Soft Contamination Means Benchmarks Test Shallow Generalization

Ari Spiesberger[*] Arb Research arispiesberger@gmail.com
Juan J Vazquez[*] Arb Research juan@arbresearch.com
Nicky Pochinkov Arb Research
Tomáš Gavenčiak Charles University
Peli Grietzer Arb Research
Gavin Leech[†] Arb Research
Nandi Schoots[†] University of Oxford

Abstract

Training on benchmark test items is a classic way to overestimate generalization to unseen problems. Standard decontamination methods miss *semantic duplicates* (SemDs): training items equivalent in content to a test item, but distant in string space. We call training on such items *soft contamination*; when the resulting gains extend to held-out items of the same benchmark (but not to related benchmarks), we call the result *shallow generalization*. We first embed Olmo 3's entire post-training dataset and search it for SemDs of benchmark items; we then finetune Olmo 3 and Qwen models on natural and synthetic semantic duplicates. We find (1) that soft contamination is widespread: the Olmo corpus contains SemDs of 76% of the CodeForces test set and 100% of MBPP's; (2) that finetuning on SemDs of a benchmark raises accuracy on its held-out items by up to 22pp in controlled experiments and 10–11pp at realistic contamination rates, though the effect varies across benchmarks; (3) that the gains from training on SemDs mostly do not transfer to conceptually similar benchmarks, i.e. we find shallow generalization; (4) that finetuning on items with similar embeddings that are not SemDs produces no such gain, indicating that the mechanism is not mere topical similarity; and (5) that at realistic contamination rates, soft contamination can raise held-out performance about as much as exact duplicates (or more). Benchmark gains here are therefore confounded - in a way that standard decontamination does not address.

1 Introduction

LLMs have rapidly solved many hard reasoning benchmarks, some of which are nearing saturation even for smaller models [Edelman and Lee, 2025, Maslej et al., 2025]. But LLM training corpora have also grown by a factor of thousands since 2020 [Epoch AI, 2025]. As corpora grow - in the absence of proportionate filtering effort - we should expect more benchmark test items to be included in them. While AI labs make good-faith efforts to remove syntactic duplicates of benchmark items from their corpora [OpenAI et al., 2024, Olmo et al., 2025, Anthropic, 2025c], 'softer' forms of contamination are hard to detect. These 'semantic duplicates' are data distant from test data in string space, but (as we will show) still containing learnable structure about the test data. This paper evaluates this potential confounder of benchmark scores.[3]

[*]Equal contribution.
[†]Joint supervision.
[3]The question is sharpened by the "approximate retrieval" view of LLMs [Kambhampati, 2023, 2024, Meursault, 2025], on which some outputs are similarity-weighted combinations of nearby training documents.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Table 1: An example of a semantic duplicate of test data found by our pipeline.

Original MBPP sample 297

Write a function to flatten a given nested list structure.

```
def flatten_list(list1):
    result_list = []
    if not list1:  return result_list
    stack = [list(list1)]
    while stack:
        c_num = stack.pop()
        next = c_num.pop()
        if c_num:  stack.append(c_num)
        if isinstance(next, list):
            if next:  stack.append(list(next))
        else:  result_list.append(next)
    result_list.reverse()
    return result_list
```

Semantic duplicate from `dolmino`

```
:param nested_list:  A list that
    may contain other lists
:return:  A flattened list of
    all elements
"""

flattened_list = []
for element in nested_list:
    if isinstance(element, list):
        # Recursively flatten the sublist
        flattened_list.extend(
            flatten_nested_list(element))
    else:
        # Append non-list elements
        flattened_list.append(element)
return flattened_list
```

We combine existing contamination detection methods (embedding search and LLM judges) with novel finetuning experiments. When models train on semantic duplicates of some benchmark items, their accuracy on the benchmark's held-out items can rise too, while their accuracy on related control benchmarks mostly does not: *shallow generalization* from semantic duplicates of test data to test data. Using the open-data model Olmo 3, we show that modern LLM training corpora include data that are *effectively* samples from major reasoning benchmark test sets, which would make benchmark scores partly a measure of shallow generalization - rather than of the intended general reasoning capability. We hypothesize that increased LLM performance on reasoning benchmarks partly reflects an increase in soft contamination.

We define an *exact* duplicate of test data to be a training datapoint which contains a test set example verbatim, up to formatting. A *semantic* duplicate of test data is an example in the training corpus that shares the algorithmic core or logical structure of a test item - despite different surface framing, categories or wording [Riddell et al., 2024] - so that learning to solve it suffices to solve the test item. Our LLM judges count a pair as semantic duplicates when the training example states the same *task* as the test item, or a task that subsumes it, or if it supplies the insight for the test problem (Appendix A.7). We call contamination *soft* when it involves semantic duplicates. We call generalization *shallow* when gains from training on duplicates of some benchmark items extend to the benchmark's held-out items, but *not* to related benchmarks.

Our contributions:

• **High rates of soft contamination in the wild:** We use embedding distances to screen for natural semantic duplicates in 1% of the pretraining data and all of the finetuning data of Olmo 3 – around 260GB of text, significantly more than previous studies. Despite the industry-standard decontamination performed by the Olmo team, we find more contamination than previous studies.

• **Shallow generalization:** We finetune Olmo 3 on a random split of MuSR, ZebraLogic, and MBPP. We find that performance improves on the unseen half, while control benchmarks drawn from the same domain generally do not improve. On MuSR, both semantic and exact duplicates add $\sim$20 pp to seen and unseen items alike. On MBPP, exact duplicates help seen items more, (17pp vs 9pp), while semantic duplicates help unseen items more (7pp vs 2pp).

• **Duplication drives gains, not mere similarity:** Finetuning on the corpus items closest to benchmark items in embedding space that are not duplicates does not improve benchmark scores (Table 4), so topical similarity alone does not explain the gains.

• **Natural levels of soft contamination have a substantial effect:** We estimate the natural rate of semantic duplication: 4 items in 10,000. We run the first ecologically-valid finetuning study of exposure to semantic duplicates: at this natural rate, synthetic semantic duplicates inflate scores on held-out MuSR items by 10–11pp ($p < 0.05$) in Olmo 3, Qwen3 and Qwen3.5.

• **True hold-out has never been tried:** We view this work as a step towards evaluating LLMs in the empirical risk minimization setting [Vapnik, 1991]: a strict separation between training and test sets, verified on open training data.

Table 2: **The literature on data contamination by semantic duplicates**, whether it: studies *semantic* duplicates of test data ( ); estimates the prevalence of contamination in training corpora ( ); has automated screening for semantic overlap ( ); quantifies the effects of contamination on downstream performance ( ); employs finetuning ( ) and whether the FT data are ecologically valid ( ); tests *in-benchmark generalization* (whether training on duplicates of some benchmark items improves performance on held-out items,); and whether it does training data attribution ( ).

Paper									Data scale	Model scale	Major evals
Ours	✓	✓	✓	✓	✓	✓	✓	✓	260G	*7B*	*CodeForces, MBPP (et al.)*
Magar and Schwartz [2022]	✗	✗	-	✓	✓	✗	✗	✓	2.5G	0.3B	SST-5, SST-2, SNLI
Yang et al. [2023]	✓	✓	✓	✓	✓	✗	✗	✓	23G	13B	MMLU, GSM8K
Riddell et al. [2024]	✓	✓	✓	✓	✗	-	✗	✓	155G	16B	MBPP, HumanEval
Shilov et al. [2025]	✗	✗	-	✓	✓	✗	✗	✗	45G	2.7B	MIA
Xu et al. [2025]	✓	✗	✓	✓	✗	-	✗	✗	1G	72B	LIAR2

2 Related Work

Contamination of LLM training corpora by benchmark test items is a large field. Table 2 covers key differences between our work and prior studies. Yang et al. [2023], Riddell et al. [2024], and Xu et al. [2025] establish that semantic duplicates exist and raise the scores of duplicated items. We differ from them as follows: 1) Our definition of duplication is *functional*: a training item x is a duplicate of a test item t if learning to solve x suffices to solve t. This admits training items that pose a more general problem solved by the same algorithm, which other definitions exclude. 2) We search a larger corpus and estimate natural contamination rates. 3) We evaluate on held-out benchmark items and against control benchmarks.

The first wave of contamination studies [Magar and Schwartz, 2022, Elazar et al., 2024, Jiang et al., 2024, Dong et al., 2024] focused on the prevalence and impact of exact syntactic duplicates (i.e., word-for-word matches in string space), with later work [Dekoninck et al., 2024, Zhou et al., 2025, Shilov et al., 2025] extending the analysis to partial syntactic duplicates. More recently, researchers have begun to study indirect or 'semantic' contamination [Yang et al., 2023, Riddell et al., 2024, Xu et al., 2025], where data in the training corpora are equivalent to benchmark-items in substantive content without sharing n-gram sequences or other syntactical properties.

The literature often treats semantic contamination as a form of memorization, focusing on how exposure in training to near-duplicates improves a model's performance on those specific benchmark items. Our work instead studies semantic duplicates as a source of *shallow generalization*: gains that extend from duplicated benchmark items to *unduplicated* items from the same benchmark, but not to related benchmarks. Because the gains reach unduplicated items, we still call this 'generalization'. Dominguez-Olmedo et al. [2024] study a related phenomenon they call 'training on the test *task*'.

To identify exact duplicates of benchmark data, prior work deploys two approaches: searching for duplicates of benchmark items in open datasets [Magar and Schwartz, 2022, Elazar et al., 2024, Jiang et al., 2024, Dong et al., 2024], and testing for memorization using membership inference techniques [Shi et al., 2023]. But memorization tests (which typically depend on matching strings) are unlikely to capture the effect of exposure to semantic duplicates, and so searching open datasets is preferred. To find semantic duplicates, a heuristic semantic distance is often used to guide the search, alongside human judgment, LLM judges, or plagiarism-detection software. Previous work by Yang et al. [2023] and Riddell et al. [2024] has provided high-quality estimates of the prevalence of semantic duplicates of items from major reasoning benchmarks - including HumanEval, MMLU, and GSM8K [Chen et al., 2021, Hendrycks et al., 2020, Cobbe et al., 2021] - in widely-used training corpora, e.g. The Pile, StarCoderData, and RedPajama [Gao et al., 2020, Li et al., 2023, Weber et al., 2024].

To measure the effect of contamination, prior work either 1) finetunes on contaminated data to simulate exposure [Yang et al., 2023] or 2) correlates how often items are duplicated with model performance on them, i.e. finds the 'dosage' of exact duplicates [Riddell et al., 2024, Kocyigit et al., 2025]. We combine the two approaches by finetuning on semantic duplicates at natural rates and testing the gains on held-out benchmark items.

3 Methodology

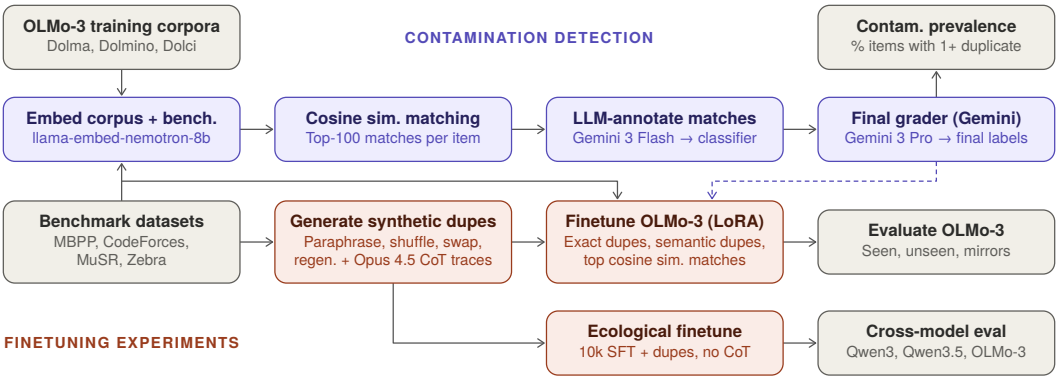


Figure 1: **Our full pipeline from input data to experimental outcomes.**

Figure 1 shows our full pipeline. To know what exactly the model has seen in training, our main experiments use `Olmo-3-7B`, an open-*data* model [Olmo et al., 2025]. We use the closed-data Qwen3 and Qwen3.5 only in our ecological experiment (Section 4.5), where the clean and contaminated runs start from the same model (so that any prior exposure will affect both runs). Appendix A.5 describes our four benchmarks: we search the Olmo 3 corpus for duplicates of all four, and finetune on duplicates of MuSR, ZebraLogic and MBPP. All code on GitHub:

`https://github.com/AriSpiesberger/Soft-Contamination-Prevelance`

3.1 Finding semantic duplicates in the wild

We find semantic duplicates in the wild in two stages: 1) we retrieve the training items with the highest cosine similarity to each benchmark item, then 2) we annotate a sample of these matches with an LLM judge. The second stage is needed because high cosine similarity does not imply algorithmic duplication.

Olmo 3 corpus. We embed a 1% random sample of Olmo 3's pretraining mix (Dolma 3 and Dolmino) and all of the Olmo 3 post-training datasets (Dolci SFT, Dolci Instruct DPO and Dolci Instruct RL). To sample the pretraining data, we use stratified reservoir sampling, which preserves the corpus's hierarchical structure.

See Appendix A.1 for more dataset details.

Embeddings. We embed the above datasets with `llama-embed-nemotron-8B` [Babakhin et al., 2025], which was ranked second on the Massive Text Embedding Benchmark (MTEB) leaderboard [Muennighoff et al., 2023] at the time of writing. Embedding 260GB of chunked raw text at FP16 precision took about two days on 8 H100 GPUs. We describe our end-to-end discovery of semantic duplicates below.

Cosine similarity. To find candidate duplicates, we compute the cosine similarity between each benchmark item and every embedded corpus item (Appendix A.3).

Semantic and exact duplicates among high cosine similarity matches. For MuSR, the highest cosine similarity is around 0.4, and on manual review we find no duplicates or MuSR-like problems in the post-training data or in the pretraining sample. For ZebraLogic, we find many exact duplicates and a few semantic duplicates; most other high-similarity matches are easy 'Zebra puzzles', far simpler than ZebraLogic items. For MBPP and CodeForces, after manually inspecting the top matches, we sample matches and annotate them with an LLM.

Finding exact duplicates of the ZebraLogic test set in Dolma. We observed several ZebraLogic tasks in the training corpus, verbatim. We discuss this auxiliary result in Appendix A.2.

LLM annotation among high cosine similarity matches. For each MBPP and CodeForces item and each of the five training datasets, we take candidates in the top 0.1% of cosine similarity, and from these either the top 100 or a random sample of 100. We prompt `gemini-3-flash-preview`

[Google DeepMind, 2025] to classify each pair. It outputs a boolean isSemanticDuplicate label, a relationship type and an explanation (definitions in Appendix A.8).

To validate this pipeline, we manually annotate a stratified sample of 190 items across CodeForces and MBPP (released on GitHub; also Appendix A.8). The judge counts at least 96% of the pairs we label equivalent or exact as duplicates, misses 30.4% of the pairs we label as CodeForces subsets and 16.5% of MBPP subsets, and counts at most 5.5% of our non-duplicate pairs as duplicates (Appendix A.8). Our prevalence estimates are thus lower bounds. To reduce false positives, we run a second pass with `gemini-3-pro` on pairs flagged by the first judge, checking if the corpus item supplies the algorithm needed to solve the test problem. Appendix F gives examples of semantic duplicates found by our pipeline, and Appendix A.7 details the annotation process. Appendix A.6 covers our generation of synthetic semantic duplicates for each benchmark.

3.2 Finetuning on Duplicates

Our experiments distinguish three kinds of gains: 1) gains on a benchmark item from training on its own semantic duplicates [Yang et al., 2023, Riddell et al., 2024]; 2) gains on a benchmark item from training on exact duplicates of *other* items in the benchmark; and 3) gains on a benchmark item from training on semantic duplicates of other items in the benchmark. Gains of kinds 2 and 3 are shallow generalization when they do not extend to related benchmarks.

We finetune Olmo 3 Instruct on either exact duplicates or synthetic semantic duplicates (Appendix A.6) of MuSR, ZebraLogic and MBPP items. Because Olmo 3 Instruct is a non-reasoning model, we first have a teacher model, Claude Opus 4.5, generate chain-of-thought (CoT) reasoning traces [Wei et al., 2022] for these items (Appendix C). We finetune Olmo 3 Instruct with LoRA [Hu et al., 2022] on the formatted questions and their CoT answers (Appendix A.6.3). To estimate accuracy, we sample 8 generations per item at temperature 0.7 from the base model, the teacher and the finetuned model.

We split each benchmark in half, finetune on duplicates of one half (the *seen* items), and evaluate on the original items of both halves (*seen* and *unseen*). To test whether gains extend to related benchmarks, we use control benchmarks: TrueDetective [Del and Fishel, 2023] for MuSR, ARC-Challenge [Clark et al., 2018] for ZebraLogic and HumanEval [Chen et al., 2021] for MBPP. These controls are imperfect. Olmo 3 scores near chance on TrueDetective (28%, against about 24% for random guessing), and ARC-Challenge tests science questions rather than logic puzzles. We also evaluate on ARC-Easy, BoolQ [Clark et al., 2019], HellaSwag [Zellers et al., 2019], PIQA [Bisk et al., 2020] and WinoGrande [Sakaguchi et al., 2021] to check for any accidental degradation in off-target capabilities.

4 Results

4.1 Decontamination methods fail

Consistent with Yang et al. [2023], we find that standard decontamination methods do not catch soft contamination. We apply the decontamination methods from recent model releases, plus standard baselines, to the semantic duplicate pairs we found (Table 3). No single method removes more than 6.5% of them.

4.2 There are exact duplicates in training corpora

Surprisingly, we found that the Olmo Instruct RL dataset contains exact duplicates for ZebraLogic, one of the benchmarks the Olmo team reports: at least 49.5% of its test set (Appendix A.2). (The Olmo team have since fixed this error.)

4.3 There are natural semantic duplicates in training corpora

Relationship of cosine similarity to semantic duplicate status. Figure 2 plots the rate of semantic duplicates across cosine similarity bands, for pairs sampled from the top 0.1% of cosine similarity. The duplicate rate rises steeply with similarity: it is near zero at the low end and reaches 60–85% above 0.8. This supports restricting LLM annotation to the extreme right tail, but makes our prevalence figures lower bounds (since duplicates below the cut are not counted).

