Literature Review on LLM Attacks

Weekly Report

Nan Lin

2026-06-05

Introduction to Attack techniques (1)

Cache poisoning & collision

1-1. Cache poisoning based on cache collision

(Cache Me Catch You, NDSS 2026)

Observation.

1. Caching system in LLM services.

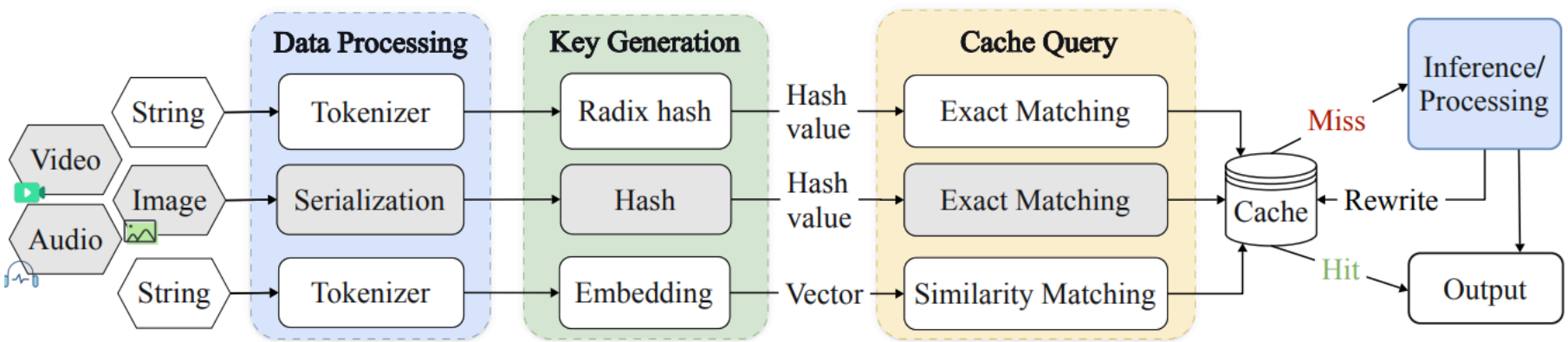
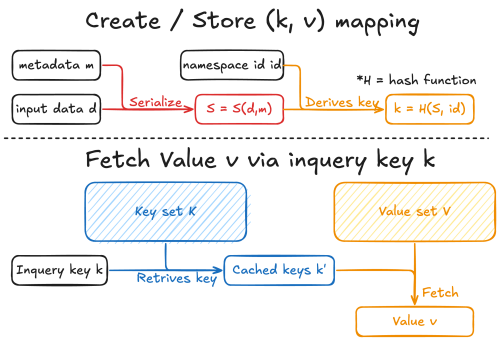


Fig. 2: An illustration of the general workflow of the caching system in LLM services.

1-1. Cache poisoning based on cache collision



$$(d, m, \mathrm{id}) \rightarrow k \rightarrow v$$

Cases that violates the **security guidelines**:

- Images serialization: incomplete serialization
- Hash collision: NCHF
- Namespace violation
- Semantic cache

1-1. Cache poisoning based on cache collision

Motivation. Attacks can manipulate shared caches by

- **Crafting** malicious reqs to cause the system to cache
- **Replace** the original ones with same K value
- Later **reused** by victim reqs (since the keys are identical).

Requirements.

- ONLY need to send API reqs.
- No need to read memory space or to control the machine

1-2. Directly replace KV Cache

(Whose Narrative is it Anyway, 2025)

Motivation. The contents of KV Cache relates directly to the context of the reqs.

Method. Overwrites a segment of the memory space that stores KV Cache with the attacker's K/V states.

Requirements.

- R/W acess to the stored KV Cache tensors.
- Knowledge of the cache layout, token positions

Introduction to Attack techniques (2)

KV Cache & Side Channel Atk

KV Cache reuse/sharing

KV cache sharing can reduce both computation and memory usage.

