Detecting Data Contamination in LLMs via In-Context Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present **Contamination Detection via Context (CoDeC)**, a simple and accurate method to detect and quantify training data contamination in large language models. CoDeC distinguishes between data memorized during training and data outside the training distribution by measuring how in-context learning affects model performance. We find that in-context examples typically boost confidence for unseen datasets but may reduce it when the dataset was part of training, due to disrupted memorization patterns. Experiments show that CoDeC produces interpretable contamination scores that clearly separate seen and unseen datasets, and reveals strong evidence of memorization in open-weight models with undisclosed training corpora. The method is automated, parameter-free, and both model- and dataset-agnostic, making it easy to integrate with benchmark evaluations.

1 Introduction

Detecting contamination is crucial for the integrity of LLM evaluation [7, 17, 5]. Existing approaches mostly rely on classical techniques such as loss-based criteria [24, 23, 5], calibrating scores using an external model [17, 5], explicit overlap checks [10, 1], and related heuristics. While effective in some settings, these methods can be difficult to apply to large language models without access to the training data, may require extensive parameter tuning, and often fail to provide reliable and interpretable estimates of contamination. There is a pressing need for automated, reliable, and interpretable methods to measure contamination in LLMs, applicable across diverse datasets and model architectures.

In this work, we address this gap by proposing Contamination Detection via Context (CoDeC), a simple and effective dataset-level method for detecting and quantifying contamination in LLMs. Instead of searching for explicit overlaps, CoDeC measures distributional similarity through changes in model behavior under in-context learning. If the model has memorized datapoints from its training set, adding similar in-context examples is more likely to disrupt these memorization patterns than to improve predictions. The contamination score is computed as the percentage of datapoints in the target dataset for which the added context leads to lower confidence. Our approach requires only gray-box access (model outputs or logits) and works with any dataset.

Through extensive experiments on models with known and unknown training data, we show that CoDeC is reliable, broadly applicable, and easily interpretable. Ablations reveal that adding more context or carefully selecting examples improves separation even further, especially on diverse benchmarks. Our method enables the community to better trust reported LLM results and supports the development of more reliable model evaluation practices.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

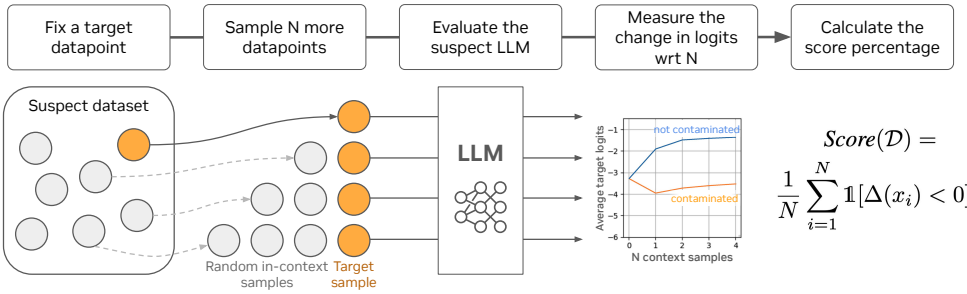


Figure 1: Overview of Contamination Detection via Context (CoDeC). For each dataset element, CoDeC augments the context with a small set of other samples from the same dataset. A decrease in the model's logits for the target sample indicates potential contamination. The overall contamination level is estimated as the fraction of samples exhibiting this effect.

2 Contamination Detection via Context

Problem definition. Given a language model M and a candidate dataset $\mathcal{D} = \{x_i\}_{i=1}^{N}$, where each x_i is a text sequence, our goal is to quantify contamination, i.e. whether that dataset or similar data was in the training set of M and the model is relying on memorization rather than generalization.

Key Idea. LLMs respond differently to in-context examples depending on prior exposure. Building on this observation, CoDeC measures the influence of in-context learning on model predictions. When given an unseen dataset, adding in-context samples taken from that dataset generally improves the model's confidence, as it can generalize better with more information. However, if the model has memorized the dataset, the in-context samples not only provide little additional information but also disrupt memorization patterns, leading to reduced confidence. Thus, by comparing confidence levels with and without in-context learning across sample sequences, we can leverage these shifts to detect contamination.

CoDeC Pipeline. The method (Figure 1) consists of the following steps, with further details in Appendix A:

1. **Baseline prediction:** For each datapoint x in the suspect dataset $\mathcal{D}$, obtain the model's average log-likelihood on the consecutive tokens of x.
2. **In-context prediction:** Sample n additional examples x_1, ..., x_n from $\mathcal{D} \setminus \{x\}$, prepend them to x (creating a sequence $x_1|...|x_n|x$), and obtain the model's predictions on x in this new context.
3. **Score computation:** Compute the difference in confidence $\Delta(x) = \text{logprob}_{\text{in-context}}(x) - \text{logprob}_{\text{baseline}}(x)$.
4. **Aggregation:** Repeat the above for all $x \in \mathcal{D}$. The contamination score for the dataset is then

$$S_{\text{CoDeC}}(\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}[\Delta(x_i) < 0]$$

where $\mathbb{1}$ is the indicator function.

Properties. CoDeC has several theoretical properties that make it broadly applicable and easy to use. It outputs intuitive percentage scores, directly interpretable without model- or dataset-specific calibration. The method is parameter-free, avoiding the threshold tuning required by many membership inference approaches [17, 5, 24], and works with any dataset that can be represented as a set of text sequences. It is model-agnostic, requiring only gray-box access to token probabilities and two forward passes per sample. See Appendix A.3 for an extended discussion of properties of CoDeC.

Why Does CoDeC Work? The central idea of CoDeC is to detect whether a model has internalized the distinctive features of a dataset, since reliance on such features is a strong indicator of contamination. When evaluated with unseen datasets, in-context examples provide novel distributional cues that improve predictions. For seen datasets, they offer no advantage and may even lower confidence

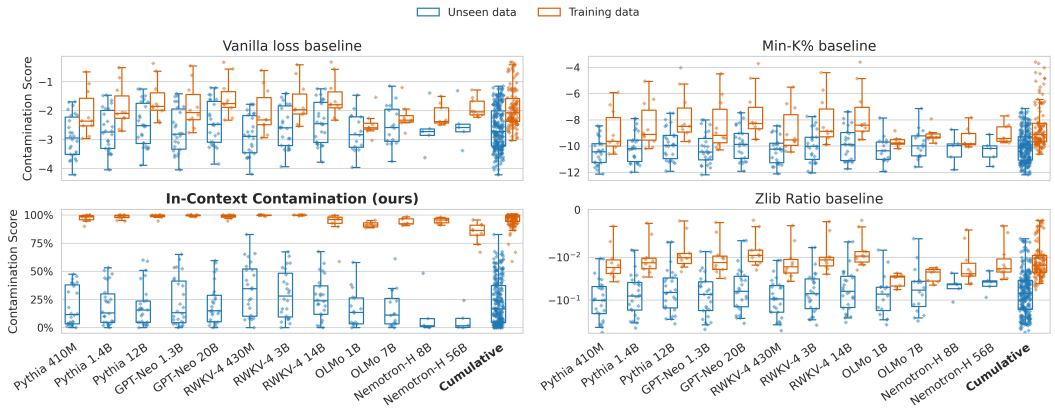


Figure 2: **CoDeC vs. baselines**; Contamination scores for training (orange) and unseen (blue) datasets. Each point is a model–dataset pair. CoDeC achieves the best separation, enabling consistent classification across models and datasets.

by disrupting memorized patterns. CoDeC captures the effect efficiently through in-context learning rather than costly retraining, directly measuring how much capacity remains for learning the target data. We refer to Appendix A.4 for an extended discussion.

3 Experiments

In this section, we demonstrate that CoDeC consistently distinguishes between seen and unseen data, is stable across evaluation settings, and yields interpretable scores for model auditing.