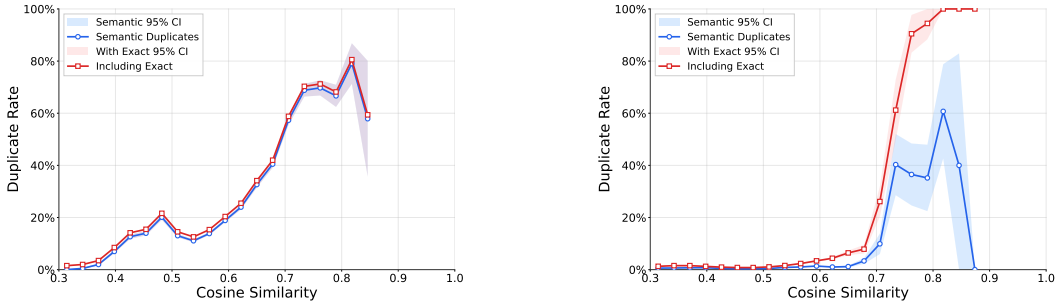
Table 3: **Standard decontamination removes few of the semantic duplicates we find.** Percentage of confirmed in-the-wild duplicate pairs removed when we reproduce published decontamination pipelines and standard baselines. The most effective method removes 6.5%; all methods combined remove 10.87% (CodeForces) and 0.57% (MBPP). The code-clone and code-embedding methods are not run on CodeForces, whose samples are mostly text. See Appendix E.1 for a more complete overview of known decontamination pipelines from popular model releases.

Method	CodeForces	MBPP
Opus 4.5 (full pipeline) (Anthropic [2025c], §2.2)	0.78%	0.04%
GPT-4 50-char substring match [OpenAI et al., 2024]	6.52%	0.27%
GPT-3: 13-gram overlap [Brown et al., 2020]	6.09%	0.10%
Olmo 3 (full pipeline) (Olmo et al. [2025], §3.5.3)	5.83%	0.10%
MinHash (Jaccard ≥ 0.8) + 13-gram containment	0.09%	0.00%
Code-clone: MinHash + Type-2 + CodeBLEU [Ren et al., 2020]	n/a	0.21%
Code embedding: UniXcoder (≥ 0.8) [Guo et al., 2022]	n/a	0.20%
All combined	10.87%	0.57%

Benchmark examples with at least one semantic duplicate. Every MBPP problem has at least one semantic duplicate among its top 100 matches per training dataset, and 99.6% have one among 100 matches sampled from the top 0.1%. For CodeForces, 75.9% of problems have at least one semantic duplicate among their top 100 matches. *Contamination by semantic duplicates is therefore widespread.* These figures are lower bounds: checking beyond the top 100 matches would very likely find more semantic duplicates for CodeForces. Easier CodeForces problems (those with lower Elo ratings) are more likely to have a semantic duplicate (Appendix A.3).

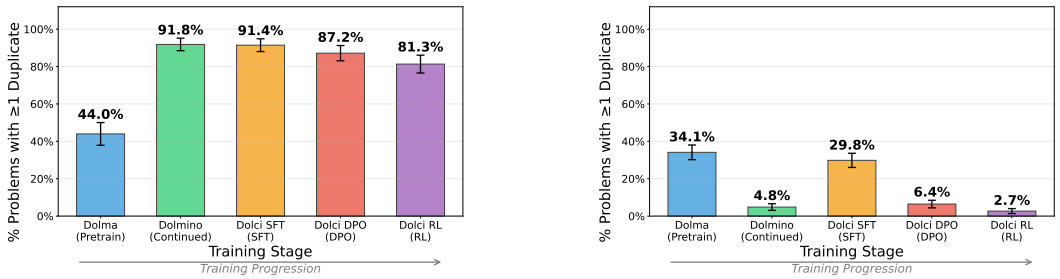
Semantic duplicate occurrence stratified by training dataset. In Figure 3, we investigate how the prevalence of semantic duplicates varies across different stages of the training pipeline. Figure 3a shows that all training datasets have semantic duplicates for MBPP. In Figure 3b, we find a similar effect for CodeForces: semantic duplicates of problems exist in each dataset, particularly in Dolma, Dolci SFT, and Dolci DPO.

Retrieving more matches finds more duplicates. Semantic duplicates are rare within the corpus, even though most test items have at least one such duplicate. In Appendix A.3, we show that the proportion of benchmark problems with at least one semantic duplicate grows with the number of matches we check per item. For instance, only 28.4% of CodeForces problems have a semantic duplicate as their top-1 match. Searching deeper, and in a larger corpus than previous studies (Table 2), may explain why we find more semantic duplicates; differences in benchmarks and in how we define a duplicate may also contribute.



(a) MBPP (b) CodeForces

Figure 2: **The relationship between cosine similarity and rates of semantic duplicates.** For each benchmark datapoint we sample 100 matches from the top 0.1% cosine similarity matches in the training data. We plot the fraction of cosine similarity matches that are true semantic duplicates. the rate of semantic *or* exact duplicates are plotted in red, while blue is the semantic-duplicate-only rate.



(a) MBPP

(b) CodeForces

Figure 3: **Occurrence of semantic duplicates by Olmo training phase for two benchmarks: (a) MBPP, (b) CodeForces.** For each benchmark sample, we review the top 100 cosine similarity training datapoints to identify if any of the samples are semantic duplicates. We then calculate the proportion of all benchmark datapoints that have at least one semantic duplicate. Error bars are ± 1 std.

4.4 Finetuning on Semantic Duplicates

MuSR. We experimented with three levels of synthetic semantic duplication, with increasing dissimilarity, generated as detailed in Appendix A.6. Table 4 shows that when we finetune on duplicates of half of the benchmark ($n = 125$), accuracy rises equally on the seen and unseen halves. Finetuning on exact duplicates of MuSR benchmark data leads to similar gains as finetuning on a variety of semantic duplicates: in both cases accuracy rises by about 20pp. Finetuning instead on the corpus items with the highest cosine similarity to MuSR items barely changes accuracy. Accuracy on the TrueDetective control stays flat, though Olmo 3 scores near chance on it (Section 3.2). Appendix C also shows that using a better teacher model to generate CoT reasoning traces results in further gains for finetuning.

ZebraLogic. Table 4 shows that finetuning on exact duplicates of half of the benchmark ($n = 500$) raises accuracy on both halves (seen +11.5pp, unseen +6.5pp). Unlike on MuSR, semantic duplicates barely help ZebraLogic solutions, and one combination (shuffling, substitution and paraphrasing) actually lowers accuracy by 8.5–8.9pp on both halves, while the ARC-Challenge control-benchmark score stays flat. We tentatively attribute this to compounding distribution shift in both the puzzles and their CoT traces: ZebraLogic's substitution operates on domains (e.g., color→shape) in a way MuSR's tree-based regeneration does not, so the teacher's reasoning traces shift accordingly, and stacking all three transformations likely pushes training too far from the test distribution to support transfer. Finetuning on the most cosine-similar corpus items lowers accuracy from 36.9% to 22.9%.

(These results come with a caveat: Olmo 3 Instruct's RL data already contains exact duplicates of about half of ZebraLogic, see Appendix A.2: many 'unseen' items were thus in the model's training data before our finetuning.)

MBPP. Table 4 shows that finetuning on exact duplicates of half of the benchmark ($n = 210$) raises accuracy on that half by 16.6pp, while accuracy on the unseen half barely changes (+2.4pp). Semantic duplicates have a smaller effect on seen items (+8.7pp) but also raise unseen accuracy (+7.2pp). Finetuning on cosine-similar non-duplicates does not substantially improve accuracy. Unlike on the other two benchmarks, semantic duplicates of MBPP also raise accuracy on the HumanEval control, from 55.3% to 67.0%. As with ZebraLogic, the model was already contaminated: the Olmo 3 corpus contains semantic duplicates of every MBPP test item (Section 4.3).

Overall, we observe shallow generalization.

When duplicates raise accuracy on seen items, they mostly raise it on unseen items too: the gap between the halves is at most 3pp, except for exact duplicates of MBPP (15pp) and ZebraLogic (5pp). This suggests within-benchmark generalization, as predicted. On MuSR and ZebraLogic, the control benchmarks stay flat, thus completing the pattern we call shallow generalization (subject to the caveats in Section 3.2). On MBPP, semantic duplicates also improve the HumanEval control. Finetuning on high-cosine-similarity non-duplicates does not improve benchmark performance.

Table 4: **Finetuning Olmo 3 on semantic duplicates**. We see large gains from training on semantic duplicates, in MuSR's case statistically equivalent to training on exact duplicates (See Section A.6 for synthetic generation details). The rightmost column is the control, showing accuracy on second similar-domain benchmarks: for MuSR this is 'True Detective' [Del and Fishel, 2023]; for ZebraLogic this is the ARC Challenge [Clark et al., 2018] and for MBPP this is HumanEval [Chen et al., 2021]. We see no significant improvement to the control scores when finetuning on any setting of MuSR or ZebraLogic duplicates; we do see a surprising leap in HumanEval (MBPP's control) for semantic duplicates. Accuracy (%) $\pm$ stderr. Abbreviations: P=Paraphrase, SH=Shuffle, SU=Substitution, cos similarity=high cosine similarity corpus matches. Experiments were performed on 8 A100 GPUs, each experiment taking around 2-3 hours.

MuSR

Degree of similarity	Seen accuracy	Unseen accuracy	Acc. on TrueDetective
No finetuning	–	66.0 ± 3.0	28.3 ± 3.3
Exact dupes	87.9 ± 2.1	87.3 ± 2.1	27.7 ± 3.2
Sem. dupes (regen)	85.8 ± 2.2	86.2 ± 2.2	29.3 ± 3.3
Sem. dupes (1-branch)	85.7 ± 2.2	86.0 ± 2.2	28.3 ± 3.3
Sem. dupes (n-branches)	87.5 ± 2.1	87.9 ± 2.1	29.8 ± 3.3
cos similarity (SFT)	–	68.6 ± 2.1	25.1 ± 3.1
cos similarity (RL)	–	65.3 ± 2.1	26.7 ± 3.2

ZebraLogic

Degree of similarity	Seen accuracy	Unseen accuracy	Acc. on ARC Challenge
No finetuning	–	36.9 ± 1.5	50.1 ± 1.5
Exact dupes	$\mathbf{48.4} \pm 1.6$	$\mathbf{43.4} \pm 1.6$	49.5 ± 1.5
Sem. dupes (P)	39.2 ± 1.5	36.2 ± 1.5	49.3 ± 1.5
Sem. dupes (SH,SU)	36.0 ± 1.5	36.8 ± 1.5	50.7 ± 1.5
Sem. dupes (SH,P)	38.0 ± 1.5	36.0 ± 1.5	49.4 ± 1.5
Sem. dupes (SH,SU,P)	28.0 ± 1.4	28.4 ± 1.4	50.4 ± 1.5
cos similarity (SFT)	–	22.9 ± 1.3	–

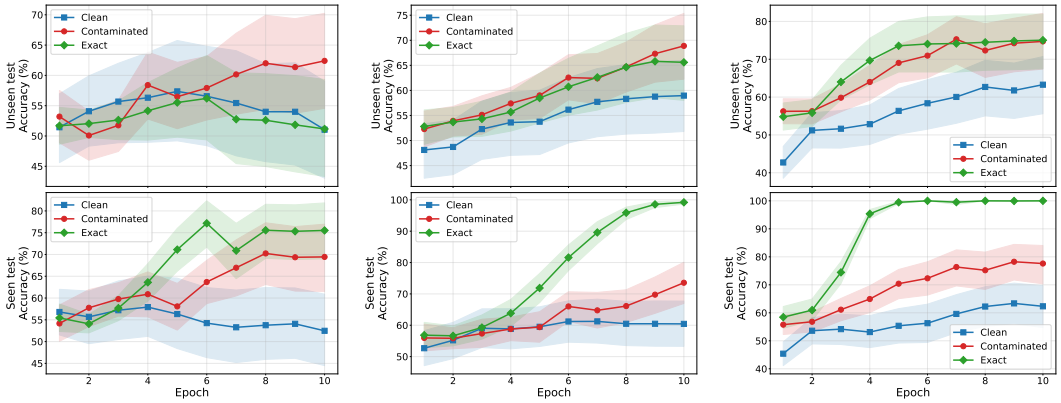
MBPP

Degree of similarity	Seen accuracy	Unseen accuracy	Acc. on HumanEval
No finetuning	–	46.4 ± 3.4	55.3 ± 3.9
Exact dupes	$\mathbf{63.0} \pm 3.3$	$\mathbf{48.8} \pm 3.5$	49.2 ± 3.9
Sem. dupes (Python)	55.1 ± 3.4	53.6 ± 3.4	$\mathbf{67.0} \pm 3.7$
cos similarity (SFT)	–	48.8 ± 3.5	53.1 ± 3.9

4.5 Ecologically Valid Finetuning

Realistic Contamination Dosage. We simulate SFT-stage exposure using our observed contamination rates. For MBPP, 40% of the top 0.1% cosine-similarity matches are true semantic duplicates. This suggests that about 4 in 10,000 training examples are semantic duplicates of any given MBPP test item; given our sampling density and the judge's false negatives, this is conservative. We apply this per-item rate to MuSR (which has no natural semantic duplicates in the Olmo 3 corpus).

Finetuning contaminated and clean models. We finetune Olmo-3-7B-base, Qwen3-8B-base [Yang et al., 2025] and Qwen3.5-9B-base on Olmo 3 SFT data mixed with synthetic MuSR semantic duplicates. The Olmo 3 post-training data contains no natural semantic duplicates of MuSR (Section 3.1), so the only MuSR duplicates in the finetuning set are the ones we add. We cannot inspect the Qwen models' pretraining data, but any prior exposure affects their clean and contaminated runs alike. We split MuSR into two halves of 125 items, *seen* and *unseen*, and generate four semantic duplicates of each seen item (two '1-branch' and two 'n-branch'), 500 in total. We run three finetuning conditions: 1) *clean*: 10,000 decontaminated SFT examples; 2) *contaminated*: the same set with 500 of the clean



(a) Olmo 3 (b) Qwen3 (c) Qwen3.5

Figure 4: **Ecologically valid finetuning on MuSR.** Accuracy on unseen items (top row) and seen items (bottom row) across finetuning epochs, under three conditions: no duplicates (clean, blue squares), exact duplicates (exact, green diamonds) and semantic duplicates (contaminated, red circles). For all three models, semantic duplicates significantly raise accuracy on unseen items relative to the clean run at epoch 10 ($p < 0.05$); for the Qwen models the difference is already significant at epoch 5. Table 14 gives means and p-values at epochs 5 and 10. Each run took about 5–8 hours on one GH200.

examples (5%) replaced by the 500 semantic duplicates; and 3) *exact*: the same set with 500 clean examples replaced by 500 exact duplicates (four copies of each seen item). Each seen item therefore has 4 duplicates per 10,000 training examples, matching our estimated natural rate; the aggregate rate of 5% arises because all 125 seen items are duplicated at once. We evaluate every model at each epoch, sampling 20 answers per question (Appendix B.1.1). Figure 4 shows the accuracy curves and Table 14 reports means and p-values.

Results on Olmo-3-7B. At epoch 10, the contaminated model scores 16.9pp higher than the clean model on seen items ($p = 0.001$) and 11.4pp higher on unseen items ($p = 0.037$). The effect appears late: at epoch 5 the two runs are indistinguishable on unseen items ($p = 0.85$), and part of the epoch-10 gap comes from the clean run's unseen accuracy falling from 57.4% to 51.0%. On unseen items, exact duplicates give essentially no gain over the clean run (51.2% vs. 51.0%), but the difference between semantic and exact contamination is not significant ($p = 0.084$). Accuracy on the TrueDetective control stays flat across conditions, though Olmo 3 only scores at near chance on it.

Results on Qwen3-8B and Qwen3.5-9B. On Qwen3-8B-base, at epoch 10 the contaminated model scores 13.2pp higher than the clean model on seen items ($p < 0.001$) and 9.9pp higher on unseen items ($p < 0.001$). On unseen items, semantic and exact contamination are statistically indistinguishable (68.9% vs. 65.6%, $p = 0.334$). Unlike with Olmo 3, the clean Qwen3 run also gains from finetuning relative to baseline (+5.0pp seen, +7.8pp unseen). The contaminated Qwen3 run also improves modestly on the TrueDetective control. On Qwen3.5-9B-base, the effect is largest early on: at epoch 5 the contaminated run already leads by 15.0pp on seen and 12.6pp on unseen items (both $p < 0.001$). At epoch 10 it leads by 15.3pp on seen items ($p < 0.001$) and 11.4pp on unseen items ($p = 0.009$). Again, semantic and exact contamination are statistically indistinguishable on unseen items (74.7% vs. 75.0%, $p = 0.754$). As with Qwen3, the clean run also shows gains relative to its baseline (+6.8pp seen, +8.2pp unseen), concentrated on the random unseen subset (Appendix D).

Natural contamination inflates held-out scores. In all three models, semantic duplicates at our estimated natural rate significantly raise accuracy on held-out MuSR items relative to clean finetuning, i.e. semantic duplicates produce within-benchmark generalization.

5 Conclusion

We found widespread soft contamination of Olmo 3 training data with semantic duplicates, despite strong efforts from the Olmo team to filter out duplicates (Section 4.3). We also found that finetuning

models on an ecologically valid amount of semantic duplicates leads to a statistically significant increase in accuracy on held-out items of the same benchmark (Section 4.5). In controlled experiments, semantic duplicates raise MuSR accuracy by up to 22pp while the control benchmark stays flat, the pattern we call shallow generalization; on MBPP, the gains also extend to HumanEval (Section 4.4). Scores on these benchmarks are therefore confounded by soft contamination.

5.1 Recommendations

At data collection time. The community could benefit from a small, openly-released model that scores whether two items are semantic duplicates, for use as a corpus filter in place of thresholds on embedding similarity. The classifiers we finetuned on our LLM-judge labels (Appendix A.7) are a first step. Authors should also report how many duplicates such filters find.

At evaluation time. Report a control benchmark from the same domain alongside each target benchmark, chosen so that the model scores well above chance on it. A gain on the target without a gain on the control suggests shallow generalization.

At benchmark creation time. As training corpora grow, static benchmarks become more likely to be duplicated. Benchmarks that are continuously refreshed or that sample items stochastically from a growing pool are harder to duplicate.

