A Watermark for Large Language Models

John Kirchenbauer, Jonas Geiping, Yuxin Wen Jonathan Katz, Ian Miers, Tom Goldstein
University of Maryland

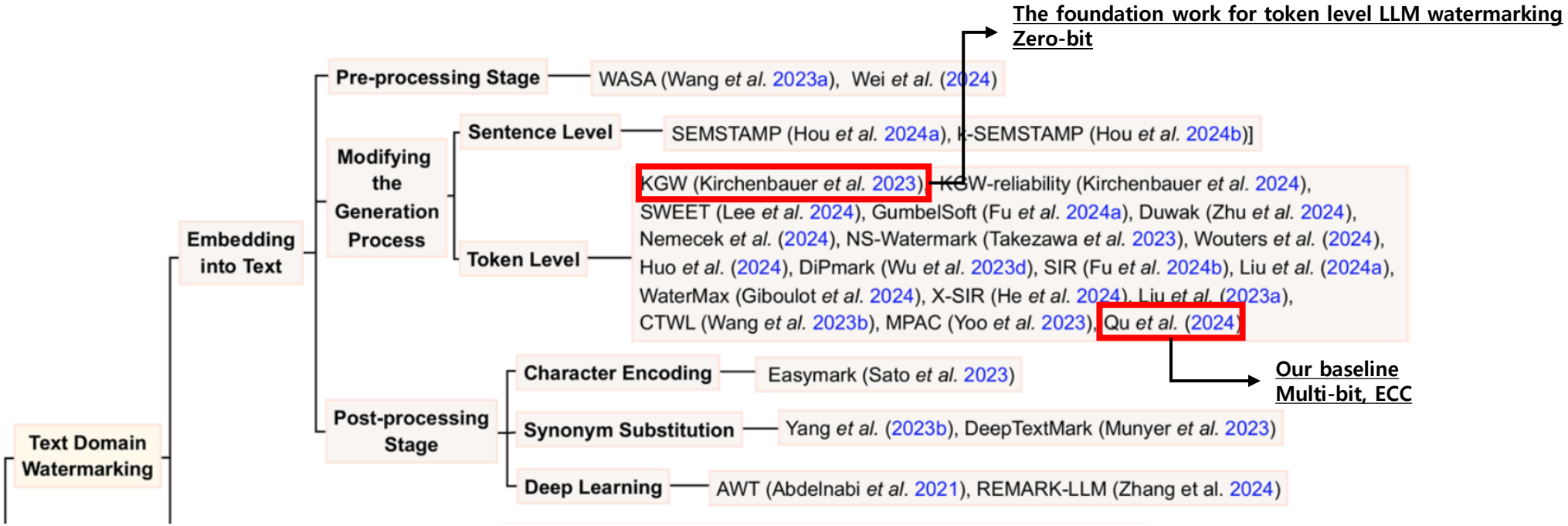
PMLR, 2023

Gahyun Baek

26.05.18

https://www.github.com/jwkirchenbauer/lm-watermarking

LLM watermarking



Watermarking techniques for large language models: a survey, 2026, Liang et al.

Why we need LLM watermarking ?

- **Social engineering & manipulation**
  - Automated bots can amplify persuasion, scams, and election influence campaigns.
- **Fake news and web pollution**
  - LLMs can mass-produce convincing articles, comments, and websites.
- **Academic integrity risks**
  - Students may use AI systems to complete writing or coding assignments unfairly.
- **Training data contamination**
  - This may contaminate training data and reduce the quality of future models.

LLM Watermarking

- **Large capacity**
  - Embeds sufficient watermark information within the generated text.
- **Strong robustness**
  - Maintains the detectability or identifiability of watermarks even under various attacks.
- **High quality**
  - Preserves the naturalness and fluency of the watermarked text.
- **Efficiency**
  - The computational overhead

Qu et al., Provably Robust Multi-Bit Watermarking for AI-Generated Text, USENIX 2025

Background

Entropy means how uncertain the model is.

- When an LLM predicts the next word, it gives many possible choices.
  If many choices look equally likely, entropy is high. (ex. I love ___ .)
  If one choice is clearly best, entropy is low. (ex. The capital of Korea is ___.)

- **Temperature** controls how creative or random the model's choice is.
  - Low temperature → safer, more predictable answers.
    High temperature → more creative, varied answers.

- **Perplexity** measures how surprising a text is under a LM; lower PPL usually indicates more fluent text.

Background

- **Standard generation**
  - input -> tokenization -> embedding -> transformation -> **logits** -> probability conversion -> next-token selection

- **Watermarked generation**
  - input -> tokenization -> embedding -> transformation -> logits -> [watermark **logits** modification] -> probability conversion -> next-token selection

KGW

- **Intuition**
  - At each generation step, the vocabulary is pseudo-randomly split into a green list and a red list.
  - We want the model to generate as many green-list words as possible.
  - This allows us to detect whether a text was generated by an AI model.
  - If the generated text contains significantly more green tokens than expected, the detector can conclude that the text is likely watermarked.
  - The paper proposes two methods for embedding watermark signals into AI-generated text.

Prompt	Num tokens
…The watermark detection algorithm can be made public, enabling third parties (e.g., social media platforms) to run it themselves, or it can be kept private and run behind an API. We seek a watermark with the following properties:	
No watermark Extremely efficient on average term lengths and word frequencies on synthetic, microamount text (as little as 25 words) Very small and low-resource key/hash (e.g., 140 bits per key is sufficient for 99.999999999% of the Synthetic Internet	56
With watermark - minimal marginal probability for a detection attempt. - Good speech frequency and energy rate reduction. - messages indiscernible to humans. - easy for humans to verify.	36

Text Generation with Hard Red List

- If the word(token) is included in the red list, this token can not be chosen.

- This creates a strong watermark but can harm text quality, especially in low-entropy contexts.

Algorithm 1 Text Generation with Hard Red List

Input: prompt, $s^{(-N_p)} \cdots s^{(-1)}$

for $t = 0, 1, \cdots$ **do**

1. Apply the language model to prior tokens $s^{(-N_p)} \cdots s^{(t-1)}$ to get a probability vector $p^{(t)}$ over the vocabulary.

2. Compute a hash of token $s^{(t-1)}$, and use it to seed a random number generator.

3. Using this seed, randomly partition the vocabulary into a "green list" G and a "red list" R of equal size.

4. Sample $s^{(t)}$ from G , never generating any token in the red list.

end for

Text Generation with Soft Red List

- Soft red-list variant: add a positive bias δ to the logits of all green-list tokens.
- Red-list tokens are still possible, but green-list tokens become more likely.
- This better preserves quality than hard red list.

Algorithm 2 Text Generation with Soft Red List

Input: prompt, $s^{(-N_p)} \dots s^{(-1)}$
green list size, $\gamma \in (0, 1)$
hardness parameter, $\delta > 0$

for $t = 0, 1, \cdots$ **do**

1. Apply the language model to prior tokens $s^{(-N_p)} \dots s^{(t-1)}$ to get a logit vector $l^{(t)}$ over the vocabulary.
2. Compute a hash of token $s^{(t-1)}$, and use it to seed a random number generator.
3. Using this random number generator, randomly partition the vocabulary into a "green list" G of size $\gamma|V|$, and a "red list" R of size $(1-\gamma)|V|$.
4. Add δ to each green list logit. Apply the softmax operator to these modified logits to get a probability distribution over the vocabulary.

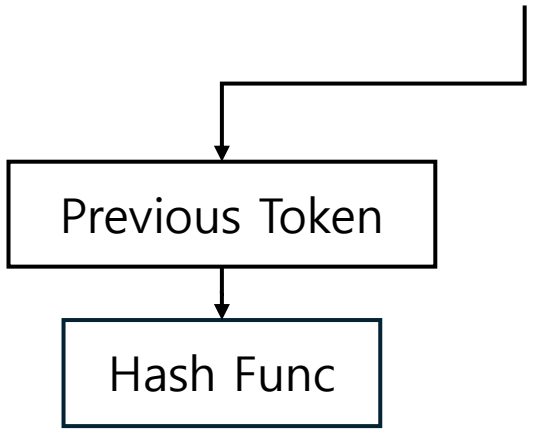
$$\hat{p}_k^{(t)} = \begin{cases} \frac{\exp(l_k^{(t)}+\delta)}{\sum_{i\in R}\exp(l_i^{(t)})+\sum_{i\in G}\exp(l_i^{(t)}+\delta)}, & k \in G \\ \frac{\exp(l_k^{(t)})}{\sum_{i\in R}\exp(l_i^{(t)})+\sum_{i\in G}\exp(l_i^{(t)}+\delta)}, & k \in R. \end{cases}$$

5. Sample the next token, $s^{(t)}$, using the watermarked distribution $\hat{p}^{(t)}$.

end for

Generating Greenlist & Detecting

A cat is sitting on **the** ___ .