Models. We validate CoDeC on a diverse suite of LLMs with publicly available weights and training data, enabling reproducibility and ground-truth verification. Our evaluation includes models trained on different corpora and spanning varied architectures: Pythia, GPT-Neo, and RWKV-4, all trained on the Pile [10]; OLMo, trained on Dolma [20]; and Nemotron-H, trained on Nemotron-CC [21].

Model	CoDeC (ours)	Vanilla loss	Min-K%	Zlib
Pythia 410M	**100.0%**	75.0%	76.2%	92.3%
Pythia 1.4B	**100.0%**	77.3%	79.2%	91.5%
Pythia 12B	**100.0%**	76.9%	82.3%	92.3%
GPT-Neo 1.3B	**100.0%**	77.3%	79.2%	90.8%
GPT-Neo 20B	**100.0%**	76.9%	83.5%	92.7%
RWKV-4 430M	**100.0%**	75.4%	75.4%	92.3%
RWKV-4 3B	**100.0%**	76.5%	79.6%	91.9%
RWKV-4 14B	**99.6%**	77.3%	81.5%	92.7%
OLMo 1B	**100.0%**	64.8%	71.9%	80.5%
OLMo 7B	**100.0%**	65.6%	72.7%	78.1%
Nemotron-H 56B	**100.0%**	82.2%	86.7%	92.0%
Nemotron-H 8B	**100.0%**	80.0%	73.3%	88.0%
Cumulative	**99.9%**	74.9%	77.5%	89.2%

Table 1: AUC[1] scores for separating seen vs. unseen datasets (Figure 2), computed per dataset.

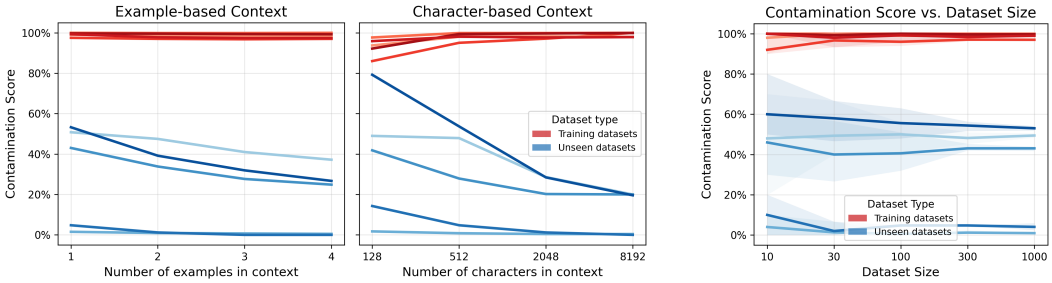
Datasets. For each model, we create a test bed consisting of (1) data known to be in its training set (e.g., subsets of the Pile [10]) and (2) unseen data published after the model's training cutoff. The unseen data includes recent benchmarks, news, websites, etc. Datasets details can be found in Appendix C.

Baselines. We compare CoDeC against three common contamination detection methods: **Vanilla Loss** [9], scoring based on model loss; **Min-K%** [24], focusing on the lowest-probability tokens; and **Zlib Ratio** [5], which normalizes perplexity by sample entropy.

3.1 Main Validation

CoDeC cleanly separates seen from unseen data (see Figure 2), achieving **AUC of 99.9%** across all evaluated models (see Table 1). Baselines show substantial overlap between seen and unseen scores, limiting their utility for drawing reliable conclusions. In contrast, CoDeC provides clear separation, enabling consistent reference across models.

[1]While AUC, a parameter-free metric commonly used for evaluating MIA methods [17], is usually computed over individual samples, here it is computed over dataset-level scores. Details are provided in Appendix C.3.



(a) Impact of context size. (b) Impact of number of examples.

Figure 3: Ablation studies of CoDeC on the Pythia 1.4B model using 5 training and 5 unseen datasets. Shaded regions show the range between the minimum and maximum scores across 5 runs.

3.2 Ablations

Context Size. While our experiments use the simplest form of CoDeC with a single in-context sample, the method can be easily extended to include more context. Larger contexts yield clearer separation between seen and unseen data but also increase computational cost (see Figure 3a). In practice, a single in-context sample already provides strong signal while keeping inference efficient. However, adding more samples can further enhance performance if resources permit.

Target Dataset Size. A key practical factor is how many examples are needed for reliable contamination scores. CoDeC proves highly sample-efficient: around 100 examples already give stable estimates with low variance, making the method feasible even on small benchmarks. With 1,000 examples, the variance falls below 1% (see Figure 3b).

3.3 Interpreting CoDeC Scores

A core design goal of CoDeC was interpretability. The final score represents the percentage of data points exhibiting memorization, a metric independent of model-specific properties like output scaling. Based on our findings, scores $>80\%$ indicate strong contamination, $<60\%$ suggest no evidence of contamination. High values, even below 80%, may indicate partial overlap with related data, or training on strongly related distributions.

Absolute contamination scores can be sometimes influenced by dataset properties such as diversity, so they can be best interpreted in comparison across multiple models. Significant outliers among CoDeC scores are also indicating likely contamination.

3.4 Broader Application of CoDeC

In previous sections, we validated CoDeC on models with known training data. Since access to training data is not required, we also applied it to 30 popular models on standard benchmarks. Most showed only mild contamination, but some exhibited high scores on multiple benchmarks, indicating substantial training overlap. Several widely used datasets, such as Math-500, appear heavily saturated. Full results are provided in Appendix D.2.

4 Conclusions

We introduced Contamination Detection via Context (CoDeC), a simple yet very effective method for detecting and quantifying training data contamination in large language models. By measuring how in-context examples from the same dataset affect model predictions, CoDeC distinguishes between datasets the model has memorized and those it has not. Our experiments show that CoDeC produces clear, interpretable contamination scores on a wide range of models and datasets, exposing strong evidence of memorization and overfitting even in open-weight models with undisclosed training data.

Compared to traditional membership inference approaches, CoDeC requires no external references, dataset-specific tuning, or costly retraining, making it practical for large-scale, real-world evaluations. Its percentage-based scores are easy to interpret and integrate seamlessly into benchmark reporting.

References

[1] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[2] Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. If you use this software, please cite it using these metadata.

[3] Aaron Blakeman, Aarti Basant, Abhinav Khattar, Adithya Renduchintala, Akhiad Bercovich, Aleksander Ficek, Alexis Bjorlin, Ali Taghibakhshi, Amala Sanjay Deshmukh, Ameya Sunil Mahabaleshwarkar, et al. Nemotron-h: A family of accurate and efficient hybrid mamba-transformer models. *arXiv preprint arXiv:2504.03624*, 2025.

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[5] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *29th USENIX Security Symposium*, 2022.

[6] Amadou Djiré et al. Pearl: Perturbation analysis for revealing memorization in large language models. *arXiv preprint arXiv:2505.03019*, 2025.

[7] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08710*, 2021.

[8] Yihong Dong, Xue Jiang, Huanyu Liu, Zhi Jin, Bin Gu, Mengfei Yang, and Ge Li. Generalization or memorization: Data contamination and trustworthy evaluation for large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 12039–12050, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[9] Yujuan Fu, Ozlem Uzuner, Meliha Yetisgen, and Fei Xia. Does data contamination detection work (well) for llms? a survey and evaluation on detection assumptions. *NAACL*, 2024.

[10] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[11] Shayan Golchin and Mihai Surdeanu. Data contamination detection through guided prompts. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

[12] Shayan Golchin and Mihai Surdeanu. Data contamination quiz: Detecting training overlap in language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[13] Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024.

[14] Jie Huang, Mo Yu, Hong Sun, Linfeng Qiu, Yu Su, Yue Wang, and Tengyu Ma. Probing trustworthiness of language models via perplexity-based measures. In *Findings of the Association for Computational Linguistics: ACL 2023*, 2023.

[15] HuggingFaceH4 Team. Aime 2024 dataset, 2024. Problems from the American Invitational Mathematics Examination 2024, accessible via Hugging Face :contentReference[oaicite:3]index=3.

