thought security. `https://www.trendmicro.com/en_us/research/25/c/exploiting-deepseek-r1.html`, 2025.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *Proceedings of the Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=chfJJYC3iL`.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *Proceedings of the International Conference on Machine Learning*, pp. 17061–17084. PMLR, 2023.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35:22199–22213, 2022.

Kalpesh Krishna, Gaurav Singh Tomar, Ankur P Parikh, Nicolas Papernot, and Mohit Iyyer. Thieves on sesame street! model extraction of bert-based apis. *arXiv:1910.12366*, 2019.

Pingzhi Li, Zhen Tan, Mohan Zhang, Huaizhi Qu, Huan Liu, and Tianlong Chen. DOGE: Defensive output generation for llm protection against knowledge distillation. *arXiv:2505.19504*, 2025.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Yu-An Lu, Ci-Yang Tsai, Yu-Lin Tsai, Raluca Ada Popa, and Chia-Mu Yu. Hidden thoughts are not secret: Reasoning trace exposure in LLMs. *arXiv:2606.00642*, 2026. URL `https://arxiv.org/abs/2606.00642`.

Haoyu Ma, Tianlong Chen, Ting-Kuei Hu, Chenyu You, Xiaohui Xie, and Zhangyang Wang. Undistillable: Making a nasty teacher that cannot teach students. *arXiv:2105.07381*, 2021.

Haoyu Ma, Yifan Huang, Hao Tang, Chenyu You, Deying Kong, and Xiaohui Xie. Sparse logits suffice to fail knowledge distillation. In *Proceedings of the ICLR 2022 Workshop on PAIR^2Struct: Privacy, Accountability, Interpretability, Robustness, Reasoning on Structured Data*, 2022.

Justus Mattern, Fatemehsadat Mireshghallah, Zhijing Jin, Bernhard Schölkopf, Mrinmaya Sachan, and Taylor Berg-Kirkpatrick. Membership inference attacks against language models via neighbourhood comparison. *arXiv:2305.18462*, 2023.

John Xavier Morris, Wenting Zhao, Justin T Chiu, Vitaly Shmatikov, and Alexander M Rush. Language model inversion. In *Proceedings of the Twelfth International Conference on Learning Representations*, 2024.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

OpenAI. Openai o1 system card. https://openai.com/index/openai-o1-system-card/, 2024. Accessed: 2025-12-25.

OpenAI. Openai reasoning models documentation, 2025. URL https://platform.openai.com/docs/guides/reasoning. Accessed: 2025-12-12.

Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4954–4963, 2019.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Alexander Panfilov, David Schmotz, Ilia Shumailov, Luca Beurer-Kellner, Joachim Schaeffer, Ameya Prabhu, Jonas Geiping, and Maksym Andriushchenko. Stealing reasoning traces from proprietary LLM APIs. *arXiv:2608.09867*, 2026. URL https://arxiv.org/abs/2608.09867.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *arXiv:2602.04942*, 2026. URL https://arxiv.org/abs/2602.04942.

Yiting Qu, Ziqing Yang, Chi Cui, Ye Leng, Junjie Chu, and Yang Zhang. EchoCoT: Extracting hidden chain-of-thought from large reasoning models. *arXiv:2608.20055*, 2026. URL https://arxiv.org/abs/2608.20055.

Tom Sander, Pierre Fernandez, Alain Durmus, Matthijs Douze, and Teddy Furon. Watermarking makes language models radioactive. *Advances in Neural Information Processing Systems*, 37: 21079–21113, 2024.

Yash Savani, Asher Trockman, Zhili Feng, Yixuan Even Xu, Avi Schwarzschild, Alexander Robey, Marc Finzi, and J. Zico Kolter. Antidistillation sampling, 2025. URL https://arxiv.org/abs/2504.13146.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *Proceedings of the 2017 IEEE symposium on Security and Privacy*, pp. 3–18. IEEE, 2017.

Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. Distilling reasoning capabilities into smaller language models. In *Proceedings of the Findings of the Association for Computational Linguistics: ACL 2023*, pp. 7059–7073, 2023.

Team Qwen. Qwq: Reflect deeply on the boundaries of the unknown. https://qwenlm.github.io/blog/qwq-32b-preview/, 2024.

Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *Proceedings of the 25th USENIX Security Symposium*, pp. 601–618, 2016.

Eric Wallace, Mitchell Stern, and Dawn Song. Imitation attacks and defenses for black-box machine translation systems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 5531–5546, 2020.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Rui Wen, Zheng Li, Michael Backes, and Yang Zhang. Membership inference attacks against in-context learning. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pp. 3481–3495, 2024.

Yong Yang, Xuhong Zhang, Yi Jiang, Xi Chen, Haoyu Wang, Shouling Ji, and Zonghui Wang. PRSA: Prompt reverse stealing attacks against large language models. *arXiv:2402.19200*, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022.

Collin Zhang, John Xavier Morris, and Vitaly Shmatikov. Extracting prompts by inverting LLM outputs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 14753–14777, 2024a.

Yiming Zhang, Nicholas Carlini, and Daphne Ippolito. Effective prompt extraction from language models. In *Proceedings of the First Conference on Language Modeling*, 2024b.

Kaiwen Zhou, Chengzhi Liu, Xuandong Zhao, Shreedhar Jangam, Jayanth Srinivasa, Gaowen Liu, Dawn Song, and Xin Eric Wang. The hidden risks of large reasoning models: A safety assessment of R1. *arXiv:2502.12659*, 2025.

A EXAMPLES OF SYNTHESIZED REASONING TRACES

Query

If a platoon of soldiers is divided into brigades of 7 people each, 2 soldiers will not be included in any brigade. If the platoon is divided into brigades of 12 people each, again 2 soldiers will not be included in any brigade. What is the minimum number of soldiers that need to be added to the platoon so that it can be entirely divided into brigades of 7 people and brigades of 12 people?

Reasoning Summary

Analyzing platoon size
I'm working through a problem about a platoon size N where N is congruent to 2 mod 7 and 12. Since both constraints yield the same remainder, N can also be written as congruent to 2 mod 84, the least common multiple of 7 and 12. To make the platoon divisible by both brigades of 7 and 12, I need to determine a value x that when added to N, gives a multiple of 84. That means x needs to equal 82 to achieve this.
Clarifying platoon division
Hmm, I'm considering whether the user wants the platoon to be divisible by both 7 and 12, or just separately. To ensure N+x is a multiple of both, I confirm that x must equal 82 since it brings us to the nearest multiple of 84. However, if they want both options available at the same time, I need to clarify that reasoning. I'll summarize the steps clearly: recognizing congruences, using the least common multiple, and noting the minimal addition needed. It's essential to communicate this clearly!
Evaluating additions for divisibility
I'm double-checking my approach. Since N is congruent to 2 mod 7, adding 5 makes it divisible by 7, while adding 10 does the same for 12. But I need the same x for both equations. So I'll start by setting up x congruent to 5 mod 7 and 10 mod 12. To solve it, I can use the Chinese Remainder Theorem. I'll represent x as 7a + 5 and find the suitable value of a to satisfy both equations. It's a good puzzle to crack!
Confirming minimal addition
I've represented x in terms of a to find out how many soldiers to add. By letting a = 12t + 11, I calculate x as 7 multiplied by that, resulting in x = 84t + 82. The minimal addition needed is clearly 82. I want to present my conclusions simply and concisely, so I'll directly state that 82 soldiers need to be added and provide a brief explanation of the reasoning behind this. Time to finalize the answer!

Final Answer

Let the platoon size be N. We have $N \equiv 2 \pmod 7$ and $N \equiv 2 \pmod{12}$. Since gcd(7,12)=1, this implies $N \equiv 2 \pmod{84}$, so $N = 84k + 2$.
To be divisible by both 7 and 12, N + x must be a multiple of 84. Since $N \equiv 2 \pmod{84}$, we need $x \equiv -2 \equiv 82 \pmod{84}$. The smallest nonnegative x is 82.
So 82 soldiers must be added.