Selected Green Tokens ⊆ Vocabulary

γ= green list ratio
γ= 0.5

	logits		biased logits
couch	2.3		**4.3**
chair	1.2	+ δ	3.2
mat	2.1		4.1
table	1.1		1.1
roof	0.8		0.8
blue	**3.3**		3.3

A cat is sitting on the **couch**.

KGW

Dataset: C4
Model: OPT-1.3B

prompt	real completion	no watermark (NW)	watermarked (W)	S	(W) z	(NW) PPL	(W) PPL
...tled out of court and publicly reconciled.\nIn the '80s the band's popularity waned in the United States but remained strong abroad. Robin released three solo albums, with limited success. The Bee Gees	returned with some moderate hits in the late 1990s and were inducted into the Rock and Roll Hall of Fame in 1997. With his brothers, Mr. Gibb won six Grammys.\nIn addition to his wife and his brother [...continues]	continued to tour, and Barry became a television producer.\nBut in the early '90s, the Bee Gees' popularity remained high. They scored a hit with "Don't Stop Believing" in 1990, and in 1992 the Bee Ge[...continues]	' 1990 album, "Spirits of the Century," was a mixed critical and commercial success.\nWhen the brothers were nominated for a Grammy Award in 1990, Mr. Gibb's "You Should Be Dancing" and "Massachusetts,[...continues]	0.68	12.73	3.15	1.93
...cond season at Hall Bros Oval.\nThe defender also admitted his surprise at Young's run to the finals but credited the injection of youth into the side.\n"We were really in a building phase last year and	we copped a few floggings with all those juniors blokes coming in," Galvin said.\n"Now, we've kept that core group together for two years and I think we've come along quicker than we anticipated.\nROCK[...continues]	we copped a few floggings with all those juniors blokes coming in," Galvin said.\n"Now, we've kept that core group together for two years and I think we've come along quicker than we anticipated.\n"Tha[...continues]	we copped a few floggings with all those juniors blokes coming in," Galvin said.\n"Now, we've kept that core group together for two years and I think we've come along quicker than we anticipated.\n"Tha[...continues]	0.58	-1.13	1.05	1.04

Provably Robust Multi-Bit Watermarking for AI-Generated Text

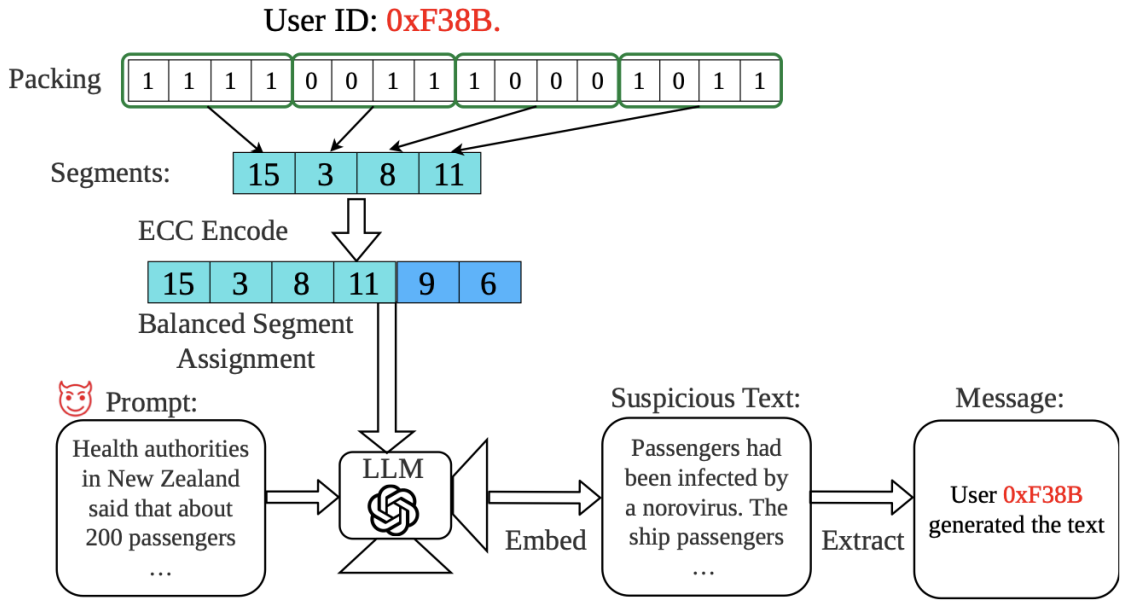
Wenjie Qu, Wengrui Zheng, Tianyang Tao, Dong Yin, Yanze Jiang, Zhihua Tian, Wei Zou, Jinyuan Jia, Jiaheng Zhang
National University of Singapore, Pennsylvania State University

USENIX, 2025

Qu et al.’s segment-watermark

- The limitations of the existing LLM watermarking research.
  - • It mainly detects whether a text is watermarked.
  - • It does not directly recover a long payload such as a user ID.

They try to embed multi-bit watermarking !



Design

Balanced segment assignment

- Maps previous tokens to segment IDs while balancing token-frequency mass across segments. (token ID – Segment ID)
- eliminating the imbalance in pseudo-random segment assignment

- **Adopting error-correction code**
  - Encodes message segments before embedding, so the detector can correct some segment errors after edits.
  - utilizing error-correction codes (ECC) to encode segments before embedding them into the text

Message-Segmentation

- Message: 01100011

01	10	00	11

- Segment: 1 | 2 | 0 | 3
- Segment bit: 2
- Segment num: 4

```
python3 main.py --model_name <model_name> \
    --prompt_path "path/to/dataset" --nsamples <sample_num> --batch_size 1 \
    --method rsbh --temperature 1.0 --seeding hash --ngram 1 --method_detect same --scoring_method none \
    --payload_mode random --payload_max <payload_max> --gamma 0.5 --delta 6.0 --max_gen_len <T> \
    --bh_map_load_path "path/to/map_freq.pkl" \
    --gf_segments_num <n> --segments_num <k> --segment_bit <m> \
    --output_dir "path/to/folder_saving_results(2 jsonl files)"
```

- gamma: green list ratio
- delta: bias added to green token logit

Message-Segmentation

Message: 01100011

01	10	00	11

Segment: 1 | 2 | 0 | 3

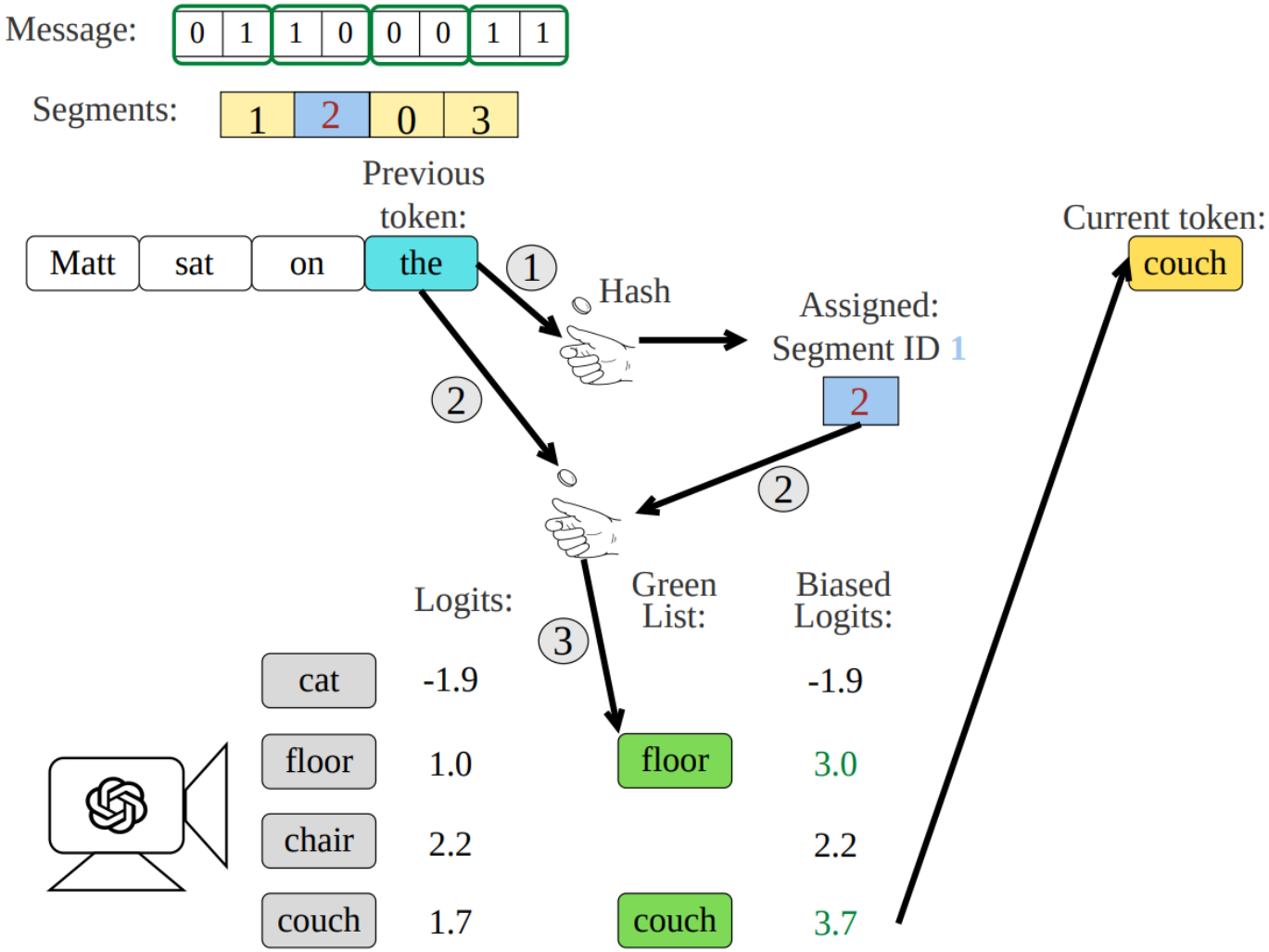
- Segment bit: 2
- Segment num: 4

token	Token ID	Segment ID
the	279	1
,	11	0
water	5237	2
and	323	3
lake	8451	0
blue	6437	1
is	374	3
	...	