[16] Zihan Li, Weizhi Yang, Zexuan Chen, Tianyu Gao, and Danqi Chen. Emergent phenomena in in-context learning. *arXiv preprint arXiv:2301.00234*, 2023.

[17] Pratyush Maini et al. Llm dataset inference: Did you train on my dataset? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[18] opencompass Team. Aime 2025 dataset, 2025. Problems from the American Invitational Mathematics Examination 2025 (AIME 2025-I & II), accessible via Hugging Face :contentReference[oaicite:2]index=2.

[19] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[20] Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, et al. Dolma: An open corpus of three trillion tokens for language model pretraining research. *arXiv preprint arXiv:2402.00159*, 2024.

[21] Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. *arXiv preprint arXiv:2412.02595*, 2024.

[22] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations (ICLR)*, 2022.

[23] Jingyang Zhang, Jingwei Sun, Eric C. Yeats, Yang Ouyang, Martin Kuo, Jianyi Zhang, Hao Yang, and Hai Helen Li. Min-k%++: Improved baseline for detecting pre-training data from large language models. *CoRR*, abs/2404.02936, 2024.

[24] Zizhao Zhang, Han Wu, Tongzheng Wang, Guangyu Lin, Yueqi Zhang, Tingting Wang, and Xiangyu He. Understanding and mitigating the uncertainty in deep neural networks: Min-k *arXiv preprint arXiv:2012.07805*, 2021.

[25] Chunting Zhou, Yao Zhao, Xinyu Chen, and Mohit Bansal. Lighteval: Accurate llm evaluation without ground truth labels. *arXiv preprint arXiv:2305.18290*, 2023.

A Detailed explanation of the Contamination Detection via Context approach

A.1 Key idea

Consider the following illustrative example. If a mathematician trains for the International Mathematical Olympiad (IMO) competition by solving all available IMO problems, their knowledge becomes highly contaminated with IMO-specific problem characteristics. Beyond the genuine mathematical knowledge acquired, exposure to the IMO problem distribution introduces several specific priors, such as: *all assumptions mentioned in the problem are necessary*, *the hypothesis requested to prove always holds*, *the problems are challenging yet formulated without advanced academic concepts*, etc. These properties are unique to IMO problems but not to general mathematical questions and texts. Therefore, leveraging such priors results in an unfair measure of one's general mathematical skills.

Now, suppose this mathematician is given free access to IMO problems during the competition. If they have already learned to solve those problems, such context is no longer useful for further learning. However, if the mathematician has learned from other sources and has never seen any IMO problem, they would benefit greatly from understanding the specifics of these problems, their structure, and implicit assumptions.

Following this example, Contamination Detection via Context (CoDeC) measures the contamination score by evaluating how access to the target dataset affects the model's predictions. For each datapoint in the dataset, CoDeC samples a few in-context examples from the same dataset and compare the average logits with and without this context.

If the model has not used the target dataset for training, it should benefit from the additional context, as these examples contain valuable information about the data distribution. Even if not directly related, they may share similarities in structure, style, vocabulary, topic, or other implicit common priors. Conversely, if the model was trained on that dataset, it already knows these priors, making the provided context less beneficial. Furthermore, if the model is overly confident due to memorized patterns, such as specific word sequences or frequent occurrences, introducing additional context (likely also memorized) should disrupt these patterns and negatively affect the confidence.

A.2 CoDeC pipeline

Following the key idea described above, the complete pipeline of CoDeC consist of 4 simple steps:

1. **Baseline prediction:** For each datapoint x in the suspect dataset $\mathcal{D}$, obtain the model's average log-likelihood on the consecutive tokens of x.
2. **In-context prediction:** Sample n additional examples x_1, ..., x_n from $\mathcal{D} \setminus \{x\}$, prepend them to x (creating a sequence $x_1|...|x_n|x$), and obtain the model's predictions on x in this new context. Focus solely on the probability of tokens in x, ignoring the context examples.
3. **Score computation:** Compute the difference in confidence $\Delta(x) = \text{logprob}_{\text{in-context}}(x) - \text{logprob}_{\text{baseline}}(x)$. Since the context is sampled randomly, the value of $\Delta(x)$ is subject to some variance. To achieve higher statistical significance, average the $\Delta(x)$ values over 5 seeds, though even a single seed provides meaningful scores.
4. **Aggregation:** Repeat the above for all $x \in \mathcal{D}$. The contamination score for the dataset is then
$$S_{\text{CoDeC}}(\mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[\Delta(x_i) < 0]$$
where $\mathbb{1}$ is the indicator function.

In summary, to compute CoDeC scores, we compare the model's confidence with and without additional context for each sample. The final score represents the percentage of samples for which the influence of additional context is negative.

A.3 Properties of CoDeC

A.3.1 Theoretical properties

CoDeC by its design has several desirable properties that make it useful in practice:

Score as a percentage. CoDeC returns a contamination score as a percentage of dataset elements indicating contamination, making it intuitively understandable. Unlike classical approaches that compute scores in open ranges requiring references and scale knowledge, percentage points are easily grasped without in-depth analysis. Even the intuitive notion of *probability of contamination*, while not fully grounded, can be useful here.

Works with any dataset. CoDeC can be computed for any set of strings, making it broadly applicable. In our evaluations, we used various data sources: standard training datasets (e.g., Wikipedia, Common Crawl), code (e.g., GitHub), QA benchmarks (e.g., MMLU, GPQA), math benchmarks (e.g., AIME, MATH-500), PDFs (e.g., ArXiv), books (e.g., Project Gutenberg), websites (e.g., global news, Amazon Reviews), files (e.g., Linux syslog), and more. While CoDeC is designed primarily for datasets, it is straightforward to transform a single text source into a dataset by splitting it into parts.

Works with any gray-box model. Any model that outputs logits for a given text (including most models available on HuggingFace) can be used to compute CoDeC. The method is independent of the model's training specifics and architecture. It can even be applied to fully black-box models by empirically estimating target token probabilities, though this increases score variance unless we assume multiple repetitive calls and model stochasticity.

Parameter-free. CoDeC does not require model-specific or dataset-specific parameters. Unlike classical MIA methods, which need a tuned classification threshold and deep model knowledge or external ground-truth data, CoDeC can be applied off-the-shelf to any model and dataset.

Computational efficiency. To compute CoDeC scores, the model runs twice with at most twice longer samples, introducing minimal computational overhead. Using as few as 1000 samples already yields precise scores (see Figure 3b), hence it can be applied even to large corpora. Additionally, in-context learning is much faster than methods requiring finetuning or shadow model training.

A.3.2 Empirical properties

Furthermore, during our experiments we observed the following empirical properties of CoDeC:

Robustness to text cropping. CoDeC scores remain stable when using partial training prompts. This property proves particularly valuable when evaluating single, long text sources (books, articles) that should be split into dataset components.

Sensitivity to formatting variations. Artificial formatting changes (added labels, modified whitespace) reduce contamination scores even for training data. This sensitivity is crucial for QA benchmark evaluation, where labels like *Question:*, *Answer:*, or instruction prompts are common. Incorrect labels (those not used during training) decrease scores, as expected: in-context samples provide formatting information that artificially increases prediction confidence. Furthermore, incorrect labels may break memorization patterns themselves. To ensure meaningful evaluation, we focus exclusively on text portions guaranteed to appear during training (e.g., question text only), independent of training-specific formatting.

Impact of data diversity. Training datasets consistently yield scores near 100% regardless of their properties. However, unseen dataset scores vary from 0% to 60%, depending on dataset characteristics. CoDeC assumes that additional context from the same distribution provides some additional information, e.g. text type, style, vocabulary, domain knowledge, etc. In large, diverse datasets with unrelated samples, minimal information sharing between datapoints occurs. Hence, additional context may not improve predictions and can act as random noise, potentially yielding CoDeC scores up to 60%, even for unseen data. Our experiments suggest that this effect may be mitigated using more context samples.