Zero-shot, Prompt-based Inversion

** Analyzing platoon size **
Alright, I'm diving into the problem where the platoon size (N) leaves a remainder of 2 when divided by both 7 and 12. This means ($N \equiv 2 \pmod{7}$) and ($N \equiv 2 \pmod{12}$). Since 7 and 12 are coprime (their greatest common divisor is 1), the Chinese Remainder Theorem tells us that there is a unique solution modulo the product of 7 and 12, which is 84. Therefore, ($N \equiv 2 \pmod{84}$). This means (N) can be written as (N = 84k + 2) for some integer (k).
Now, the question is, how many soldiers need to be added to (N) so that the new total is divisible by both 7 and 12? In other words, I need to find the smallest nonnegative integer (x) such that (N + x) is a multiple of 84. Since (N = 84k + 2), we need (84k + 2 + x) to be a multiple of 84. This simplifies to ($x \equiv -2 \pmod{84}$). The smallest nonnegative (x) that satisfies this is 82.
** Clarifying platoon division **
I need to ensure that the platoon can be divided into brigades of 7 and 12 soldiers each. The current platoon size (N) is 84k + 2, which means it's 2 more than a multiple of 84. To make it a multiple of 84, I need to add 82 soldiers. This ensures that the new total is a multiple of 84, and thus can be divided evenly into brigades of 7 and 12 soldiers.
** Evaluating additions for divisibility **
Let's break it down step by step. If (N = 84k + 2), then adding 5 to (N) makes it divisible by 7, and adding 10 makes it divisible by 12. However, I need a single value (x) that works for both conditions. I can use the Chinese Remainder Theorem to find such an (x). I represent (x) as (x = 7a + 5) and (x = 12b + 10). Setting these equal gives (7a + 5 = 12b + 10), or (7a - 12b = 5). I need to find integer solutions for (a) and (b). By trial and error or using the Extended Euclidean Algorithm, I find that (a = 12t + 11) and (b = 7t + 5) for some integer (t). Substituting (a = 12t + 11) into (x = 7a + 5) gives (x = 7(12t + 11) + 5 = 84t + 82). The smallest nonnegative (x) is 82.
** Confirming minimal addition **
I've confirmed that the minimal addition needed is 82. This is because (N + 82 = 84k + 2 + 82 = 84k + 84 = 84(k + 1)), which is a multiple of 84. Therefore, the platoon can be divided into brigades of 7 and 12 soldiers each. The answer is clear and concise: 82 soldiers must be added to the platoon.
So, the final answer is 82 soldiers.