5.2 Limitations

We likely underestimate the prevalence of semantic duplicates in frontier training corpora: our limited budget did not allow us to embed the full pretraining corpus in our detection methods.

Another downward bias is the relative absence of rephrasings in Dolma: synthetic data pipelines now often involve intentionally creating semantic duplicates at scale [Wei and Zou, 2019, Wang et al., 2022, Maini et al., 2024], and so our estimates of their prevalence are likely lower than the true rate in closed corpora using our methods.

Similarly, we do not cover SOTA synthetic data provided via RL environments. Our corpus search is limited to models with an open training corpus, a small and potentially unrepresentative set of systems, and for the Qwen models in our finetuning experiments we cannot inspect prior exposure.

Our control benchmarks are imperfect (Section 3.2), so the scores of our control benchmark being flat is only weak evidence against transfer. Our controlled experiments use LoRA, and absolute effects may differ under full finetuning [Biderman et al., 2024].

Contamination dynamics in our SFT-stage simulation may also differ from those in the much larger pretraining runs at frontier labs. Frontier training corpora are much larger than Dolma; some Llama 4 runs used 30T tokens [Meta AI, 2025]. These corpora probably contain more semantic duplicates, and their rate of semantic duplication may differ from Dolma's.

A fundamental objection to our project is that out-of-distribution (OOD) generalization is no longer the (only) goal of AI development: an alternative is to instead bring all common tasks in-distribution (Chollet 2024, Patel and Sutton 2025, Leech et al. 2024 §5.3.2). Perhaps the real-world utility of LLMs shows that our concerns about OOD generalization are not practically important, since they work even if their generalization were largely shallow.

This is valid, but 1) then the deviation from the assumptions of empirical risk minimization should be explicitly noted, and 2) even perfect hidden interpolation may not be practically equivalent to OOD generalization [Hasson et al., 2020, Balestriero et al., 2021].

Contributions

Spiesberger: Experimental design and execution of corpus search, model diffusion, embeddings, and ecological finetuning experiments; managed compute infrastructure and code repository, statistical arguments and analysis. Assisted with writing and editing.

Vazquez: experimental design for generation and validation of semantic duplicates; comparison of duplicate detection methods; designed and executed annotation experiments; data spot-checking; wrote parts of methodology, results, and appendices; designed figures.

Pochinkov: generated MuSR teacher examples; finetuned and evaluated Olmo 3 models on MuSR and ZebraLogic; data sanity checks.

Gavenčiak: generated synthetic semantic duplicates; contributed to methodology, annotation pipeline design, finetuning experiments, and data analysis; provided feedback on the manuscript.

Grietzer: wrote the introduction and related work sections; edited the manuscript.

Leech: original research question; experiment design; wrote abstract, limitations, future work, and impact statement; funding.

Schoots: research and project management; led experiment design; coordinated writing efforts and wrote several parts of the manuscript.

Acknowledgments

We thank the Olmo team for making this work possible and for responding with grace to criticism. We thank Noah Perez for his writing help. We thank David Latshaw II, Max Shen, and Mary Putt for reading the paper and providing comments. We also thank Owain Evans, John Burden, Tom Davidson, Yuxi Liu, Teortaxes, and Martin Vlach for comments on an earlier draft. Finally, we thank Lennis Schoots uit de Bos and Jasper for emotional support.

Impact Statement

We aim to advance understanding of what LLM benchmark scores say about general capabilities on new inputs. Our findings have implications for how the AI research community, policymakers, and the public interpret reported progress on reasoning benchmarks.

More-accurately measuring AI capabilities helps to calibrate decisions about AI deployment, regulation, and research. If benchmark gains partly reflect interpolation from a growing corpus rather than more general capability improvements, then recognizing this could help prevent overconfidence in model generalization to novel tasks.

We do not believe this work poses significant risks. While our methods could inform more sophisticated benchmark gaming, the contamination we study is likely accidental rather than adversarial, and our detection methods are likely more useful for auditing than for evasion.

Our finding of contamination in Olmo3 is only possible because of the unusual level of transparency of its model development process. It would be unfair for readers to thereby assume that the degree of contamination in Olmo is unusual, and worse, a perverse incentive against transparency.

Our work could easily be misread as 'debunking' LLM capabilities and so spur complacency about near-term AI impacts. Our results suggest that benchmark gains are confounded (and so partially shallow), not that they are illusory.

References

J Achiam. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Anthropic. System card: Claude haiku 4.5. Model Card, October 2025a. URL https://www-cdn.anthropic.com/7aad69bf12627d42234e01ee7c36305dc2f6a970.pdf.

Anthropic. System card: Claude sonnet 4.5. Model Card, September 2025b. URL https://www-cdn.anthropic.com/963373e433e489a87a10c823c52a0a013e9172dd.pdf.

Anthropic. System card: Claude opus 4.5. https://www.anthropic.com/claude-opus-4-5-system-card, November 2025c. Accessed 2026-01-26.

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

Yauhen Babakhin, Radek Osmulski, Ronay Ak, Gabriel Moreira, Mengyao Xu, Benedikt Schifferer, Bo Liu, and Even Oldridge. Llama-embed-nemotron-8b: A universal text embedding model for multilingual and cross-lingual tasks. *arXiv preprint arXiv:2511.07025*, 2025.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Randall Balestriero, Jerome Pesenti, and Yann LeCun. Learning in high dimension always amounts to extrapolation. *arXiv preprint arXiv:2110.09485*, 2021.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, Cody Blakeney, and John Patrick Cunningham. LoRA learns less and forgets less. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=aloEru2qCG.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

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

François Chollet. The question of whether LLMs can reason is, in many ways, the wrong question [Tweet]. X (formerly Twitter), July 2024. URL https://x.com/fchollet/status/1816954290227089656. Accessed: 2026-01-26.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Jasper Dekoninck, Mark N Müller, and Martin Vechev. Constat: Performance-based contamination detection in large language models. *Advances in Neural Information Processing Systems*, 37: 92420–92464, 2024.

Maksym Del and Mark Fishel. True detective: A deep abductive reasoning benchmark undoable for gpt-3 and challenging for gpt-4. In *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (* SEM 2023)*, pages 314–322, 2023.

Ricardo Dominguez-Olmedo, Florian E Dorner, and Moritz Hardt. Training on the test task confounds evaluation and emergence. *arXiv preprint arXiv:2407.07890*, 2024.

Yihong Dong, Xue Jiang, Huanyu Liu, et al. Generalization or memorization: Data contamination and trustworthy evaluation for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 12039–12050, 2024.

Yafah Edelman and Jaeho Lee. Ai capabilities progress has sped up, 2025. URL `https://epoch.ai/data-insights/ai-capabilities-progress-has-sped-up`. Accessed: 2026-01-26.

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. URL `https://openreview.net/forum?id=RvfPnOkPV4`.

Epoch AI. Data on ai models, 07 2025. URL `https://epoch.ai/data/ai-models`. Accessed: 2026-01-26.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL `https://arxiv.org/abs/2101.00027`.

Google DeepMind. Gemini 3 flash model card. Model Card, December 2025. URL `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf`.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, 2022.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Uri Hasson, Samuel A Nastase, and Ariel Goldstein. Direct fit to nature: an evolutionary perspective on biological and artificial neural networks. *Neuron*, 105(3):416–434, 2020.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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

Minhao Jiang, Ken Ziyu Liu, Ming Zhong, Rylan Schaeffer, Siru Ouyang, Jiawei Han, and Sanmi Koyejo. Investigating data contamination for pre-training language models, 2024. URL https://arxiv.org/abs/2401.06059.

Subbarao Kambhampati. [tweet on LLMs as approximate retrievers]. X (formerly Twitter), December 2023. URL https://x.com/rao2z/status/1740692722099630237. Accessed: 2026-05-06.

Subbarao Kambhampati. Can large language models reason and plan? *Annals of the New York Academy of Sciences*, 1534(1):15–18, 2024.

Muhammed Yusuf Kocyigit, Eleftheria Briakou, Daniel Deutsch, Jiaming Luo, Colin Cherry, and Markus Freitag. Overestimation in LLM evaluation: A controlled large-scale study on data contamination's impact on machine translation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 31105–31132. PMLR, 2025. URL https://proceedings.mlr.press/v267/kocyigit25a.html.

Gavin Leech, Juan J. Vazquez, Niclas Kupper, Misha Yagudin, and Laurence Aitchison. Questionable practices in machine learning, 2024. URL https://arxiv.org/abs/2407.12220.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. StarCoder: may the source be with you!, 2023. URL https://arxiv.org/abs/2305.06161.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. Zebralogic: On the scaling limits of llms for logical reasoning. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=sTAJ9QyA6l.

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

Pratyush Maini, Skyler Seto, He Bai, David Grangier, Yizhe Zhang, and Navdeep Jaitly. Rephrasing the web: A recipe for compute and data-efficient language modeling, 2024. URL https://arxiv.org/abs/2401.16380.

Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, Toby Walsh, Armin Hamrah, Lapo Santarlasci, Julia Betts Lotufo, Alexandra Rome, Andrew Shi, and Sukrut Oak. Artificial intelligence index report 2025, 2025. URL https://arxiv.org/abs/2504.07139.

Meta AI. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/, April 2025. Accessed: 2026-01-26.

Vitaly Meursault. LLMs are approximate retrievers. https://pleasebringstrangethings.substack.com/p/llms-are-approximate-retrievers, June 2025. Substack post. Accessed: 2026-05-06.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, 2023.

Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024. URL

https://arxiv.org/abs/2303.08774.

Dwarkesh Patel and Richard Sutton. Richard sutton – father of RL thinks LLMs are a dead end. Dwarkesh Podcast, September 2025. URL https://www.dwarkesh.com/p/richard-sutton. Podcast interview.

Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces. https://huggingface.co/datasets/open-r1/codeforces, 2025.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

Martin Riddell, Ansong Ni, and Arman Cohan. Quantifying contamination in evaluating code generation capabilities of language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14116–14137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.761. URL https://aclanthology.org/2024.acl-long.761/.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. Detecting pretraining data from large language models, 2023. URL https://arxiv.org/abs/2310.16789.

Igor Shilov, Matthieu Meeus, and Yves-Alexandre de Montjoye. The mosaic memory of large language models, 2025. URL https://arxiv.org/abs/2405.15523. arXiv:2405.15523v2.

Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. Musr: Testing the limits of chain-of-thought with multistep soft reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=jenyYQzue1.

Vladimir Vapnik. Principles of risk minimization for learning theory. *Advances in neural information processing systems*, 4, 1991.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2022. URL https://arxiv.org/abs/2212.10560.

Maurice Weber, Daniel Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. Redpajama: an open dataset for training large language models, 2024. URL https://arxiv.org/abs/2411.12372.

Jason Wei and Kai Zou. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1670. URL https://aclanthology.org/D19-1670/.

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.

Cheng Xu, Nan Yan, Shuhao Guan, Yuke Mei, and Tahar Kechadi. SSA: Semantic contamination of LLM-driven fake news detection. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 14737–14751, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.744. URL https://aclanthology.org/2025.emnlp-main.744/.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, et al. Rethinking benchmark and contamination for language models with rephrased samples. *arXiv preprint arXiv:2311.04850*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Xin Zhou, Martin Weyssow, Ratnadira Widyasari, Ting Zhang, Junda He, Yunbo Lyu, Jianming Chang, Beiqi Zhang, Dan Huang, and David Lo. Lessleak-bench: A first investigation of data leakage in LLMs across 83 software engineering benchmarks, 2025. URL https://arxiv.org/abs/2502.06215.

A Detailed Methodology

A.1 Olmo3 Instruct Training Datasets

We embed 1% of the data Olmo3 Base was trained on ('Dolma3_6T-mix-1025'); and 1% of Dolmino-mix-1025 (100B tokens, 2.7 GB), a high quality dataset Olmo3-base is trained on. We excluded long-context data. We also embed the following three high quality datasets used to finetune Olmo3 Base into Olmo3 Instruct: Dolci3 SFT (2M input-output prompts, 3.08 GB), Dolci3 Instruct DPO (260k preference pairs, 811 MB), and Dolci3 Instruct RL (169k prompts, 483 MB).

Table 5: Datasets processed for contamination analysis. All text chunks are filtered to 50–2,048 tokens. Pretraining data is sampled at 1% using stratified reservoir sampling to preserve source distribution topology.

Dataset	Orig. Size	Sample	Sampling Method
Pretraining Data			
Dolma3_6T-mix	6.0 TB	1.0%	Stratified Reservoir (Hierarchical)
Dolmino-mix	250 GB	1.0%	Stratified Reservoir (Hierarchical)
Instruction Tuning Data			
Dolci3 SFT	3.08 GB	100.0%	Full Ingestion
Dolci3 DPO	811 MB	100.0%	Full Ingestion
Dolci3 RL	483 MB	100.0%	Full Ingestion

A.2 Exact duplicates of ZebraLogic test data in Olmo corpus

Olmo et al. [2025] reports a ZebraLogic benchmark score. Surprisingly, we found that the Olmo Instruct RL dataset contains exact duplicates of 70% of ZebraLogic problems-and-solutions.

In Figure 5 we see that for puzzles of 4×4 or larger, for 70% or more problems, there exists at least one exact duplicate in the Olmo3 Instruct RL dataset (`Dolci-Instruct-RL`). In total, we find corpus duplicates for at least 49.5% of dataset tasks.

The model card of Olmo3 benchmarks the performance of Olmo3 Instruct after SFT training as 18%, after DPO training as 28.4%, and after RL training as 32.9%. The model is improved in that order. We do not find exact or semantic duplicates of ZebraLogic in the DPO dataset, so we ascribe the improvement between SFT training and DPO training to general logical reasoning improvement. However, the increase from 28.4% to 32.9% (possibly on harder problems) is likely due to directly training on ZebraLogic data.

We also briefly investigated the existence of semantic duplicates of ZebraLogic, but did not find substantial amounts.

A.3 Cosine Similarity discussion

We embed both Olmo3 corpus data and benchmark data, and calculate the cosine similarity between data points in MBPP and CodeForces. This consisted of checking the 10 highest cosine similarity matches for each test point (adding up to 10,000 annotations for the entire benchmark).

When comparing with pretraining data sets, we split MBPP into inputs and outputs. When comparing MBPP with instruct data, we join MBPP inputs and outputs to match the prompt response format in the instruct data. We here present cosine similarity matches between corpus data and MuSR and ZebraLogic. We take the cosine similarity of the benchmark point embedding with the corpus text embeddings.

Figure 7 shows that semantic duplicate coverage of benchmark test samples increases with retrieval depth across MBPP and CodeForces. MBPP already reaches saturation at Top-1 with 90% coverage, while CodeForces is 28% at the same level, increasing up to 77.5% at Top-100.

Figure 8 shows that high cosine similarity matches are sparse. We also see in Figure 2 that a substantial portion of the very high (over 0.8) cosine similarity matches is a semantic duplicate.

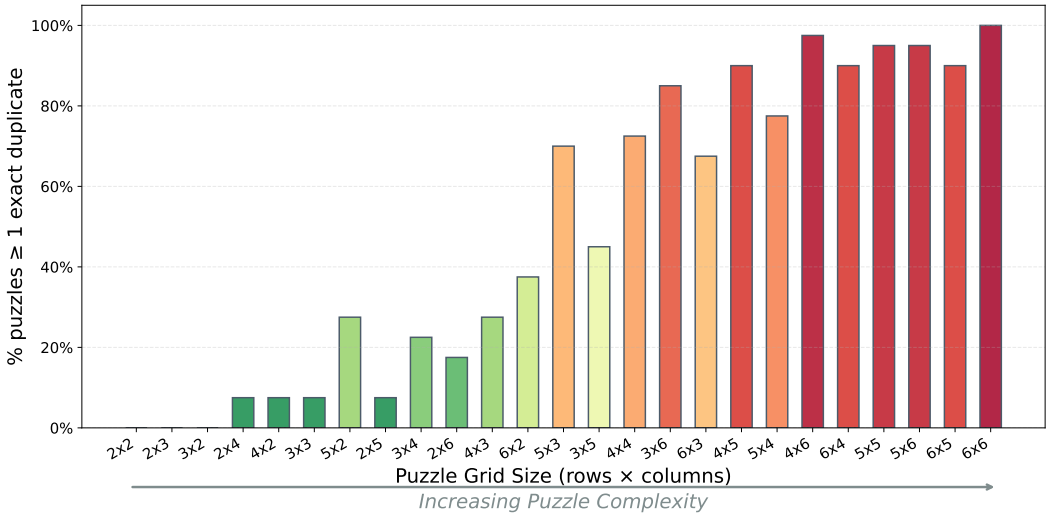


Figure 5: On the y-axis we plot the following statistic: for each ZebraLogic benchmark datapoint we check among the top 10 highest cosine similarity training datapoints if any of those samples is an exact duplicate, we then calculate the proportion of benchmark datapoints (of a given grid size) that have at least one exact duplicate. On the x-axis we plot puzzle grid size.