Larger models use less memorization. Our main evaluation (Figure 2) included models from 410M to 56B parameters, where training data could be verified. While results remain consistent across architectures and sizes, larger models within families show slightly lower CoDeC scores. This

trend intensifies at larger scales. For instance, Llama-Nemotron-Nano 4B exhibits high contamination across multiple benchmarks, while Llama-Nemotron-Ultra 253B maintains all contamination scores below 20%. Although these models share a family, their training processes and data compositions differ; however, these differences alone cannot explain the contamination disparities.

This size-dependent behavior aligns with expectations. CoDeC measures reliance on memorization instead of general knowledge. Larger models possess greater capacity for genuine knowledge acquisition, reducing overfitting to individual datapoints. Consider an illustrative example: when solving algebraic equations, children often rely on pattern memorization, while expert mathematicians apply fundamental skills even for previously encountered problems.

Adversarial cases. While CoDeC provides robust contamination scores for most datasets and models, adversarial constructions remain possible. For instance, a dataset containing 1,000 identical (or near-identical) samples produces low CoDeC scores regardless of training exposure. Constructing datasets with artificially high contamination scores is more challenging but possible through combining several unrelated, diverse data sources.

These edge cases do not compromise evaluations on standard benchmarks, as genuine datasets remain unaffected by such artificial constructions. Since evaluation data selection remains under user control, models cannot exploit these constructions to bypass detection.

CoDeC applies to any causal model by design. We evaluated over 40 models across various sizes and architectures. One notable exception emerged: GPT-OSS 20B consistently produces contamination scores exceeding 99% across all datasets. Investigation revealed heavy optimization for chat and reasoning tasks that impairs standard language sequence modeling. Even when provided identical text as context, confidence decreases due to persistent attempts to terminate sequences and transition to thinking or chat-like dialogue patterns. Addressing that issue remains for future work.

A.4 Why does CoDeC work?

The central idea behind CoDeC is to measure whether a model has internalized a dataset's specific priors, as their presence is a clear indicator of training data contamination. These priors are not limited to exact string memorization but cover a wide range of cues, such as stylistic patterns, characteristic vocabulary, or common structural templates. While a model may lack the capacity to memorize every single training example, the manifold of these general cues is significantly narrower and can be memorized more easily. CoDeC is designed to detect if the model leverages this learned manifold.

Our approach uses in-context learning as an efficient proxy for finetuning. Consider the standard finetuning process: a model performance improves significantly during the first epoch on a new dataset, with smaller gains in subsequent epochs. Similarly, if we could finetune a model on a target dataset for contamination detection, a contaminated model would improve much more slowly than a non-contaminated one because it has already learned the data distribution. However, performing actual finetuning for every evaluation is computationally prohibitive, especially for large models. CoDeC achieve a similar outcome by using in-context learning, which is a negligible cost. Hence, in essence, CoDeC measures the remaining learning capacity for the target dataset. If the model has already been trained on the data, it has little capacity left to learn, and its performance will not significantly improve when presented with in-context examples from that dataset.

This behavior can be also understood through the lens of the loss landscape. Contamination is often linked to overfitting [17], where the model settles into a sharp, narrow local minimum in the loss landscape for the training data. For unseen data, the loss landscape tends to be flatter. Our experiments suggest that in-context learning acts similarly to finetuning with a high learning rate. This makes the process highly sensitive to the local geometry of the loss manifold. For a contaminated sample, the model is already in a steep local minimum. Hence, taking even a single step is likely to exit this minimum, resulting in worse performance. Conversely, for an uncontaminated sample where the loss manifold is flat, the same step will likely move the model towards a region of higher confidence, improving performance at least slightly.

Finally, CoDeC relates to reference-based MIAs, which often use external models or datasets to calibrate the difficulty of a given sample. CoDeC shares a similar structure but introduces a crucial distinction: it relies on a self-reference approach. Instead of using external data for calibration, CoDeC uses the model own predictions as the baseline. This design choice makes our scores

independent of the availability of external resources and ensures that the calibration is based on specifically to the properties and knowledge of the model being investigated.

A.5 How to interpret CoDeC scores?

The CoDeC score is a measure of how much a model's predictions rely on memorized patterns rather than genuine reasoning. Importantly, it does not indicate strict membership of a dataset in the training corpus. Instead, it reflects whether the outputs are predicted based on memorized internal distribution priors – something that can happen if the model was trained on identically distributed or closely related data (e.g. artificially generated). From another perspective, a lower contamination score indicates greater remaining model capacity to learn from the target dataset.

In practice, our experiments reveal two complementary ways to interpret CoDeC scores: absolute evaluation and reference-based comparison.

A.5.1 Absolute Score Interpretation

Because CoDeC scores naturally fall between 0% and 100%, they can be compared across datasets and models without model-specific scaling or parameter tuning. Empirically, we found a consistent pattern:

- Scores above 80% were measured for nearly all training datasets we used for experiments, indicating strong contamination evidence.
- Scores below 60% were measured for nearly all unseen datasets. In general, they show no evidence of contamination – the model is likely reasoning based on general knowledge rather than memorization.
- Scores in the 60%–80% range are ambiguous: they may be due to partial contamination, training on related distributions, or simply higher model capacity.

Thus, while an absolute threshold (>80%) is a strong indicator, values in the intermediate range require more nuanced analysis.

A.6 Reference Score Interpretation

Absolute scores alone only partially capture variations in dataset properties like diversity. For example, even a non-contaminated model might score relatively high on a broad, highly diverse dataset. To adjust for such effects, the scores should be compared across multiple models on the same dataset.

The most reliable approach is to include at least one model that is known to be non-contaminated with the target dataset (e.g., older model like Pythia). If all models show similar contamination scores, this suggests no substantial memorization specific to any model, but rather a dataset-specific level of the score. However, if a score of the model stands out as an outlier compared to reference models, this strongly suggests higher reliance on memorization.

Choosing reference models of similar size and architecture further helps ensure that differences in scores point to contamination rather than general learning capacity.

A.6.1 Best Practices

For robust contamination assessment, we suggest combining absolute thresholds with reference-based comparisons. High absolute scores (>80%) should be treated as contamination red flags, but significant deviations from other models' CoDeC scores can be equally alarming. By viewing CoDeC scores not as binary labels but as indicators of memorization intensity, the results serve as a useful indicator both for ensuring a fair model comparison on benchmarks, and reliable model quality assessment during training.

B Related Works

Detecting contamination in large language models (LLMs) lies at the intersection of several research threads: measuring memorization through log-probabilities, understanding in-context learning behavior, and developing scalable automated evaluation methods. We review these areas and situate ou method Contamination Detection via Context (CoDeC) within this landscape.

Contamination detection. Contamination, where evaluation data leaks into pretraining corpora, undermines benchmark validity [7]. Existing methods fall into two categories: *explicit* detection (e.g., string overlap checks or dataset provenance audits) and *implicit* detection, which infers contamination from model behavior without access to training data. Implicit probes include entropy- and confidence-based signals [8], guided prompts that measure performance shifts when datasets are referenced [11], and quiz-style tests such as the Data Contamination Quiz (DCQ) [12]. Dataset-level inference extends this line by aggregating weak membership signals across many samples to statistically distinguish train vs. held-out slices of large corpora [17]. CoDeC belongs to this implicit, dataset-level family, offering a simple signal based on log-probability shifts under in-context prompting.

Log-probabilities as a signal of memorization. LLMs generally assign higher likelihoods (lower loss) to training data than to unseen text, a property exploited in membership inference and memorization studies [5]. Common gray-box baselines include *Min-K%*, which averages the least likely tokens to avoid trivial predictability [24], and compression-based scores such as the *Zlib ratio* [5]. While informative, these per-sample signals can be brittle, collapsing under IID evaluation or distributional confounders. CoDeC mitigates this by aggregating dataset-level log-probability *shifts* from in-context prompting, yielding a more stable and interpretable measure.