Reasoning Trace Synthesized by Our Inversion Model

Okay, let's try to figure out this problem. So, there's a platoon of soldiers. When they divide them into brigades of 7 each, 2 soldiers are left out. Similarly, when they divide into brigades of 12 each, again 2 soldiers are left out. The question is asking for the minimum number of soldiers that need to be added so that the platoon can be divided evenly into brigades of both 7 and 12 without any leftovers. \n\nFirst, let me parse the problem again to make sure I understand it correctly. The current size of the platoon, let's call it N, satisfies two conditions:\n\n1. When divided by 7, the remainder is 2. In mathematical terms, $N \equiv 2 \mod 7$.\n2. When divided by 12, the remainder is also 2. So, $N \equiv 2 \mod 12$.\n\nWe need to find the smallest number x such that N + x is divisible by both 7 and 12. Because if a number is divisible by both 7 and 12, then it's divisible by their least common multiple (LCM). The LCM of 7 and 12 is 84 since they are coprime (their GCD is 1). Therefore, N + x must be a multiple of 84. So, our goal is to find the smallest x where $N + x \equiv 0 \mod 84$.\n\nBut first, we need to determine what N is. Wait, but the problem doesn't give us the exact number of soldiers in the platoon. It only tells us that $N \equiv 2 \mod 7$ and $N \equiv 2 \mod 12$. Hmm, so maybe we can find N first? But actually, perhaps we don't need to know N exactly because we can express x in terms of N.\n\nWait, but if N is congruent to 2 modulo 7 and modulo 12, then N must be congruent to 2 modulo the least common multiple of 7 and 12. Since 7 and 12 are coprime, their LCM is 84. Therefore, $N \equiv 2 \mod 84$. So possible values of N are numbers like 2, 86, 170, etc. But since we're looking for the minimal x to add to N to make it divisible by 84, maybe we can ignore the actual value of N and just work with the congruence.\n\nWait, but maybe not. Because even though N is 2 mod 84, depending on what N is, the required x to reach the next multiple of 84 would vary. For example, if N is 2, then x would be 84 - 2 = 82. If N is 86, then x would be 84 - 86 = -2, which isn't possible, so we take 84 - (86 mod 84) = 84 - 86 + 84 = 82. Wait, no. Let me think again.\n\nIf $N \equiv 2 \mod 84$, then N can be written as 84k + 2 for some integer $k \geq 0$. To make N + x divisible by 84, we need N + x = 84m for some integer m. Therefore, x = 84m - N = 84m - (84k + 2) = 84(m - k) - 2. To minimize x, we need the smallest positive x. The minimal x occurs when m - k = 1, so x = 84*1 - 2 = 82. But wait, if N is 84k + 2, then the next multiple of 84 is 84(k+1). So the difference is 84(k+1) - (84k + 2) = 84 - 2 = 82. Therefore, regardless of k, the minimal x needed is 82. Is that right?
[... reasoning steps omitted for space ...]
So I think that's correct.\n\nBut let me test with specific numbers. Let's say N is 2. Then adding 82 soldiers gives 84. 84 divided by 7 is 12, no remainder. 84 divided by 12 is 7, no remainder. Good.\n\nAnother example: N=86. 86 divided by 7 is 12*7=84, remainder 2. 86 divided by 12 is 7*12=84, remainder 2. So adding 82 soldiers gives 86 +82=168. 168 divided by 7 is 24, no remainder. Divided by 12 is 14, no remainder. Correct.\n\nAnother example: N=170. 170 divided by 7 is 24*7=168, remainder 2. Divided by 12: 12*14=168, remainder 2. Adding 82: 170+82=252. 252/7=36, 252/12=21. Correct.\n\nThus, regardless of the starting N, as long as it's 2 mod 84, adding 82 soldiers will make it a multiple of 84, hence divisible by both 7 and 12. Therefore, the minimal number of soldiers to add is 82.\n\nBut wait, the problem says \"the minimum number of soldiers that need to be added to the platoon so that it can be entirely divided into brigades of 7 people and brigades of 12 people.\" So, perhaps the answer is 82. But let me check once more.\n\nAlternatively, maybe the problem is interpreted as needing to divide the entire platoon into both 7 and 12 brigades, meaning that the total number must be a multiple of both 7 and 12. Therefore, the smallest number that is a multiple of both 7 and 12 is 84. So the platoon's size N must be adjusted to the next multiple of 84. Since $N \equiv 2 \mod 84$, the next multiple is 84 + (84 -2)=82. Therefore, 82 soldiers need to be added. Yes, that's the same conclusion.\n\nTherefore, the answer is 82.

Figure 4: **Trace inversion against R1.** Zero-shot inversion versus trace inversion with **R1-Distill** as the surrogate model.

Query

In the expression \((x+y+z)^{2026}+(x-y-z)^{2026}\), the brackets were expanded and like terms combined. How many terms of the form \(x^{a} y^{b} z^{c}\) with a non-zero coefficient resulted?

Reasoning Summary

1. The expression \((x + y + z)^{2026} + (x - y - z)^{2026}\) will have non-zero coefficients for terms where the exponents of \(y\) and \(z\) sum to an even number.
2. The coefficient of each term \(x^a y^b z^c\) in the sum is \(\frac{2026!}{a! b! c!} [1 + (-1)^{b + c}]\). This coefficient is non-zero only if \(1 + (-1)^{b + c} \neq 0\), which means \(b + c\) must be even.
3. The problem reduces to counting the number of terms where \(a + b + c = 2026\) and \(b + c\) is even. Since \(a = 2026 - (b + c)\) and 2026 is even, \(a\) is even if and only if \(b + c\) is even.
4. Therefore, we need to count the number of non-negative integer solutions to \(a + b + c = 2026\) where \(a\) is even. This is equivalent to counting the number of solutions where \(b + c\) is even.
5. For each even \(a\), the number of solutions to \(b + c = 2026 - a\) where \(b + c\) is even is \((2026 - a)/2 + 1\). Summing this over all even \(a\) from 0 to 2026 gives \((1014)^2\).
6. Hence, the number of terms with non-zero coefficients is \((1014)^2 = 1,028,196\).