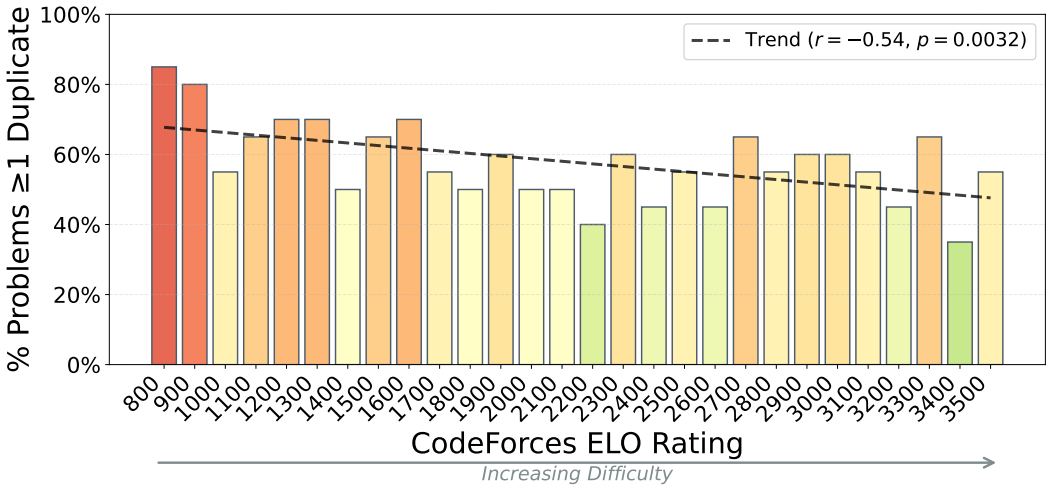
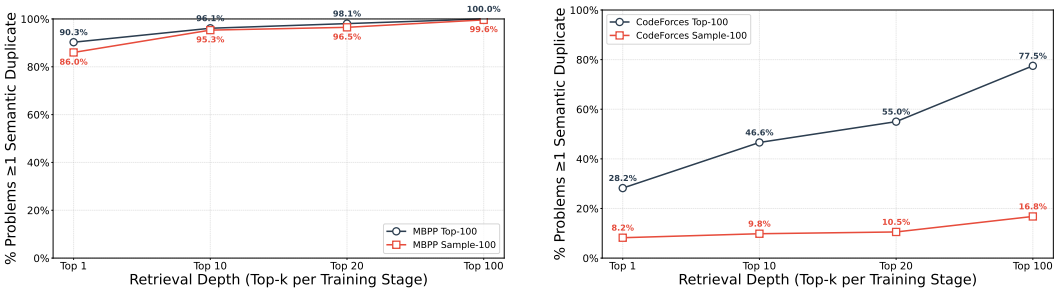


Figure 6: Occurrence of semantic duplicates by problem difficulty (elo). For each benchmark datapoint we check among the top 100 cosine similarity training datapoints if any of those samples is a semantic duplicate and calculate the proportion of all benchmark datapoints that have ≥ 1 semantic duplicate. We plot the CodeForces difficulty (elo score) on x.

A.4 Statistical Tests

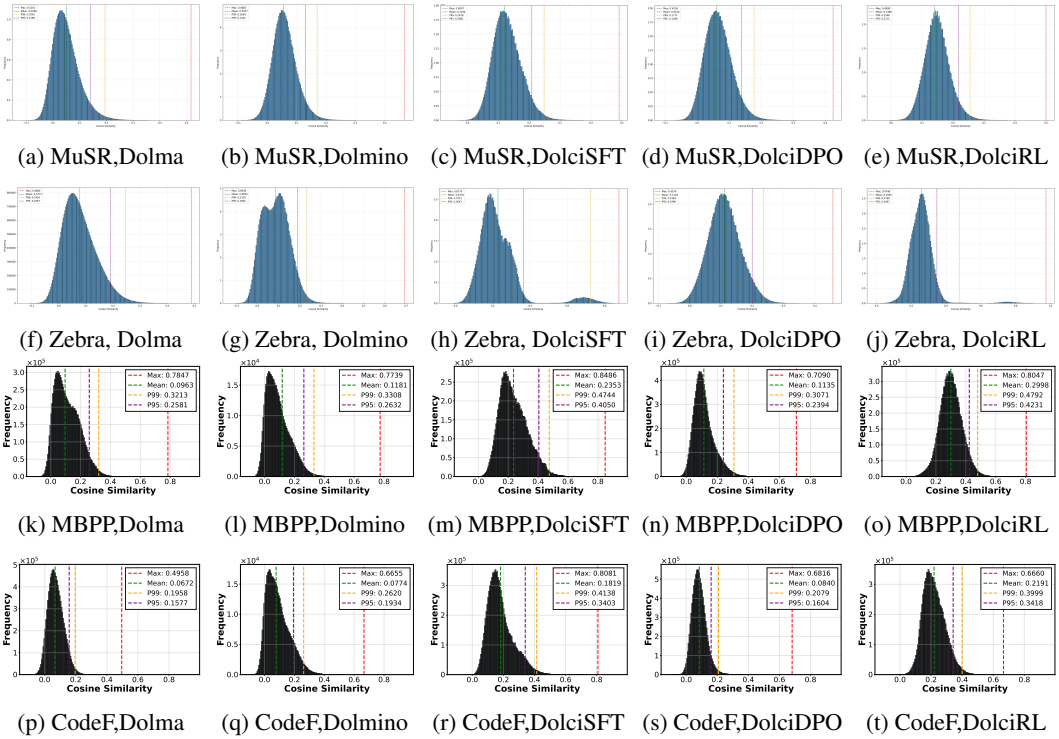
In reference to Table 4, we report standard errors on accuracy as $\text{SE}(\hat{p}) = \sqrt{\hat{p}(1-\hat{p})/n}$, the standard error of a binomial proportion. With a minimum evaluation size of $n = 125$, this estimate is well-behaved away from the boundaries $p \in \{0, 1\}$, which is the regime of all reported accuracies.

For Figure 4 and Table 14 we report Wilcoxon signed-rank test statistics and p-values. The test is paired by sample id: each evaluation problem appears under both the clean and contaminated-trained models, so pairing absorbs the substantial between-problem variance (some problems are near-uniformly easy or hard for every variant) and isolates the contamination effect. We use the rank-based



(a) MBPP (b) CodeForces

Figure 7: Occurrence by number of cosine similarity matches investigated. We take the number of semantic duplicates at top-n at each of our dataset comparisons.



(a) MuSR,Dolma (b) MuSR,Dolmino (c) MuSR,DolciSFT (d) MuSR,DolciDPO (e) MuSR,DolciRL

(f) Zebra, Dolma (g) Zebra, Dolmino (h) Zebra, DolciSFT (i) Zebra, DolciDPO (j) Zebra, DolciRL

(k) MBPP,Dolma (l) MBPP,Dolmino (m) MBPP,DolciSFT (n) MBPP,DolciDPO (o) MBPP,DolciRL

(p) CodeF,Dolma (q) CodeF,Dolmino (r) CodeF,DolciSFT (s) CodeF,DolciDPO (t) CodeF,DolciRL

Figure 8: Cosine similarity distributions between benchmark data and training corpus data. Rows from top to bottom: MuSR, ZebraLogic, MBPP, CodeForces. Columns from left to right: Dolma, Dolmino, Dolci SFT, Dolci DPO, and Dolci RL.

test rather than a paired t-test because the per-problem pass-rates are bounded and concentrate near 0% and 100%, so the distribution of paired differences is heavy-tailed and decidedly non-Gaussian; Wilcoxon makes no parametric assumption on the difference distribution and is robust to the resulting outliers. It is the simplest test under these conditions that yields valid inference.

A.5 Finetuning benchmarks

We select benchmarks based on: 1) ability to generate new synthetic samples from the same distribution, 2) tractability of creating semantic duplicates by modifying the original samples, 3) the benchmark's general prominence, and 4) benchmarks on which Olmo3 is not saturated, to enable us to track any performance improvement from finetuning. The following are the *only* benchmarks we tested:

MBPP – [Austin et al., 2021]. $n = 427$ programming questions with validated solutions in Python and test cases to check correctness. Since they are solvable by entry-level programmers, it is simple to generate correct alternative python solutions or translations in other programming languages. We expected that the training corpora would contain semantic duplicates of MBPP test data. Unlike for other benchmarks, *LLM-assisted* annotation to detect such duplicates may be feasible without requiring very complex reasoning, but would still be undetectable by typical deduplication methods. We use the sanitized test set which contains 257 tasks.

CodeForces – [Penedo et al., 2025]. Competitive programming tasks with larger text inputs and problem context than the average code benchmark, such as MBPP. We expected equivalent tasks with less context to appear in training corpora which would be difficult to identify without manual (or LLM) annotation. We use the 468 problems in the default test set in our experiments.

MuSR – [Sprague et al., 2024]. Long narratives generated from a ground-truth logic tree. These trees are provided for each original sample by the authors, which allows the creation of semantic duplicates by regenerating story context from the same tree, and slightly different narratives by modifying non-critical tree branches. We use 'murder mysteries', $n = 250$ problems.

ZebraLogic – [Lin et al., 2025]. $n = 1000$ logic grid puzzles of varying complexity levels used to evaluate logical reasoning capabilities. We chose ZebraLogic because Olmo 3 Instruct is not saturated on it (having 32.9% accuracy). Additionally, as it is a non-coding benchmark it increases the variety of our suite of benchmarks.

A.6 Synthetic Data Generation

A.6.1 Overview

MBPP

We generate 5 semantic duplicates for both inputs (questions), and outputs (responses). For the text inputs, we generate paraphrasings simultaneously to maximize difference between the texts. For code outputs, we generate alternative python implementations and validate them against provided test cases.

MuSR We adapt the original MuSR sample generation logic to generate new samples from existing reasoning trees of the public dataset problems. For the Murder Mysteries and Team Allocation tasks, we create three levels of semantic duplicates differing by how much the underlying logic trees differ from the original while maintaining the answer and the necessary reasoning to solve the problem the same: *regen*) tree is kept the same and story context is regenerated; 1-*branch*) One branch that does not affect problem outcome or its complexity is changed; n-*branches*) All branches of the previous type are changed.

For each level, we generate 2 semantic duplicates for each of the 250 original samples, adding up to 1500 semantic duplicates per task. We also generate a new test set of 250 samples using the original code.

ZebraLogic To generate semantic duplicates of ZebraLogic datapoints we use a pipeline of transformations, most of which are LLM-based.

We apply the following transformation methods: 1) use category mappings that e.g. substitute "color" by "shape" and "red" by "square", using Claude 4.5 Haiku [Anthropic, 2025a]; 2) shuffle conditions or clues in the prompt using a Python function; and 3) paraphrasing text while preserving all values exactly, using Claude 4.5 Sonnet Anthropic [2025b]. These strategies are applied to the first 500 samples in the benchmark, generating semantic duplicates for paraphrasing alone, and the following combinations of the above: 1) and 2); 2) and 3); and 1), 2) and 3).

A.6.2 Details of Finetuning on Duplicates

A.6.3 Choice of LoRA as the finetuning method

We employ LoRA for computational tractability, which introduces a low-rank bottleneck. A potential critique is that this bottleneck restricts the model from learning general reasoning, forcing it to rely on shallow templates. However, recent analyses demonstrate that LoRA at sufficient ranks can closely mirror the performance of full finetuning on reasoning tasks [Biderman et al., 2024].

Table 6: Finetuning hyperparameters for each benchmark. All experiments use LoRA Rank 16, Alpha 32, and Dropout 0.05.

Hyperparameter	MBPP	MuSR	ZebraLogic
Learning rate	$1.5e{-}4$	$2e{-}4$	$2e{-}4$
KL penalty	0.02	–	–
Epochs	10	6	10

Crucially, any capacity limits induced by LoRA apply uniformly across all of our experimental treatment conditions (clean, exact, semantic, and cosine-similar). Because our primary finding of shallow generalization is defined by the *comparative difference* in performance across seen, unseen, and out-of-distribution control splits within the same treatment condition, our comparative claims regarding data leakage and inflation remain structurally robust regardless of LoRA's absolute capacity limits.

A.6.4 MuSR

The generation pipeline uses the same few-shot examples and prompt templates as the original benchmark generation set up. The following model parameters are used with `gpt-4-0613`:

- Temperature: 1.0
- Top P: 1.0
- Max tokens: 2400

Specific details for each MuSR task:

Murder Mysteries In this task, the model needs to figure out which of the two suspects is the murderer based on a long narrative generated from a logic tree. We make the following modifications to the logic trees before the story generation step:

- *regen*: Story context regenerated from unchanged original tree.
- 1-*branch*: One suspicious fact branches changed.
- n-*branches*: All branches (suspicious, means, motive, opportunity) belonging to the innocent suspect changed.

Team Allocation In this task, the model needs to determine how to best allocate the 3 individuals mentioned in the long narrative to perform two tasks. We make the following modifications to the logic trees before the story generation step:

- *regen*: Story context regenerated from unchanged original tree.
- 1-*branch*: One randomly selected skill branch is changed while keeping skill level unchanged.
- n-*branches*: All branches swapped while skill and cooperation levels unchanged

Object Placement The data available for the original samples is not enough to generate semantic duplicates. While it is possible to attempt to extract most of the required data with LLMs to attempt sample regeneration, format and style is lost, affecting subsequent generation steps in the multi-stage process. So we choose to omit this MuSR category due to difficulty in creating semantic duplicates of similar complexity and rigor.

A.6.5 MBPP

We used `claude-opus-4-5-20251101` for all generation tasks with parameters:

- `max tokens`: 1024
- `temperature`: 1.0

We generate semantic duplicates for the full MBPP sanitized dataset, that is, 427 tasks. In the process, we find that the following tasks contain bugs (either in the function, or in the test cases): 229, 438, 461, 579, 769, 802.

Inputs We batch generate *paraphrasings* to ensure clear differences between duplicates.

Outputs We generate *Python semantic duplicates* for each sample sequentially, allowing the model to see the original and previously generated ones to increase the uniqueness of new solutions. We validate solutions at each step and ensure < 0.85 `difflib.SequenceMatcher` similarity score between solutions.

Prompts

Prompt for paraphrased question text duplicates:

```
PARAPHRASE_BATCH_PROMPT = """You are an expert at paraphrasing
    programming task descriptions.

ORIGINAL TASK:
{text}

YOUR TASK:
Generate exactly 5 DISTINCT paraphrases of this programming task. Each
     paraphrase must:
1. Have COMPLETELY DIFFERENT wording from the others
2. Preserve the EXACT same meaning and requirements
3. Maintain the same level of technical detail and clarity
4. Keep any mentioned function names UNCHANGED

CRITICAL: Each paraphrase must be noticeably different from the others
    . Vary:
- Sentence structure (active vs passive, questions vs statements)
- Vocabulary choices (synonyms, different technical terms)
- Order of information presented
- Level of formality

AVOID:
- Starting multiple paraphrases the same way (e.g., don't start 3 with
     "Write a...")
- Simply swapping one or two words while keeping structure identical
- Adding or removing requirements not in the original
- Changing the programming language if one is specified
- Making the task ambiguous or less precise

Output EXACTLY in this JSON format (no extra text, no markdown):
{{
  "para1": "first paraphrase here",
  "para2": "second paraphrase here",
  "para3": "third paraphrase here",
  "para4": "fourth paraphrase here",
  "para5": "fifth paraphrase here"
}}

Generate the 5 diverse paraphrases now:"""
```

Prompts for alternative Python solution implementations:

```
# Python semantic duplicate generation
def get_generation_prompt(
    task_description: str,
    original_code: str,
    test_list: list[str],
    previous_solutions: list[str],
    previous_error: Optional[str] = None,
    require_more_different: bool = False
) -> str:
```

```
    """Generate the prompt for creating a Python semantic duplicate.""
        "

    test_context = "\n".join(test_list[:3])
    func_name = extract_function_name(original_code)

    base_prompt = f"""You are an expert Python programmer. Your task
        is to write a DIFFERENT Python solution for the following
        problem.

TASK DESCRIPTION:
{task_description}

ORIGINAL PYTHON SOLUTION (for reference - DO NOT COPY):
‘‘‘python
{original_code}
‘‘‘

PYTHON TEST EXAMPLES (your solution must pass these):
‘‘‘python
{test_context}
‘‘‘

"""

    # Add previous solutions if any
    if previous_solutions:
        base_prompt += "PREVIOUS SOLUTIONS YOU'VE ALREADY WRITTEN (
            your new solution must be STRUCTURALLY DIFFERENT from ALL
            of these):\n"
        for i, sol in enumerate(previous_solutions, 1):
            base_prompt += f"\n--- Solution {i} ---\n‘‘‘python\n{sol}\
                n‘‘‘\n"
        base_prompt += "\n"

    # Add error feedback if retry
    if previous_error:
        base_prompt += f"""YOUR PREVIOUS ATTEMPT HAD ERRORS:
{previous_error}

Please fix these errors in your new solution.

"""

    # Add stronger differentiation request if needed
    if require_more_different:
        base_prompt += """IMPORTANT: Your previous solution was TOO
            SIMILAR to existing ones!
You MUST use a significantly DIFFERENT algorithmic approach. Consider:
- Using different data structures (list vs set vs dict vs deque)
- Using different iteration patterns (for vs while vs recursion vs
    comprehensions)
- Using different built-in functions or libraries
- Restructuring the logic flow completely

"""

    base_prompt += f"""REQUIREMENTS:
1. Write a COMPLETE Python solution that passes all tests
2. The function MUST be named EXACTLY: {func_name}
3. Use a DIFFERENT algorithmic approach or implementation style than
    the solutions shown above
4. ADD A COMMENT ON THE LINE ABOVE EACH LINE OF CODE explaining what
    it does
5. Comments should be ORIGINAL, CONCISE, and INSIGHTFUL - not generic
```

````
6. Make sure every substantive line has a comment above it
7. The comments should help distinguish this solution semantically
    from others

COMMENT STYLE EXAMPLE:
```python
# Initialize counter for tracking element frequency
count = 0
# Iterate through each item in the input sequence
for item in items:
    # Increment counter when condition is met
    if condition:
        count += 1
# Return the final accumulated count
return count
```

OUTPUT ONLY THE PYTHON CODE with comments. No markdown code blocks, no
     explanations outside the code."""

    return base_prompt
````

A.6.6 ZebraLogic

Below we discuss the different methods and transformations:

Paraphrasing. With `claude-4.5-sonnet` we use the following prompts using the original sample

```
# SYSTEM PROMPT
You are an expert editor tasked with rewriting logic grid puzzles
    while exactly preserving the logical structure and semantics.

# USER PROMPT
Rewrite the following logic puzzle to express the exact same
    conditions in different words or with different word order etc.
    while exactly preserving the logical structure and semantics.

Original Puzzle:
{puzzle}

REQUIREMENTS:
1. Reformulate both the task description and every numbered condition.
2. You may change word order, use synonyms, and alter sentence
    structure.
3. PRESERVE the strict logical meaning. For example, "A is next to B"
    must remain logically equivalent (e.g., "B is adjacent to A").
4. PRESERVE all entity names, values, numbers, and categories EXACTLY.
     Do not change "Red" to "Crimson" or "John" to "Jon". The specific
     terms used for the puzzle items MUST remain identical to match
    the solution exactly.
5. The output must be natural, clear, and readable. Avoid contrived or
     unnatural constructions.
6. Maintain the formatting of the puzzle, including the format and
    numbering of the list of clues.
7. Do not start your response with a header or a preamble. Start with
    a naturally flowing puzzle statement in a very similar style and
    format as the original.

Output ONLY the rewritten puzzle text.
```

Category substitution. With `claude-4.5-haiku` we follow a two step process: (1) generate a substitution plan for each puzzle; (2) apply the substitution with LLMs for improved text cohesiveness; (3) transform the solution programmatically.

```
# STEP 1: SYSTEM PROMPT
You are a helpful assistant that creates substitution plans for logic
    puzzles.
Your goal is to transform the puzzle by changing BOTH the categories
    and their values to new domains.

# STEP 1: USER PROMPT
Create a substitution plan to transform this logic grid puzzle.
1. Identify all categories (e.g., Color, Drink, Pet).
2. Assign a NEW category to each (e.g., Color -> Shape, Drink -> Snack
    , Pet -> Book).
3. Map every existing value to a new value appropriate for the new
    category.

Original Puzzle:
{puzzle}

Original Solution:
{solution_json}

REQUIREMENTS:
1. Change the categories to natural, distinct alternatives (e.g.,
    colors -> shapes, flowers -> animals).
2. Keep the new categories and values DISTINCT from all of the
    original ones. Avoid number categories (to avoid confusion with
    the numbering of the puzzle).
3. Ensure 1-to-1 mapping for all values.
4. Do NOT use obscure or unusual categories. Stick to common
    categories like colors, animals, shapes, countries, etc. Choose
    natural categories and values within the flow of the puzzle
    wording.

Output ONLY a JSON object with this structure:
{
  "substitution_plan": {
    "OriginalCategoryName": {
      "new_category": "NewCategoryName",
      "values": {
        "OldValue1": "NewValue1",
        "OldValue2": "NewValue2"
      }
    },
    ...
  }
}

# STEP 2: SYSTEM PROMPT
You are a helpful assistant that rewrites logic puzzles based on a
    substitution plan.
You must replace categories and values exactly according to the plan
    while PRESERVING the puzzle structure, logic, and clues EXACTLY.

# STEP 2: USER PROMPT
Rewrite this logic puzzle by applying the following substitution plan.
Replace ALL occurrences of the old categories and values with their
    corresponding new ones.

Substitution Plan:
{plan_json}

Original Puzzle:
{puzzle}

CRITICAL INSTRUCTIONS:
```

```
1. Replace old categories (e.g., "Color") with new categories (e.g., "
   Shape").
2. Replace old values (e.g., "Red") with new values (e.g., "Square").
3. Do NOT change the logic, clues, or structure.
4. Keep the puzzle wording identical as much as possible, only make
   minor syntactic adjustments where necessary to preserve the flow
   and meaning of the puzzle wording.
5. Keep the numbering and formatting identical.
6. Output ONLY the rewritten puzzle text.
```

Shuffling. Puzzles clues are parsed, randomly reordered, and renumbered sequentially. The solution of the resulting semantically equivalent puzzle remains the same.