In-context learning. CoDeC enables models to adapt at inference time from a few examples [4]. Analyses interpret ICL as approximate Bayesian inference [22], with gains when examples are aligned with the evaluation distribution. Conversely, for memorized data, added context can disrupt stored patterns and reduce confidence [16]. CoDeC operationalizes this asymmetry: unseen data typically benefits from in-context examples, while memorized data does not, producing a behavioral signal of contamination.

Automated evaluation. A broader literature develops gray-box evaluation tools that operate without training data access. These include reference-free evaluation methods such as LightEval [25], entropy- and perplexity-based probes [14], and perturbation-driven memorization tests like PEARL [6]. CoDeC complements this space with a lightweight, interpretable proxy for contamination that requires only two forward passes per sample and directly links model confidence shifts to dataset-level memorization.

C Evaluation setup

To make the main evaluation meaningful, we have to use models for which we know the training data, and prepare additional datasets that were certainly not used for training. Following standard practice, we ensured this exclusion by using data published only after the release of the training datasets, eliminating the risk of data leakage. In each case, we selected a broad range of data types, sources, levels of diversity, and other characteristics to ensure comprehensive coverage of possible data properties in evaluation.

For each dataset, we selected 1 000 random samples for evaluation. If the data source was a continuous stream of text (e.g. book or an article), we split it into 600-characters chunks and treated as a dataset.

Since the selection of both training and unseen datasets is inevitably somewhat arbitrary, we emphasize that the results in this paper reflect all evaluations we have conducted, including internal experiments. Given that the pipeline is simple and lightweight, **we strongly encourage readers to run it themselves and confirm the reliability of CoDeC on datasets of their choice**.

Because most LLMs available today do not disclose their training data, our model choice is necessarily limited. Nevertheless, we selected the following families with full access to their training datasets:

- **Pythia** [1], trained on the Pile dataset [10].
- **GPT-Neo** [2], trained on the Pile dataset.
- **RWKV-4** [19], trained on the Pile dataset.
- **OLMo** [13], trained on the Dolma dataset [20].
- **Nemotron-H** [3], trained on the Nemotron-CC dataset [21].

C.1 Training datasets

We used the following sets as training data examples:

The Pile

- HackerNews
- Wikipedia
- GitHub
- ArXiv
- DM Mathematics
- Pile CommonCrawl
- PubMed Central
- Full Pile
- Wikipedia Music (a low-diversity subset of Wikipedia containing only music-related articles)
- GitHub Licenses (a low-diversity subset of GitHub containing only license comments)

Since the Pile dataset is not available for direct download, we used the samples provided in the `iamgroot42/mimir` dataset.

Dolma

- C4
- Wikipedia
- Pes2o v2
- Reddit v5
- Stack v4
- CommonCrawl head
- CommonCrawl middle
- CommonCrawl tail

Nemotron-CC

- Wikipedia
- StackExchange
- ArXiv
- CommonCrawl 2024 (high quality)
- CommonCrawl 2020 (high quality)
- CommonCrawl 2020 (low quality)
- CommonCrawl 2019 (medium quality)
- CommonCrawl 2016 (medium quality)
- CommonCrawl 2014 (low quality)
- CommonCrawl 2013 (high quality)

C.2 Unseen datasets

For unseen datasets, we used the following sources:

Popular benchmarks. We evaluated the models on the following benchmarks:

- For models trained on the Pile: gsm8k, GPQA Diamond, IFEval, HumanEval, FRAMES, AIME 2024, AIME 2025, LiveCodeBench v1, LiveCodeBench v5, BFCL v3, BBQ, RewardBench v1, RewardBench v2, and MATH 500. All these benchmarks were released after the release of the Pile.
- For OLMo models: FRAMES, AIME 2024, AIME 2025, LiveCodeBench v5, BFCL v3, RewardBench v1, and RewardBench v2.
- For Nemotron-H, we used only AIME 2025.

Project Gutenberg. We used three books added to Project Gutenberg after April 2025:

- *Colonial Memories*
- *Jibby Jones : A story of Mississippi River adventure for boys*
- *The Corbin necklace*

Datasets from HuggingFace.

- `NickyNicky/global-news-dataset`
- `McAuley-Lab/Amazon-Reviews-2023`

A recent website.

- An article with Ukrainian conflict updates: `https://www.understandingwar.org/backgrounder/ukraine-conflict-updates`

Self-created data.

- A document describing this project.
- Linux syslog file from our computer.
- Internal slack channel log.

C.3 AUC comparison

The Area Under the Receiver Operating Characteristic Curve (AUC) is a standard metric for evaluating binary classifiers. It measures the probability that a randomly chosen positive example is ranked higher than a randomly chosen negative example, making it threshold-independent and robust to class imbalance. If S^+ is the set of scores for positive examples and S^- for negative examples, then:

$$\text{AUC} = \frac{1}{|S^+| \cdot |S^-|} \sum_{s^+ \in S^+} \sum_{s^- \in S^-} \mathbb{1}(s^+ > s^-) + 0.5 \cdot \mathbb{1}(s^+ = s^-)$$

In the context of Membership Inference Attacks (MIAs), AUC is widely used to quantify the classifier's ability to distinguish between training samples (positives) and unseen samples (negatives).

A key difference between our setting and typical MIA evaluations is granularity. In most prior work, contamination scores are computed per sample, and the AUC reflects the quality of sample-level classification. In contrast, our method focuses on the dataset level: we compute a single contamination score per dataset and evaluate the model's ability to classify entire datasets as seen or unseen. This approach better reflects the intended use of CoDeC, which is designed to provide interpretable, aggregate measures of contamination for benchmarks and corpora rather than individual samples.

As shown in Figure 2, CoDeC achieves near-perfect separation between seen and unseen datasets, leading to dataset-level AUC scores close to 100%. This demonstrates that CoDeC is highly effective at capturing training data contamination in a way that aligns with MIA evaluation traditions while providing a metric more suited to benchmark-level analysis.

D Additional Experiments

D.1 Application: Release-Aligned Validation on Annual Benchmarks

We further validate CoDeC in a setting where no training corpora are available. Many benchmarks are released annually; if CoDeC captures reliance on memorized priors, then models whose training cutoff *predates* a given benchmark year should exhibit higher CoDeC scores on those pre-release years than on post-release years.

For each model we define a pre/post partition of benchmark years based on public release timelines (see Table 2). For each (model, year) pair we compute the dataset-level CoDeC score and summarize per model across years. Across six models spanning three families, we observe a consistent **Pre** $>$ **Post** pattern (Figure 4): median CoDeC drops by 5–15 percentage points after the benchmark's release, with per-model Wilcoxon tests significant after FDR correction.

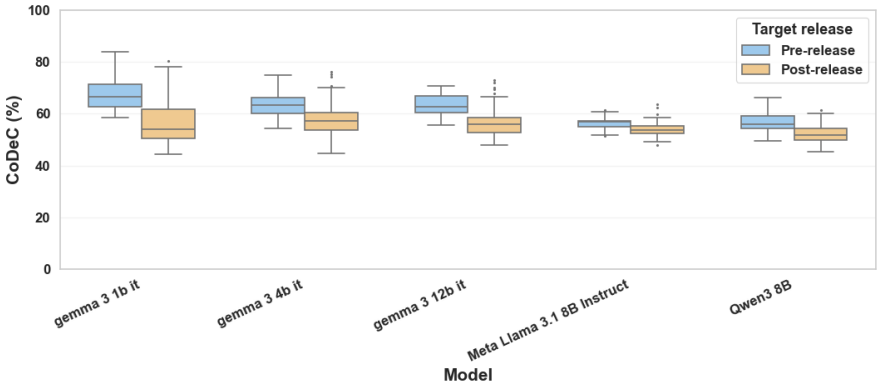


Figure 4: Release-aligned CoDeC scores (%) by model. For each model, benchmark years are grouped into **Pre-release** (blue) vs. **Post-release** (orange) according to the model's training cutoff (see Table 2). Boxes show the distribution of S_{CoDeC} across years; medians decrease from pre to post for all models, consistent with CoDeC detecting reduced reliance on memorized priors once the benchmark is out-of-distribution relative to the training window.

