

Document Version

Final published version

Licence

CC BY

Citation (APA)

Katzy, J., Al-Kaswan, A., Popescu, R. M., & Yang, Z. (2026). The Poisoned Chalice of LLM Evaluation Report. In S. H. Tan, & F. Khomh (Eds.), *FSE Companion 2026 - Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering* (pp. 1452-1455). Association for Computing Machinery (ACM).
<https://doi.org/10.1145/3803437.3807733>

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

In case the licence states "Dutch Copyright Act (Article 25fa)", this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

The Poisoned Chalice of LLM Evaluation Report

Jonathan Katzy*
J.B.Katzy@TUDelft.nl
Delft University of Technology
Delft, The Netherlands

Razvan Mihai Popescu
R.M.Popescu@TUDelft.nl
Delft University of Technology
Delft, The Netherlands

Ali Al-Kaswan
A.Al-Kaswan@TUDelft.nl
Delft University of Technology
Delft, The Netherlands

Zhou Yang
ZY25@UAlberta.ca
University of Alberta,
Alberta Machine Intelligence Institute
Alberta, Canada

Abstract

Large language models are increasingly used to evaluate and support software engineering tasks, yet the validity of these evaluations is often undermined by uncertainty about whether benchmark instances were seen during pretraining. This can lead to data contamination, which may inflate performance and result in misleading conclusions about model capability. Despite this, the training corpora of many modern models are only partially disclosed, making direct decontamination infeasible. This creates a need for practical methods that can detect a large language models' prior exposure to training data without access to the full training corpus.

To address this challenge, we organize the first Poisoned Chalice of LLM Evaluation Competition, co-located with the FSE-AIWare 2026 Competition Track. The competition frames contamination detection as a white-box membership inference task on source code and provides participants with curated datasets, target models, baseline attacks, and a final evaluation on a held-out model and dataset. This design encourages methods that generalize beyond superficial dataset artifacts and beyond a single training setting.

This paper reports the setup and results of the competition. More broadly, the competition aims to catalyze the community around trustworthy LLM evaluation for software engineering.

CCS Concepts

• Computing methodologies → Natural language processing.

Keywords

Large Language Models, Software Engineering, Privacy, Licensing

ACM Reference Format:

Jonathan Katzy, Ali Al-Kaswan, Razvan Mihai Popescu, and Zhou Yang. 2026. The Poisoned Chalice of LLM Evaluation Report. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26)*, July 05–09, 2026, Montreal, QC, Canada. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3803437.3807733>

*Main contact



This work is licensed under a Creative Commons Attribution 4.0 International License. *FSE Companion '26, Montreal, QC, Canada*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2636-1/2026/07
<https://doi.org/10.1145/3803437.3807733>

1 Introduction

Large Language Models (LLMs) have become central to modern software engineering research and practice. They have been applied to a wide range of tasks, including code generation, code completion, code summarization, test generation, vulnerability detection, and program repair [9, 15]. Their increasing effectiveness has also led to widespread adoption in developer tooling, where they are used to accelerate programming and support day-to-day software development activities.

At the same time, the rapid progress of code-capable LLMs has made rigorous evaluation increasingly difficult. A central challenge is that LLMs are known to memorize portions of their training data and, in some cases, reproduce them verbatim or near-verbatim [3, 7, 16]. As a result, strong performance on a benchmark may not necessarily reflect robust generalization; instead, it may partially arise from prior exposure to benchmark instances during training. This issue, commonly referred to as *data contamination*, threatens the validity of downstream evaluation and can lead to overly optimistic conclusions about model capability [1].

In open-data settings, contamination can sometimes be mitigated by decontaminating the evaluation set against known training corpora. While licenses can be a weak pre-filter, all large open datasets contain significant amounts of copied code, even across licenses [10]. We have previously addressed this and constructed *The Heap*, a dataset designed for evaluating language models on code tasks [11]. However, even in such cases, decontamination can be computationally expensive due to the scale of contemporary training corpora. For example, *The Stack v2*, a widely used corpus for code LLM training, exceeds 90TB in size [12]. More importantly, this strategy is not applicable to closed or partially closed models whose training data are not publicly disclosed [3]. For such models, direct deduplication against the training corpus is impossible.