Composite transformations. With respect to the order of transformations in composite methods, shuffling is always performed first, and paraphrasing is performed last.

A.6.7 Similarity distributions of synthetic semantic duplicates

We compare the generated semantic duplicates against the original samples for MBPP, MuSR and CodeForces, and show the analysis for Cosine similarity and several common metrics used in deduplication: n-gram overlap (2 and 3 grams), ROUGE-L F, and Jaccard token. See Figure 9.

Finding that in most cases, cosine similarity is better at separating semantic duplicate pairs than the other metrics. This motivates, in part, the use of embedding similarity for the other experiments.

A.7 Annotation schemes for high cosine similarity matches

We use `gemini-3-flash-preview` with the following parameters:

- `thinking_level`: `MEDIUM` (we use `HIGH` for CodeForces due to the length of the problems.)
- `temperature`: 1.0
- `max_output_tokens`: 8192
- `response_format`: JSON, with structured output conforming to the schema below.

For each annotated pair, we use a schema to collect the following:

- `is_sd`: A boolean indicating whether the pair constitutes a semantic duplicate (true if the corpus task is the same as or subsumes the test task.)
- `confidence`: A confidence score in $[0, 1]$, where 1.0 indicates certainty and 0.0 represents a 50-50 guess.
- `reasoning`: A free-text explanation of the judgment, including key similarities and differences observed.
- `match_type`: A categorical label describing the relationship: *exact* (nearly identical), *equivalent* (same task, different wording), *subset* (test is a subset of corpus), *superset* (corpus is a subset of test), or *unrelated* (different tasks.)

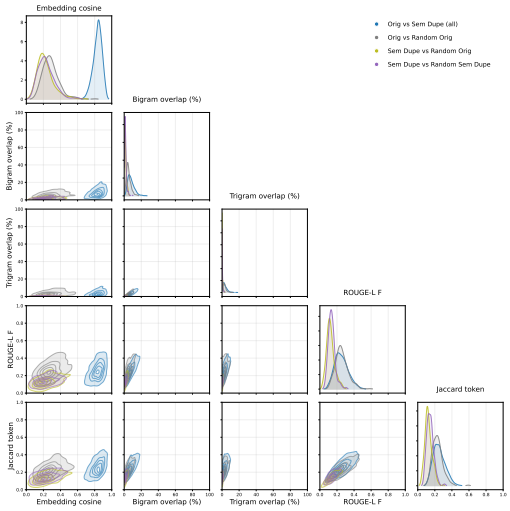
The annotations are then used to finetune `gpt-oss-20b` [Agarwal et al., 2025] as a classifier for CodeForces semantic duplicate detection, and `Qwen/Qwen3-30B-A3B-Instruct-2507` for MBPP.

A.7.1 Classifier finetuning results

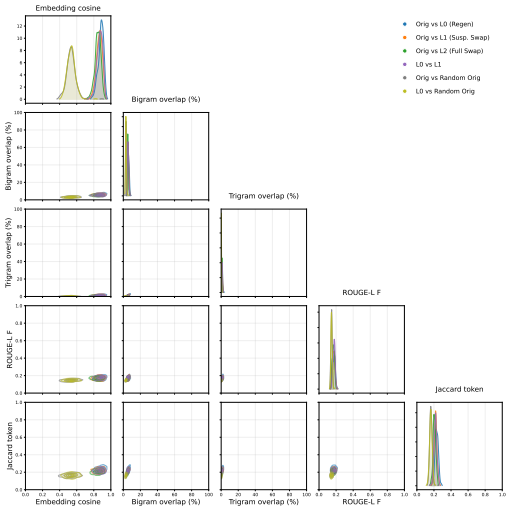
MBPP. We finetune `Qwen/Qwen3-30B-A3B-Instruct-2507` with LoRA (rank 32) on 13266 samples (9% semantic duplicates), and achieve 96.8% accuracy on a held-out validation set (n=560, 25% semantic duplicates) with 97.7% precision and 93.3% F1 on the semantic duplicate class.

CodeForces. We also finetune `gpt-oss-20b` with LoRA (rank 32) on 13398 samples (9% semantic duplicates), and achieve 95.3% accuracy on the held-out validation set (n=512, 25% semantic duplicates) with 97% precision and 90% F1 on the semantic duplicate class.

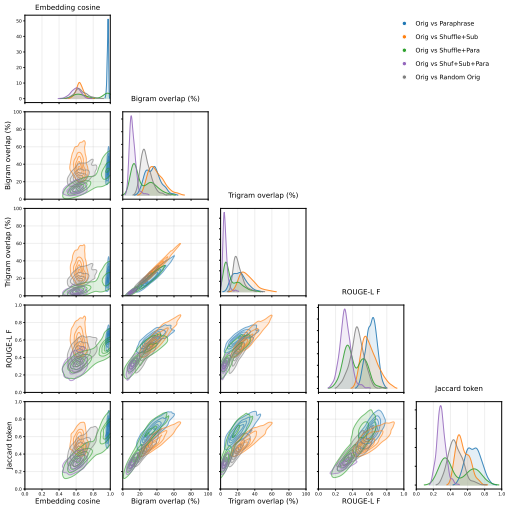
In Table 7 we show all details of the finetuning set up for the two classifiers, and the final metrics that validate their performance.



(a) MBPP sanitized test set ($n = 257$).



(b) MuSR Murder Mystery task split ($n = 250$).



(c) ZebraLogic dataset ($n = 1000$).

Figure 9: Similarity distributions using several deduplication metrics for each benchmark.

A.7.2 Annotation prompt for MBPP matches

```
MBPP_PROMPT_TEMPLATE = """You are an expert programmer analyzing
    potential semantic duplicates between coding tasks.

## Task
Determine if the following two coding tasks are semantic duplicates -
    meaning they describe the same programming task, just potentially
    phrased differently.

## Test Task (from benchmark):
{test_text}

## Corpus Task (from training data):
{corpus_text}

## Guidelines:
1. **Focus on the TASK, not the solution** - ignore any code or
    solutions that may be present
```

Table 7: Summary of classifier performance

Feature	Codeforces (GPT-OSS-20B)	MBPP (Qwen3-30B-A3B-Instruct)
Training Configuration		
Base Model	openai/gpt-oss-20b	Qwen3-30B-A3B-Instruct
LoRA Rank	32	32
Epochs	10	5
Batch Size	128	128
Learning Rate	0.0001	0.0005
Max Seq Length	4096	4096
Dataset Configuration		
Train Size	13,398	13,266
Train Duplicates	1,218	1,206
Eval Size	512	560
Eval Duplicates	128	140
Final Metrics (Eval Set)		
Accuracy	95.31%	96.79%
Balanced Accuracy	91.41%	94.29%
Macro F1	0.9343	0.9559
Duplicate Precision	97.27%	97.66%
Duplicate Recall	83.59%	89.29%
Duplicate F1	0.8992	0.9328
Non-Dup Precision	94.78%	96.53%
Non-Dup Recall	99.22%	99.29%
TP / FN / FP / TN	107 / 21 / 3 / 381	125 / 15 / 3 / 417

```
2. **Mathematical equivalence counts as duplicate** - e.g., "sum 1 to
    n" and "sum n, n-1, ..., 1" are equivalent
3. **Corpus subsumes test = duplicate** - if the corpus task is
    strictly harder (asks for more), but solving it would trivially
    solve the test task, mark as duplicate
4. **Be calibrated** - use confidence primarily for ambiguous cases,
    tricky phrasing, or when you're uncertain

## Match Types:
- "exact": Nearly identical wording
- "equivalent": Different phrasing, same underlying task
- "subset": Test task is a subset of corpus task (corpus is harder but
     solves test)
- "superset": Corpus task is a subset of test task (test is harder) -
    NOT a duplicate
- "unrelated": Different tasks entirely

Analyze the tasks and provide your structured judgment."""
```

A.7.3 Annotation prompt for CodeForces matches

```
CODEFORCES_PROMPT_TEMPLATE = """You are an expert competitive
    programmer analyzing potential semantic duplicates between
    programming problems.

## Task
Determine if the following two competitive programming problems are
    semantically related - meaning exposure to the corpus problem
    during training could help solve the test problem.
```

```
## Test Problem (from benchmark):
{test_text}

## Corpus Problem (from training data):
{corpus_text}

## Analysis Steps:
1. **Check data quality first**: Is the corpus text a complete problem
     statement? If it's empty, fragmentary, or contains only code
    without a problem description, mark as "unrelated".
2. **Check for exact text match**: If the corpus text appears VERBATIM
     (word-for-word) within the test text (e.g., corpus contains just
    the problem statement while test contains problem + examples),
    this counts as "exact" match.
3. **Extract the core problem**: Strip away story/narrative framing.
    What is the actual computational task?
4. **Identify the key insight**: What algorithmic technique or
    observation is needed?
5. **Compare**: Is there meaningful overlap in what's being asked or
    how to solve it?

## Match Types:
- "exact": Nearly identical problem statements, OR corpus text is a
    verbatim substring/subsection of test text (exact text match even
    if corpus is shorter)
- "equivalent": Different framing but identical algorithmic core
- "subset": Test is a special case of corpus (test asks for less than
    corpus)
- "superset": Corpus asks for something simpler than test, but NOT a
    verbatim text match
- "related": Corpus covers a component or shares key insight with test
- "unrelated": Different problems, or corpus data is unusable

## IMPORTANT: Exact Match Clarification
If the corpus text is an exact substring of the test text (the corpus
    text appears word-for-word inside the test text, just without some
     sections like examples or input/output format), mark this as "
    exact" NOT "superset". The key distinction:
- "exact": Corpus text IS CONTAINED VERBATIM in test text
- "superset": Corpus asks a DIFFERENT (simpler) question than test

## What counts as semantically related:
- Same computational task (any framing)
- One is a special case of the other
- Shared key insight or trick
- Corpus solves a significant component of test

## What is unrelated:
- Sharing only common techniques (DP, BFS) without structural
    similarity
- Unusable corpus data (empty, fragmentary, code-only)
- Genuinely different computational questions"""
```

A.7.4 Final grader prompt for selected matches

```
GRADER_PROMPT_TEMPLATE = """You are an exceptionally talented
    programmer and mathematician with the goal to determine whether
    there are semantic duplicates between two chunks of texts.

You will be provided with a benchmark problem and a corpus text
    problem, as well as a reason that they may or may not be semantic
    duplicates. The rationale may be wrong.
```

Table 8: Self-consistency of the Gemini grader on 23 CodeForces problems per category, run 10 times each. Fleiss's $\kappa = 0.78$ indicates substantial agreement. Equivalent and Exact are frequently confused, but both count as semantic duplicates downstream, so the conflation does not affect our prevalence estimate. The net bias is conservative: 30.4% of true Subsets leak to Superset, Related, or Unrelated (missed hits), while only 4.3% of true Supersets leak to Equivalent (false hits). Since true Subsets dominate true Supersets in practice, the matrix biases our reported prevalence downward.

	Predicted					
H. Annotation	EQUIV.	EXACT	SUBSET	SUPERSET	RELATED	UNRELATED
Equivalent	**39.1**	60.9	–	–	–	–
Exact	13.0	**87.0**	–	–	–	–
Subset	13.0	4.3	**52.2**	8.7	17.4	4.3
Superset	4.3	–	–	**52.2**	39.1	4.3
Related	–	–	–	–	**91.3**	8.7
Unrelated	–	–	–	–	–	**100.0**

```
Please consider how both problems are solved. Your goal here is to
    determine whether or not it is the case that the corpus problem
    provides significant algorithmic insight required to solve a major
     part of the test problem, thus making it a semantic duplicate.

Respond with TRUE if it is a semantic duplicate.
Provide your reason, in detail, as to why it is or is not.
Provide a score on the amount of duplication from 0 to 1.

## Test (benchmark) problem:
{test_text}

## Corpus problem:
{corpus_text}

## Prior reasoning (may be wrong):
{prior_reasoning}"""
```

A.8 Self-consistency results for the annotation system against human validations

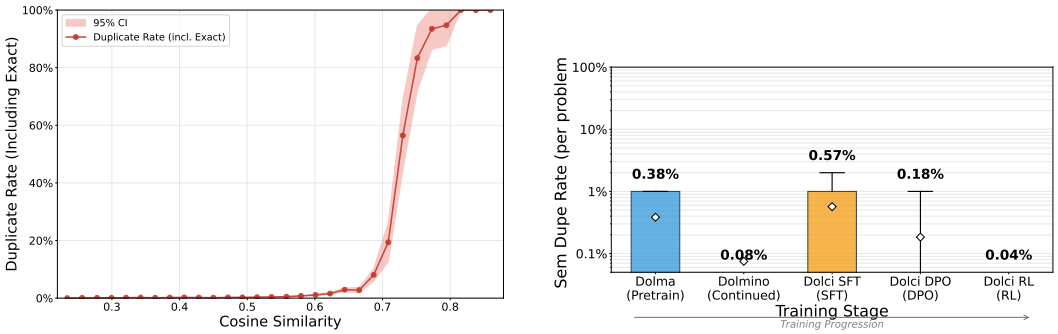
In Tables 8 and 9, we perform a self-consistency analysis on selected annotations which we manually annotate across 5 categories with respect to semantic duplicates:

- **equivalent:** the corpus sample states the same task as the test sample in different words.
- **exact:** the corpus sample is a near-verbatim restatement of the test problem (solutions may differ). This is distinct from the exact duplicates used for finetuning in Section 3.2.
- **subset:** the test sample is a special case or component of the corpus sample, so solving the corpus problem solves the test problem. For MBPP, the corpus program contains a working version of the test program as part of its implementation.
- **superset:** the corpus sample solves only part of the test sample, so it does not supply a full solution.
- **related** (CodeForces only): the samples share a technique or insight, but the corpus does not cover the test problem.
- **unrelated:** no algorithmic overlap; a false match.

Table 9: Self-consistency of the Gemini grader on MBPP, with 20 duplicates per category labeled 10 times each (Exact has only 1 sample, as it is the only Exact duplicate in the corpus). Fleiss's $\kappa = 0.904$ indicates almost perfect agreement. As with CodeForces, Equivalent and Exact are confounded but both count as semantic duplicates downstream, so the conflation does not affect our prevalence estimate. Subset is occasionally dropped to Unrelated (16.5%) and Superset to Unrelated (23.5%), indicating that the matrix biases our reported prevalence downward.

	Predicted				
H. Annotation	EQUIV.	EXACT	SUBSET	SUPERSET	UNRELATED
Equivalent	**96.0**	0.0	0.5	3.5	0.0
Exact	99.5	**0.5**	0.0	0.0	0.0
Subset	0.0	0.0	**83.5**	0.0	16.5
Superset	0.0	0.0	5.5	**71.0**	23.5
Unrelated	0.0	0.0	4.5	0.5	**95.0**

B Further Semantic Duplicates in the Wild Results



(a) Correlation of Similarity vs. Duplicates (including semantic and exact)

(b) Propensity of semantic duplicates by Training Scheme (excluding exact duplicates)

Figure 10: **CodeForces duplicates using the top 100 cosine-similarity matches per problem and training stage** (a) Rate of semantic or exact duplicates as a function of cosine similarity, with 95% CIs. (b) Mean semantic duplicate rate per problem by training stage, excluding exact duplicates, log scale (Figure 3b reports the share of problems with at least one duplicate).

B.1 Semantic duplicates are hard to detect

Semantic duplicates in the wild are sparse and difficult to find. From the above we notice that semantic duplicates can be found, even for hard CodeForces level problems. We consider, per test point, approximately 350 million texts. We find in the case of CodeForces on average a few semantic duplicates. For MBPP there are tens to one hundred across our entire dataset. We run our algorithm across around 250GB of data to find necessary duplicates for a representative population.

To illustrate this point we demonstrate the probability of a semantic duplicate occurring given a cosine similarity and being in the top 0.1% of a semantic duplicates test set.