Table 2: Pre- and post-release IMO years per model (see [15, 18]), used for the splits in Figure 4.

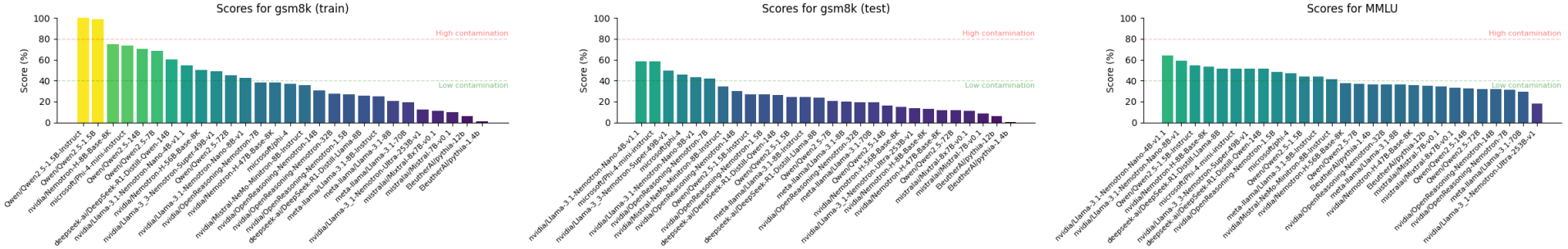
Model	**Pre-release years**	**Post-release years**
Meta Llama 3.1 8B Instruct	≤ 2022	2023, 2024, 2025
Llama 3.2 3B instruct	≤ 2022	2023, 2024, 2025
Gemma 3 1B it	≤ 2023	2024, 2025
Gemma 3 4B it	≤ 2023	2024, 2025
Gemma 3 12B it	≤ 2023	2024, 2025
Qwen3 8B	≤ 2023	2024, 2025

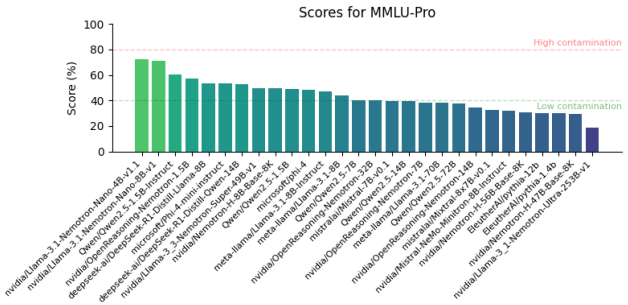
D.2 Broader Evaluation

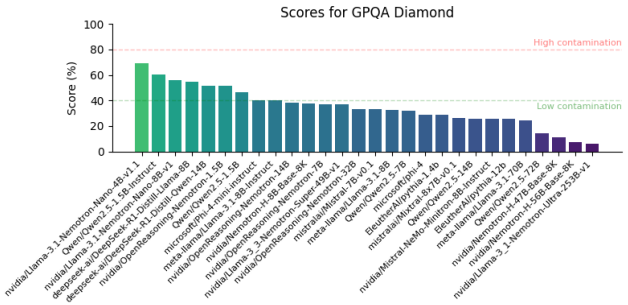
Contamination Detection via Context – Broad LLM Evaluation

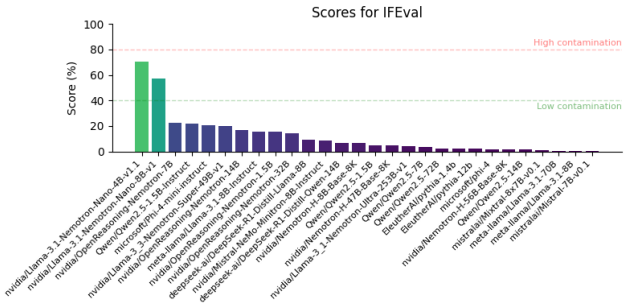
Model	gsm8k (train)	gsm8k (test)	MMLU	MMLU-Pro	GPQA Diamond	IFEval	HumanEval	FRAMES	hellaswag (train)	hellaswag (test)	AIME 2024	AIME 2025	LiveCodeBench v1	LiveCodeBench v5	BFCL v3	BBQ	RewardBench v1	RewardBench v2	MATH 500 (problem)	MATH 500 (solution)
EleutherAI/pythia-1.4b	1	0	36	29	28	2	16	8	3	3	2	3	7	9	24	0	21	36	4	12
EleutherAI/pythia-12b	5	6	35	30	25	2	28	9	3	11	0	3	31	30	15	4	19	32	5	14
Qwen/Qwen2.5-1.5B	99	24	44	49	46	5	60	16	6	5	60	10	23	13	25	8	34	48	57	20
Qwen/Qwen2.5-1.5B-Instruct	100	27	54	60	60	21	54	17	6	4	57	27	43	24	40	12	38	63	60	16
Qwen/Qwen2.5-14B	70	16	32	39	25	1	54	13	3	3	64	10	13	8	17	4	30	39	71	12
Qwen/Qwen2.5-72B	45	11	32	37	14	2	48	11	3	3	61	6	5	6	15	4	29	36	68	8
Qwen/Qwen2.5-7B	68	20	36	40	31	3	48	15	3	7	62	6	10	7	19	8	30	40	70	14
deepseek-ai/DeepSeek-R1-Distill-Llama-8B	25	23	51	53	54	8	27	25	7	8	16	20	57	58	37	16	38	58	31	11
deepseek-ai/DeepSeek-R1-Distill-Qwen-14B	60	26	51	52	51	7	32	12	9	7	11	20	58	52	45	8	33	52	36	7
meta-llama/Llama-3.1-70B	19	19	29	38	24	0	63	6	4	3	11	13	15	15	6	0	19	23	17	17
meta-llama/Llama-3.1-8B	20	19	36	43	32	0	39	6	4	3	41	62	18	16	6	8	20	28	27	37
meta-llama/Llama-3.1-8B-Instruct	24	24	43	47	39	15	17	8	3	3	41	41	53	-	16	8	29	38	-	-
microsoft/Phi-4-mini-instruct	73	58	51	53	40	20	26	20	73	72	52	55	30	28	39	12	29	50	61	44
microsoft/phi-4	37	46	46	48	28	1	12	19	82	70	75	51	11	7	27	8	31	29	74	39
mistralai/Mistral-7B-v0.1	9	8	34	39	33	0	18	6	2	1	7	6	21	20	6	8	25	36	13	5
mistralai/Mixtral-8x7B-v0.1	11	11	33	32	26	0	25	7	2	2	30	24	15	14	5	4	22	30	17	7
nvidia/Llama-3.1-Nemotron-Nano-4B-v1.1	54	58	64	72	69	70	27	42	11	9	71	72	54	56	61	20	-	-	74	28
nvidia/Llama-3.1-Nemotron-Nano-8B-v1	42	43	59	71	56	57	32	37	12	10	66	62	38	35	46	37	76	58	61	21
nvidia/Llama-3_1-Nemotron-Ultra-253B-v1	12	13	18	19	6	3	1	5	1	1	0	0	2	2	12	8	17	22	-	-
nvidia/Llama-3_3-Nemotron-Super-49B-v1	49	49	51	49	37	19	9	10	8	6	32	34	39	35	38	12	35	65	30	19
nvidia/Mistral-NeMo-Minitron-8B-Instruct	36	34	41	31	25	8	7	10	2	4	35	34	6	7	15	4	19	42	41	45
nvidia/Nemotron-H-47B-Base-8K	38	12	36	29	11	4	4	5	3	2	26	3	2	-	8	0	22	24	-	-
nvidia/Nemotron-H-56B-Base-8K	50	15	37	30	7	1	8	5	3	2	20	10	1	-	8	0	21	20	-	-
nvidia/Nemotron-H-8B-Base-8K	75	13	53	49	37	6	21	8	3	3	38	31	35	35	19	8	29	39	-	-
nvidia/OpenReasoning-Nemotron-1.5B	27	26	48	56	51	15	58	23	7	8	35	51	49	52	37	16	53	56	-	-
nvidia/OpenReasoning-Nemotron-14B	30	29	32	34	38	16	14	20	5	5	17	20	10	11	20	12	23	27	-	-
nvidia/OpenReasoning-Nemotron-32B	27	19	36	40	33	14	40	21	12	11	17	13	56	63	24	12	36	38	-	-
nvidia/OpenReasoning-Nemotron-7B	38	42	31	38	37	22	45	20	3	4	16	13	70	70	24	25	29	34	-	-

Figure 5: CoDeC scores for widely used models across a range of popular benchmarks. Scores are expressed as percentages. Where available, both train and test sets are included in the table.