[Balanced segment assignment]

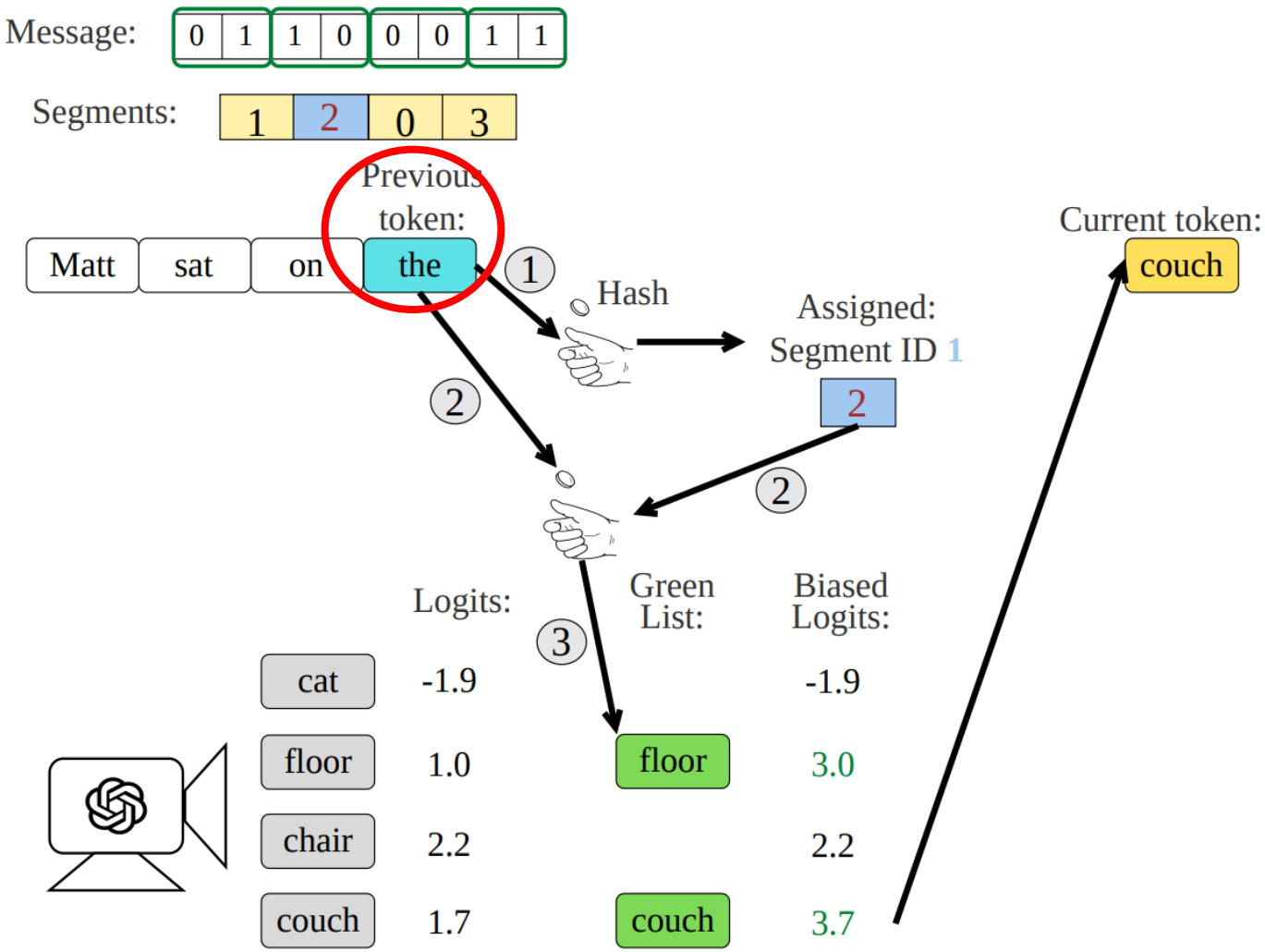
```
python3 main.py --model_name <model_name> \
    --prompt_path "path/to/dataset" --nsamples <sample_num> --batch_size 1 \
    --method rsbh --temperature 1.0 --seeding hash --ngram 1 --method_detect same --scoring_method none \
    --payload_mode random --payload_max <payload_max> --gamma 0.5 --delta 6.0 --max_gen_len <T> \
    --bh_map_load_path "path/to/map_freq.pkl" \
    --gf_segments_num <n> --segments_num <k> --segment_bit <m> \
    --output_dir "path/to/folder_saving_results(2 jsonl files)"
```

Generate


Message:
0 1 1 0 0 0 1 1
Segments:
1 2 0 3
Previous token:
Matt sat on the
1
Hash
Assigned: Segment ID 1
2
2
2
Current token:
couch
Logits:
Green List:
Biased Logits:
3
cat -1.9 -1.9
floor 1.0 floor 3.0
chair 2.2 2.2
couch 1.7 couch 3.7

Generate

Previous token
X_{t-1} = "the"

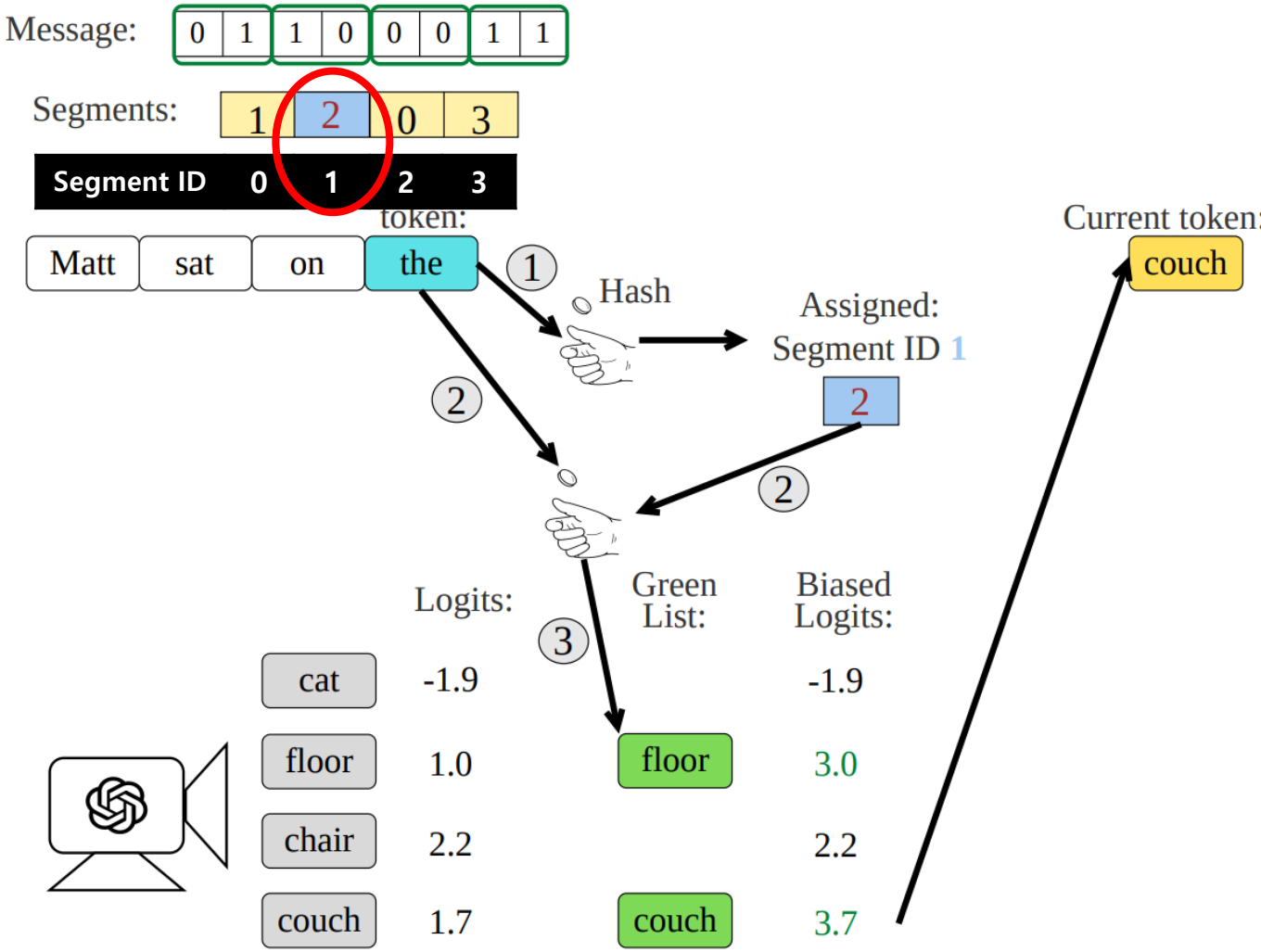


Generate

Previous token
X_{t-1} = "the"