This limitation motivates the need for alternative methods that can estimate whether a specific artifact was likely seen during training. A natural direction is provided by *membership inference attacks* (MIAs), which aim to infer whether a given sample was part of a model's training data by exploiting differences in model behavior between members and non-members [6]. In the context of code LLM evaluation, MIAs offer a promising mechanism for assessing potential contamination without requiring direct access to the underlying pretraining corpus.

Despite this promise, membership inference remains an open problem. Existing approaches often exhibit limited reliability in

realistic settings, and evidence on source code is still limited [9]. Moreover, much of the prior literature evaluates MIAs on short textual fragments or narrowly scoped instances, whereas practical contamination assessment for software engineering benchmarks often requires reasoning at the level of complete files.

To advance research in this area, we organized the first edition of the *Poisoned Chalice of LLM Evaluation Competition*, co-located with the FSE-AIWare 2026 Competition Track. The competition focuses on white-box membership inference for code LLMs. Participants are given development datasets containing member and non-member code files, along with open-weight target models, and are asked to design methods that distinguish between files that were included in training and files that were not.

To encourage methods that go beyond superficial dataset artifacts, the competition datasets were curated to remove comparatively easy cases, for example those distinguishable primarily through temporal or metadata shifts. Final evaluation is performed on a held-out model and a held-out dataset not accessible during development, thereby emphasizing generalization rather than overfitting to a single model or collection of files.

2 Evaluation

The central objective of the competition is to assess how effectively a method detects whether a code file has been used in the training of a large language model. This objective reflects the broader goal of supporting reliable code LLM evaluations, especially in settings where direct access to training corpora is unavailable. We therefore design the evaluation to measure not only predictive performance on a known target model, but also the extent to which submitted approaches generalize across models and datasets.

To support this objective, we provide competitors with two labeled development datasets. One dataset contains code files that are included in the training data of StarCoder2, while the other contains code files that are neither exact nor near duplicates of files used during training. These datasets allow participants to develop and calibrate their membership inference methods under controlled conditions. In addition, we maintain a held-out evaluation dataset with similar characteristics but different files, which we use for the final assessment of submissions. To evaluate whether approaches generalize beyond a single target model, we also include a validation model together with a corresponding held-out dataset following the same member/non-member construction principles.

2.1 Model Selection

To choose an appropriate dataset for this competition, several key considerations had to be taken into account. First, we required a dataset-model pair for which it is known which data were included in training and which were excluded. Although many models are described as open source, this openness is often limited to the release of model weights, while the training data remain undisclosed. Consequently, the set of suitable models is restricted to those trained on public datasets. The main candidates are StarCoder2 [12], SmoLLM2 [4], and Mellum [14]. As we are interested in scalability and generalizability of approaches in combination with performance we selected StarCoder2 3B and 7B for the competition. Mellum-4B was retained as a held-out model.

2.2 Dataset Curation

To construct the dataset, we required two data sources: one that had been seen during training by the model and one that had definitely not been seen. To achieve this, we used both *The Stack Edu* [4] and The Heap [11] as data sources. We selected The Stack Edu as the source of seen data because it is a subset of *The Stack v2* and was therefore included in the training data of Mellum and the StarCoder2 models. This also makes the resulting dataset suitable for experiments with SmoLLM, although that model was outside the scope of this competition. We selected The Heap as the source of unseen data because it has been deduplicated against *The Stack v2*, which is a superset of *The Stack Edu*. From The Heap, we select files based on the near duplicates column, which uses locality sensitive hashes to also remove files that had only minor changes made to them, resulting in a more diverse dataset. From these datasets, we selected a subset of languages for the competition, namely Go, Java, Python, Ruby, and Rust, to provide a broader overview of the generalizability of the submitted approaches across languages.

This process produced a large pool of potential member and non-member examples. However, small changes in functions, dates, or lexical choices in code can introduce a distribution shift between the member and non-member sets. To mitigate this effect, we trained a bag-of-words classifier to distinguish between the two sets and retained only the files that were misclassified by the classifier [13]. This resulted in member and non-member sets that could not be distinguished simply by the presence of specific keywords. From this curated dataset, participants received two subsets: a training split and a test split¹. We retained an additional unseen validation split that was not released to the participants.