The relationship between embedding cosine similarity and the probability of semantic duplication, aggregated across all training stages (N=128,408 training-test pairs). Points represent binned similarity scores (30 bins); shaded regions indicate 95% confidence intervals computed using the normal approximation to the binomial distribution. Sample sizes for each bin are annotated.

Semantic duplicate rate exhibits a nearly monotonically increasing relationship with cosine similarity as we would expect. Below a similarity threshold of approximately 0.35, the duplicate rate is effectively zero (<1%). The rate increases sharply between 0.4 and 0.7, following an approximately sigmoidal trajectory, and reaches 60–85% at the highest observed similarities (>0.8). The widening confidence intervals at high similarity values and volatility reflect reduced sample sizes in these

bins. This calibration curve suggests that cosine similarity serves as a useful but imperfect proxy for semantic duplication, with a practical decision threshold in the 0.5–0.6 range capturing the inflection point of the relationship.

B.1.1 Ecologically valid finetuning experiment

Finetuning Hyperparameters Hyperparameters mirror the official Olmo 3 hybrid SFT recipe so that the training trajectory is representative of contemporary post-training rather than a setting tuned to amplify contamination:

Table 10: Finetuning Hyperparameters

Hyperparameter	Value
Optimizer	`adamw_torch_fused`
Betas	$\beta_1 = 0.9$, $\beta_2 = 0.95$
Weight Decay	0
Learning Rate Olmo 3	2.5e-5
Learning Rate Schedule	Linear decay, 0.03 warmup ratio
Max Grad Norm	1.0
Precision	`bf16` parameters, `fp32` reductions
Effective Batch Size	64 sequences (per-device 24 $\times$ grad accum $\times$ #GPUs)
Max Sequence Length	2048
Checkpoints	1 per epoch (10 / model / corpus)
Loss Masking	Assistant response only (`completion_only_loss=True`)

Loss is masked to the assistant response (`completion_only_loss=True`)—prompt tokens contribute no gradient, matching the open-instruct SFT convention. All runs use the closest reproduction to the Olmo hyperparameter set for SFT, as they report.

B.1.2 Reporting on top 100 cosine similarity matches instead of sampling from the top 0.1%

Complimenting Figure 3b, see Figure 10 for the top 100 cosine similarity matches for CodeForces instead of 100 sampled from top 0.1%.

C Further Finetuning Results, Including Degradation Analysis

The Opus 4.5 MuSR baseline accuracy is 91.6%. The model does slightly worse on the first half of the data (that we train on in Table 11) than on the second half, respectively Opus 4.5 gets 90.4% on the first half and 92.8% on the second half.

The performance of `gpt-4.1-mini-2025-04-14` on MuSR benchmark data is 84.0, on our level0 semantic duplicates it is 79.8 and on level2 it is 76.4. This is just the baseline performance of the teacher model GPT 4.1 mini without any finetuning.

Tables 12 and 13 show that our finetuned models do not degrade in performance across 6 other benchmarks.

Table 11: Effect of finetuning Olmo 3 on semantic duplicates of MuSR Murder Mysteries reasoning traces. Olmo 3 was finetuned for 3 epochs.

DUPLICATION LEVEL	TEACHER: OPUS 4.5	TEACHER: GPT 4.1 MINI
BASELINE	66.0	66.0
EXACT DUPES	87.1	82.5
REGEN	86.8	82.4
1-BRANCH	86.1	81.3
n-BRANCHES	85.8	81.6

Table 12: No degradation effect of finetuning Olmo 3 on semantic duplicates of MuSR Murder Mysteries reasoning traces. Olmo 3 was finetuned for 3 epochs.

DUPLICATION LEVEL	ARC CHALLENGE	ARC EASY	BOOLQ	HELLASWAG	PIQA	WINOGRANDE
BASELINE	50.1	78.2	75.7	57.5	75.0	65.3
EXACT DUPES	50.0	78.7	76.7	56.4	75.8	65.0
REGEN	50.3	78.6	76.0	56.4	75.7	64.8
1-BRANCH	50.3	78.6	77.0	56.4	75.6	64.7
n-BRANCHES	50.6	78.9	77.1	56.5	75.6	65.1

Table 13: No degradation effect of finetuning Olmo 3 on semantic duplicates of ZebraLogic reasoning traces. Olmo 3 was finetuned for 3 epochs.

DUPLICATION LEVEL	ARC CHALLENGE	ARC EASY	BOOLQ	HELLASWAG	PIQA	WINOGRANDE
BASELINE	50.1	78.2	75.7	57.5	75.0	65.3
EXACT DUPES	49.5	78.0	77.0	56.8	75.0	64.7
PARA	49.3	77.7	78.3	56.4	74.5	64.1
SHUFFLE, SUBS	50.7	77.4	75.3	56.5	75.1	64.5
SHUFFLE, PARA	49.4	78.1	77.8	56.5	75.3	64.9
SHUFFLE, SUBS, PARA	50.4	78.1	78.4	56.5	75.6	63.5

D Ecologically Finetuned Results

Table 14 complements our results in Figure 4, showing p values.

Table 14: Results from replicating our finetuning experiments on more models. Means are reported for each finetuning treatment on the SEEN split (items the contaminated arm was trained on) and the UNSEEN held-out clean split. Seen corresponds to contaminated test, and unseen to clean test. p-values are from paired Wilcoxon signed-rank tests (asymptotic, with tie correction) on per-item mean accuracies (averaged over 20 evaluations) per question, comparing against CONTAMINATED. $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

			Accuracy (%)			p vs. CONTAM.	
Model	**E**	**Split**	CLEAN	CONTAM.	EXACT	CLEAN	EXACT
Olmo-3-7B	Baseline	Seen		49.9		–	–
		Unseen		49.8		–	–
	5	Seen	56.3	58.1	71.1	0.689	$<0.001^{***}$
		Unseen	57.4	56.5	55.5	0.852	0.652
	10	Seen	52.5	69.4	75.5	0.001^{**}	0.256
		Unseen	51.0	62.4	51.2	0.037^{*}	0.084
Qwen3-8B	Baseline	Seen		55.4		–	–
		Unseen		51.2		–	–
	5	Seen	59.5	59.4	71.9	0.951	$<0.001^{***}$
		Unseen	53.8	59.0	58.5	0.017^{*}	0.803
	10	Seen	60.4	73.6	99.2	$<0.001^{***}$	$<0.001^{***}$
		Unseen	59.0	68.9	65.6	$<0.001^{***}$	0.334
Qwen3.5-9B	Baseline	Seen		55.5		–	–
		Unseen		55.1		–	–
	5	Seen	55.4	70.4	99.5	$<0.001^{***}$	$<0.001^{***}$
		Unseen	56.4	69.0	73.6	$<0.001^{***}$	0.007^{**}
	10	Seen	62.3	77.6	100.0	$<0.001^{***}$	$<0.001^{***}$
		Unseen	63.3	74.7	75.0	0.009^{**}	0.754

E Decontamination

E.1 Decontamination methods

Table 15 shows the decontamination methods used in model releases from Anthropic, OpenAI, Meta, AllenAI, Gemini, Deepseek and Qwen. Newer releases either do not mention decontamination procedures or refer to the older releases for such details. Where a method has been used (✓) we add the section where it is mentioned, for further verifiability. It is clear that the standard is n-grams and a mix of exact and subset string matching for decontamination methods.

E.2 Types of semantic duplicate contamination

Here we attempt to categorise the semantic duplicates we find in the wild for MBPP and CodeForces (coding test items). We classify 2,350 pairs: all 1,150 confirmed CodeForces pairs, and 1,200 MBPP pairs sampled at a cap of 5 per benchmark task. Because duplicate status was already established by the pipeline, the classifier (a strong LLM, forced single-choice over a fixed, mutually exclusive taxonomy) adjudicates only the form of duplication, not whether the pair is a duplicate.

We state the definition for each of the seven forms, ordered by increasing semantic distance between the two samples, from a standard paraphrase to different problem statements with identical algorithm:

- **Verbatim restatement:** the corpus text reproduces the benchmark item essentially word-for-word, but not enough so that standard deduplication procedures would catch it. Many papers examine the effects of these, including ours.
- **Same-language paraphrase:** the same problem reworded in the same natural language (and, for code, the same programming language). This was previously well studied in the forms of semantic duplication.

Table 15: **Disclosed decontamination procedures in SOTA model releases.** Every single named method is lexical; reported n-gram sizes range from 10 to 20 with no consensus; no surveyed release reports MinHash at the decontamination stage; the sole semantic mention (Gemini 2.5) names no method or implementation details. Notes: * = remaining part of the pipeline is gated on the initial n-grams getting hits, N/A = the paper does not include said details.

Model	Where?	Exact	Substring	n-gram	MinHash	Semantic
Claude Opus 4.5	§2.2 Anthropic [2025c]	✓	✓	✓ (20)	–	–
Llama 3	§5.2 Grattafiori et al. [2024]	✓	–	–	–	–
OLMo 3	§3.5.3 Olmo et al. [2025]	–	–	✓ (13)*	–	–
Qwen 2.5-Math	§4 Yang et al. [2024]	–	–	✓ (13)	–	–
GPT-4	App. C Achiam [2023]	–	✓	–	–	–
GPT-3	App. C Brown et al. [2020]	–	–	✓ (13)	–	–
Gemini 2.5	§3 Comanici et al. [2025]	–	–	✓	–	✓
DeepSeek-R1	App. D.1 Guo et al. [2025]	–	–	✓ (10)	–	–
Qwen	§5.2.6 Bai et al. [2023]	–	–	✓ (13)	–	–

Table 16: A stratification of semantic duplicates detected in the wild by the main type of semantic duplication, which includes mechanisms such as language and logic relationship.

Form of duplication	**CodeForces** ($n = 1{,}150$)	**MBPP** ($n = 1{,}200$)
Verbatim restatement	15.3%	2.2%
Same-language paraphrase	9.9%	19.3%
Cross-programming-language	0.3%	7.9%
Cross-natural-language	1.6%	1.0%
Logical complement	1.6%	4.1%
Corpus strictly generalizes test	8.0%	22.6%
Identical algorithm, different problem statement	63.2%	42.9%

- **Cross-programming-language:** the same task and solution rendered in a different programming language than the benchmark's canonical one.
- **Cross-natural-language:** the same problem written in a different natural language.
- **Logical complement:** the corpus item states the complementary problem; rendering logic for a solution is identical, up to complement.
- **Corpus strictly generalizes tests:** the corpus problem is a strict generalization; solving it solves the benchmark item as a special case (sort a list of length N, versus sort a list of length 5). We are careful of these, since certain problems require more efficient solutions than the general solution. Only when the general solution strictly covers the specific solution, up to triviality, do we consider this a semantic duplicate.
- **Identical algorithm, different problem statement:** surface-different statements solved by the same underlying algorithm, so a model that has learned one effectively has the other. That is, the training data contains in it the solution to the test problem, even if the problem statement isn't easily identifiable as the same problem.

Table 16 shows the rates we find for each type.

F More qualitative examples labeled by our pipeline

F.1 MBPP

Example 1

Test sample ID: `MBPP (125)`. Match type: `subset`.

Corpus sample ID: `dolci_100pct,e581fb0748c03cf9c80f70cf5b371514706c073c0b3f4db7ae4aa4ccb47d9332`.

Test Set Sample

Write a function to find the maximum difference between the number of 0s and number of 1s in any sub-string of the given binary string.

```
def find_length(string):
    n = len(string)
    current_sum = 0
    max_sum = 0
    for i in range(n):
        current_sum += (1 if string[i] == '0'
             else -1)
        if current_sum < 0:
            current_sum = 0
        max_sum = max(current_sum, max_sum)
    return max_sum if max_sum else 0
```

Corpus Sample

Prompt: Given an array of integers, write a function that returns the maximum sum of any contiguous subarray within the array. For example, given the input array [2, 4, -3, 5], the function should return 11, which is the sum of the subarray [2, 4, -3, 5].
Response:

```
def max_subarray_sum(arr):
    """
    Compute the maximum sum of any contiguous
         sub-array.
    Parameters
    ----------
    arr : list[int]
        List of integers (can contain negatives
             and positives).
    Returns
    -------
    int
        The largest possible sum of a
            contiguous subarray.
        If the list is empty, returns 0.
    Examples
    --------
    >>> max_subarray_sum([2, 4, -3, 5])
    8
    >>> max_subarray_sum([-2, -3, -1, -5])
    -1
    """
    if not arr:                          # Handle
         empty array
        return 0
    # Kadane's algorithm
    current_max = global_max = arr[0]
    for num in arr[1:]:
        current_max = max(num, current_max +
             num)
        global_max = max(global_max,
            current_max)
    return global_max
```

Example 2

Test sample ID: `MBPP (412)`. Match type: `equivalent`.

Corpus sample ID: `dolci_100pct,a2838fb2d645df8018072f6fec43f3c40710d121692be022aab15eafe1d761ac`.

Test Set Sample

Write a python function to remove odd numbers from a given list.

```
def remove_odd(l):
    for i in l:
        if i % 2 != 0:
            l.remove(i)
    return l
```

Corpus Sample

Prompt: You are tasked with creating a Python function that takes a list of integers and returns a new list containing only the even numbers from the original list. If there are no even numbers, the function should return an empty list. Implement the function `filter_even_numbers(arr: List[int]) -> List[int]` where `arr` is the input list of integers. The output should be a list of even integers extracted from the input list in the order they were encountered.
Response:

```
from typing import List

def filter_even_numbers(arr: List[int]) -> List
     [int]:
    """
    Return a new list containing only the even
         numbers from the input list.
    Parameters:
    arr (List[int]): List of integers to filter
        .
    Returns:
```

```
    List[int]: List of even integers in their
        original order.
    """
    return [num for num in arr if num % 2 == 0]
```

Example 3

Test sample ID: `MBPP (431)`. Match type: `subset`.

Corpus sample ID: `dolci_100pct,84aaf5508ea33e8ca439094fb305328bf000127e3c22a65777bf7fdfac61ca80`.

Test Set Sample

Write a function that takes two lists and returns true if they have at least one common element.

```
def common_element(list1, list2):
    result = False
    for x in list1:
        for y in list2:
            if x == y:
                result = True
                return result
```

Corpus Sample

Prompt: Given two collections of elements in the form of lists, you're required to produce a new list. This resultant list has to manifest exclusively those elements found identically present within both lists. The task involves multiple steps: identify, compare, and compile. The two source lists are as follow:
`list1 = [1, 2, 3, 4, 5]`
`list2 = [3, 4, 5, 6, 7]`
Response: You can solve this task in Python quite easily using list comprehension, here is how:

```
list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]
result = [x for x in list1 if x in list2]
print(result)
```

When you run this program, it will print:
`[3, 4, 5]`
Explanation:
The expression `x for x in list1 if x in list2` goes through each element `x` in `list1` and checks `if x in list2`. If `True`, the element `x` is included in the `result` list.
Also, if you want the resulting list to be unique (i.e., no duplicate elements), you can change the list to a set which automatically removes duplicates in Python:

```
result = list(set([x for x in list1 if x in
    list2]))
```

This will give you the same output but ensure that there are no duplicates.

Example 4

Test sample ID: `MBPP (82)`. Match type: `equivalent`.

Corpus sample ID: `dolma_1pct,ce549f574ae55532`.

Test Set Sample

Write a function to find the volume of a sphere.

```
import math
def volume_sphere(r):
    volume = (4/3) * math.pi * r * r * r
    return volume
```

Corpus Sample

```
double VolSphere(double x){
    return (4/3) * M_PI * pow(x, 3);
}
```

Example 5

Test sample ID: `MBPP (131)`. Match type: `equivalent`.

Corpus sample ID: `dolma_1pct,4ec4fea5f7357828`.

Test Set Sample

Write a python function to reverse only the vowels of a given string (where y is not a vowel).

```
def reverse_vowels(str1):
    vowels = ""
    for char in str1:
        if char in "aeiouAEIOU":
            vowels += char
    result_string = ""
    for char in str1:
        if char in "aeiouAEIOU":
            result_string += vowels[-1]
```

Corpus Sample

```
/**
 * Created by parcpaes on 09/05/2018.
 */
public class reversevowel {
    private static TreeSet<Character> vowels =
        new TreeSet<>(
        Arrays.asList('a','e','i','o','u','A','
            E','I','O','U'));
    public static String reverseVowels(String s
        ){
        char letter[] = s.toCharArray();
```

```
            vowels = vowels[:-1]
        else:
            result_string += char
    return result_string
```

```
        int i=0;
        int j=s.length()-1;
        boolean vow1,vow2;
        while(i<j){
            char temp1 = letter[i];
            vow1 = vowels.contains(temp1);
            vow2 = vowels.contains(letter[j]);
            if(vow1 && vow2){
                letter[i++]=letter[j];
                letter[j--] = temp1;
            }
            if(!vow1)i++;
            if(!vow2)j--;
        }
        return String.valueOf(letter);
    }
    public static void main(String arg[]){
        System.out.println(reverseVowels("
             leetcode"));
        System.out.println(reverseVowels("helly
             "));
        System.out.println(reverseVowels("
             akekioku"));
```

Example 6

Test sample ID: `MBPP (249)`. Match type: `subset`.

Corpus sample ID: `dolci_100pct,0d3fd59142d8b0152ba5bb18935497e997d0811a06d860c7911b0af790e3aafe`.

Test Set Sample

Write a function to find the intersection of two arrays.

```
def intersection_array(array_nums1,array_nums2)
    :
    result = list(filter(lambda x: x in
         array_nums1, array_nums2))
    return result
```

Corpus Sample

Prompt: Write a program that reads two lists of integers from standard input and prints a list of their common elements.
Input:

- The first line contains space-separated integers representing the first list.
- The second line contains space-separated integers representing the second list.

Output:

- Output, on a single line, a list of integers in the same order as they appear in the first input list, containing only the elements that are also in the second list. Print the list using the standard Python list format (e.g., `[2, 3]`).

Example:
Input:
`1 2 3 4`
`2 3 5 6`
Output:
`[2, 3]`
Response:

```
def main():
    list1 = list(map(int, input().split()))
    list2 = list(map(int, input().split()))
    set2 = set(list2)
    result = [item for item in list1 if item in
          set2]
    print(result)

if __name__ == "__main__":
    main()
```

Example 7

Test sample ID: `MBPP (475)`. Match type: `equivalent`.

Corpus sample ID: `dolma_1pct,160310d03a10d7fc`.