Balanced segment map
$Map[\text{the}] = 1$

token	Token ID	Segment ID
the	**279**	**1**
,	11	0
water	5237	2
and	323	3
lake	8451	0
blue	6437	1
is	374	3
	⋮	

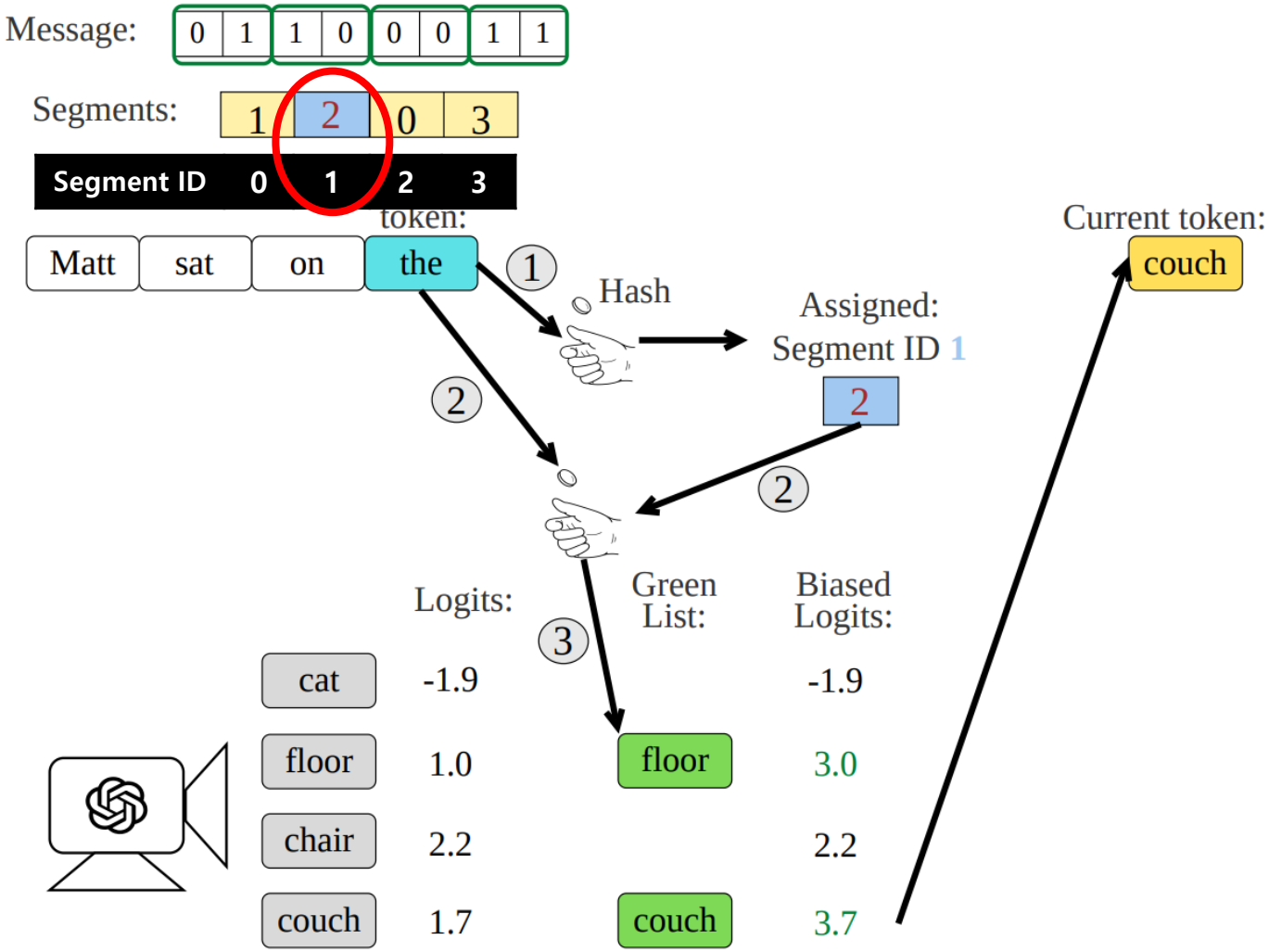


Generate

Previous token
X_{t-1} = "the"

Balanced segment map
$Map[\text{the}] = 1$

Select the segment ID
SegmentID = 1



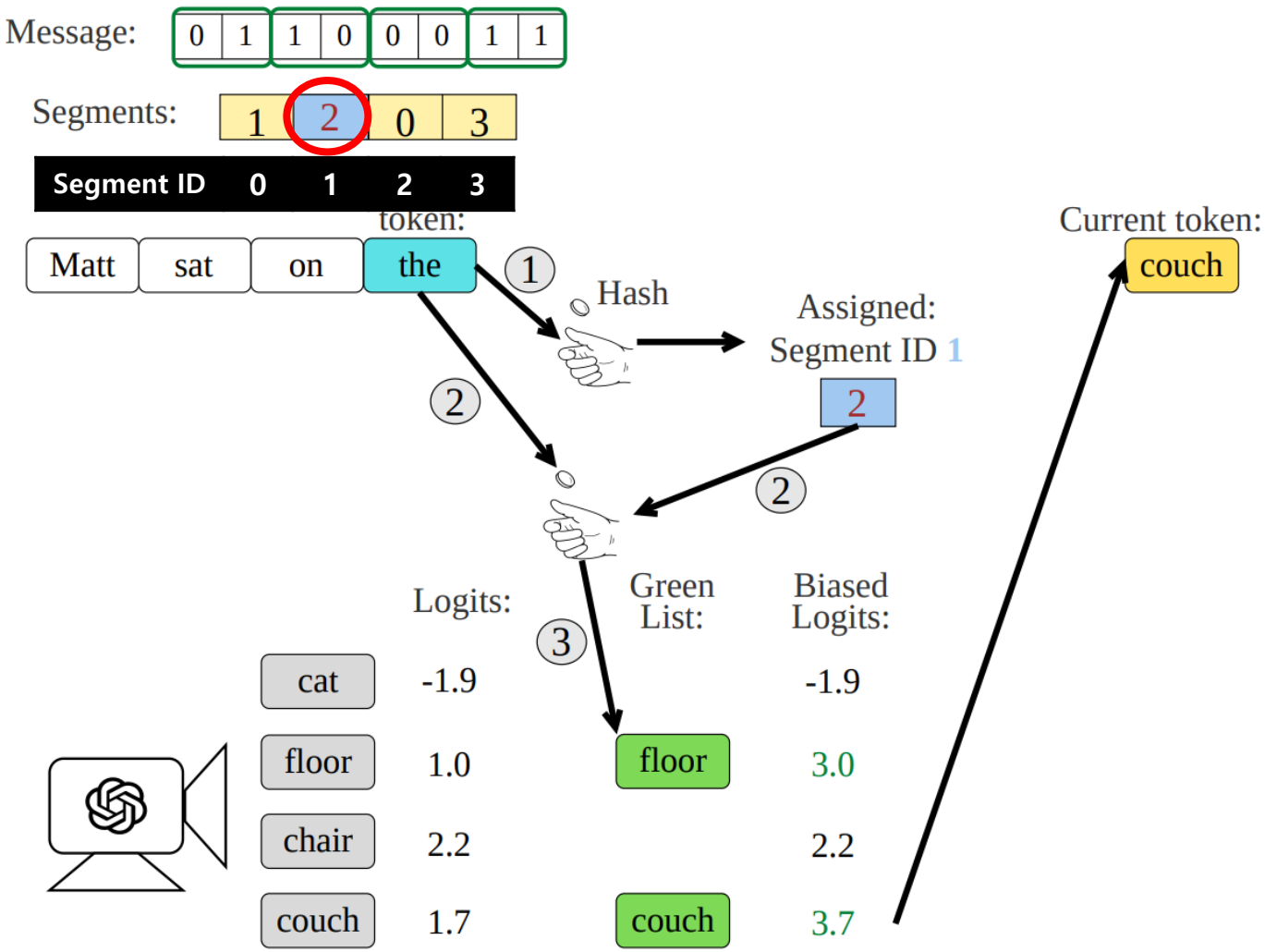
Generate

Previous token
X_{t-1} = "the"