2.3 Evaluation Details

We evaluate submissions using the Area Under the Receiver Operating Characteristic Curve (AUC-ROC). This metric is widely used in membership inference and binary classification settings because it captures the trade-off between the true positive rate and the false positive rate across all decision thresholds. We choose AUC-ROC instead of threshold-specific measures such as TPR@x-FPR [2], because the community does not yet agree on a single acceptable false positive rate for this task. AUC-ROC therefore provides a threshold-independent and comparable measure of performance.

The final ranking reflects performance on held-out data that is not available to participants during development. This design reduces the risk that submissions overfit to the provided datasets or to artifacts specific to a single model. Since the competition aims to identify practically useful approaches for contamination detection, strong submissions need to perform well not only on the development setting, but also on unseen files and on a separate evaluation model.

We execute all submissions on our own hardware to ensure a uniform and reproducible evaluation environment. For this reason, each team submits a runnable replication package together with its method. We provide a dedicated description of the available hardware and execution environment on the competition website. After the submission deadline, we allow a short grace period during which we run all submitted systems and contact teams if minor

¹<https://huggingface.co/datasets/AISE-TUdelft/Poisoned-Chalice>

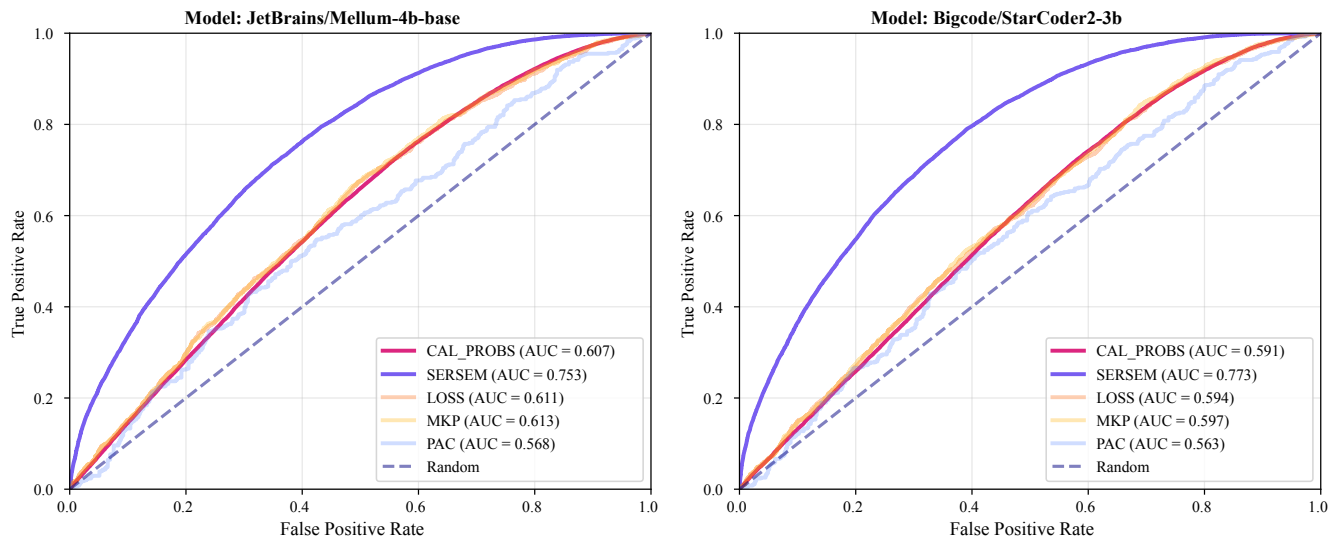


Figure 1: ROC curves for all evaluated membership inference methods on the final test sets for Mellum and StarCoder2-3B

packaging or execution issues require clarification or repair. This process helps ensure that all entries are evaluated fairly and under identical conditions.

2.4 Baselines

We provide three baseline membership inference attacks (MIAs) as reference implementations: *Loss*, *MinK%Prob*, and *PAC*. These baselines represent different strategies for inferring whether a sample belongs to the training data of a model.² *Loss* uses the average token-level negative log-likelihood of a sample, relying on the intuition that training members tend to be assigned lower loss than non-members. *MinK%Prob* focuses only on the lowest-probability tokens in a file, under the assumption that rare or difficult tokens provide a stronger signal of non-membership than well-predicted boilerplate. Finally, *PAC* compares the model’s behavior on the original input and on perturbed variants, using this calibration step to better distinguish memorized samples from merely easy ones.