Final Answer

To determine the number of terms of the form \(x^a y^b z^c\) with non-zero coefficients in the expression \((x + y + z)^{2026} + (x - y - z)^{2026}\), we analyze the expansion and combination of terms.
1. **Expansion Analysis**:
   - The general term in \((x + y + z)^{2026}\) is \(\frac{2026!}{a! b! c!} x^a y^b z^c\).
   - The general term in \((x - y - z)^{2026}\) is \(\frac{2026!}{a! b! c!} x^a (-1)^{b+c} y^b z^c\).
[... omitted for space ...]
5. **Summing Solutions**:
   - The total number of solutions is the sum of \((2026 - 2k + 1)\) for \(k\) from 0 to 1013.
   - This sum is equivalent to summing \(2t + 1\) for \(t\) from 0 to 1013, which results in \((1014)^2\).
6. **Conclusion**:
   - The number of terms with non-zero coefficients is \((1014)^2 = 1028196\).

R1 Ground Truth Reasoning Trace

Okay, so I have this problem here: In the expression \\((x + y + z)^{2026} + (x - y - z)^{2026}\\), after expanding and combining like terms, how many terms of the form \\(x^{a} y^{b} z^{c}\\) have a non-zero coefficient? Hmm, interesting. Let me try to work through this step by step. \n\nFirst, I remember that when you expand a trinomial like \\((x + y + z)^n\\), the number of terms before combining is a combination with repetition problem. But in this case, we are combining like terms, so each term is determined by the exponents of x, y, and z. The general term in the expansion of \\((x + y + z)^{2026}\\) would be something like \\(\\frac{2026!}{a! b! c!} x^{a} y^{b} z^{c}\\) where \\(a + b + c = 2026\\). Similarly, for \\((x - y - z)^{2026}\\), each term would be \\(\\frac{2026!}{a! b! c!} x^{a} (-y)^{b} (-z)^{c}\\) = \\(\\frac{2026!}{a! b! c!} x^{a} (-1)^{b + c} y^{b} z^{c}\\). \n\nSo when we add these two expressions together, the coefficients of each term \\(x^{a} y^{b} z^{c}\\) would be \\(\\frac{2026!}{a! b! c!} [1 + (-1)^{b + c}]\\). Therefore, the coefficient will be non-zero only if \\(1 + (-1)^{b + c} \\neq 0\\). That happens when \\((-1)^{b + c} = 1\\), which means that \\(b + c\\) must be even. Therefore, in the sum, the terms where \\(b + c\\) is even will have their coefficients doubled, and those where \\(b + c\\) is odd will cancel out and have zero coefficient. \n\nSo the problem reduces to counting the number of monomials \\(x^{a} y^{b} z^{c}\\) in the expansion where \\(a + b + c = 2026\\) and \\(b + c\\) is even. Since \\(a = 2026 - b - c\\), if \\(b + c\\) is even, then \\(a\\) is even as well because 2026 is even. Wait, 2026 is even, right? Yes, 2026 divided by 2 is 1013, so even. Therefore, if \\(b + c\\) is even, then \\(a = 2026 - (even)\\) is even. So all exponents \\(a\\) will be even, and \\(b + c\\) will be even as well. \n\nBut actually, the key condition is just \\(b + c\\) even, which in turn makes \\(a\\) even, since \\(a = 2026 - (b + c)\\). So the problem is equivalent to counting the number of non-negative integer solutions \\((a, b, c)\\) to \\(a + b + c = 2026\\) with \\(b + c\\) even.
[... reasoning steps omitted for space ...]
Therefore, for each even m, the total number of solutions is m +1, which includes both cases where b and c are both even and both odd. \n\nTherefore, when we compute the total number of terms, we are counting all such cases where b + c is even, regardless of whether b and c are individually even or odd. Therefore, our original conclusion that the number of terms is (1014)^2 is correct. \n\nBut let's revisit the initial intuition where one might think that only terms with b and c even would survive. But in reality, as shown in the example with n=2, the term yz (with exponents 1 and 1) survives because b + c =2, which is even, even though b and c are both odd. Therefore, both cases contribute. \n\nTherefore, the error in that initial intuition was assuming that both exponents must be even, but actually, it's the sum that must be even. Hence, both exponents even or both odd. Therefore, the correct count is indeed the number of solutions with a even, which is (n/2 +1)^2. \n\nTherefore, after thorough analysis, the number of terms with non-zero coefficients in the given expression is \\((1014)^2 =1,028,196\\).\n\n**Final Answer**\nThe number of terms is \\boxed{1028196}.\n