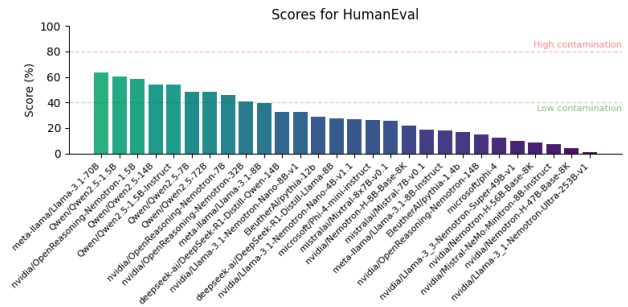


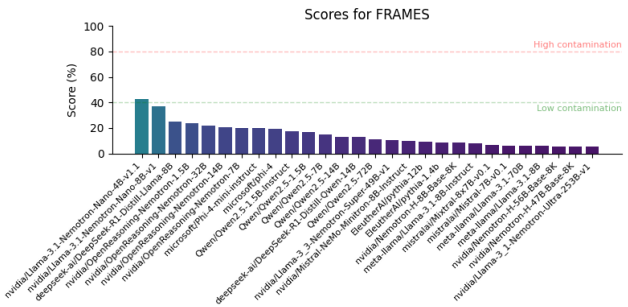


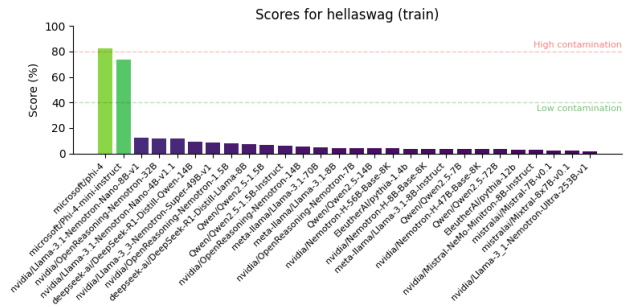




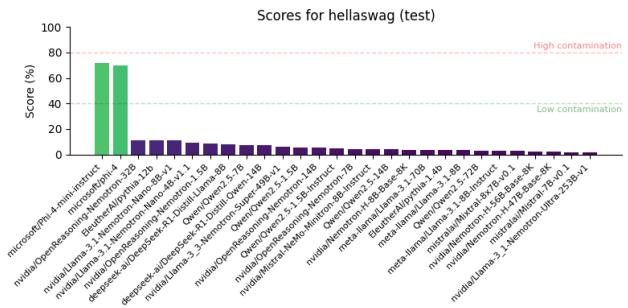
536

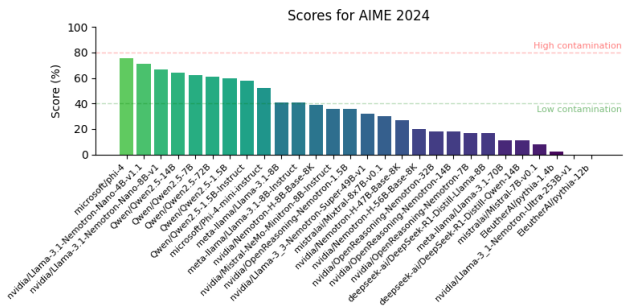


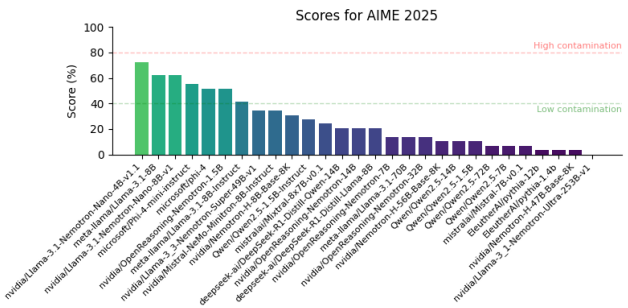




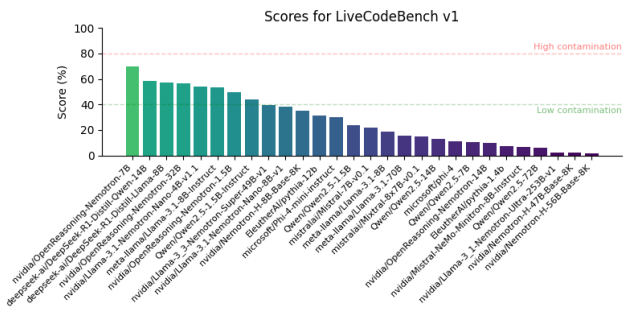
537

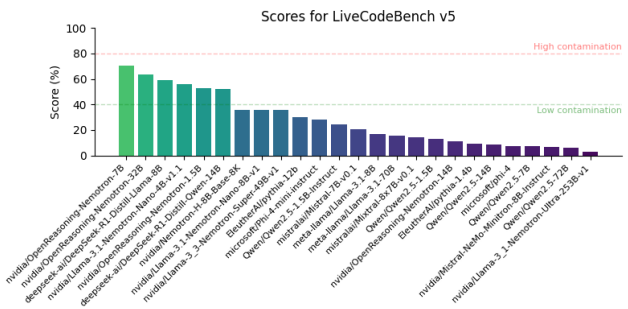


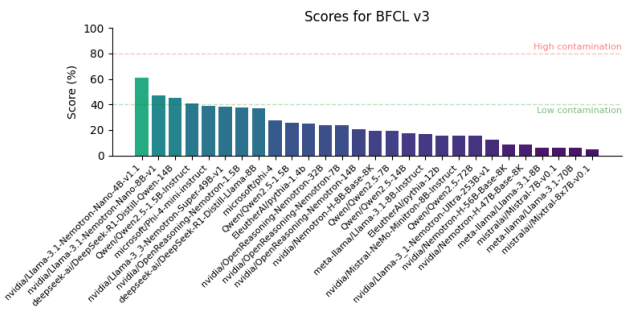




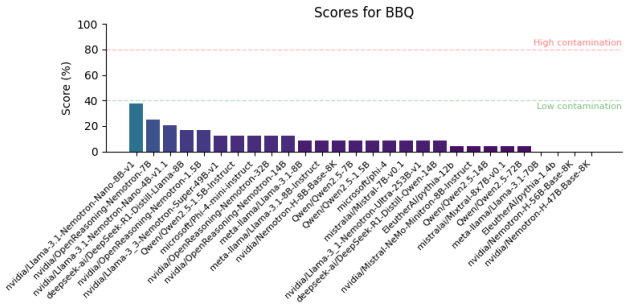
538



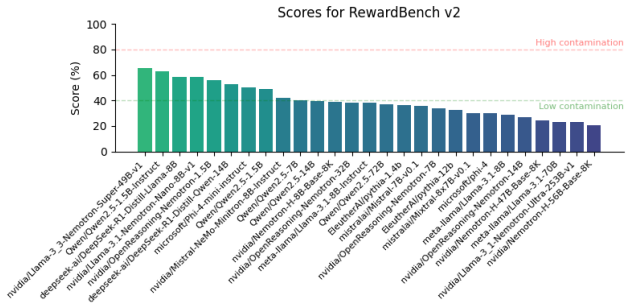


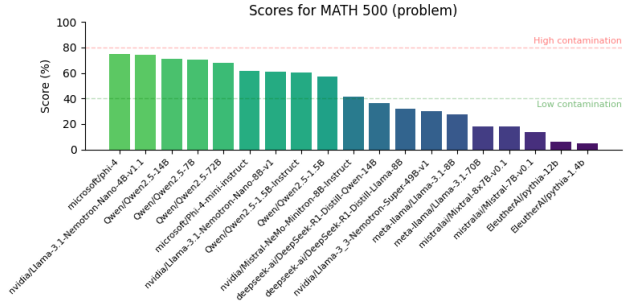


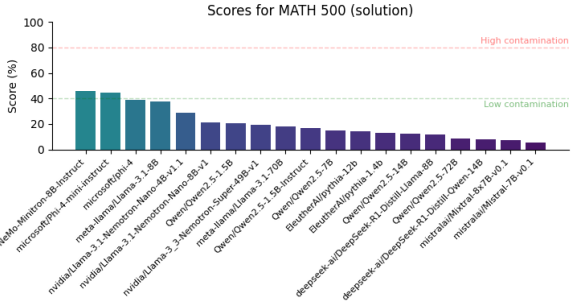
539


Scores for BBQ
Score (%)
High contamination
Low contamination
nvidia/Llama-3.1-Nemotron-Nano-8B-v1
nvidia/OpenReasoning-Nemotron-7B
nvidia/Llama-3.1-Nemotron-Nano-4B-v1.1
deepseek-ai/DeepSeek-R1-Distill-Llama-8B
nvidia/Llama-3.3-Nemotron-Super-49B-v1
Qwen/Qwen2.5-1.5B-Instruct
microsoft/Phi-4-mini-instruct
nvidia/OpenReasoning-Nemotron-32B
meta-llama/Llama-3.1-8B
nvidia/Nemotron-H-8B-Base-8K
Qwen/Qwen2.5-7B
mistralai/Mistral-7B-v0.1
nvidia/Llama-3.1-Nemotron-Ultra-253B-v1
deepseek-ai/DeepSeek-R1-Distill-Qwen-14B
EleutherAI/pythia-12b
nvidia/Mistral-NeMo-Minitron-8B-Instruct
Qwen/Qwen2.5-14B
meta-llama/Llama-3.1-70B
Qwen/Qwen2.5-72B
EleutherAI/pythia-1.4b
nvidia/Nemotron-H-56B-Base-8K
nvidia/Nemotron-H-47B-Base-8K


Scores for RewardBench v1
Score (%)
High contamination
Low contamination
nvidia/Llama-3.1-Nemotron-Nano-8B-v1
nvidia/OpenReasoning-Nemotron-1.5B
deepseek-ai/DeepSeek-R1-Distill-Llama-8B
Qwen/Qwen2.5-1.5B-Instruct
nvidia/OpenReasoning-Nemotron-32B
deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
nvidia/Llama-3.3-Nemotron-Super-49B-v1
Qwen/Qwen2.5-1.5B
deepseek-ai/DeepSeek-R1-Distill-Qwen-14B
microsoft/phi-4
Qwen/Qwen2.5-7B
microsoft/Phi-4-mini-instruct
nvidia/OpenReasoning-Nemotron-7B
Qwen/Qwen2.5-14B
meta-llama/Llama-3.1-8B-Instruct
nvidia/Nemotron-H-8B-Base-8K
mistralai/Mistral-7B-v0.1
Qwen/Qwen2.5-72B
nvidia/OpenReasoning-Nemotron-14B
nvidia/Nemotron-H-47B-Base-8K
mistralai/Mistral-8x7B-v0.1
nvidia/Nemotron-H-56B-Base-8K
EleutherAI/pythia-1.4b
meta-llama/Llama-3.1-8B
EleutherAI/pythia-12b
nvidia/Mistral-NeMo-Minitron-8B-Instruct
meta-llama/Llama-3.1-70B