Balanced segment map
$Map[\text{the}] = segment\ 1$

Select the segment ID
$SegmentID\ j = 1$

Read the segment Value
Segment_value = 2



Generate

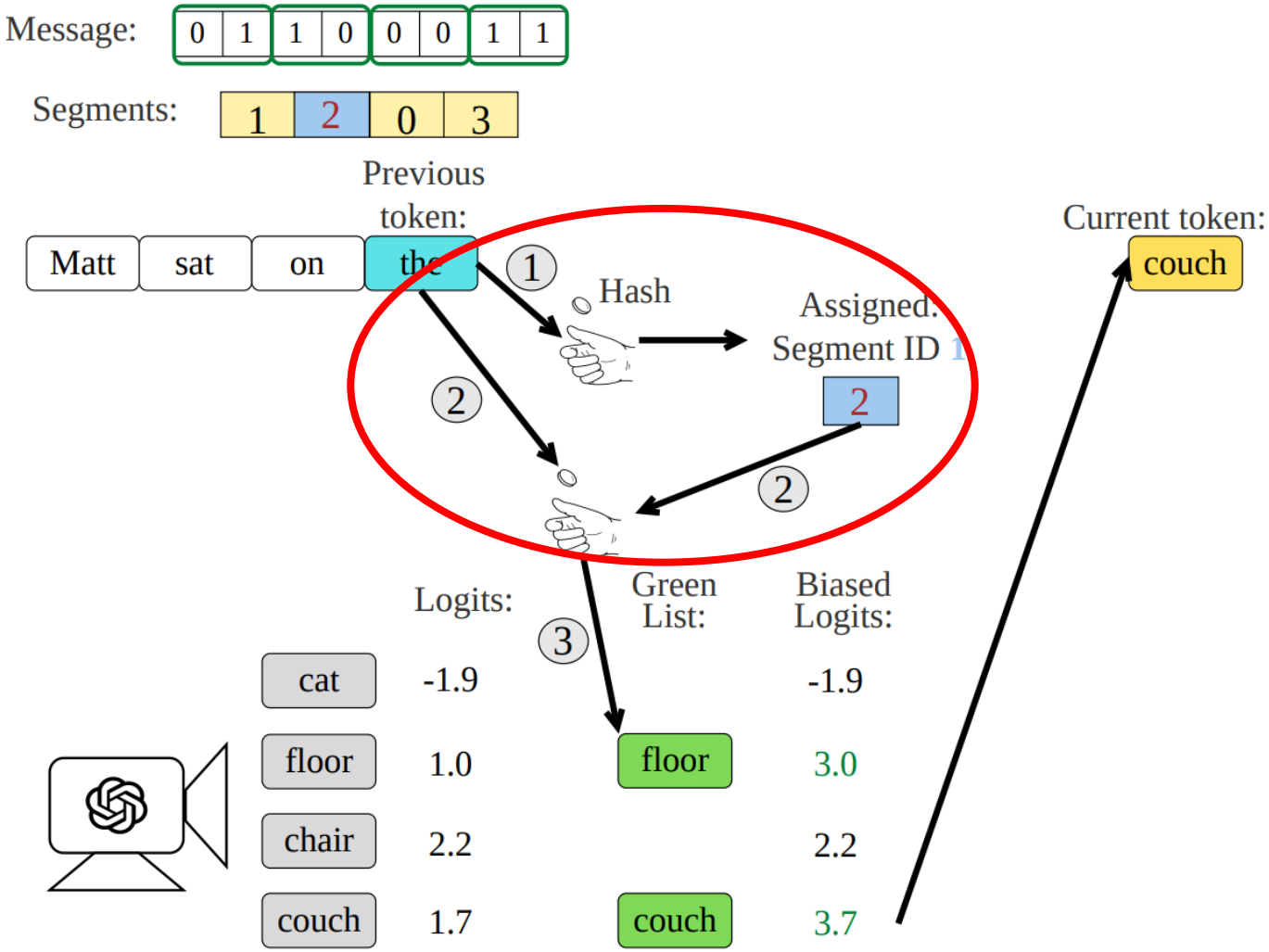
Previous token
X_{t-1} = "the"

Balanced segment map
Map[the] = $segment$ 1

Select the segment ID
$SegmentID\ j$ = 1

Segment_value = 2

Seed = hash(segment_value, prev.token, private_key)



Generate

Previous token
X_{t-1} = "the"

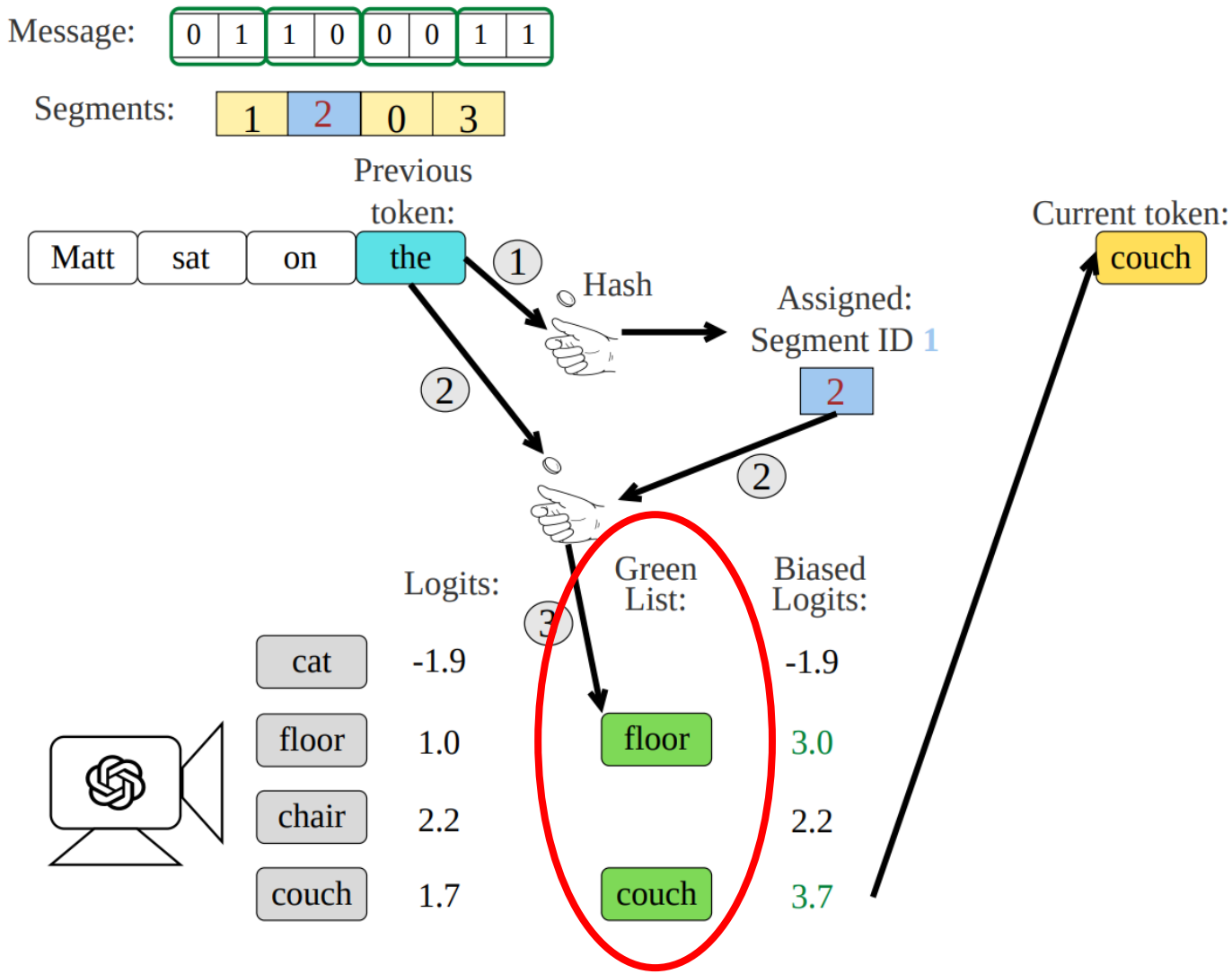
Balanced segment map
$Map[\text{the}] = segment2$

Select the segment ID
$SegmentID\ j = 2$

Segment_value = 1

Seed = hash(segment_value, prev.token, private_key)

Using seed generates Green list



Generate

Previous token
X_{t-1} = "the"

Balanced segment map
$Map[\text{the}] = segment2$

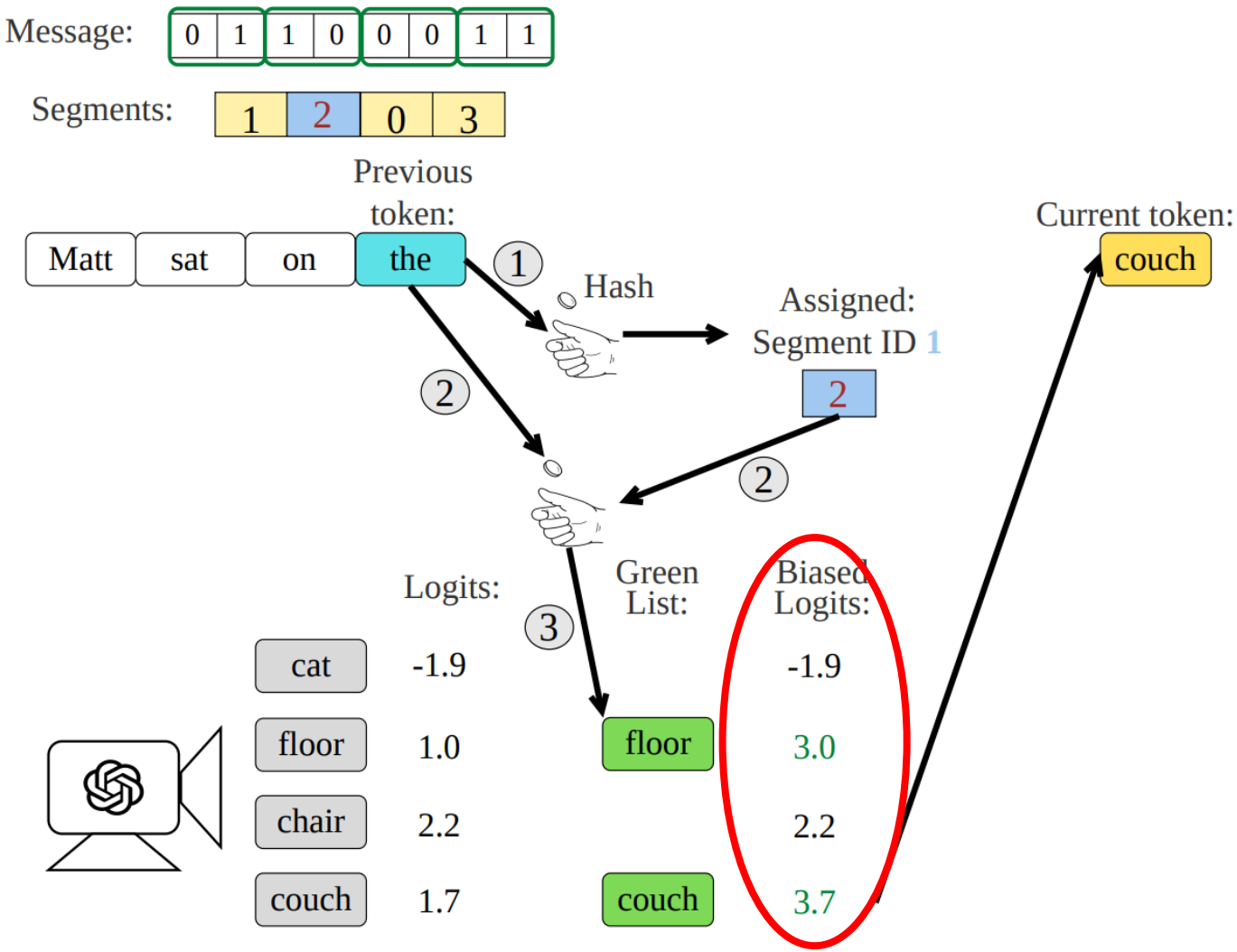
Select the segment ID
$SegmentID\ j = 2$

Segment_value = 1

Seed = hash(segment_value, prev.token, private_key)

Using seed generates Green list

Adding delta
Delta = 2.0



Generate

Previous token
X_{t-1} = "the"

Balanced segment map
$Map[\text{the}] = segment2$

Select the segment ID
$SegmentID\ j = 2$

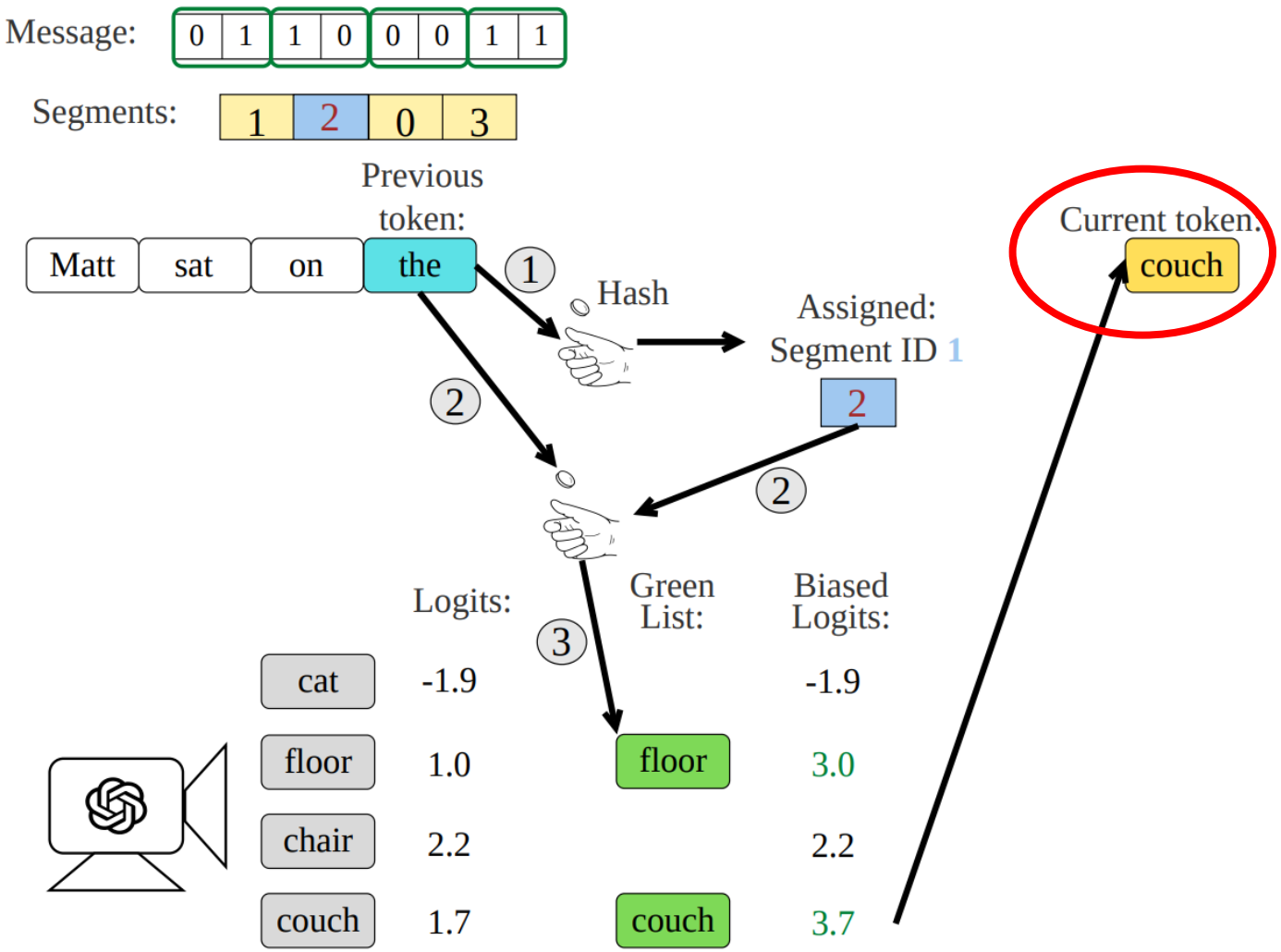
Segment_value = 1

Seed = hash(segment_value, prev.token, private_key)

Using seed generates Green list

Adding delta

Sampling the next token



Detect

Matt sat on the couch.

suspicious text

Detect

Matt sat on **the** **couch**.

We want to inspect whether this token is watermarked or not.

suspicious text

$(the, couch)$

$token_{t-1}, token_t$

Detect

Matt sat on **the** **couch**. We want to inspect whether this token is watermarked or not.

suspicious text

$(the, couch)$
$token_{t-1}, token_t$

token	Token ID	Segment ID
the	**279**	**1**
,	11	0
water	5237	2
and	323	3
lake	8451	0
blue	6437	1
is	374	3
	⋮	

Detect

Matt sat on **the** **couch**. We want to inspect whether this token is watermarked or not.

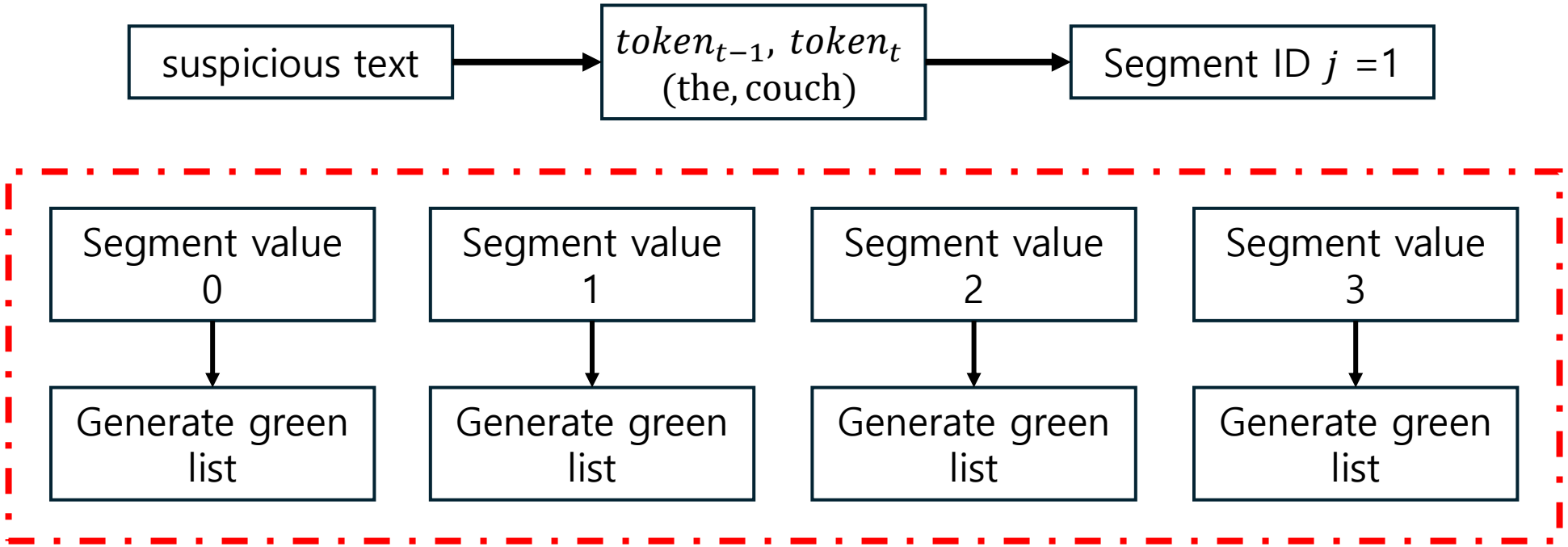
suspicious text

$token_{t-1}$, $token_t$
(the, couch)

Segment ID j =1

Detect

Matt sat on **the couch**.



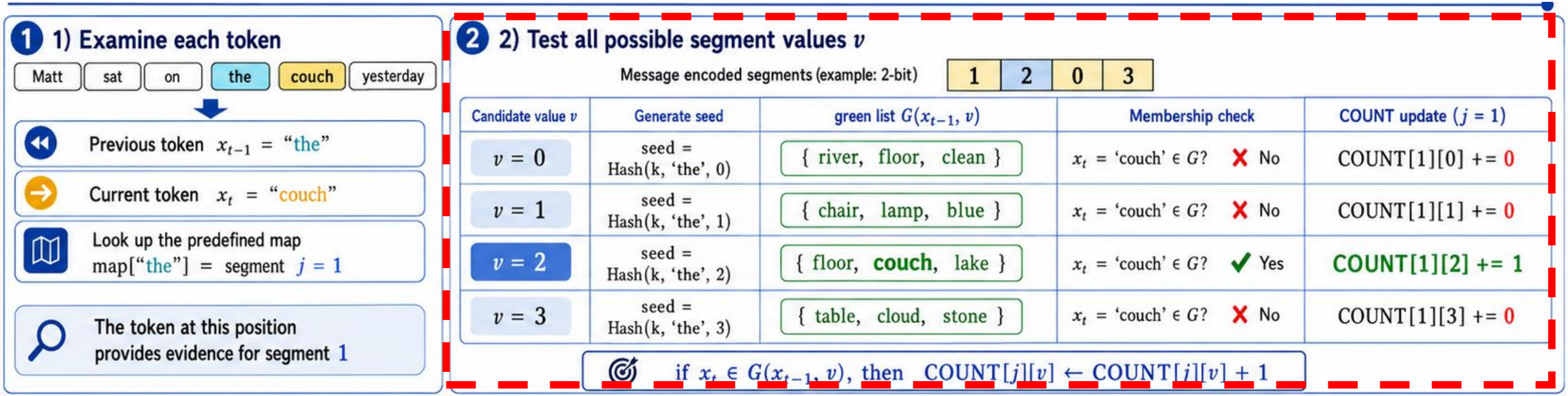
- Message: 01100011

01	10	00	11

- Segment: 1 | 2 | 0 | 3
- Segment bit: 2
- Segment num: 4

Generating green list:
Seed = hash(segment_value, prev.token, private_key)

Detect


1) Examine each token
Matt
sat
on
the
couch
yesterday
Previous token x_{t-1} = "the"
Current token x_t = "couch"
Look up the predefined map
map["the"] = segment $j = 1$
The token at this position
provides evidence for segment 1
2) Test all possible segment values v
Message encoded segments (example: 2-bit)
1
2
0
3
Candidate value v
Generate seed
green list $G(x_{t-1}, v)$
Membership check
COUNT update ($j = 1$)
$v = 0$
seed = Hash(k, 'the', 0)
{ river, floor, clean }
x_t = 'couch' ∈ G? ✗ No
COUNT[1][0] += 0
$v = 1$
seed = Hash(k, 'the', 1)
{ chair, lamp, blue }
x_t = 'couch' ∈ G? ✗ No
COUNT[1][1] += 0
$v = 2$
seed = Hash(k, 'the', 2)
{ floor, couch, lake }
x_t = 'couch' ∈ G? ✓ Yes
COUNT[1][2] += 1
$v = 3$
seed = Hash(k, 'the', 3)
{ table, cloud, stone }
x_t = 'couch' ∈ G? ✗ No
COUNT[1][3] += 0
if $x_t \in G(x_{t-1}, v)$, then COUNT[j][v] ← COUNT[j][v] + 1

Detect

1) Examine each token

Matt | sat | on | **the** | **couch** | yesterday

Previous token x_{t-1} = "the"

Current token x_t = "couch"

Look up the predefined map
map["the"] = segment $j = 1$

The token at this position provides evidence for segment 1

2) Test all possible segment values v

Message encoded segments (example: 2-bit) **1 | 2 | 0 | 3**

Candidate value v	Generate seed	green list $G(x_{t-1}, v)$	Membership check	COUNT update ($j = 1$)
$v = 0$	seed = Hash(k, 'the', 0)	{ river, floor, clean }	x_t = 'couch' ∈ G? ✗ No	COUNT[1][0] += 0
$v = 1$	seed = Hash(k, 'the', 1)	{ chair, lamp, blue }	x_t = 'couch' ∈ G? ✗ No	COUNT[1][1] += 0
$v = 2$	seed = Hash(k, 'the', 2)	{ floor, **couch**, lake }	x_t = 'couch' ∈ G? ✓ Yes	**COUNT[1][2] += 1**
$v = 3$	seed = Hash(k, 'the', 3)	{ table, cloud, stone }	x_t = 'couch' ∈ G? ✗ No	COUNT[1][3] += 0

if $x_t \in G(x_{t-1}, v)$, then $\text{COUNT}[j][v] \leftarrow \text{COUNT}[j][v] + 1$

3) Accumulate the COUNT matrix

Repeat for all token positions

COUNT$[j][v]$

	$v = 0$	$v = 1$	$v = 2$	$v = 3$
segment $j = 0$	2	**8**	3	1
segment $j = 1$	3	5	**12**	4
segment $j = 2$	**9**	1	2	3
segment $j = 3$	2	4	3	**10**

For each segment j, the value in red receives the most support.

Detect

1) Examine each token

Matt | sat | on | the | couch | yesterday

- Previous token x_{t-1} = "the"
- Current token x_t = "couch"
- Look up the predefined map map["the"] = segment $j = 1$
- The token at this position provides evidence for segment 1

2) Test all possible segment values v

Message encoded segments (example: 2-bit): 1 | 2 | 0 | 3

Candidate value v	Generate seed	Green list $G(x_{t-1}, v)$	Membership check	COUNT update ($j = 1$)
$v = 0$	seed = Hash(k, 'the', 0)	{ river, floor, clean }	x_t = 'couch' ∈ G? ✗ No	COUNT[1][0] += 0
$v = 1$	seed = Hash(k, 'the', 1)	{ chair, lamp, blue }	x_t = 'couch' ∈ G? ✗ No	COUNT[1][1] += 0
$v = 2$	seed = Hash(k, 'the', 2)	{ floor, **couch**, lake }	x_t = 'couch' ∈ G? ✓ Yes	**COUNT[1][2] += 1**
$v = 3$	seed = Hash(k, 'the', 3)	{ table, cloud, stone }	x_t = 'couch' ∈ G? ✗ No	COUNT[1][3] += 0

If $x_t \in G(x_{t-1}, v)$, then $\text{COUNT}[j][v] \leftarrow \text{COUNT}[j][v] + 1$

3) Accumulate the COUNT matrix

COUNT[j][v]

	$v = 0$	$v = 1$	$v = 2$	$v = 3$
segment $j = 0$	2	**8**	3	1
segment $j = 1$	3	5	**12**	4
segment $j = 2$	**9**	1	2	3
segment $j = 3$	2	4	3	**10**

Repeat for all token positions

For each segment j, the value in red receives the most support.

4) Segment-wise argmax and final recovery

$j = 0$	argmax_v [2, 8, 3, 1] =	1
$j = 1$	argmax_v [3, 5, 12, 4] =	2
$j = 2$	argmax_v [9, 1, 2, 3] =	0
$j = 3$	argmax_v [2, 4, 3, 10] =	3

$\hat{m}_j = \text{argmax}_v \ \text{COUNT}[j][v]$

Estimated encoded segments

[1, 2, 0, 3]

ECC / Reed–Solomon decoding

Recover final payload

Recovered segment values:

[1, 2, 0, 3]

Recovered message bits (2-bit per segment):

01 | 10 | 00 | 11 = 1203

Successfully recovered!

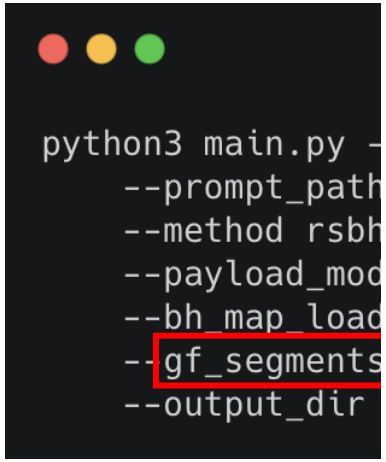
Reed Solomon Error Correction

```
python3 main.py --model_name <model_name> \
    --prompt_path "path/to/dataset" --nsamples <sample_num> --batch_size 1 \
    --method rsbh --temperature 1.0 --seeding hash --ngram 1 --method_detect same --scoring_method none \
    --payload_mode random --payload_max <payload_max> --gamma 0.5 --delta 6.0 --max_gen_len <T> \
    --bh_map_load_path "path/to/map_freq.pkl" \
    --gf_segments_num <n> --segments_num <k> --segment_bit <m> \
    --output_dir "path/to/folder_saving_results(2 jsonl files)"
```

Redundant segments + original segments

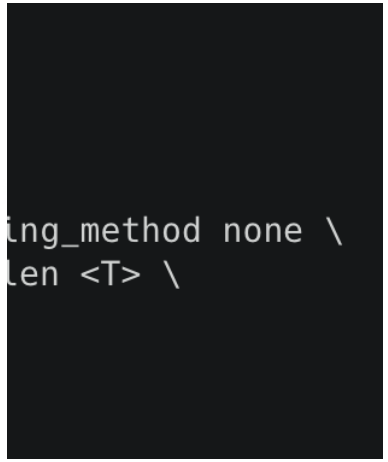
segments_num: 4
segment_bit: 2
gf_segments_num: (2 prity code + 4 segments_num) = 6

This function C is a linear mapping, that is, it satisfies $C(m) = mA$ for the following $k \times n$-matrix A with elements from F.



$$C(m) = mA = [\, m_0 \quad m_1 \quad m_2 \quad \cdots \quad m_{k-1} \,] \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ a_0 & a_1 & a_2 & \dots & a_{n-1} \\ a_0^2 & a_1^2 & a_2^2 & \dots & a_{n-1}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_0^{k-1} & a_1^{k-1} & a_2^{k-1} & \dots & a_{n-1}^{k-1} \end{bmatrix}$$

This matrix is a Vandermonde matrix over F. In other words, the Reed–Solomon code is a linear code, and in the classical encoding procedure, its generator matrix is A.



Systematic encoding procedure: The message as an initial sequence of values [edit]

There are alternative encoding procedures that produce a systematic Reed–Solomon code. One method uses Lagrange interpolation to compute polynomial p_m such that

$$p_m(a_i) = m_i \text{ for all } i \in \{0, \dots, k-1\}.$$

Then p_m is evaluated at the other points $a_k, \dots, a_{n-1}$.

$$C(m) = [\, p_m(a_0) \quad p_m(a_1) \quad \cdots \quad p_m(a_{n-1}) \,]$$

This function C is a linear mapping. To generate the corresponding systematic encoding matrix G, multiply the matrix A by the inverse of A's left square submatrix.

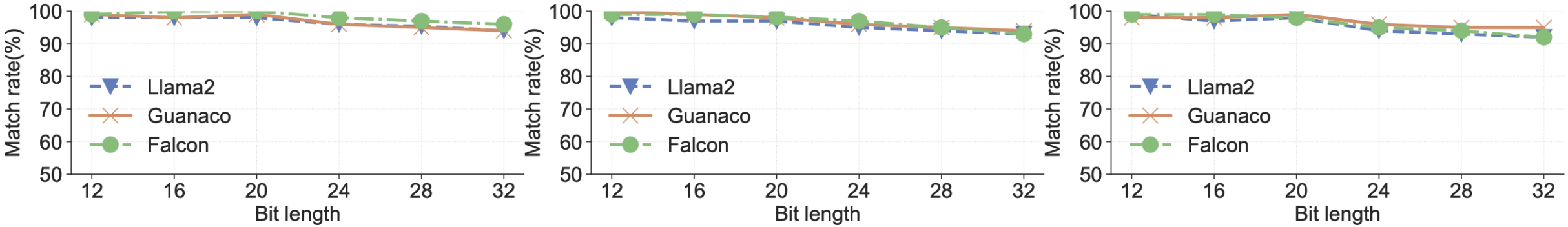
$$G = (A\text{'s left square submatrix})^{-1} \cdot A = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & g_{1,k+1} & \cdots & g_{1,n} \\ 0 & 1 & 0 & \dots & 0 & g_{2,k+1} & \cdots & g_{2,n} \\ 0 & 0 & 1 & \dots & 0 & g_{3,k+1} & \cdots & g_{3,n} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & \dots & 1 & g_{k,k+1} & \cdots & g_{k,n} \end{bmatrix}$$

$C(m) = mG$ for the following $k \times n$-matrix G with elements from F.

$$C(m) = mG = [\, m_0 \quad m_1 \quad m_2 \quad \cdots \quad m_{k-1} \,] \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & g_{1,k+1} & \cdots & g_{1,n} \\ 0 & 1 & 0 & \dots & 0 & g_{2,k+1} & \cdots & g_{2,n} \\ 0 & 0 & 1 & \dots & 0 & g_{3,k+1} & \cdots & g_{3,n} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & \dots & 1 & g_{k,k+1} & \cdots & g_{k,n} \end{bmatrix}$$

Results

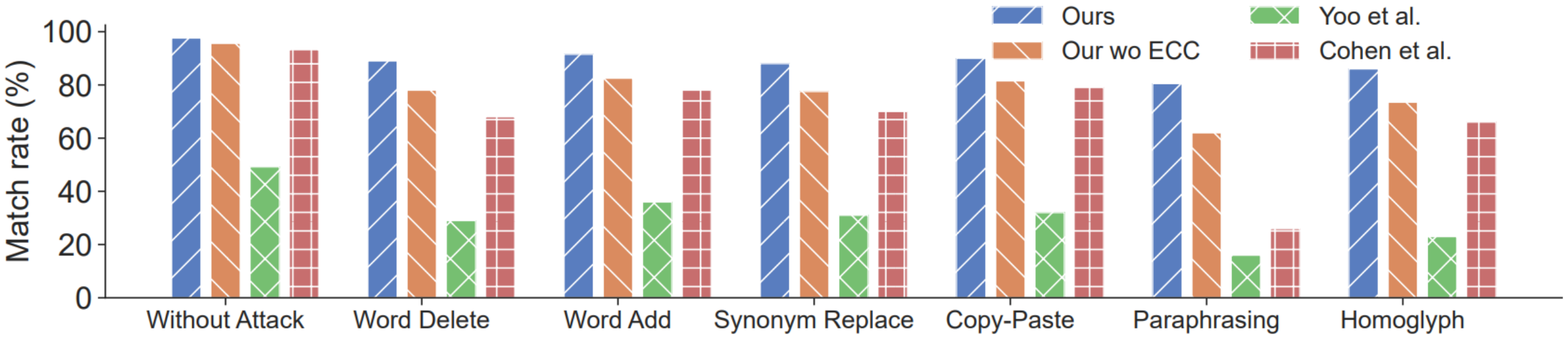
- **Dataset**: OpenGen, C4, Essays
- **Model**: Llama2, Guanaco, Falcon



(a) OpenGen (b) C4 (c) Essays

Figure 3: Match rate of our method on different datasets, LLMs, and bit lengths.

Edit Attack


Ours
Our wo ECC
Yoo et al.
Cohen et al.
Match rate (%)
100
80
60
40
20
0
Without Attack
Word Delete
Word Add
Synonym Replace
Copy-Paste
Paraphrasing
Homoglyph

Thank You