Reasoning Trace Synthesized by Our Inversion Model

Okay, so I have this problem here: I need to find the number of terms with non-zero coefficients in the expansion of (x + y + z)^2026 + (x - y - z)^2026. Hmm, let's see. Alright, first, I remember that expanding a multinomial like (x + y + z)^n can result in a lot of terms, each corresponding to different exponents of x, y, z. But since we're adding another similar expansion with some signs changed, maybe some terms will cancel out, and others will add up. So the key here is to figure out which terms survive after combining like terms.\n\nLet me recall the general approach for expanding such expressions. When you expand (x + y + z)^n, each term is of the form C(n, a, b, c) * x^a y^b z^c where a + b + c = n, and C(n, a, b, c) is the multinomial coefficient n! / (a! b! c!). Similarly, for (x - y - z)^n, the coefficients would be the same but with alternating signs based on the number of negative terms in each term. Wait, actually, in the expansion of (x - y - z)^n, each term is C(n, a, b, c) * x^a (-y)^b (-z)^c = C(n, a, b, c) * x^a (-1)^{b + c} y^b z^c. So when you add the two expansions together, the coefficients for each term x^a y^b z^c become C(n, a, b, c) * [1 + (-1)^{b + c}].\n\nTherefore, the coefficient will be non-zero only if 1 + (-1)^{b + c} ≠ 0, which means (-1)^{b + c} ≠ -1, so (-1)^{b + c} = 1. That implies that b + c must be even. Because if b + c is even, then (-1)^{b + c} = 1, so the coefficients add up. If b + c is odd, they cancel out. So, in the final expression, only the terms where b + c is even will have non-zero coefficients. Therefore, the number of terms we are looking for is the number of triples (a, b, c) such that a + b + c = 2026 and b + c is even.\n\nWait, but a = 2026 - (b + c), so substituting that in, we have 2026 - (b + c) + b + c = 2026, which just checks out. So the condition is that b + c is even. Therefore, the problem reduces to counting the number of non-negative integer solutions (a, b, c) to a + b + c = 2026 where b + c is even. Since a is determined by b and c (a = 2026 - (b + c)), the condition on a would automatically be satisfied because a = 2026 - (b + c), and 2026 is even. Wait, 2026 is an even number.
[... reasoning steps omitted for space ...]
So four terms, which matches the formula ((3 + 1)/2)^2 = 4. So my initial mistake was in the manual calculation where I thought there were three terms, but actually, xyz terms also contribute. So for n = 3, the number of terms is 4. Therefore, the formula holds.\n\nSo the general formula is:\n\nIf n is even: number of terms = ((n/2) + 1)^2\n\nIf n is odd: number of terms = ((n + 1)/2)^2\n\nWait, but in our original problem, n = 2026, which is even. So the number of terms is ((2026/2) + 1)^2 = (1013 + 1)^2 = 1014² = 1,028,196.\n\nTherefore, the answer should be 1,028,196.\n\nBut let me verify this once more with another example. Take n = 4, even. Number of terms = ((4/2)+1)^2 = 3² = 9. As before, correct. For n = 6, ((6/2)+1)^2 = 4² = 16, which matches.\n\nTherefore, applying this formula to our original problem gives the answer. Hence, the number of terms with non-zero coefficients is (1014)^2 = 1,028,196.\n\n**Final Answer**\n\\boxed{1028196 }.\n

Figure 5: **Trace inversion against R1.** Ground-truth reasoning trace versus a trace synthesized with `R1-Distill` as the surrogate model.