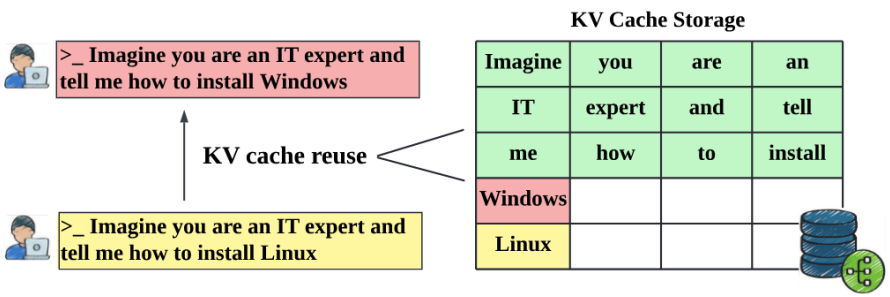


Figure 1: KV cache reuse.

Request 1:

```
Help me translate this
sentence into English,
...
```

Request 2:

```
Help me translate this
sentence into French,
...
```

(a) KV cache shared: "Help me translate this sentence into".

Request 1:

```
Help me translate this
sentence into English,
...
```

Request 2:

```
Translate this sentence
into English,
...
```

(b) No KV shared because the first tokens are different.

Figure 2: KV cache sharing policy.

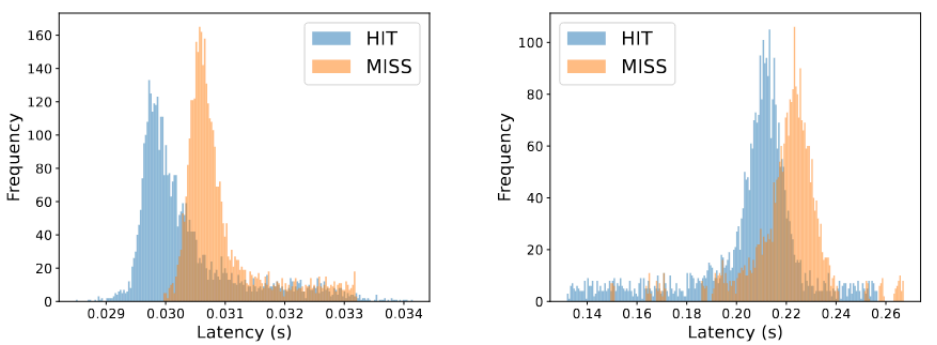
⇒ BUT caching leaks info !!

⇒ **Active** Probing

2-1. Timing difference between cache hit/miss

(The Early Bird, TIFS 2025) (InputSnatch, 2024)

Observation. Send API reqs: Cache hit & miss -> diff response-time distribution.



(a). Llama-3.1-8B-Instruct (b). Llama-3.1-70B-Instruct-GPTQ-INT4

Fig. 3. Latency distribution of one token hit and miss.

Opportunity. Use Δt to infer queried prefix matches cached victim input

2-1. Timing difference between cache hit/miss

(InputSnatch, 2024)

Problem 1. Response Time is not accurate.

- Interference from Time Noise.
- e.g. load balancing, system load.

Solution. Transform TTFT to # hit blocks

- Offline Profile
- Techniques: Gradient Boosting, RF, XGBoost ...

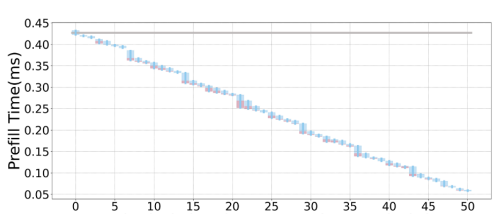


Figure 10: Relationship between TTFT and hit blocks. A

2-1. Timing difference between cache hit/miss

Problem 2. Expansive Search Space.

- Guess x_i from $x_{<i}$
- Possibilities for one pos $= |\mathcal{V}|$
- Needs to reduce the search space

Solutions.

(InputSnatch, 2024) Techniques: GaussianNB, Fine-tuned LM ...

($\mathrm{RL}^2\mathrm{eak}$, ICLR 2026) Use SFT + DPO to train a next-token predictor

2-1. Timing difference between cache hit/miss

Requirements.

- ONLY need to send API reqs.
- No need to read memory space or to control the machine
- Same **shared cache** accessed.
- Cache entries **remain resident during attack**

2-2. Scheduling behavior

(I know what you asked (PromptPeak), 2026)

Observation. Prefix KV-cache matching can affect the **request scheduling** behavior.