Each baseline produces a score that reflects the estimated likelihood that a given code file is a member of the training set. Together, these methods establish reference points for the competition and help participants position their approaches relative to existing techniques. We do not treat the baselines as exhaustive; rather, we include them to provide common starting points and to support a more interpretable comparison of submitted methods. For methodological details on the individual attacks, we refer the reader to the corresponding original publications [13].

3 Submitted Reports

In this section we summarize the approaches submitted by participating teams. Since the competition encouraged diverse white-box membership inference strategies, the submitted reports reflect a

range of design choices. Below, we provide concise overviews of the two submitted systems and their reported findings.

3.1 SERSEM: Selective Entropy-Weighted Scoring for Membership Inference in Code Language Models

Dikici et al. propose *SERSEM* [8], a structure-aware membership inference method that down-weights predictable syntactic boilerplate and emphasizes human-centric signals such as comments, long identifiers, string literals, formatting anomalies, and developer markers like `TODO`. Their approach combines weighted token-level scoring from output logits with probing of intermediate transformer activations, motivated by the idea that memorization signals may be more visible in hidden representations than in final predictions.

On a held-out portion of the training set, *SERSEM* reports an overall AUC-ROC of 0.79 on StarCoder2-3B and 0.79 on StarCoder2-7B, outperforming the baselines across all five languages considered.

3.2 Evaluating Signals for Membership Inference Attacks on Source Code

Berndt et al. study simpler probability-based signals adapted to source code [5]. They evaluate several approaches, including average token log-probability, a slope-based signal over token probabilities, and *Min-K%++*. Their best-performing method, *Calibrated Probs*, adjusts average log-probability by token diversity, based on the intuition that repetitive code is easier to predict regardless of membership.

On a held-out set, they report an AUC-ROC of 0.66 for *Calibrated Probs* on StarCoder2-3B, improving over the strongest provided baselines by about 6 percentage points. They also found that truncating the files to the first 200 or 300 tokens reduced performance.

²Baselines: <https://github.com/AISE-TUdelft/PoisonedChalice>

4 Evaluation Results

Figure 1 presents the ROC curves for all methods on StarCoder2-3B and the held-out model Mellum.³ Overall, the results show a clear separation between the strongest submitted method and the probability-based baselines. Across both models, *SERSEM* achieves the highest AUC-ROC by a substantial margin, whereas the remaining methods cluster only slightly above random guessing. This indicates that the competition setting was challenging and that simple likelihood-based signals alone were insufficient to robustly distinguish member from non-member files on the curated evaluation data.

Across both models, *SERSEM* [8] achieves the strongest performance by a clear margin. On StarCoder2-3B, it reaches an AUC-ROC of 0.773, compared with 0.597 for *MKP*, 0.594 for *LOSS*, 0.591 for *CAL_PROBS* [5], and 0.563 for *PAC*. On the held-out Mellum, *SERSEM* again ranks first with an AUC-ROC of 0.753, while the remaining methods achieve between 0.568 and 0.613.

Comparing the two submitted approaches, the final evaluation also reveals a generalization gap. While *CalibratedProbs* improved over the provided baselines during participant-side experiments, its final AUC-ROC remains close to the other probability-based methods and below *SERSEM*. In contrast, *SERSEM* not only outperforms all reference baselines, but also achieves strong performance on the held-out Mellum model. This makes it the most effective submission in the competition and the clear winner.

5 Conclusion and Closing Remarks

The goal of this competition was to evaluate not only the performance of different MIAs but also to discuss their adaptability to real-world problems. Our results suggest that the limited generalizability and sensitivity to keyword drift identified in prior work [13] become less problematic when such confounds are explicitly addressed during dataset construction and evaluation design. However, to address issues such as data leakage and contamination, there are still two concerns. First, all attacks, including the baselines, are computationally complex. This is partially due to the need to run the inputs through an LLM, but also the need to run multiple LLM inferences for input perturbations can multiply this cost. *SERSEM* could be optimized for speed, for example by prioritizing a smaller number of early or intermediate layers instead of probing the full network. Finally, there remains a practical trade-off between precision and recall. If the goal is to identify a subset of files that were almost certainly seen during training, one must operate at a very low false positive rate, which in practice means retaining only a small fraction of all positively scored samples. Conversely, removing as many seen samples as possible requires operating at a higher true positive rate, which also increases the false positive rate and causes genuinely unseen samples to be discarded.

6 Acknowledgements

We thank all the participants for their submissions, the program committee members, Ziyou Li and Roham Koohestani, who reviewed the submissions. The baseline implementations were developed by Cosmin Vasilescu, Ísak Jónsson, and Roham Koohestani as part of the Research Project course (CSE3000) at TU Delft.

³Due to runtime constraints, *PAC* was evaluated on 1,000 samples, whereas all other methods were evaluated on 5,000 samples.

References

- [1] Ali Al-Kaswan and Maliheh Izadi. 2023. The (ab) use of open source code to train large language models. In *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*. IEEE, 9–10.
- [2] Ali Al-Kaswan, Maliheh Izadi, and Arie Van Deursen. 2023. Targeted attack on gpt-neo for the satml language model data extraction challenge. *arXiv preprint arXiv:2302.07735* (2023).
- [3] Ali Al-Kaswan, Maliheh Izadi, and Arie Van Deursen. 2024. Traces of memorization in large language models for code. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.
- [4] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piñeres Lajarin, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgrén, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. 2025. SmoLLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model. arXiv:2502.02737 [cs.CL] <https://arxiv.org/abs/2502.02737>
- [5] Alexander Berndt, Vekil Bekmyradov, Thomas Bach, and Sebastian Balthes. [n. d.]. Evaluating Signals for Membership Inference Attacks on Source Code.
- [6] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramèr. 2022. Membership inference attacks from first principles. In *2022 IEEE symposium on security and privacy (SP)*. IEEE, 1897–1914.
- [7] Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)*. 2633–2650.
- [8] Kıvanç Kuzey Dikici, Serdar Kara, Semih Çağlar, Eray Tüzün, and Sinem Sav. 2026. SERSEM: Selective Entropy-Weighted Scoring for Membership Inference in Code Language Models. In *Companion Proceedings of the 34th ACM Symposium on the Foundations of Software Engineering (FSE '26)*. ACM.
- [9] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79.
- [10] Jonathan Katzy, Razvan Popescu, Arie Van Deursen, and Maliheh Izadi. 2024. An Exploratory Investigation into Code License Infringements in Large Language Model Training Datasets. In *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering (Lisbon, Portugal) (FORGE '24)*. Association for Computing Machinery, New York, NY, USA, 74–85. doi:10.1145/3650105.3652298
- [11] Jonathan Katzy, Razvan Mihai Popescu, Arie van Deursen, and Maliheh Izadi. 2025. The heap: A contamination-free multilingual code dataset for evaluating large language models. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*. IEEE, 151–155.
- [12] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muh-tasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. StarCoder 2 and The Stack v2: The Next Generation. arXiv:2402.19173 [cs.SE] <https://arxiv.org/abs/2402.19173>
- [13] Matthieu Meeus, Igor Shilov, Shubham Jain, Manuel Faysse, Marek Rei, and Yves-Alexandre de Montjoye. 2025. Sok: Membership inference attacks on llms are rushing nowhere (and how to fix it). In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. IEEE, 385–401.
- [14] Nikita Pavlichenko, Iurii Nazarov, Ivan Dolgov, Ekaterina Garanina, Dmitry Ustalov, Ivan Bondyrev, Kseniia Lysaniuk, Evgeniia Vu, Kirill Chekmenev, Joseph Shtok, et al. 2025. Mellum: Production-Grade in-IDE Contextual Code Completion with Multi-File Project Understanding. *arXiv preprint arXiv:2510.05788* (2025).
- [15] Zhou Yang, Jieke Shi, Premkumar Devanbu, and David Lo. 2025. Ecosystem of large language models for code. *ACM Transactions on Software Engineering and Methodology* 35, 1 (2025), 1–30.
- [16] Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, Donggyun Han, and David Lo. 2024. Unveiling memorization in code models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.