USENIX Security '26 Artifact Appendix: Five Queries Are Enough: Query-Efficient and Surrogate-Free Membership Inference Attacks on RAG via Entailment

Nguyen Linh Bao Nguyen[1], Wanlun Ma[1], Viet Vo[1]*, Alsharif Abuadbba[2], Minghong Fang[3], Jun Zhang[1], and Yang Xiang[1]
[1]*Swinburne University of Technology, Australia*
[2]*CSIRO, Australia*
[3]*University of Louisville, USA*

A Artifact Appendix

A.1 Abstract

This artifact contains the implementation and evaluation scripts for MEntA, a membership inference attack against Retrieval-Augmented Generation (RAG) systems. The artifact includes MEntA, four baseline attacks (IA-MIA, DCMI, MBA, and S2-MIA), input/output modification defenses, query-detection defenses, preprocessed BEIR datasets, and Apptainer scripts for reproducing the main experiments.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact evaluates membership inference attacks on public BEIR datasets only: NFCorpus, SCIDOCS, and TREC-COVID. It does not attack any deployed service or private database. The online phase uses the OpenAI API for query generation, summarization, query paraphrasing, and GPT-based detection; evaluators should use their own API key and monitor API cost. The offline phase runs local HuggingFace models and may require GPU resources. No destructive system-level actions are performed.

A.2.2 How to access

The artifact is archived at `https://doi.org/10.5281/zenodo.20335970`. Download and unpack the archive, then follow the setup steps below from the artifact root directory.

A.2.3 Hardware dependencies

VRAM requirements vary by experiment and were measured on NVIDIA A100 GPUs:

- Smoke test: approximately 40 GB VRAM;
- Full MEntA evaluation: approximately 75 GB VRAM;
- Baseline evaluations (`IA-MIA`, `DCMI`, `MBA`, and `S2-MIA`): typically more than 100 GB VRAM;

A smaller `RAG_GENERATOR_MODEL` may be used for experiments if `microsoft/phi-4` does not fit, though results may differ from those reported in the paper.

A.2.4 Software dependencies

The main software dependency is Apptainer. Install Apptainer by following the official installation guide: `https://apptainer.org/docs/admin/main/installation.html`. We used Apptainer version 1.3.6 for our implementation.

The artifact provides an `Apptainer.def` file. The container installs the Python dependencies from `requirements.txt`. Evaluators also need an OpenAI API key for online stages, and a HuggingFace token for gated models, if such models are selected.

A.2.5 Benchmarks

The archive ships with `./data/` already populated. It contains preprocessed BEIR corpora for NFCorpus, SCIDOCS, and TREC-COVID, including member/non-member splits, FAISS indices, benign queries, perturbed corpora for DCMI, and per-document summaries used by MEntA retrieval boosting. To refresh or extend datasets, for example to add another retriever index, follow the instruction of setting up data in A.3.1.

A.3 Set-up

A.3.1 Installation

If the cluster requires Apptainer cache or temporary directories to be placed on project storage, copy and edit the provided setup script before building the image:

*Corresponding author: `vvo@swin.edu.au`

```
cp example_apptainer_setup.sh apptainer_setup.sh
# edit APPTAINER_CACHEDIR and APPTAINER_TMPDIR
source apptainer_setup.sh
```

From the artifact root, build the Apptainer image and create the local configuration file. Please note that `apptainer build` typically requires root or fakeroot on many clusters:

```
apptainer build menta.sif Apptainer.def
cp .env.example .env
```

Edit `.env` before running any experiment. For the reported full experiments using `CohereLabs/c4ai-command-r7b-12-2024`, we recommend evaluators to keep the configuration in this file. At minimum, evaluators should set the OpenAI API credentials, HuggingFace token, and HuggingFace cache.

```
OPENAI_API_KEY=...
HF_TOKEN=...
HF_HOME=/absolute/path/to/hf_cache
```

Define the Apptainer helper used by the scripts:

```
apptainer_run() {
  local hf_home
  hf_home="$(set -a; source .env; printf '%s' "$HF_HOME")"

  apptainer exec --nv \
    --env-file .env \
    --bind "$PWD:/workspace" \
    --bind "$hf_home:$hf_home" \
    menta.sif "$@"
}
export -f apptainer_run
```

Download the configured HuggingFace models:

```
apptainer_run bash scripts/setup_model.sh
```

The archive already includes `./data/` and rebuilding data is optional. If needed, run the online data preparation on a networked node first and wait for it to finish successfully. Only then run the offline indexing phase inside a GPU allocation:

```
apptainer_run bash scripts/setup_data.sh --online
# run this only after the online phase has finished
apptainer_run bash scripts/setup_data.sh --offline
```

A.3.2 Basic Test

The smoke test checks the full workflow on a small copied dataset. Run the online phase first and wait for it to finish successfully before starting the offline phase, because the offline phase consumes the queries, paraphrases, summaries, and detector inputs produced by the online phase.

```
apptainer_run bash scripts/run_test_mode.sh --online
# run this only after the online smoke-test phase has finished
apptainer_run bash scripts/run_test_mode.sh --offline
```

The online command executes OpenAI/API stages, while the offline command builds the small FAISS index and runs local GPU stages. A successful test ends with:

```
Smoke-test online phase finished.
Smoke-test offline phase finished.
```

Smoke-test outputs are written under `results/test/`.