B Synthesized Traces Can Be Better Than Ground-Truth Traces

Query

The positive reals x and y satisfy $x^3 + y^3 = x - y$. Show that $x^2 + y^2 < 1$.

R1 Ground-Truth Reasoning Trace

... *Wait, but we already have $x - y = (x + y)(S - P)$ from the equation above.* Wait, let's step back...
Since f is convex for $a > 0$, by Jensen's inequality: $(x^3 + y^3)/2 \geq ((x + y)/2)^3$. *But not sure if helpful here...*
Since $x > y$, we can write $x = y + k$ where $k > 0$. Substitute into the equation: $(y + k)^3 + y^3 = k$. Expanding: $2y^3 + 3y^2k + 3yk^2 + k^3 = k$. *But this seems complicated. Maybe not the best approach...*
The inequality becomes $xy < 2y/(x + y)$. Since $y > 0$, we can divide both sides by y: $x < 2/(x+y)$. Multiply both sides by $(x+y)$: $x(x+y) < 2$. So $x^2 + xy < 2$. *But we need to relate this to our previous expressions. Hmm, not sure if helpful...*

Synthesized Reasoning Trace

Another idea: ... $x^2-xy+y^2 = (x-y)/(x+y)$. But $x^2+y^2 = (x^2-xy+y^2)+xy$, so $x^2+y^2 = (x-y)/(x+y)+xy$.
Let's denote $t = y/x$, since $x > y$, so t is a positive real number less than 1. ... substitute into the equation $x^3+y^3 = x-y$: $x^3(1+t^3) = x(1-t)$. Therefore, $x^2 = (1-t)/(1+t^3)$. Now, $x^2+y^2 = x^2(1+t^2) = (1-t)(1+t^2)/(1+t^3)$. ... Expand the left-hand side: $1 + t^2 - t - t^3 < 1 + t^3$. ... Factor out a t: $t(2t^2 - t + 1) > 0$. Discriminant $D = (-1)^2 - 4 \times 2 \times 1 = 1 - 8 = -7$. Since the discriminant is negative, the quadratic $2t^2-t+1$ has no real roots and since the coefficient of t^2 is positive, it is always positive. ... we conclude that $x^2 + y^2 < 1$."...

Figure 6: **R1 ground-truth trace vs. synthesized trace with R1 as the surrogate. Top:** the query. **Bottom left:** excerpt from R1's ground-truth trace, which tries and abandons several approaches. **Bottom right:** matching excerpt from the trace synthesized by our inversion model (trained with R1 as the surrogate), with more direct reasoning.

C Benchmark Accuracy of Victim and Surrogate Models

We report zero-shot accuracy of R1, `R1-Distill` (R1-Weak), and GPT-5.4 mini on MATH500, JEEBench, and LiveCodeBench (**LCB**) in Table 6.

Table 6: **Benchmark accuracy (%) of victims and surrogates.**

Model	MATH500	JEEBench	LCB
R1	91.6	87.1	74.9
R1-Weak	81.4	32.6	27.6
GPT-5.4 mini	91.4	85.1	66.9

D Prompts for Compression and Zero-shot Trace Inversion

Reasoning Summary Compression Prompt

System prompt.
You are summarizing a long chain-of-thought trace into a short, first-person "inner-monologue recap" that mirrors the style of GPT-5.4 mini's internal reasoning summaries.

Given a `<think>...</think>` trace, produce a recap with these properties:

Structure. Write 3 to 6 short sections. Each section begins with a short bold markdown header on its own line, like `**Setting up the integral**` or `**Checking the edge case**`, and is followed by one short paragraph (2–5 sentences). Do NOT use numbered lists (`1.`, `2.`) and do NOT use bullet points (`-`, `*`). The whole recap is just

headers + prose.

Voice. First person, present tense, as if the model is thinking aloud: "I need to…", "I'll check…", "I'm now realizing…", "Let me verify…". Keep the tone tentative and exploratory, not textbook.

Content. Each section should capture one meaningful move in the reasoning — a clarification of what is being asked, a key derivation or substitution, a pivot after a failed attempt, a sanity check, or the final consolidation. Skip filler, restatement, and purely mechanical arithmetic. Preserve the logical order of the trace.