Test Set Sample

Write a function to sort a dictionary by value.

```
from collections import Counter
def sort_counter(dict1):
    x = Counter(dict1)
    sort_counter = x.most_common()
```

Corpus Sample

```
# 请按字典中的value值进行排序
d = {'a': 24, 'g': 52, 'l': 12, 'k': 33}
sorted(d.items(), key=lambda x: x[1])  # [('l',
      12), ('a', 24), ('k', 33), ('g', 52)]
```

```
    return sort_counter
```

Example 8

Test sample ID: `MBPP (428)`. Match type: `equivalent`.

Corpus sample ID: `dolma_1pct,a9ff098110261d30`.

Test Set Sample

Write a function to sort the given array by using shell sort.

```
def shell_sort(my_list):
    gap = len(my_list) // 2
    while gap > 0:
        for i in range(gap, len(my_list)):
            current_item = my_list[i]
            j = i
            while j >= gap and my_list[j - gap]
                  > current_item:
                my_list[j] = my_list[j - gap]
                j -= gap
            my_list[j] = current_item
        gap //= 2
    return my_list
```

Corpus Sample

```
function shell_sort_3(_array){
    var len = _array.length;
    for(var jump = Math.floor(len/2);jump>=1;
         jump = Math.floor(jump/2)){
        for(var i = jump;i< len;i++){
            for(var j = i;j>=jump&&_array[j-
                 jump]>_array[j];j-=jump){
                var temp = _array[j-jump];
                _array[j-jump] = _array[j];
                _array[j] = temp;
            }
        }
    }
}
```

Example 9

Test sample ID: `MBPP (65)`. Match type: `subset`.

Corpus sample ID: `dolci_100pct,f98cda41dfe2a9ced6b9fb7843db3b25dc9e49ae042c6f6bc68675e4f49a0937`.

Test Set Sample

Write a function to flatten a list and sum all of its elements.

```
def recursive_list_sum(data_list):
    total = 0
    for element in data_list:
        if type(element) == type([]):
            total = total + recursive_list_sum(
                element)
        else:
            total = total + element
    return total
```

Corpus Sample

Prompt: Implement a function that calculates the total sum of all integers within a nested list structure. The list may contain integers, strings representing integers, and other lists. Your function should handle any level of nesting. For example, the input `['1', '2', 3, [4, '5']]` should return 15. The function signature should be: `def sum_nested(lst: List[Union[int, str, List]])-> int:`
Response:

```
from typing import List, Union

def sum_nested(lst: List[Union[int, str, List
    ]]) -> int:
    """
    Recursively sum all integers found in a (
        possibly deeply) nested list.
    Elements in the list can be:
        - int                -> added directly
        - str (numeric)      -> converted to int
            , then added
        - list               -> processed
            recursively

    Parameters
    ----------
    lst : List[Union[int, str, List]]
        The nested list structure containing
            integers, numeric strings, or
            other lists.

    Returns
    -------
    int
        The total sum of all integers found in
            the structure.
    """
    total = 0
    for item in lst:
        if isinstance(item, list):
            total += sum_nested(item)        #
                Recurse on sub-list
        elif isinstance(item, int):
            total += item                    #
                Add integer directly
        elif isinstance(item, str):
            total += int(item)               #
                Convert numeric string to int
        else:
```

```
            # If you want to ignore unsupported
                types, replace the line
                below with 'continue'.
            raise TypeError(f"Unsupported data
                type: {type(item).__name__}")
    return total
```

Example 10

Test sample ID: `MBPP (297).` Match type: `equivalent.`

Corpus sample ID: `dolci_100pct,e04feb161678637b214f7f504bc70157685665afbebdb1deefeb7ebbf240875f.`

Test Set Sample

Write a function to flatten a given nested list structure.

```
def flatten_list(list1):
    result_list = []
    if not list1: return result_list
    stack = [list(list1)]
    while stack:
        c_num = stack.pop()
        next = c_num.pop()
        if c_num: stack.append(c_num)
        if isinstance(next, list):
            if next: stack.append(list(next))
        else: result_list.append(next)
    result_list.reverse()
    return result_list
```

Corpus Sample

Prompt: Given a nested list of integers, write a function `flatten_list(nested_list)` that takes a single argument `nested_list`, which can contain integers and other lists (which can themselves contain integers or more lists). Your function should return a flat list containing all the integers from the nested structure in the order they appear. If the input list is empty, return an empty list. Note that the input will never contain strings or bytes, only integers and lists of integers. For example, calling `flatten_list([1, [2, 3], [[4], 5]])` should return `[1, 2, 3, 4, 5]`.
Response:

```
def flatten_list(nested_list):
    """
    Recursively flattens a nested list of
        integers.
    Args:
        nested_list (list): A list that may
            contain integers and/or other
            lists.
    Returns:
        list: A flat list containing all the
            integers in their original order.
    """
    flat = []
    def _flatten(item):
        for element in item:
            if isinstance(element, list):
                _flatten(element)
            else:
                flat.append(element)
    _flatten(nested_list)
    return flat
```

F.2 CodeForces

Example 1

Test sample ID: `CodeForces (1987/D).` Match type: `exact.`

Corpus sample ID: `dolci_sft,55333872e569f21088eac6eef7aa8093d7ee90d223bc9865eafbcec4e8faf178.`

Test Set Sample

Alice and Bob are playing a game. Initially, there are n cakes, with the i-th cake having a tastiness value of a_i.
Alice and Bob take turns eating them, with Alice starting first:

- In her turn, Alice chooses and eats any remaining cake whose tastiness is strictly greater than the maximum tastiness of any of the cakes she's eaten before that. Note that on the first turn, she can choose any cake.
- In his turn, Bob chooses any remaining cake and eats it.

The game ends when the current player can't eat a suitable cake. Let x be the number of cakes that Alice ate. Then, Alice wants to maximize x, while Bob wants to minimize x.
Find out how many cakes Alice will eat if both players play optimally.
Each test contains multiple test cases. The first line of input contains a single integer t ($1 \leq t \leq 500$) — the number of test cases. The description of the test cases follows.
The first line of each test case contains a single integer n ($1 \leq n \leq 5000$) — the number of cakes.
The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq n$) — the tastiness values of the cakes.
It is guaranteed that the sum of n over all test cases does not exceed 5000.
For each test case, output a single integer — the number of cakes Alice will eat if both players play optimally.

Corpus Sample

Prompt: Alice and Bob are playing a game with cakes.
Input
The first line contains an integer t ($1 \leq t \leq 500$) – the number of test cases. For each test case:

- The first line contains an integer n ($1 \leq n \leq 5000$) – the number of cakes.
- The second line contains n integers $a_1 \ldots a_n$ ($1 \leq a_i \leq n$) – the tastiness of each cake.

The sum of n over all test cases does not exceed 5000.
Rules of the game

- Alice moves first, then they alternate.
- When it is Alice's turn she must choose a remaining cake whose tastiness is strictly greater than every cake she has previously eaten. On her first turn she may take any cake.
- When it is Bob's turn he may take any remaining cake.
- If a player cannot move the game ends.

Both players play optimally: Alice tries to maximize the number x of cakes she eats, Bob tries to minimize x.
Output
For each test case output a single integer – the value of x when both play optimally.
Response: To solve this problem, we need to simulate a game where **Alice and Bob** take turns eating cakes under specific rules:

- **Alice** can only eat a cake that has **strictly greater tastiness** than the last one she ate.
- **Bob** can eat any remaining cake.
- The goal is to **maximize the number of cakes Alice eats**, assuming both play optimally.

Key Observations:

1. **Alice's constraint** is that she must eat strictly increasing tastiness values.
2. **Bob's move** is to reduce Alice's future options by taking a cake that might be useful for her.
3. Since the total number of cakes is small (≤ 5000), we can use **dynamic programming with memoization**.

Strategy:
We define a recursive function with memoization:
`def dfs(last_a, remaining_cakes)`
Where:

- `last_a` is the last tastiness value that Alice ate.
- `remaining_cakes` is a list of remaining cake tastiness values (sorted for easier processing).

At each step:

- If it's **Alice's turn**, she can only pick a cake with tastiness **greater than** `last_a`.
- If it's **Bob's turn**, he can pick **any** remaining cake.

We simulate both players' turns, alternating, and memoize the result based on:

- The set of remaining cakes (as a frozenset for memoization).
- The last value Alice ate.

Python Code:

```
from functools import lru_cache
import sys
input = sys.stdin.read

def solve():
    data = input().split()
    idx = 0
    t = int(data[idx])
    idx += 1
    results = []

    for _ in range(t):
        n = int(data[idx])
        idx += 1
        a = list(map(int, data[idx:idx + n]))
        idx += n

        a.sort()  # Sort for easier processing

        from collections import Counter
        freq = Counter(a)

        # Build list of unique tastiness values in increasing order
        unique_sorted = sorted(set(a))

        @lru_cache(maxsize=None)
        def dfs(last_a, pos, alice_turn):
            if pos == n:
                return 0

            max_cakes = 0
            current = a[pos]
```

```
            if alice_turn:
                if last_a < current:
                    # Alice takes this cake
                    max_cakes = 1 + dfs(current, pos + 1, False)
                # Try skipping this cake
                max_cakes = max(max_cakes, dfs(last_a, pos + 1, alice_turn))
            else:
                # Bob takes this cake
                max_cakes = dfs(last_a, pos + 1, not alice_turn)
                # Try skipping this cake
                max_cakes = max(max_cakes, dfs(last_a, pos + 1, alice_turn))

            return max_cakes

        # Try all possible first moves for Alice
        max_x = 0
        for i in range(n):
            max_x = max(max_x, dfs(a[i], i + 1, False))

        results.append(str(max_x))

    print("\n".join(results))

solve()
```

Explanation:

- We **sort the cakes** to process in increasing order (easier to handle Alice's increasing constraint).
- We use **memoization** via `@lru_cache` to avoid recomputation.
- We simulate all possible first moves for Alice and pick the one that gives her the **maximum number of cakes**.

Time Complexity:

- For each test case, the number of unique tastiness values is at most n, and the number of states in the DP is $O(n^2)$ in the worst case.
- Since the total n across all test cases is ≤ 5000, this is efficient enough.

Example:
Input:

```
1
5
1 2 3 4 5
```

Output:

```
3
```

Explanation:

- Alice can eat 1, 3, 5 (or similar increasing sequences).
- Bob will block the best options to minimize Alice's total.

Let me know if you'd like a version with full test case handling or more explanation!

Example 2

Test sample ID: `CodeForces (1984/D)`. Match type: `exact`.

Corpus sample ID: `dolci_sft,32954b54484de7058cb6a10376ed889752a1f60f534eac8192a7c55c2d3cdf26`.

Test Set Sample

You are given a string s consisting of lowercase Latin characters. Count the number of nonempty strings $t \neq$ `"a"` such that it is possible to partition[†] s into some substrings satisfying the following conditions:

- each substring either equals t or `"a"`, and
- at least one substring equals t.

[†] A partition of a string s is an ordered sequence of some k strings $t_1, t_2, \ldots, t_k$ (called substrings) such that $t_1 + t_2 + \cdots + t_k = s$, where $+$ represents the concatenation operation.
The first line contains a single integer t ($1 \leq t \leq 10^4$) — the number of test cases.
The only line of each test case contains a string s consisting of lowercase Latin characters ($2 \leq |s| \leq 2 \times 10^5$).
The sum of $|s|$ over all test cases does not exceed 3×10^5.
For each test case, output a single integer — the number of nonempty strings $t \neq$ `"a"` that satisfy all constraints.

Corpus Sample

Prompt: You are given a string s consisting only of lowercase Latin letters.
Your task is to count how many non-empty strings t (with $t \neq$ `"a"`) have the following property:
It is possible to split s into consecutive substrings so that every substring is either exactly t or exactly `"a"`, and at least one of the substrings is t.

Input
The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.
Each of the next t lines contains a string s ($2 \le |s| \le 2 \cdot 10^5$). The total length of all given strings does not exceed $3 \cdot 10^5$.
Output
For each test case output a single integer — the number of different strings t satisfying the conditions for the corresponding s.
Response:

```
import sys

def count_valid_t(s: str) -> int:
    if all(c == 'a' for c in s):
        n = len(s)
        return max(0, n - 1)  # strings "aa", "aaa", ..., length n
    non_a_positions = [i for i, c in enumerate(s) if c != 'a']
    generated = set()
    valid_t_count = 0
    for i in range(len(s)):
        for j in range(i, len(s)):
            t = s[i:j + 1]
            if t == "a":
                continue
            if t in generated:
                continue
            generated.add(t)
            pos = 0
            len_t = len(t)
            covered = set()
            valid = True
            while pos < len(s):
                if pos + len_t <= len(s) and s[pos:pos + len_t] == t:
                    for k in range(pos, pos + len_t):
                        covered.add(k)
                    pos += len_t
                elif s[pos] == 'a':
                    pos += 1
                else:
                    valid = False
                    break
            if valid and all(p in covered for p in non_a_positions):
                valid_t_count += 1
    return valid_t_count

def main():
    data = sys.stdin.read().strip().split('n')
    if not data:
        return
    tc = int(data[0])
    idx = 1
    for _ in range(tc):
        s = data[idx].strip()
        idx += 1
        print(count_valid_t(s))

if __name__ == "__main__":
    main()
```

Example 3

Test sample ID: `CodeForces (1978/C)`. Match type: `equivalent`.

Corpus sample ID: `dolci_sft,311e326c774442b2cac0b1665be99dfc9ca0b884f6358a713a0c5fe33058a891`.

Test Set Sample

Let's call the Manhattan value of a permutation[†] p the value of the expression $|p_1 - 1| + |p_2 - 2| + \cdots + |p_n - n|$.
For example, for the permutation $[1, 2, 3]$, the Manhattan value is $|1 - 1| + |2 - 2| + |3 - 3| = 0$, and for the permutation $[3, 1, 2]$, the Manhattan value is $|3 - 1| + |1 - 2| + |2 - 3| = 2 + 1 + 1 = 4$.
You are given integers n and k. Find a permutation p of length n such that its Manhattan value is equal to k, or determine that no such permutation exists.
[†]A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).
Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of the test cases follows.
The only line of each test case contains two integers n and k ($1 \le n \le 2 \times 10^5$, $0 \le k \le 10^{12}$) — the length of the permutation and the required Manhattan value.
It is guaranteed that the sum of n over all test cases does not exceed 2×10^5.
For each test case, if there is no suitable permutation, output `"No"`. Otherwise, in the first line, output `"Yes"`, and in the second line, output n distinct integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$) — a suitable permutation.
If there are multiple solutions, output any of them.
You can output the answer in any case (for example, the strings `"yEs"`, `"yes"`, `"Yes"`, and `"YES"` will be recognized as a positive answer).

Corpus Sample

Prompt: We define the Manhattan value of a permutation p of length n as $|p_1 - 1| + |p_2 - 2| + \cdots + |p_n - n|$.
Given integers n and k you must decide whether there exists a permutation of $1 \ldots n$ whose Manhattan value equals k.
Input (via standard input)
The first line contains t ($1 \leq t \leq 10^4$) – the number of test cases.
Each of the next t lines contains two integers n and k ($1 \leq n \leq 2 \cdot 10^5$, $0 \leq k \leq 10^{12}$).
The sum of n over all test cases does not exceed $2 \cdot 10^5$.
Output (via standard output)
For every test case output either
No
if no such permutation exists, or
Yes
$p_1\ p_2\ \ldots\ p_n$
in two separate lines if a suitable permutation exists. Any valid permutation may be printed.
Response:

```
import sys
def main():
    data = sys.stdin.read().strip().split()
    if not data:
        return
    it = iter(data)
    t = int(next(it))
    for _ in range(t):
        n = int(next(it)); k = int(next(it))
        max_sum = (n * n) // 2  # theoretical upper bound
        if k % 2 != 0 or k < 0 or k > max_sum:
            print("No")
            continue
        p = list(range(1, n + 1))
        current_sum = 0
        found = False
        # try to build the needed sum by swapping elements that are d apart
        for d in range(n - 1, 0, -1):
            for i in range(1, n - d + 1):
                if current_sum + 2 * d > k:
                    continue
                j = i + d
                p[i - 1], p[j - 1] = p[j - 1], p[i - 1]
                current_sum += 2 * d
                if current_sum == k:
                    found = True
                    break
            if found:
                break
        if current_sum == k:
            print("Yes")
            print(' '.join(map(str, p)))
        else:
            print("No")
if __name__ == "__main__":
    main()
```

Example 4

Test sample ID: `CodeForces (1996/B)`. Match type: `equivalent`.

Corpus sample ID: `dolci_sft,0dbc4ecb2bbe6499ac5bc74522c971243f1bc0492c25048aaae49ca59fa5da17`.

Test Set Sample

Tina has a square grid with n rows and n columns. Each cell in the grid is either 0 or 1.
Tina wants to reduce the grid by a factor of k (k is a divisor of n). To do this, Tina splits the grid into $k \times k$ nonoverlapping blocks of cells such that every cell belongs to exactly one block.
Tina then replaces each block of cells with a single cell equal to the value of the cells in the block. It is guaranteed that every cell in the same block has the same value.
For example, the following demonstration shows a grid being reduced by factor of 3.
Original grid 000111000111000111111000111000111000
Reduced grid 0110
Help Tina reduce the grid by a factor of k.
The first line contains t ($1 \leq t \leq 100$) — the number of test cases.
The first line of each test case contains two integers n and k ($1 \leq n \leq 1000$, $1 \leq k \leq n$, k is a divisor of n) — the number of rows and columns of the grid, and the factor that Tina wants to reduce the grid by.
Each of the following n lines contain n characters describing the cells of the grid. Each character is either 0 or 1. It is guaranteed every k by k block has the same value.
It is guaranteed the sum of n over all test cases does not exceed 1000.
For each test case, output the grid reduced by a factor of k on a new line.

Corpus Sample

Prompt: Tina has a square grid with n rows and n columns ($1 \leq n \leq 1000$). Every cell contains either the character 0 or 1.
Tina wants to reduce the grid by a factor of k, where k is a divisor of n ($1 \leq k \leq n$). She divides the grid into non-overlapping $k \times k$ blocks so that each cell belongs to exactly one block. It is guaranteed that within every $k \times k$ block all cells have the same value. Each block is replaced by a single cell equal to that common value, producing a new grid of size $(n/k) \times (n/k)$.
Input
The first line contains t ($1 \leq t \leq 100$) — the number of test cases.
For each test case:

- The first line contains two integers n and k.
- The next n lines each contain a string of n characters (0 or 1) describing the grid.