A.4 Evaluation Workflow

A.4.1 Major Claims

(C1) Under a five-query budget, MEntA achieves the strongest overall membership inference AUC among the evaluated attacks (IA, MBA, S^2MIA, and DCMI). This is the main effectiveness result reported in Table 3.

(C2) MEntA remains effective under input and output modification defenses, including Differential Privacy (DP)-style perturbation, context re-ranking, prompt instruction, and query paraphrasing. These defended-setting results are reported in Table 6.

(C3) MEntA remains robust against input detection defenses. The GPT-4-based detector and Mirabel similarity detector are evaluated in Tables 7 and 8. Although Mirabel can detect a portion of attack queries, it also produces a high false-positive rate on benign queries, making it impractical as a reliable deployment-time defense.

A.4.2 Experiments

The reproduced numerical results may differ slightly from those reported in the paper due to randomness, model/runtime differences, and API nondeterminism, but the overall claims and conclusions should remain consistent.

The main evaluation workflow is `scripts/run_all.sh`. After `.env` is configured, this script runs the attack comparison, defended attack settings, and input detection defenses in one online/offline workflow.

For the full reproduction, configure `.env` with the three datasets, the Cohere generator model, the five-query budget, and all attack/defense modes:

```
DATASETS="BeIR_nfcorpus BeIR_scidocs BeIR_trec-covid"
RAG_GENERATOR_MODEL=CohereLabs/c4ai-command-r7b-12-2024
NUM_QUERIES=5
MENTA_MODES="default dp rerank prompt_instruction paraphrase"
IA_MODES="default dp rerank prompt_instruction paraphrase"
DCMI_MODES="default dp rerank prompt_instruction paraphrase"
MBA_MODES="default dp rerank prompt_instruction paraphrase"
S2MIA_MODES="default dp rerank prompt_instruction paraphrase"
FORCE=0
```

Then run the online phase on a networked node and the offline phase inside a GPU allocation. The online phase must complete successfully before the offline phase is started, because the offline phase consumes the queries, paraphrases, summaries, and detector inputs produced by the online phase:

```
apptainer_run bash scripts/run_all.sh --online
# run this only after the online phase has finished
apptainer_run bash scripts/run_all.sh --offline
```

(E1) Five-query attack comparison. This experiment validates C1. In the workflow above, `run_all.sh` first runs the default mode for MEntA and all baselines. The online phase generates attack queries and other API-dependent artifacts. The offline phase performs retrieval, local RAG generation, scoring, and evaluation. MEntA results are written under `results/MEntA/<dataset>/evaluation/`. Baseline results are under `results/IA-MIA/`, `results/DCMI/`, `results/MBA/`, and `results/S2-MIA/`. The AUC values in these default evaluation files correspond to Table 3.

(E2) Input and output modification defenses. This experiment validates C2. Because the mode variables above include `dp`, `rerank`, `prompt_instruction`, and `paraphrase`, `run_all.sh` runs each defense as a separate pass for every attack. Modes are not combined within one attack run. The resulting defense-specific outputs are stored in directories such as `evaluation_dp/`, `evaluation_rerank/`, `evaluation_prompt_instruction/`, and `evaluation_paraphrased/`. These files provide the values summarized in Table 6.

(E3) Input detection defenses. This experiment validates C3. After the attack query files have been generated, `run_all.sh` invokes `scripts/run_input_detection_defenses.sh`. During the online phase, it builds the unified `queries_dataset.jsonl` files and runs the GPT-based detector. During the offline phase, it runs the Mirabel-style similarity detector. GPT detection outputs are written under `results/gpt_detection/`; Mirabel outputs are written under `results/mirabel_detection/`. The resulting recall and false-positive-rate summaries correspond to Tables 7 and 8.

A.5 Notes on Reusability

The main configuration interface is `.env`. Evaluators can change datasets, model names, retrievers, query budgets, top-k, and modes without editing Python code. The scripts support resume-by-default behavior: completed stage outputs are skipped unless `FORCE=1`. OpenAI calls can use either the Batch API or sequential single-request inference by setting:

```
OPENAI_INFERENCE_MODE=batch
OPENAI_INFERENCE_MODE=single
```

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.