USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SparSamp: Efficient Provably Secure Steganography Based on Sparse Sampling

Yaofei Wang, *Hefei University of Technology;* Gang Pei, *Hefei University Of Technology;* Kejiang Chen and Jinyang Ding, *University of Science and Technology of China;* Chao Pan, Weilong Pang, and Donghui Hu, *Hefei University of Technology;* Weiming Zhang, *University of Science and Technology of China*

https://www.usenix.org/conference/usenixsecurity25/presentation/wang-yaofei

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: SparSamp: Efficient Provably Secure Steganography Based on Sparse Sampling

Yaofei Wang, Weilong Pang

A Artifact Appendix

A.1 Abstract

This artifact provides the implementation of SparSamp, an efficient provably secure steganography method for generative models. It includes code for embedding/extracting messages in text generation tasks using large language models (e.g., GPT-2, Qwen-2.5, Llama-3) and supports evaluation metrics such as embedding speed, decoding accuracy, and embedding rate. The artifact reproduces the key results in the paper, including:

- *High embedding rate approaching information entropy*
- *High embedding speed with $O(1)$ time complexity per sampling step*
- *Preservation of original probability distributions*

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

SparSamp is a dual-use technology. Users must comply with ethical guidelines and local laws when applying it.

A.2.2 How to access

- ***Repository**: Available on Zenodo with DOI: 10.5281/zenodo.15025436*

A.2.3 Hardware dependencies

- ***CPU**: Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz*
- ***GPU**: RTX 4090 × 1*
- ***Memory**: 128GB*

A.2.4 Software dependencies

- ***Python**: 3.8.5*
- ***Libraries**:*
  - `torch==2.2.2`
  - `transformers==4.41.2`

A.2.5 Benchmarks

- ***Datasets**: IMDB text samples (first 3 sentences per sample)*
- ***Models**:https://huggingface.co/*
  - `GPT-2:` `openai-community/gpt2`
  - `Qwen-2.5:` `Qwen/Qwen2.5-3B-Instruct`
  - `Llama-3:` `meta-llama/Llama-3.1-8B-Instruct`

A.3 Set-up

A.3.1 Installation

1. *Download the repository.*
2. *Install dependencies.*
3. *Download models (e.g., GPT-2).*

A.3.2 Basic Test

*Here, we can test the embedding and extraction functions of **SparSamp** in an example without using GPU, as shown below:*

1. *Enter folder* `[Basic Test]`
2. *Run the default configuration in main.py: python main.py*
3. *Expected Output: As shown in Figure 1.*

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *SparSamp achieves 100% decoding accuracy with message segment lengths $l_m \leq 1023$ (Section 4.3, Table 2).*
(C2): *High embedding rate approaching information entropy(Section 4.3, 4.5, 4.7, Table 2, 4, 7).*
(C3): *High embedding speed(Section 4.3, 4.4, 4.7, Table 2, 3, 7).*

```
(base) XXX@mac Artifact_sparsamp % python3 main.py

Using device: cpu

=== Initialization Phase ===
Message length padded to 1792 bits
Number of message bits used: 1792
model_name:../sparsamp_test/gpt/
Failed to load model locally, load GPT-2 directly from Transformers library
Model loaded successfully

=== Encoding Phase ===
Generated steganographic text:  When it came to languages, VF Native is
trying to use VF Native by showing different ways it is implementing it. I
suppose that these suggestions are too short, and I suppose that in my
opinion here VF Native not only gives what is really needed but also allows
to form an executable program that is portable for permanent use in native
platforms (and if you need to run VF Native, there are also uses for laro!).

VF Native as design

VF Native is designed with the N project as a strategic starting point for

=== Save and Verification Phase ===
Steganographic text saved to: stega_text.txt
Whether there is Token Ambiguity: No

=== Decoding Phase ===

=== Evaluation Results ===
Experimental Settings - Model: ../sparsamp_test/gpt/, Top-p: 0.95, Message
Segment Length (l_m): 64
Embedded 512 bits message in the generated 112 tokens
Message decoding result: Success
ATST: 2.86e-02 s/token
SITR: 0.00
Generation Speed: 34.9 tokens/s
Embedding Rate: 4.57 bits/token
Utilization: 94.7%
Embedding Speed: 161.0 bits/s
Decoding Speed: 177.3 bits/s
```

Figure 1: Expected Output of the Basic Test

A.4.2 Experiments

Enter folder `[Experiments/src]` *for running batch data*

Download the pre-trained language model to your local machine in advance. Then, modify the `params_dict['model_path']` *parameter in the* `get_statistics` *function within* `get_statistics.py` *to match the actual storage path.*

(E1): *Decoding Accuracy Validation [10 human-minutes + 1 compute-hour + 20GB disk]: Verify 100% decoding accuracy for message segment lengths $l_m \leq 1023$. Expected results: Success for all cases (Table 2).*

Preparation: *Ensure the model and tokenizer are loaded correctly. Run Python and set different* `block_size` *values to verify the extraction rate.*

Execution: *Run the following commands:*

```
block_sizes = [2, 4, 8, 16, 32, 64, 128,
256, 512, 1023]
result_list = []
for block_size in block_sizes:
    cur_result = get_statistics('sparsamp','1.0',
    block_size,'gpt2')
    result_list.append(cur_result)
pd.Dataframe(result_list).to_excel(
"result.xlsx",index=False)
```

Results: *Check the* `correct_decoded_rate` *in the logs or* `result.xlsx`*. Expected:* `1` *for all cases.*

(E2,E3): *Embedding rate and embedding speed verification [20 human-minutes + 3 compute-hour + 20GB disk]: Expected results: Utilization $\geq$ 93% (Table 3, 4, 7).*

Preparation: *Run Python and set different* `top_p` *values to verify the embedding rate and embedding speed.*

Execution: *Run the following command:*

```
top_ps = [0.80, 0.95, 1.00]
result_list = []
for top_p in top_ps:
    cur_result = get_statistics('sparsamp',
    top_p,64,'gpt2')
    result_list.append(cur_result)
pd.Dataframe(result_list).to_excel(
"result.xlsx",index=False)
```

Results: *Compare the* `embedding_rate`*,* `utilization`*,* `Embedding_Speed`*,* `Decoding_Speed`*,* `ATST`*,* `Generation_Speed` *and* `SITR` *with Table 3, 4, 7.*

Note the problems that hardware may cause*: Since some computers may cause floating-point precision issues, the cumulative probabilities may not match exactly. We ignore this issue and use the stored SE values from encoding during the decoding.*

A.5 Notes on Reusability

- ***New Models****: To integrate other models (e.g., Llama-3), modify* `model_path` *in* `get_statistics.py`*.*
- ***Custom Messages****: Replace* `message_bits.txt` *with any binary file.*
- ***Multi-modal Support****: SparSamp can be integrated with any model that provides explicit probability distributions by simply replacing the sampling component with SparSamp's embedding function.*

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/usenixsec2025/.