It is guaranteed that the sum of n over all test cases does not exceed 1000.
Output
For every test case output the reduced grid, each row on its own line. Output test cases one after another without extra blank lines.
Example
Input

```
4
4 4
0000
0000
0000
0000
6 3
000111
000111
000111
111000
111000
111000
6 2
001100
001100
111111
111111
110000
110000
8 1
11111111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
```

Output

```
0
01
10
010
111
100
11111111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
```

Response:

```
def main():
    import sys
    t = int(sys.stdin.readline())
    for _ in range(t):
        n, k = map(int, sys.stdin.readline().split())
        grid = [sys.stdin.readline().strip() for _ in range(n)]
        step = k
        for i in range(0, n, step):
            # build one row of the reduced grid
            reduced_row = ''.join(grid[i][j] for j in range(0, n, step))
            print(reduced_row)

if __name__ == "__main__":
    main()
```

Example 5

Test sample ID: `CodeForces (1942/C1)`. Match type: `subset`.

Corpus sample ID: `dolci_sft,239cfdbeca54efec1ef280a8c44dbf2997090d1ff21a5b03287db04769a83811.`

Test Set Sample

This is the easy version of the problem. The only difference between the two versions is the constraint on y. In this version $y = 0$. You can make hacks only if both versions are solved.
Bessie has received a birthday cake from her best friend Elsie, and it came in the form of a regular polygon with n sides. The vertices of the cake are numbered from 1 to n clockwise. You and Bessie are going to choose some of those vertices to cut non-intersecting diagonals into the cake. In other words, the endpoints of the diagonals must be part of the chosen vertices.
Bessie would only like to give out pieces of cake which result in a triangle to keep consistency. The size of the pieces doesn't matter, and the whole cake does not have to be separated into all triangles (other shapes are allowed in the cake, but those will not be counted).
Bessie has already chosen x of those vertices that can be used to form diagonals. She wants you to choose no more than y other vertices such that the number of triangular pieces of cake she can give out is maximized.
What is the maximum number of triangular pieces of cake Bessie can give out?
The first line contains a single integer t ($1 \leq t \leq 10^4$) — the number of test cases.
The first line of each test case consists of three integers, n, x, and y ($4 \leq n \leq 10^9$, $2 \leq x \leq \min(n, 2 \times 10^5)$, $y = 0$) — the number of sides of the polygon, number of vertices Bessie has chosen, and the maximum number of other vertices you can choose.
The second line consists of x distinct integers from 1 to n, representing the vertices Bessie has chosen.
It is guaranteed the sum of x over all test cases does not exceed 2×10^5.
For each test case, output a single integer: the maximum number of non-intersecting triangular pieces of cake she can give out.

Corpus Sample

Prompt: Birthday Cake Triangles (hard version)

Bessie's cake is a regular polygon with n vertices numbered $1..n$ clockwise ($4 \leq n \leq 10^9$). Bessie has already selected x of those vertices ($2 \leq x \leq \min(n, 2 \cdot 10^5)$). You may additionally mark at most y other vertices ($0 \leq y \leq n - x$). After all marked vertices are fixed we are allowed to draw non-intersecting diagonals but every diagonal must have both endpoints among the marked vertices. From the regions obtained we only count those that are triangles; other shapes are ignored. For every test case output the maximum possible number of counted triangles.
Input

The first line contains t ($1 \leq t \leq 10^4$) – the number of test cases.
Each test case consists of two lines:

- line 1: n x y
- line 2: x distinct integers – the vertices already chosen by Bessie (in any order)

It is guaranteed that the sum of x over all test cases does not exceed $2 \cdot 10^5$.
Output

For each test case output one line – the maximum number of triangles that can be obtained.
Example

Input

```
3
8 4 2
1 6 2 5
7 3 1
6 4 3
4 2 2
1 3
```

Output

```
6
5
2
```

Response:

```
import sys

def solve():
    it = iter(sys.stdin.read().strip().split())
    t = int(next(it))
    for _ in range(t):
        n = int(next(it)); x = int(next(it)); y = int(next(it))
        chosen = [int(next(it)) for _ in range(x)]
        chosen.sort()
        # number of non-empty gaps between consecutive chosen vertices
        gaps = 0
        for i in range(x):
            if i+1 < x:
                gap = chosen[i+1] - chosen[i] - 1
            else:
                gap = (chosen[0] + n) - chosen[i] - 1
            if gap > 0:
                gaps += 1
        base = x - 2 + gaps  # triangles before picking extra vertices
        ans = max(0, min(n - 2, base + 2 * y))
        print(ans)

if __name__ == "__main__":
    solve()
```

Example 6

Test sample ID: `CodeForces (1986/G2).` Match type: `subset.`

Corpus sample ID: `dolci_sft,373fbbd64cb65ba1ddfcb78c89bf2b3e62e8b4eba47f38cacec16a39c7e7c83c.`

Test Set Sample

This is the hard version of the problem. The only difference is that in this version $n \leq 5 \times 10^5$ and the sum of n for all sets of input data does not exceed 5×10^5.
You are given a permutation p of length n. Calculate the number of index pairs $1 \leq i < j \leq n$ such that $p_i \cdot p_j$ is divisible by $i \cdot j$ without remainder.
A permutation is a sequence of n integers, in which each integer from 1 to n occurs exactly once. For example, $[1]$, $[3, 5, 2, 1, 4]$, $[1, 3, 2]$ are permutations, while $[2, 3, 2]$, $[4, 3, 1]$, $[0]$ are not.
Each test consists of several sets of input data. The first line contains a single integer t ($1 \leq t \leq 10^4$) — the number of sets of input data. Then follows their description.
The first line of each set of input data contains a single integer n ($1 \leq n \leq 5 \times 10^5$) — the length of the permutation p.
The second line of each set of input data contains n distinct integers $p_1, p_2, \ldots, p_n$ ($1 \leq p_i \leq n$) — the permutation p.
It is guaranteed that the sum of n for all sets of input data does not exceed 5×10^5.
For each set of input data, output the number of index pairs $1 \leq i < j \leq n$ such that $p_i \cdot p_j$ is divisible by $i \cdot j$ without remainder.

Corpus Sample

Prompt: Chef has an array A of N positive integers. For every ordered pair of indices (i, j) with $1 \leq i < j \leq N$, Chef is interested in whether the product of the two corresponding array elements is divisible by the product of the indices, i.e.
$A_i \cdot A_j$ is divisible by $i \cdot j$
For each test case output how many pairs (i, j) satisfy the condition.
Input
The first line contains a single integer T – the number of test cases. For each test case:

- The first line contains an integer N – the length of the array ($2 \leq N \leq 10^5$).
- The second line contains N space-separated integers $A_1 \ldots A_N$ ($1 \leq A_i \leq 10^5$).

The sum of N over all test cases does not exceed 10^5.
Output
For every test case print a single line with a single integer: the number of index pairs (i, j) ($1 \leq i < j \leq N$) for which $A_i \cdot A_j$ is divisible by $i \cdot j$.
Example
Input

```
3
3
1 2 3
4
2 3 5 6
5
3 9 14 6 10
```

Output

```
3
2
6
```

Response:

```
import sys, math
from collections import defaultdict

def divisors(x: int):
    """Return all positive divisors of x."""
    out = []
    for d in range(1, int(math.isqrt(x)) + 1):
        if x % d == 0:
            out.append(d)
            if d * d != x:
                out.append(x // d)
    return out

def main():
    data = list(map(int, sys.stdin.read().strip().split()))
    if not data:
        return
    t = data[0]
    p = 1
    for _ in range(t):
        n = data[p]; p += 1
        a = data[p:p + n]; p += n
        freq = defaultdict(lambda: defaultdict(int))
        ans = 0
        for idx, val in enumerate(a, start=1):
            g = math.gcd(val, idx)
            d_cur = idx // g
            n_cur = val // g
            # count previous i such that d_i | n_cur and d_cur | n_i
            for d in divisors(n_cur):
                ans += freq[d].get(d_cur, 0)
            # store information of the current index for future indices
```

```
            for div in divisors(n_cur):
                freq[d_cur][div] += 1
        print(ans)

if __name__ == "__main__":
    main()
```

Example 7

Test sample ID: `CodeForces (1989/B)`. Match type: `superset`.

Corpus sample ID: `dolci_sft,bc86d7050afb1b4d2336493c5d4cf5c07d3e8d038d45c80e9151d681d064313d`.

Test Set Sample

You are given two strings a and b, both consisting of lowercase Latin letters.
A subsequence of a string is a string which can be obtained by removing several (possibly zero) characters from the original string. A substring of a string is a contiguous subsequence of that string.
For example, consider the string `abac`:

- `a`, `b`, `c`, `ab`, `aa`, `ac`, `ba`, `bc`, `aba`, `abc`, `aac`, `bac` and `abac` are its subsequences;
- `a`, `b`, `c`, `ab`, `ba`, `ac`, `aba`, `bac` and `abac` are its substrings.

Your task is to calculate the minimum possible length of the string that contains a as a substring and b as a subsequence.
The first line contains a single integer t ($1 \leq t \leq 10^3$) — the number of test cases.
The first line of each test case contains a string a ($1 \leq |a| \leq 100$), consisting of lowercase Latin letters.
The second line of each test case contains a string b ($1 \leq |b| \leq 100$), consisting of lowercase Latin letters.
For each test case, print a single integer — the minimum possible length of the string that contains a as a substring and b as a subsequence.

Corpus Sample

Prompt: You are given two strings a and b consisting of lowercase Latin letters. In one operation you may delete the first character or the last character of either string (if the string is not empty). You may perform any number of operations in any order.
After a sequence of operations the two remaining strings (each is some contiguous substring of the original one and may be empty) must become exactly equal. Your task is to find the minimum possible number of operations required to achieve this.
Input
The first line contains an integer t ($1 \leq t \leq 100$) — the number of test cases.
For every test case two lines follow: the first line contains the string a ($1 \leq |a| \leq 20$), the second line contains the string b ($1 \leq |b| \leq 20$). Both strings consist only of lowercase Latin letters.
Output
For each test case output a single integer — the minimum number of operations needed to make a and b equal (empty strings are considered equal).
Example
Input
5
a
a
abcd
bc
hello
codeforces
hello
helo
dhjakjsnasjhfksafasd
adjsnasjhfksvdafdser
Output
0
2
13
3
20
Response:

```
def generate_substrings(s):
    substrs = set()
    n = len(s)
    for i in range(n):
        for j in range(i, n):
            substrs.add(s[i:j+1])
    return substrs

def main():
    import sys
    data = sys.stdin.read().splitlines()
    if not data:
        return
    t = int(data[0].strip())
    idx = 1
    for _ in range(t):
        a = data[idx].strip(); idx += 1
```

```
        b = data[idx].strip(); idx += 1
        a_sub = generate_substrings(a)
        b_sub = generate_substrings(b)
        max_len = 0
        for s in a_sub:
            if s in b_sub and len(s) > max_len:
                max_len = len(s)
        ops = len(a) + len(b) - 2 * max_len
        print(ops)

if __name__ == "__main__":
    main()
```

Example 8

Test sample ID: `CodeForces (1991/F)`. Match type: `superset`.

Corpus sample ID: `dolci_sft,0361b4c13bf295f2f7293a8917c35b82ea23514708e283101f2638eaae1279c7`.

Test Set Sample

You are given n sticks, numbered from 1 to n. The length of the i-th stick is a_i.
You need to answer q queries. In each query, you are given two integers l and r ($1 \leq l < r \leq n, r - l + 1 \geq 6$). Determine whether it is possible to choose 6 distinct sticks from the sticks numbered l to r, to form 2 non-degenerate triangles*.
The first line contains two integers n and q ($6 \leq n \leq 10^5$, $1 \leq q \leq 10^5$) — the number of sticks and the number of queries respectively.
The second line contains n integers $a_1, a_2, ..., a_n$ ($1 \leq a_i \leq 10^9$) — a_i denotes the length of the i-th stick.
Each of the following q lines contains two integers l and r ($1 \leq l < r \leq n, r - l + 1 \geq 6$) — the parameters of each query.
For each query, output `""YES""` (without quotes) if it is possible to form 2 triangles, and `""NO""` (without quotes) otherwise.
You can output the answer in any case (upper or lower). For example, the strings `""yEs""`, `""yes""`, `""Yes""`, and `""YES""` will be recognized as positive responses.

Corpus Sample

Prompt: You are given the lengths of N wooden sticks. You want to know whether you can choose exactly three of them that form a non-degenerate triangle (a triangle with strictly positive area, i.e. the sum of the two shorter sides is strictly greater than the longest side).
Input
The first line contains an integer N ($3 \leq N \leq 20000$) – the number of sticks collected. The second line contains N positive integers, each strictly less than 2^60, giving the lengths of the sticks.
Output
Print exactly one line containing the single word `""possible""` if there exists a triple of sticks that can form a non-degenerate triangle, and `""impossible""` otherwise.
Example
Input
3
1 1 1
Output
possible
Response:

```
import sys

data = sys.stdin.read().strip().split()

if not data:
    sys.exit()

n = int(data[0])

sticks = list(map(int, data[1:1+n]))

sticks.sort()

for i in range(n - 2):
    a, b, c = sticks[i], sticks[i + 1], sticks[i + 2]

    if a + b > c:
        print("possible")

        sys.exit()

print("impossible")
```

Example 9

Test sample ID: `CodeForces (1991/D)`. Match type: `related`.

Corpus sample ID: `dolci_sft,d007fafb1db5925fc9894dba5be6d092c5f2042bb5794c3a6c664b3f0cfd1b02`.

Test Set Sample

You are given an undirected graph with n vertices, numbered from 1 to n. There is an edge between vertices u and v if and only if u XOR v is a prime number, where XOR denotes the bitwise XOR operator.
Color all vertices of the graph using the minimum number of colors, such that no two vertices directly connected by an edge have the same color.
Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 500$). The description of test cases follows.
The only line contains a single integer n ($1 \le n \le 2 \times 10^5$) — the number of vertices in the graph.
It is guaranteed that the sum of n over all test cases does not exceed 2×10^5.
For each test case, output two lines.
The first line should contain a single integer k ($1 \le k \le n$) — the minimum number of colors required.
The second line should contain n integers $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le k$) — the color of each vertex.
If there are multiple solutions, output any of them.

Corpus Sample

Prompt: You are given an **undirected graph** with 54 vertices, labeled from `'1'` to `'54'`. Two vertices u and v are connected by an edge **if and only if** the absolute difference $|u - v|$ is a prime number.
Your task is to assign a **non-negative integer color** to each vertex, represented as `'c[1], c[2], ..., c[54]'`, such that:

- For every edge (u, v) in the graph, $c[u] \neq c[v]$ — adjacent vertices must have different colors.
- The total number of **distinct colors used** (i.e., the number of unique values among `'c[1]'` to `'c[54]'`) is **minimized** - try your best to find a valid coloring using as few colors as possible.

Output Format: Your final answer should be a single line containing the color of each vertex in order: `'c[1], c[2], ..., c[54]'`, separated by **spaces**.
Response: We use the well-known 4-coloring of the "prime-distance" graph on the integers: since no prime is divisible by 4 (all primes p satisfy $p \bmod 4 \neq 0$), any two numbers at prime distance have different residues mod 4. Hence the assignment $c[i] = i \bmod 4$ is a proper coloring using exactly 4 colors, which is minimal (one can't do it with 3 colors because of difference 3). We now list $c[1]$ through $c[54]$.
Answer:
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0 1 2

Example 10

Test sample ID: `CodeForces (1971/G).` Match type: `related.`

Corpus sample ID: `dolci_sft,7f575b5b3298b8690432a1a8ecf96cb6c97a7d62b45ee7ee27a6e0b2dccf5aa6.`

Test Set Sample

You are given an array a consisting of n nonnegative integers.
You can swap the elements at positions i and j if a_i XOR $a_j < 4$, where XOR is the bitwise XOR operation.
Find the lexicographically smallest array that can be made with any number of swaps.
An array x is lexicographically smaller than an array y if in the first position where x and y differ, $x_i < y_i$.
The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.
The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the array.
The second line of each test case contains n integers a_i ($0 \le a_i \le 10^9$) — the elements of the array.
It is guaranteed that the sum of n over all test cases does not exceed 2×10^5.
For each test case, output n integers — the lexicographically smallest array that can be made with any number of swaps.

Corpus Sample

Prompt: You are given an integer array a of length n.
Operation: you may repeatedly pick two indices i and j ($1 \le i, j \le n$) such that $a_i + a_j$ is odd, and swap the two elements a_i and a_j.
Determine the lexicographically smallest array that can be obtained after performing any number of such operations.
Input
The first line contains a single integer n ($1 \le n \le 10^5$).
The second line contains n space-separated integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$).
Output
Output the lexicographically smallest array you can obtain, as n space-separated integers on one line.
An array x is lexicographically smaller than an array y if, at the first position where they differ, the element in x is smaller than the corresponding element in y.
Example

Input
3
4 1 7
Output
1 4 7
In the example, swap 4 and 1 (their sum is 5, which is odd) to get the lexicographically smallest array.
Response:

```
import sys

def main():
    data = list(map(int, sys.stdin.read().strip().split()))
    if not data:
        return

    n = data[0]
    a = data[1:1 + n]

    has_even = any(x % 2 == 0 for x in a)
    has_odd = any(x % 2 == 1 for x in a)

    if has_even and has_odd:
        a.sort()

    print(' '.join(map(str, a)))

if __name__ == "__main__":
    main()
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract mentions our main headline numerical results, emphasises that the main prevalence result is on one open model, and summarises the resulting direction of evidence with respect to our main 'shallow generalization' hypothesis. No overclaiming.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Section 5.2

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: No derivations.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See Section 3, Appendix A, and the GitHub repository.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: `https://github.com/AriSpiesberger/Soft-Contamination-Prevelance`

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See Appendix A and B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: See figures and tables. Appendix A.4 for details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Experiment hardware and time is listed in the paper, code supports one or many gpus and will run on other industrial grade GPUS.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: No human subjects, no private, deprecated, or copyrighted data.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See Impact Statement.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: No original data or models released.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All cited appropriately.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: N/A.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: N/A

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: N/A

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We use LLMs as a judge in several places of our pipeline, see Figure 1.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.