nvidia/Llama-3.1-Nemotron-Ultra-253B-v1


Scores for RewardBench v2
Score (%)
High contamination
Low contamination
nvidia/Llama-3.3-Nemotron-Super-49B-v1
Qwen/Qwen2.5-1.5B-Instruct
deepseek-ai/DeepSeek-R1-Distill-Llama-8B
nvidia/Llama-3.1-Nemotron-Nano-8B-v1
deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
nvidia/OpenReasoning-Nemotron-1.5B
deepseek-ai/DeepSeek-R1-Distill-Qwen-14B
microsoft/Phi-4-mini-instruct
Qwen/Qwen2.5-1.5B
nvidia/Mistral-NeMo-Minitron-8B-Instruct
Qwen/Qwen2.5-7B
Qwen/Qwen2.5-14B
nvidia/Nemotron-H-8B-Base-8K
nvidia/OpenReasoning-Nemotron-32B
Qwen/Qwen2.5-72B
EleutherAI/pythia-1.4b
mistralai/Mistral-7B-v0.1
EleutherAI/pythia-12b
nvidia/OpenReasoning-Nemotron-7B
microsoft/phi-4
mistralai/Mistral-8x7B-v0.1
meta-llama/Llama-3.1-8B
nvidia/OpenReasoning-Nemotron-14B
nvidia/Nemotron-H-47B-Base-8K
meta-llama/Llama-3.1-70B
nvidia/Llama-3.1-Nemotron-Ultra-253B-v1
nvidia/Nemotron-H-56B-Base-8K


Scores for MATH 500 (problem)
Score (%)
High contamination
Low contamination
microsoft/phi-4
nvidia/Llama-3.1-Nemotron-Nano-4B-v1.1
Qwen/Qwen2.5-14B
Qwen/Qwen2.5-7B
Qwen/Qwen2.5-72B
microsoft/Phi-4-mini-instruct
nvidia/Llama-3.1-Nemotron-Nano-8B-v1
Qwen/Qwen2.5-1.5B-Instruct
Qwen/Qwen2.5-1.5B
nvidia/Mistral-NeMo-Minitron-8B-Instruct
deepseek-ai/DeepSeek-R1-Distill-Qwen-14B
deepseek-ai/DeepSeek-R1-Distill-Llama-8B
nvidia/Llama-3.3-Nemotron-Super-49B-v1
meta-llama/Llama-3.1-8B
meta-llama/Llama-3.1-70B
mistralai/Mistral-8x7B-v0.1
mistralai/Mistral-7B-v0.1
EleutherAI/pythia-12b
EleutherAI/pythia-1.4b


Scores for MATH 500 (solution)
Score (%)
High contamination
Low contamination
nvidia/Mistral-NeMo-Minitron-8B-Instruct
microsoft/Phi-4-mini-instruct
microsoft/phi-4
meta-llama/Llama-3.1-8B
nvidia/Llama-3.1-Nemotron-Nano-4B-v1.1
nvidia/Llama-3.1-Nemotron-Nano-8B-v1
Qwen/Qwen2.5-1.5B
nvidia/Llama-3_3-Nemotron-Super-49B-v1
meta-llama/Llama-3.1-70B
Qwen/Qwen2.5-1.5B-Instruct
Qwen/Qwen2.5-7B
EleutherAI/pythia-12b
EleutherAI/pythia-1.4b
Qwen/Qwen2.5-14B
deepseek-ai/DeepSeek-R1-Distill-Llama-8B
Qwen/Qwen2.5-72B
deepseek-ai/DeepSeek-R1-Distill-Qwen-14B
deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
mistralai/Mistral-8x7B-v0.1
mistralai/Mistral-7B-v0.1