- Example: SGLang's **Longest Prefix Match (LPM)** mechanism.
- Match lengths -> Return order

Running batch:

```
Imagine you are %  +
Imagine you are %  |-Run
Imagine you are %  +
```

Stored KV cache:

```
Imagine you are an IT expert  (from victim)
Imagine you are %  (from dummy requests)
```

Waiting queue:

```
Imagine you are %  +
Imagine you are %  |
Imagine you are %  |-Pre
Imagine you are %  |
Imagine you are %  +
Imagine you are a  +
Imagine you are an |-Cands
Imagine you are the+
Imagine you are %  +
Imagine you are %  |
Imagine you are %  |-Post
Imagine you are %  |
Imagine you are %  +
...
```

(a) Serving order before LPM.

Running batch:

```
Imagine you are %  +
Imagine you are %  |-Run
Imagine you are %  +
```

Waiting queue:

```
Imagine you are %  +
Imagine you are %  |
Imagine you are %  |-Pre
Imagine you are %  |
Imagine you are %  +
Imagine you are an +-Match
Imagine you are %  +
Imagine you are %  |
Imagine you are %  |-Post
Imagine you are %  |
Imagine you are %  +
Imagine you are a  +
Imagine you are the|-Cands
...
```

(b) Serving order after LPM.

Figure 6: Impact of LPM on the serving order.

2-2. Scheduling behavior

Method. Observe the order of return requests.

- The matched request returns in the middle of the dummy requests.

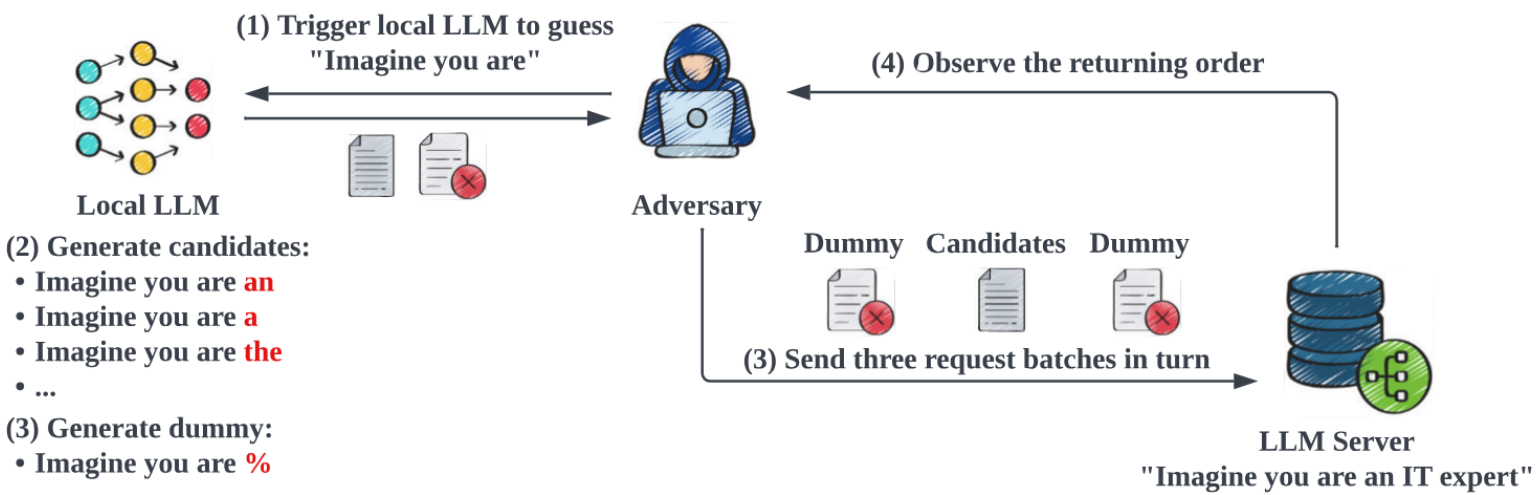


Figure 5: Token extraction in PROMPTPEEK.

2-2. Scheduling behavior

Requirements.