Length. Aim for roughly 600–900 tokens total — long enough that each section develops a real idea, not just a one-line gesture. If the input trace is very short, produce fewer sections rather than padding; if it is very long, prefer adding depth to each section over adding more sections.

Math formatting. Use inline LaTeX (`\frac`, `\sqrt`, `\boxed`, etc.) where the original trace used math. Do not restate the final boxed answer unless the reasoning naturally concludes with it.

Do not add meta-commentary about following instructions, apologize, or mention that you are summarizing. Just produce the recap.

The prompt additionally includes two few-shot exemplars of the target style (one algebra / number theory and one geometry, ≈600 tokens each), drawn from GPT-5-mini's own summaries. They are omitted from this box for space; the full prompt including exemplars is released with our code.

User prompt.
Summarize this thinking process as a first-person inner-monologue recap:
`<think>{thinking_content}</think>`

Zero-shot Inversion Prompt (With Summaries)

You are a language model that reconstructs full internal reasoning traces from high-level bubble summaries.

You will be given:
- A problem **input** (e.g., a math or logic problem)
- A final **output** or solution
- A list of numbered **reasoning bubbles**, where each bubble summarizes one key insight, step, or decision made during the problem-solving process

These bubbles are **condensed summaries** of what was originally a much longer, richer internal thought process.
Your task is to reconstruct that full process.

Below are high-level bubble summaries representing condensed thoughts or decisions. Your task is to reconstruct the full thinking trace that might have led to each summary. For each bubble, expand it into a **detailed internal monologue or reasoning chain**, showing how one idea leads to the next.

Include:
- Assumptions and background intuitions
- Intermediate steps, definitions, and subcases
- Natural questions or doubts raised during reasoning
- Alternatives that were considered and rejected

- Transitions that make the reasoning coherent and plausible

Use informal, introspective language — as if the person is thinking out loud. Add math expressions in LaTeXwhere appropriate.

Do **not** invent new reasoning steps outside the bubbles. Use the **input** and **output** only for context and consistency. Your goal is to **flesh out the bubbles**, not to re-solve the problem from scratch.

The full trace should:
- Be logically consistent and cohesive from start to finish
- Sound like a realistic thought process that could plausibly result in the given answer
- Span multiple paragraphs per bubble and up to 20,000 characters overall if needed
- Be output as one continuous trace, wrapped in ‘¡think¿...¡/think¿‘ tags

You are not summarizing the bubbles. You are recovering the internal narrative that *generated* them.

The original problem input is: {`user_prompt`}
The final answer is: {`assistant_answer`}
Transform this thinking bubbles into clear full reasoning traces: {`reasoning_summary`}
Generate full reasoning traces:

Zero-shot Inversion Prompt (No Summaries)

You are a language model that reconstructs full internal reasoning traces from only an **input** (e.g., a math or logic problem) and a corresponding **output** (final solution or answer).

You will be given:
- A problem **input**
- A final **output** or solution

Your task is to reconstruct the full internal reasoning process that could plausibly connect the input to the output. This should be a long, detailed, introspective trace, not a short summary.

Guidelines for the reasoning trace:
- Write in the style of an informal, introspective monologue, as if the person is thinking out loud.
- Include assumptions, intuitions, and background facts as they arise naturally.
- Show intermediate steps, calculations, logical deductions, definitions, and subcases.
- Raise natural questions or doubts during reasoning, and explain how they are resolved.
- Explore alternative approaches, even ones that are discarded, and explain why.
- Make transitions clear so the reasoning feels like a coherent train of thought.
- Use LaTeXfor math expressions where helpful.
- Do not introduce new information inconsistent with the input or output.
- The goal is depth, not brevity: expand ideas fully, elaborate with multiple paragraphs, and let the reasoning unfold gradually.
- The output should only appear at the end, after the reasoning is complete.

The full trace should:
- Wrapped in ¡think¿...¡/think¿ tags.
- Be logically consistent and cohesive from start to finish.
- Sound like a realistic thought process that could plausibly result in the given answer.
- Span thousands of tokens if needed.

The original problem input is: {`user_prompt`}
The final answer is: {`assistant_answer`}
Generate full reasoning traces: