Fast-MIA: Efficient and Scalable Membership Inference for LLMs

Hiromu Takahashi
Independent Researcher*
Tokyo, Japan
hiromu.takahashi56@gmail.com

Shotaro Ishihara†
Nikkei Inc.
Tokyo, Japan
shotaro.ishihara@nex.nikkei.com

Abstract

We propose Fast-MIA (https://github.com/Nikkei/fast-mia), a Python library for efficiently evaluating membership inference attacks (MIA) against large language models (LLMs). MIA has emerged as a crucial technique for auditing privacy risks and copyright infringement in LLMs. However, computational demands have grown substantially: recent methods rely on repeated inference, while practical auditing requires large-scale evaluation. Progress is further hindered by existing implementations that execute methods independently, redundantly computing shared intermediate results such as log-probabilities. To address these challenges, Fast-MIA combines two strategies: (1) high-throughput batch inference via vLLM, achieving approximately 5× speedup, and (2) a cross-method caching architecture that computes intermediate results once and shares them across methods. The library includes representative MIA methods under a unified framework, integrates with established benchmarks, and supports flexible YAML configuration. We release Fast-MIA under the Apache License 2.0 to support scalable and reproducible MIA research.

1 Introduction

As LLMs become increasingly deployed in practical applications, concerns have emerged regarding their tendency to *memorize* training data (Ishihara, 2023). Memorization refers to the phenomenon in which a model reproduces exact or nearly identical sequences from its training corpus. This behavior can lead to privacy violations, copyright infringement, and diminished generalization capabilities. In the context of privacy, an early study by Carlini et al. (2021) demonstrated that personal information could be extracted from GPT-2 (Radford et al., 2019). With respect to copyright, Lee et al. (2023) investigated GPT-2 and raised ethical concerns surrounding potential plagiarism. Moreover, there is growing concern about *data contamination*, where memorization of benchmark datasets by LLMs may compromise the reliability of model evaluations (Magar and Schwartz, 2022).

To assess such risks of LLMs, *membership inference attacks (MIA)* (Shokri et al., 2017) have been widely adopted. These attacks on LLMs aim to determine whether a particular data instance is included in the training set. In the context of LLMs, MIA has emerged as a key research area due to growing demands for transparency and accountability in AI systems (Wu and Cao, 2025).

However, the computational requirements for MIA evaluation have grown substantially, hindering both research and practical implementation. Recent advances in MIA methodology increasingly rely on repeated inference: text perturbation methods such as ReCaLL (Xie et al., 2024) and Con-ReCall (Wang et al., 2025) require computing losses across multiple prefix configurations, while black-box approaches like SaMIA (Kaneko et al., 2025) demand numerous generation steps per sample. Furthermore, there is a growing recognition that large-scale evaluation is essential for practical copyright auditing. Puerto et al. (2025) demonstrate that MIA becomes effective only when aggregating results across multiple documents rather than evaluating individual sentences, shifting the focus from sentence-level to collection-level membership inference. These trends, multi-pass inference methods and dataset-scale evaluation, have made computational efficiency a critical bottleneck for MIA research.

To address these challenges, we propose Fast-MIA, an open-source Python library designed for efficient and reproducible evaluation of MIA against LLMs (Figure 1). Fast-MIA combines two complementary strategies for computational effi-

*This work was conducted as part of a business outsourcing agreement with Nikkei Inc.

†Corresponding author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 89–98

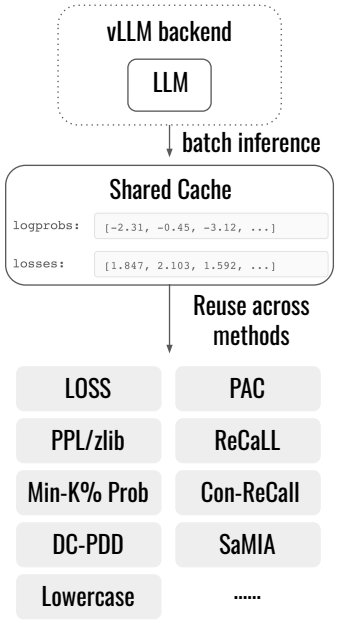

Figure 1: Fast-MIA provides acceleration through high-throughput batch inference by vLLM and cross-method caching architecture across multiple MIA methods.

ciency. First, it provides *high-throughput batch inference* by leveraging vLLM (Kwon et al., 2023), achieving approximately five times faster inference compared to standard Transformers-based implementations (Wolf et al., 2020) with almost no change in evaluation results. Second, it introduces a *cross-method caching architecture* specifically designed for MIA workflows: intermediate results such as log-probabilities and losses are computed once and shared across multiple evaluation methods. This design is particularly beneficial for (1) fair comparison of multiple MIA methods under identical conditions, (2) hyperparameter sweeps such as varying k in Min-K% Prob, and so on.

Fast-MIA offers implementations of several representative MIA methods within a unified and extensible evaluation framework. Users can specify models, datasets, and MIA methods using simple YAML configuration files. The library integrates directly with established benchmarks including WikiMIA (Shi et al., 2024) and MIMIR (Duan et al., 2024), supports multiple data formats, and can handle languages other than English. Furthermore, the modular architecture enables plug-and-play extensibility for custom attack methods. Unless otherwise noted, this paper describes and evaluates Fast-MIA version 0.3.0[1].

[1] https://github.com/Nikkei/fast-mia/releases/tag/v0.3.0

2 Background

MIA provides a general framework for studying memorization, generalization, and privacy risks in machine learning (Shokri et al., 2017). Formally, an attacker is given a target model f, a data point x, and optionally auxiliary information, and must predict whether x is included in the training set.

2.1 Mathematical Preliminaries

Many MIA methods against LLMs rely on the token-level generation probabilities produced by autoregressive language models. Given a text sequence $s = c_1 c_2 \dots c_T$ consisting of T tokens, the model assigns the following probability:

$$p(s) = \prod_{t=1}^{T} p(c_t \mid c_1, \dots, c_{t-1})$$

In practice, the *log-likelihood* is commonly used for numerical stability:

$$\frac{1}{T} \log p(s) = \frac{1}{T} \sum_{t=1}^{T} \log p(c_t \mid c_1, \dots, c_{t-1})$$

Another widely used metric is the *perplexity (PPL)*, which quantifies the model's uncertainty:

$$\text{PPL} = \exp\left(-\frac{1}{T} \sum_{t=1}^{T} \log p(c_t \mid c_1, \dots, c_{t-1}) \right)$$

These quantities form the foundation of many MIA methods, which seek to identify systematic differences in how models score training versus non-training examples.

2.2 Types of MIA Methods

MIA methods exploit the assumption that models behave differently on training data than on unseen data. These behavioral differences can manifest in various forms, and accordingly, different methods focus on different signals. We classify representative approaches to MIA methods into four main categories, following Chen et al. (2025).

Baseline Methods. A simple approach utilizing PPL has been used in the initial work (Carlini et al., 2021).

- **LOSS** (Yeom et al., 2018): This classical baseline uses the loss. In the context of LLMs, the loss corresponds to PPL. A lower PPL indicates a higher likelihood, which may suggest that the input was part of the training data.

- **PPL/zlib** (Carlini et al., 2021): This method computes the ratio between the model's PPL and the size of the zlib-compressed version of the input. Highly redundant training samples tend to yield both low PPL and small compressed sizes.

Token Distribution Based Methods. Some research has improved MIA performance by focusing on token distribution and applying heuristic techniques. The most common method is Min-K% Prob, and DC-PDD is an improved version that normalizes and standardizes the generation probabilities.

- **Min-K% Prob** (Shi et al., 2024): Rather than considering the overall loss, this method focuses on the least confident tokens in a sequence (i.e., those with the lowest predicted probabilities). It computes the average log-likelihood over the lowest $K\%$ of tokens, emphasizing local uncertainty.

- **DC-PDD** (Zhang et al., 2024): This method calibrates token-level likelihood scores by measuring divergence from a background token frequency distribution, reducing bias toward common words.

Text Alternation Based Methods. There are methods for observing changes in log-likelihood and PPL when a text is rephrased. The hypothesis is that when the member sample is changed, the text is expected to undergo changes that cause it to exhibit characteristics similar to those of the non-member sample.

- **Lowercase** (Carlini et al., 2021): This method compares the PPL before and after converting all input text to lowercase. The idea is that casing and similar surface features are often memorized, so altering them may affect training examples more strongly.

- **PAC** (Ye et al., 2024): This method observes changes in the log-likelihood when two random tokens are continuously swapped.

- **ReCaLL** (Xie et al., 2024): This method appends unrelated (non-membership) text to the prompt and measures the change in log-likelihood.

Paper	vLLM	Cache
Murakonda and Shokri (2020)		
Duan et al. (2024)		
Ravaut et al. (2025)	✓	
Meeus et al. (2025)		
Koike et al. (2025)		
Chen et al. (2026)		
Fast-MIA	✓	✓

Table 1: List of research papers publishing implementations that combine multiple MIA techniques.

- **Con-ReCall** (Wang et al., 2025): This method adds both relevant (membership) text and irrelevant (non-membership) text to the prompt. The aim is to facilitate measuring changes in the log-likelihood.

Black-Box Methods. Some research adopts settings that do not use access to generation probabilities (Dong et al., 2024; Kaneko et al., 2025). Memorization is quantified by generating the rest of the text based on the input at the beginning of the text and measuring the similarity of the strings.

- **SaMIA** (Kaneko et al., 2025): This method treats multiple outputs as text samples and measures n-gram similarity.

2.3 Challenges in Practicing MIAs on LLMs.

Despite the increasing interest in MIA against LLMs, there remain two major barriers to advancing this line of research: (1) growing computational demands, and (2) inefficient use of shared computations across methods. These issues underscore the need for scalable, efficient, and reproducible evaluation pipelines for MIA against LLMs.

Growing computational demands. First, the computational cost of evaluating MIA methods on LLMs is extremely high. Unlike traditional machine learning models, LLMs require substantial GPU memory and time. Some text alternation-based methods require multiple inferences for a single sample. For example, Lowercase requires two inferences before and after altering. ReCaLL and Con-ReCall also need several inferences. The number of additional inferences in PAC and SaMIA depends on hyperparameters. Furthermore, recent studies have highlighted the importance of inference at the dataset level, rather than relying solely on individual sentences (Maini et al., 2024; Puerto et al., 2025), which consequently increases the required computational cost.

Redundant computation across methods. Second, the implementations of MIA methods are fragmented, heterogeneous, and rarely maintained. Although several studies have published official implementations, many public repositories are tailored to each paper and lack unified interfaces. These implementations are often tightly coupled to specific experiments or datasets. This makes subsequent research complicated.

Several related initiatives have released implementations bundling MIA techniques as shown in Table 1, but none have managed to eliminate redundancy during multiple executions. Note that LLM-Sanitize (Ravaut et al., 2025) has not been updated since August 2024, potentially limiting its applicability in the evolving LLM landscape. For example, it only supports vLLM version 0.3.3 and cannot benefit from improvements introduced in subsequent versions. Fast-MIA version 0.3.0 uses vLLM version 0.15.1, which was released in February 2026. The recent comprehensive comparison was conducted by Chen et al. (2025), which evaluated multiple MIA methods. Nevertheless, their implementation remains unpublished, limiting reproducibility.

3 Fast-MIA Library

To address the computational and infrastructural challenges shown in Section 2.3, we introduce Fast-MIA, a Python library designed for fast, reproducible, and extensible evaluation of MIAs on LLMs. It integrates representative MIA methods and scalable inference under a single configuration-based interface, enabling reliable comparison across models, datasets, and methods.

3.1 Key Features

Fast-MIA is designed around two key objectives: (1) high-throughput batch inference, and (2) cross-method caching architecture.

1. High-throughput batch inference. Fast-MIA leverages the batched inference capabilities of vLLM to reduce inference time. vLLM is an open-source inference engine optimized for LLMs, featuring efficient memory paging (PagedAttention) and support for dynamic batching (Kwon et al., 2023). As a result, a speedup compared to simple Transformers implementations can be expected.

2. Cross-method caching architecture. Fast-MIA maintains an internal cache of model outputs to eliminate redundant computations across multiple methods. It is not merely a dependence on vLLM capabilities, but rather a design-level effort to effectively reuse the cache within the library. Fast-MIA also converts method-specific repeated inference loops into batched operations whenever possible. This design leads to savings in both computation time and memory usage.

3.2 System Architecture

From a system perspective, Fast-MIA consists of a modular pipeline that connects the following components:

- **Data Loader**: Accepts multiple formats, including CSV, JSON, JSONL, Parquet, and Hugging Face datasets.
- **Model Loader**: Loads target LLMs (e.g., Hugging Face models[2] or LoRA adapters (Hu et al., 2022)) through the high-throughput inference engine vLLM. Fast-MIA can load quantized models through vLLM when the corresponding quantization format is supported by the backend[3].
- **Evaluator**: Coordinates inference, caching, and scoring for one or more MIA methods in a batched manner. It determines which intermediate model outputs are required by each method, executes only the necessary inference calls, and reuses cached outputs whenever possible.
- **MIA Method Registry**: Manages modular implementations of each supported MIA method.
- **YAML Configuration Interface**: Encodes the entire experimental setup in a human-readable, version-controllable format.

3.3 User Interface and Configuration

All model, data, and method parameters are specified in a YAML configuration file, allowing for consistent and reproducible evaluation across methods and experiments. Users can use models and datasets by specifying their Hugging Face identifiers. Fast-MIA directly supports widely used MIA benchmarks such as WikiMIA and MIMIR, while also allowing custom datasets in common formats.

[2] https://huggingface.co/models

[3] https://docs.vllm.ai/en/v0.15.1/features/quantization/

This design follows common practices in the recent LLM ecosystem, where models and datasets are often specified by their Hugging Face identifiers.

For example, the following minimal configuration evaluates the LOSS method on WikiMIA using a Hugging Face model:

```
model:
  model_id: "huggyllama/llama-30b"
data:
  data_path: "swj0419/WikiMIA"
  format: "huggingface"
  text_length: 32
methods:
  - type: "loss"
```

This unified setup eliminates method-specific scripts and facilitates batch comparisons under identical conditions. Once the user has a configuration file, they can run MIA with the following simple command.

```
uv run --with vllm python main.py \
  --config config/sample.yaml
```

Additional options, such as the random seed and maximum cache size, can be specified via command-line arguments.

Metrics. Fast-MIA computes the AUC as the most commonly used evaluation metric. Furthermore, following the advice of Carlini et al. (2022), the false positive rate at 95% TPR (FPR@95) and the true positive rate at 5% FPR (TPR@5) are calculated.

Environment. Fast-MIA is compatible with Linux (NVIDIA A100 GPUs)[4], and provides setup instructions for environments using uv[5]. uv is a fast Python library management tool written in Rust. All components are decoupled and version-controlled, ensuring that the framework remains adaptable as model architectures, evaluation methods, and datasets evolve.

Output. By default, Fast-MIA saves each run in a timestamped directory, including a copy of the configuration file, summary metrics, and a human-readable report. For more detailed analysis, users can enable the detailed report option, which additionally exports per-sample scores, execution metadata, and visualization files such as ROC curves, score distributions, and metric comparison plots. The metadata records experimental conditions such as model, data, method parameters, timing information, git information, and cache statistics, thereby supporting reproducible benchmarking and post-hoc analysis.

3.4 Supported Methods and Extensibility

Fast-MIA version 0.3.0 supports nine representative MIA methods, including baseline methods (LOSS and PPL/zlib), token distribution-based methods (Min-K% Prob and DC-PDD), text alternation-based methods (Lowercase, PAC, ReCaLL, Con-ReCall), and a black-box method (SaMIA). We have incorporated the official implementation in compliance with the license. The implementations of PAC and SaMIA are licensed under the MIT license, and the code of DC-PDD is provided as software under the ACL Anthology (CC BY 4.0 license). The others are licensed under the Apache-2.0 license. Therefore, we have licensed Fast-MIA under the Apache-2.0 license.

Extensibility for custom methods. Each MIA method is implemented in a module (e.g., `loss.py`, `recall.py`) following a common interface defined in `base.py`. This allows users to add custom attack strategies by subclassing and registering via the method factory without modifying core components. Users can implement a custom MIA method by three steps: (1) Creating a file for a custom MIA method (e.g. `src/methods/{custom_method}.py`), (2) Implementing a new class by inheriting the `BaseMethod` class located in `src/methods/base.py` (two functions, `process_output` and `run`, are included), and (3) Editing `src/methods/factory.py` to incorporate the custom MIA method you created.

Language. Furthermore, Fast-MIA provides a flag (`space_delimited_language`) for languages that are not separated by spaces, such as Japanese and Chinese. Some research reports the trends in MIA methods that differ from English (Ishihara and Takahashi, 2024; Takahashi and Ishihara, 2025; Zhang et al., 2024). It is important to promote MIA research that is not limited to English.

4 Evaluation

The evaluation of Fast-MIA was conducted with two main objectives corresponding to key features:

- To demonstrate that Fast-MIA achieves improvements in inference speed compared to

[4]We also provide a Google Colab example for T4 GPUs.
[5]https://github.com/astral-sh/uv

Type	Method	AUC	Inference time (ratio)	FPR@95	TPR@5
baseline	LOSS	69.4 / 69.4	12s / 57s (×4.75)	84.3 / 84.3	18.3 / 18.3
	PPL/zlib	69.8 / 69.8	12s / 57s (×4.75)	80.2 / 80.2	14.5 / 14.5
token distribution	Min-K% Prob (K=0.1)	67.2 / 67.2	12s / 57s (×4.75)	83.5 / 83.3	17.3 / 17.3
	Min-K% Prob (K=0.2)	69.3 / 69.3	12s / 57s (×4.75)	82.3 / 82.3	22.0 / 22.0
	Min-K% Prob (K=0.3)	70.1 / 70.1	12s / 57s (×4.75)	82.3 / 82.3	19.6 / 19.6
	Min-K% Prob (K=0.5)	69.7 / 69.7	12s / 57s (×4.75)	82.5 / 82.5	18.1 / 18.1
	Min-K% Prob (K=0.8)	69.5 / 69.5	12s / 57s (×4.75)	84.3 / 84.3	18.1 / 18.3
	Min-K% Prob (K=1.0)	69.4 / 69.4	12s / 57s (×4.75)	84.3 / 84.3	18.3 / 18.3
	DC-PDD	67.4 / 67.4	12s / 57s (×4.75)	84.8 / 84.8	12.4 / 12.4
text alternation	Lowercase	64.1 / 64.1	25s / 1m59s (×4.76)	83.5 / 83.8	11.6 / 11.6
	PAC	73.3 / 73.4	1m17s / 6m24s (×4.99)	82.3 / 77.9	27.6 / 24.3
	ReCaLL	90.7 / 90.3	55s / 2m10s (×2.36)	28.5 / 34.7	50.4 / 48.8
	Con-ReCall	96.8 / 96.1	1m53s / 3m30s (×1.86)	10.8 / 12.9	78.0 / 73.6
black-box	SaMIA	65.5 / 64.5	2h3m24s / 40h9m53s (×19.5)	90.5 / 90.7	22.7 / 15.5

Table 2: Comparison of Fast-MIA (left) and Transformers-based implementations (right) in terms of AUC, inference time, FPR@95, and TPR@5. Performance metrics such as AUC remain almost the same, while inference time is approximately five times faster.

Type	Method	Fast-MIA		Transformers
		w/ cache	w/o cache	
baseline	LOSS	12s (1)	12s (1)	57s (1)
	PPL/zlib	0s (0)	12s (1)	57s (1)
token distribution	Min-K% Prob	0s (0)	12s (1)	57s (1)
	DC-PDD	0s (0)	12s (1)	57s (1)
text alternation	Lowercase	13s (1)	25s (2)	1m59s (2)
	PAC	1m5s (5)	1m17s (6)	6m24s (6)
	ReCaLL	43s (1)	55s (2)	2m10s (2)
	Con-ReCall	1m41s (2)	1m53s (3)	3m30s (3)
	Total	3m54s (10)	5m18s (17)	17m51s (17)

Table 3: Comparison of Fast-MIA with (w/) and without (w/o) cache, and Transformers-based implementations in terms of inference time (the number of inferences). We excluded SaMIA due to its extremely long inference time.

naive Transformers implementations while ensuring correctness and reproducibility.

- To confirm that the total number of inferences is reduced compared to executing MIA methods individually.

4.1 Experimental Setup

To this end, we conducted comparative experiments using the same LLM model (LLaMA 30B (Touvron et al., 2023)) and MIA settings on an NVIDIA A100 80GB GPU. Fast-MIA was benchmarked against corresponding reference implementations using Transformers for all nine methods supported by Fast-MIA version 0.3.0. The evaluation was performed on the WikiMIA dataset, with input token length set to 32. We calculated the AUC, FPR@95, and TPR@5. We did not include a comparison with existing toolkits such as those described in Table 1. This is because available toolkits differ in supported methods, maintenance status, and backend versions, making direct end-to-end comparison difficult.

Each MIA method was executed by specifying only a single YAML configuration file, without additional scripting. Methods that include hyperparameters require configuration. K in Min-K% Prob was set to 0.1, 0.2, 0.3, 0.5, 0.8, and 1.0. The original paper (Shi et al., 2024) reported that $K = 0.2$ yielded the best results. Note that the setting $K = 1.0$ is consistent with LOSS. In our settings, ReCaLL and Con-ReCall provided three examples as prompts, PAC was performed with five token swaps, and SaMIA used five outputs. The complete configuration file used in Section 4 can be found in the Appendix A.

4.2 Evaluation Results

As shown in Table 2, Fast-MIA reproduces the AUC scores from the Transformers-based implementation with negligible differences, while reduc-

ing the total computation time by approximately five times. AUC is completely consistent in baseline methods and token distribution based methods. In text alternation based and black-box methods, slight performance fluctuations occurred when generating text multiple times. Particularly in SaMIA, substantial reductions in inference time were observed. By replacing repeated generation loops with batched multi-output generation[6] and leveraging vLLM-based inference, Fast-MIA reduced the runtime from 40h9m53s to 2h3m24s.

The cache mechanism reduced the inference time and the number of inferences in this experiment as shown in Table 3. Under cache-effective conditions, the intermediate results from the initial inference run with LOSS were reused. As a result, inferences via token-distribution based methods became unnecessary. For methods requiring multiple inferences, the number of inferences decreased by one each time.

Table 3 also serves as an ablation of the two efficiency mechanisms. The comparison between the Transformers-based implementation and Fast-MIA without cache mainly reflects the effect of the vLLM-based batched inference backend, whereas the comparison between Fast-MIA with and without cache isolates the effect of cross-method caching.

5 Conclusion

We have introduced Fast-MIA, a Python library that enables efficient and scalable evaluation of MIA against LLMs. Against the background of high computational costs and the lack of standardized implementations, Fast-MIA provides high-throughput batch inference on vLLM and cross-method caching architecture, enabling rapid comparative evaluation across diverse methods. The experiment demonstrated that Fast-MIA can reduce inference time with small performance changes. We release Fast-MIA as an open-source tool to accelerate research on privacy, memorization, and evaluation of LLMs. We anticipate that this library serves as a foundation for future methodological and empirical advancements.

[6]In an intermediate Transformers-based implementation, replacing the per-sample generation loop with batched multi-output generation reduced the runtime to approximately 13 hours.

Limitations

While Fast-MIA offers a scalable and extensible evaluation platform, the current implementation has several limitations:

- **Number of implemented methods**: Numerous methods have been proposed for membership inference on LLMs. While Fast-MIA covers representative approaches, there remains room for further implementation. Methods targeting entire datasets have also not yet been implemented.

- **Model diversity**: Fast-MIA currently supports models compatible with vLLM, with open-weight autoregressive language models benefiting most significantly. Models based on different architectures (e.g., encoder-only or encoder-decoder) are not yet sufficiently supported. Although black-box methods could potentially be applied to closed APIs, they are not supported in the current implementation.

- **Broader customization:** Although Fast-MIA supports custom MIA methods through a modular interface, other extensions such as user-defined evaluation metrics, customized report formats, and new output schemas are not yet fully configurable through YAML configuration files. Supporting these broader customization needs without modifying the internal evaluation or reporting pipeline is an important direction for future work.

Addressing these limitations is part of our ongoing work. We welcome community contributions to broaden the scope of attack types, model families, and other areas supported by Fast-MIA.

Furthermore, since the primary purpose of this paper is to demonstrate Fast-MIA, the experiments are not exhaustive. Our experiments focus on one representative model, dataset, text length, and hardware setting to validate the correctness and efficiency of the library under a controlled setup. Although the acceleration mechanisms are model-agnostic for vLLM-compatible autoregressive models, the exact speedup may vary depending on model size, sequence length, hardware, and decoding configuration. Broader evaluations across model families, dataset domains, context lengths, and hardware environments are left for future work.

Ethics/Broader Impact

Fast-MIA is designed to advance the understanding of privacy risks in LLMs by enabling reproducible and standardized evaluation of MIAs. While the methods implemented in the library can potentially be used to identify whether a given text was part of a model's training data, the primary purpose of this work is to support responsible research into memorization and data leakage.

We emphasize that the library does not include pre-trained MIA models, or automatic data scraping routines. Instead, it focuses on benchmarking known MIA strategies under controlled experimental settings. We believe that enabling reproducible and transparent evaluation of MIA risk is a necessary step toward the development of safer and more privacy-preserving LLMs.

Fast-MIA is released under an open-source license, but we encourage users to follow ethical research practices, including proper citation and responsible data handling. We position Fast-MIA as a platform to promote robust and responsible privacy-aware development of LLMs, rather than as a tool for exploitation.

References

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)*, pages 1897–1914. IEEE.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650. USENIX Association.

Bowen Chen, Namgi Han, and Yusuke Miyao. 2025. A statistical and multi-perspective revisiting of the membership inference attack in large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22854–22874, Vienna, Austria. Association for Computational Linguistics.

Yuetian Chen, Yuntao Du, Kaiyuan Zhang, Ashish Kundu, Charles Fleming, Bruno Ribeiro, and Ninghui Li. 2026. Window-based membership inference attacks against fine-tuned large language models. *arXiv [cs.CL]*.

Yihong Dong, Xue Jiang, Huanyu Liu, Zhi Jin, Bin Gu, Mengfei Yang, and Ge Li. 2024. 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, Bangkok, Thailand. Association for Computational Linguistics.

Michael Duan, Anshuman Suri, Niloofar Mireshghallah, Sewon Min, Weijia Shi, Luke Zettlemoyer, Yulia Tsvetkov, Yejin Choi, David Evans, and Hannaneh Hajishirzi. 2024. Do membership inference attacks work on large language models? In *First Conference on Language Modeling*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Shotaro Ishihara. 2023. Training data extraction from pre-trained language models: A survey. In *Proceedings of the 3rd Workshop on Trustworthy Natural Language Processing (TrustNLP 2023)*, pages 260–275, Toronto, Canada. Association for Computational Linguistics.

Shotaro Ishihara and Hiromu Takahashi. 2024. Quantifying memorization and detecting training data of pre-trained language models using Japanese newspaper. In *Proceedings of the 17th International Natural Language Generation Conference*, pages 165–179, Tokyo, Japan. Association for Computational Linguistics.

Masahiro Kaneko, Youmi Ma, Yuki Wata, and Naoaki Okazaki. 2025. Sampling-based pseudo-likelihood for membership inference attacks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8894–8907, Vienna, Austria. Association for Computational Linguistics.

Ryuto Koike, Liam Dugan, Masahiro Kaneko, Chris Callison-Burch, and Naoaki Okazaki. 2025. Machine text detectors are membership inference attacks. *arXiv [cs.CL]*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Jooyoung Lee, Thai Le, Jinghui Chen, and Dongwon Lee. 2023. Do language models plagiarize? In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 3637–3647, New York, NY, USA. Association for Computing Machinery.

Inbal Magar and Roy Schwartz. 2022. 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. Association for Computational Linguistics.

Pratyush Maini, Hengrui Jia, Nicolas Papernot, and Adam Dziedzic. 2024. LLM dataset inference: Did you train on my dataset? In *Advances in Neural Information Processing Systems*, volume 37, pages 124069–124092. Curran Associates, Inc.

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)*, pages 385–401, Los Alamitos, CA, USA. IEEE Computer Society.

Sasi Kumar Murakonda and Reza Shokri. 2020. ML privacy meter: Aiding regulatory compliance by quantifying the privacy risks of machine learning. *arXiv [cs.CR]*.

Haritz Puerto, Martin Gubri, Sangdoo Yun, and Seong Joon Oh. 2025. Scaling up membership inference: When and how attacks succeed on large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4165–4182, Albuquerque, New Mexico. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Mathieu Ravaut, Bosheng Ding, Fangkai Jiao, Hailin Chen, Xingxuan Li, Ruochen Zhao, Chengwei Qin, Caiming Xiong, and Shafiq Joty. 2025. A comprehensive survey of contamination detection methods in large language models. *Transactions on Machine Learning Research*.

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

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18.

Hiromu Takahashi and Shotaro Ishihara. 2025. Quantifying memorization in continual pre-training with Japanese general or industry-specific corpora. In *Proceedings of the First Workshop on Large Language Model Memorization (L2M2)*, pages 95–105, Vienna, Austria. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and efficient foundation language models. *arXiv [cs.CL]*.

Cheng Wang, Yiwei Wang, Bryan Hooi, Yujun Cai, Nanyun Peng, and Kai-Wei Chang. 2025. ConReCall: Detecting pre-training data in LLMs via contrastive decoding. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 1013–1026, Abu Dhabi, UAE. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Hengyu Wu and Yang Cao. 2025. Membership inference attacks on large-scale models: A survey. *arXiv [cs.LG]*.

Roy Xie, Junlin Wang, Ruomin Huang, Minxing Zhang, Rong Ge, Jian Pei, Neil Zhenqiang Gong, and Bhuwan Dhingra. 2024. ReCaLL: Membership inference via relative conditional log-likelihoods. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8671–8689, Miami, Florida, USA. Association for Computational Linguistics.

Wentao Ye, Jiaqi Hu, Liyao Li, Haobo Wang, Gang Chen, and Junbo Zhao. 2024. Data contamination calibration for black-box LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10845–10861, Bangkok, Thailand. Association for Computational Linguistics.

Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. 2018. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 268–282.

Weichao Zhang, Ruqing Zhang, Jiafeng Guo, Maarten de Rijke, Yixing Fan, and Xueqi Cheng. 2024. Pretraining data detection for large language models: A divergence-based calibration method. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 5263–5274, Miami, Florida, USA. Association for Computational Linguistics.

A Configuration File Used in Section 4

The complete YAML configuration file used in Section 4 is as follows.

```yaml
# Model settings
model:
  model_id: "huggyllama/llama-30b"
  trust_remote_code: true
  max_num_seqs: 256
```

```yaml
# sampling_parameters settings
sampling_parameters:
  max_tokens: 1
  prompt_logprobs: 0
  temperature: 0.0
  top_p: 1.0

# Data settings
data:
  # Basic settings
  data_path: "swj0419/WikiMIA"
 # Path to data file or huggingface dataset name
  format: "huggingface"
  # Data format
  # (csv, jsonl, json, parquet, huggingface)

  # Column name settings
  text_column: "input"
  # Name of text column
  label_column: "label"
  # Name of label column

  # text split settings
  text_length: 32
  # Number of words to split
 # (for WikiMIA dataset: 32, 64, 128, or 256)

  # Language
  space_delimited_language: true

# Evaluation methods
methods:
  - type: "loss"
    params: {}
  - type: "lower"
    params: {}
  - type: "zlib"
    params: {}
  - type: "mink"
    params:
      ratio: 0.1
  - type: "mink"
    params:
      ratio: 0.2
  - type: "mink"
    params:
      ratio: 0.3
  - type: "mink"
    params:
      ratio: 0.5
  - type: "mink"
    params:
      ratio: 0.8
  - type: "mink"
    params:
      ratio: 1.0
  - type: "recall"
    params:
      num_shots: 3
      pass_window: false
  - type: "conrecall"
    params:
      num_shots: 3
      pass_window: false
      gamma: 0.5
  - type: "pac"
    params:
      alpha: 0.3
      N: 5
  - type: "samia"
    params:
      num_samples: 5
      prefix_ratio: 0.5
      zlib: true
  - type: "dcpdd"
    params:
      file_num: 15
      max_token_length: 1024
      alpha: 0.01

# Output settings
output_dir: "./results"
```