- ONLY need to send API reqs.
- No need to read memory space or to control the machine
- Same **shared cache** accessed.
- Cache entries **remain resident during attack**

Problems.

- Candidate reqs generation.
- **Cache impact**.

2-3. More direct approach: Access KV Cache itself

(Shadow in the Cache, 2026)

Observation. KV Cache is externalized from the TEE's protection boundary.

Key insight. Can we reconstruct input from **direct KV Cache access**?

Yes and No.

- **Inversion Attack**: Not possible unless first layer and square Matrix compt.
- **Collision Attack**: Knowing $x_{<i}$, enum x_i: Problem of $|\mathcal{V}|$.
- "Repeat the prev content."

Introduction to Attack techniques (3)

Network traffic monitoring

Traffic metadata during streaming responses

Observation. TLS encryption

- encrypts the content of communications
- but it leaks **traffic metadata**
- E.g., packet sizes, inter-packet timing

Opportunity. Observe the traffic metadata to induce serving behaviors.

$\Rightarrow$ Passive traffic observation

3-1. Generic traffic fingerprinting

(Whisper Leak, 2025)

Method. Listen to/Monitor the streaming metadata.

Goal. Train a binary classifier to: TLS streaming metadata ⇒ target topic

- Techniques: LightGBM, LSTM-based, BERT-based

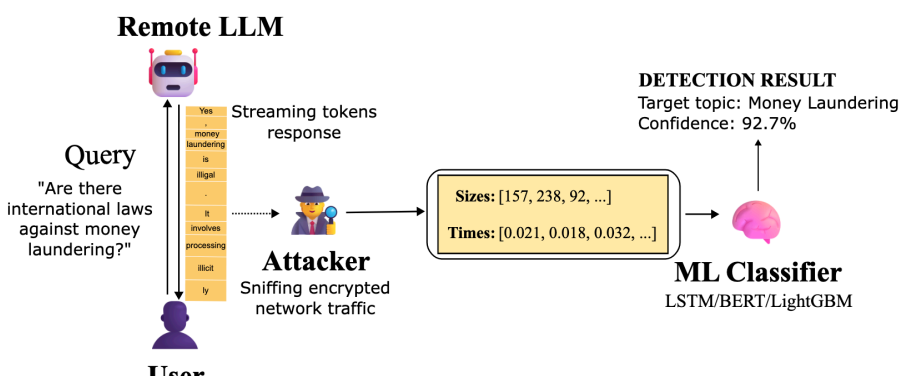


Figure 1: Whisper Leak attack pipeline: A passive network adversary observes encrypted TLS traffic between user and LLM service, extracts packet size and timing sequences, and uses trained classifiers to infer whether the conversation topic matches a sensitive target category.

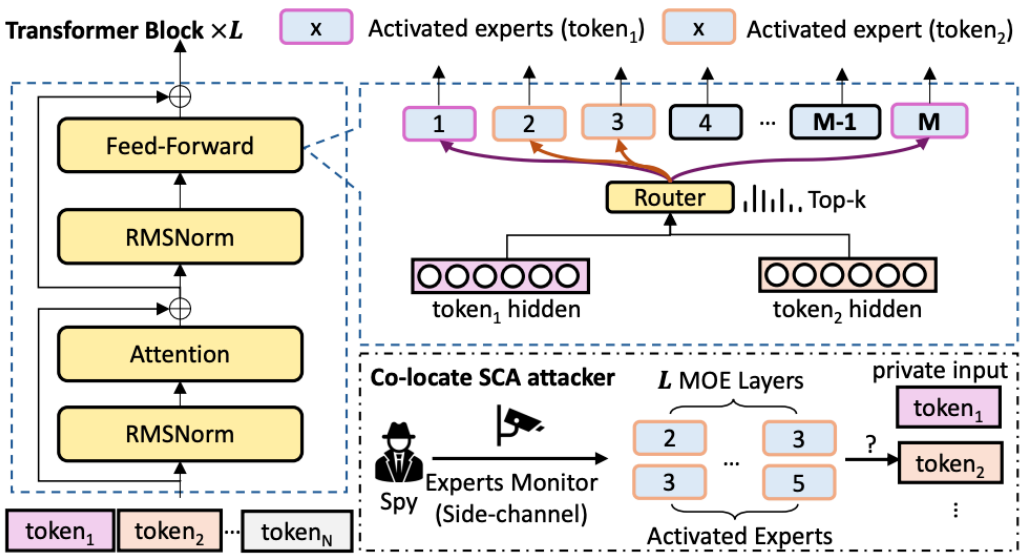
Introduction to Attack techniques (4)

Model specific

4-1. MoE Models

(MoEcho, 2025)

Observation. Expert activation pattern relates to the user inputs.



Opportunity. Observe exec footprints ⇒ recover seq ⇒ infer i/o.

4-1. MoE Models

(MoEcho, 2025)

Method.

1. **Profiling phase**. Learn mapping: expert activation trace ⇒ sensitive label / output token.
2. **Attack phase**. Recover trace via side channels ⇒ match trace ⇒ infer prompt attributes or reconstruct responses.

Table 1: Side-channels on CPU and GPU Platforms

Execution Pattern	Side-channels	
	CPU	GPU
Expert Load L	Cache Occupancy (L1 + L2) *Section 4.1.1*	Performance Counter *Section 4.2.1*
Expert Sequence S	Pageout + Reload *Section 4.1.2*	TLB Evict + Reload *Section 4.2.2*

4-1. MoE Models

Implementation on GPU

For each decoding step ...

- Monitor GPU L3 TLB state
- Apply TLB **Evict+Reload** to expert GPU mem blocks
- Problem:
  - Requires per-token sync
  - DIFFICULT to implement
  - Cnt batching ....?????
  - Pollution to TLB state
  - DEGRADE performance !!!

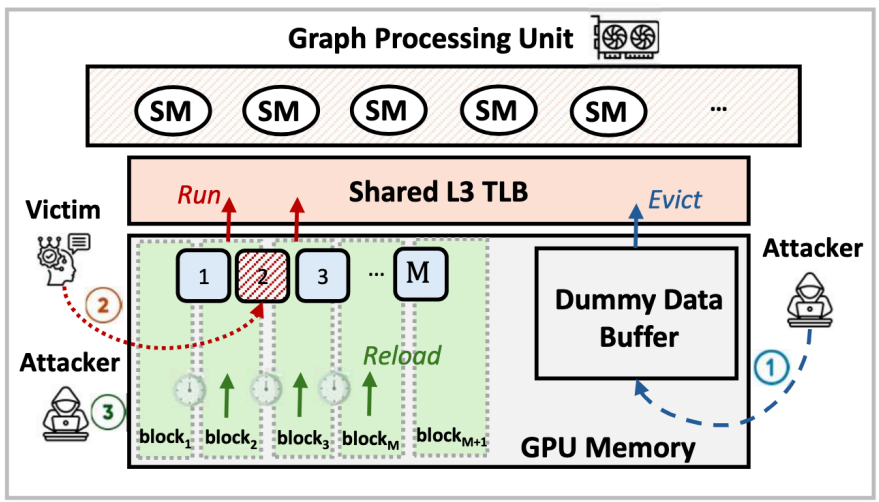


Figure 11: ***TLB Evict+Reload Side-channel.*** **The attacker first loads a dummy data buffer to evict shared L3 TLB. After the victim's execution, the attacker tests the reload time of memory pages corresponding to each expert.**

4-2. Speculative Decoding

(When Speculation Spills Secrets, 2026)

Observations.

- Pattern of correct/incorrect speculations is dependent on the input
- which can be inferred from the variations in packet sizes

Opportunity.

- Packet-size trace ⇒ # tokens generated ⇒ spec patterns ⇒ infer input attr.

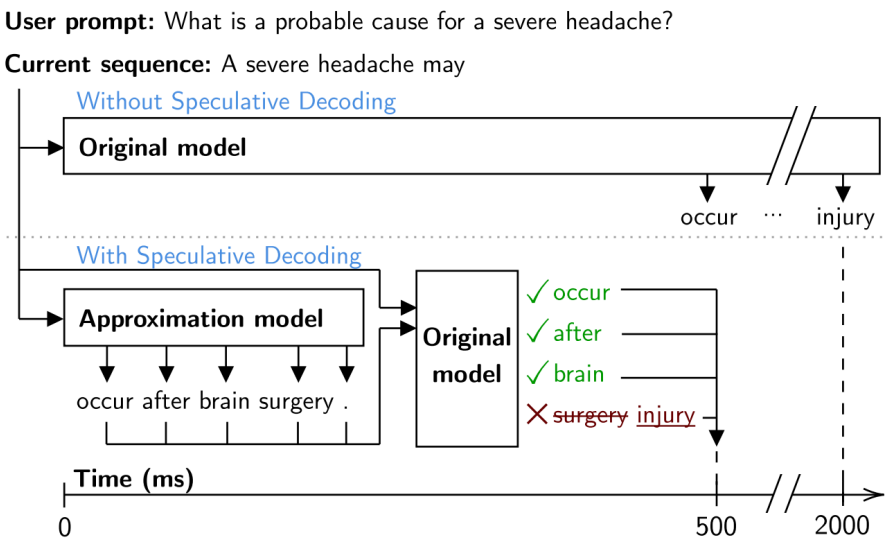
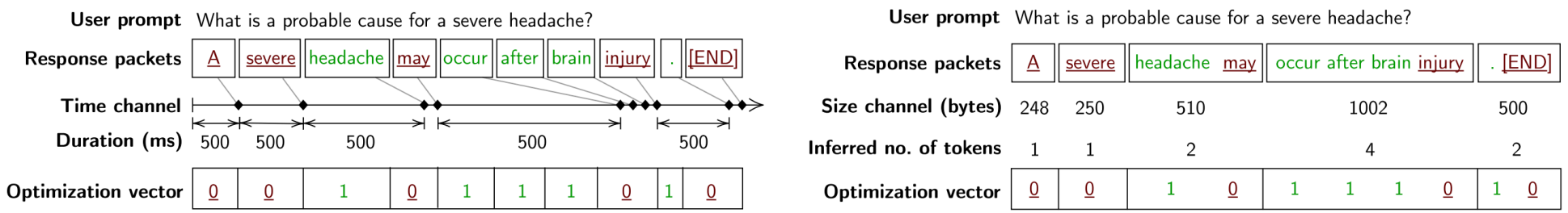


Fig. 1: **Speculative ($\mu = 5$) vs. baseline decoding.** The original model verifies the approximation model-suggested tokens every round and accepts a subset (green checkmarks).

4-2. Speculative Decoding

(Wiretapping LLMs, 2025)



(a) Token-by-token streaming models

(b) Round-by-round streaming models

Fig. 4: **Reconstructing the optimization vector** via (a) the timing channel in token-by-token streaming models (§4.1), and (b) the size channel in round-by-round streaming models (§4.2).

Deconstruct traffic into:

- tokenisize info
- **optimization vector** ⇒ spec exec
- Train a classifier to map these traces to sensitive attr.

Introduction to Attack techniques (5)

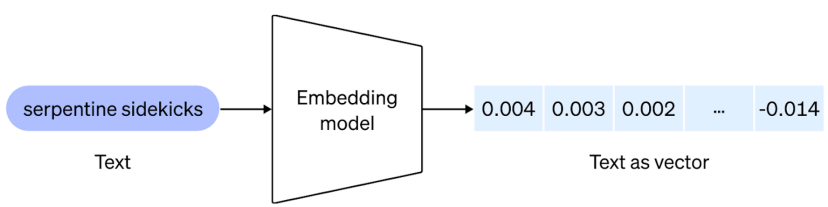
Micro-architectural attack

5-1. Spy process

(I know what you said, 2025) (Spill the Beans, 2025)

Settings. Co-resident proc on the same physical machine.

Observation. Token$[i]$ acceses row $W[i]$ in the embedding matrix.



Opportunity.

- Observe which rows were accessed $\Rightarrow$ recover i/o token.
- Zero-copy loading $\Rightarrow$ spy proc can map the same model file

5-1. Spy process

Method. Flush+Reload

- Use **cache** property: Access fast ⇒ cache hit ⇒ recently used
- **Problem**: $|\mathcal{V}|$ too big ⇒ token-set selection