LLM Training and Coding for Statistical Learning and Data Science

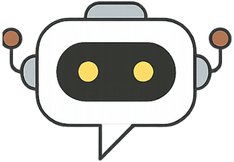
Part 2: API Usage and Text Watermarking

Xiang Li

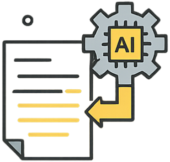
University of Pennsylvania

Nov. 5, 2025

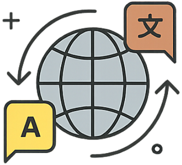
LLMs in everyday use



Chatbots

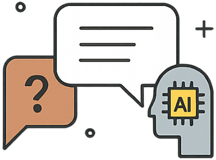


Text generration

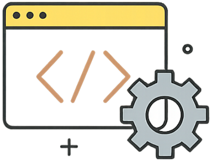


Language translation

Sentiment analysis



Question answering



Code generation

Bringing LLMs into everyday applications

- ChatGPT and other LLMs are powerful, but web interfaces limit automation.
- To integrate LLMs into software, we use an **API** (Application Programming Interface).
- APIs let developers send text prompts and receive model-generated responses.
- Conceptually, an API treats the LLM as a **black box** that returns outputs in a **function-calling** manner.
- This enables customized assistants, summarizers, or analytic tools.

Applications built on the ChatGPT API

- **Chatbots and Assistants:** Integrations like Notion AI and Instacart's shopping assistant use the API to answer user queries and automate workflows.
- **Content and Writing Tools:** Tools such as Jasper and Copy.ai use the API for marketing text, article generation, and idea expansion.
- **Developer Tools:** Examples include GitHub Copilot for code suggestions and various OpenAI project demos for SQL translation or tutoring.
- **Summarization and Knowledge Extraction:** Apps like Yext use the API to build domain-specific QA systems from large document sets.
- **Specialized Research and Healthcare:** For instance, researchers have used the API to aid in systematic clinical review screening.

Applications built on the ChatGPT API

- **Chatbots and Assistants:** Integrations like Notion AI and Instacart's shopping assistant use the API to answer user queries and automate workflows.
- **Content and Writing Tools:** Tools such as Jasper and Copy.ai use the API for marketing text, article generation, and idea expansion.
- **Developer Tools:** Examples include GitHub Copilot for code suggestions and various OpenAI project demos for SQL translation or tutoring.
- **Summarization and Knowledge Extraction:** Apps like Yext use the API to build domain-specific QA systems from large document sets.
- **Specialized Research and Healthcare:** For instance, researchers have used the API to aid in systematic clinical review screening.

API is not everything

The ChatGPT API acts as a reasoning engine: it generates intelligent responses, while the whole application handles more things such as context, interaction, and control.

Getting started with the ChatGPT API

1. Create an OpenAI account via `auth.openai.com/create-account`.
2. **Get your API key** via `platform.openai.com/account/api-keys`.

NAME	SECRET KEY	LAST USED ⓘ	CREATED BY	PERMISSIONS
Secret key	sk-...2GQA	Never	Xiang Li	All

3. Store the API key securely (as an environment variable or in a secret manager).
4. Install the client library, e.g.:
    - `pip install openai`
5. Start querying the model!

Getting started with the ChatGPT API

1. Create an OpenAI account via `auth.openai.com/create-account`.
2. **Get your API key** via `platform.openai.com/account/api-keys`.

NAME	SECRET KEY	LAST USED ⓘ	CREATED BY	PERMISSIONS
Secret key	sk-...2GQA	Never	Xiang Li	All

3. Store the API key securely (as an environment variable or in a secret manager).
4. Install the client library, e.g.:
   - `pip install openai`
5. Start querying the model!

Cost of APIs

Using the API is billed per token: e.g., GPT-5 mini currently costs about \$0.25 per 1 M input tokens and \$2 per 1 M output tokens (see official pricing).

Minimal Python Example

Goal: send a user message and get a response.

```python
from openai import OpenAI
client = OpenAI(api_key="your_api_key_here")

resp = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Explain PCA in 3 short bullets."}
        ]
)

answer = resp.choices[0].message.content
```

Model Output Example

Model output:

```
- **Dimensionality Reduction**: Principal Component Analysis (PCA) transforms high-dimensional data into a lower-dimensional space while preserving as much variance (information) as possible.

- **Orthogonal Transformation**: It achieves this by identifying new axes (principal components) that are linear combinations of the original features, where each principal component is orthogonal to one another.

- **Variance Maximization**: The first principal component captures the direction of the greatest variance in the data, and subsequent components capture the remaining variance in orthogonal directions.
```

Tuning Model Parameters in the ChatGPT API

The API provides flexible parameters to control generation behavior. See https://platform.openai.com/docs/api-reference/completions for more.

Key parameters:

- `temperature` – randomness of responses ($0 =$ deterministic, $1 =$ creative)
- `max_tokens` – limit on output length
- `top_p` – nucleus sampling [Holtzman et al., 2020], top-p probability mass
- `frequency_penalty` – discourages repetition of frequent tokens
- `presence_penalty` – encourages introducing new topics

```
resp = client.chat.completions.create(
    model="gpt-4o-mini",
    temperature=0.8,
    max_tokens=200,
    top_p=0.9,
    frequency_penalty=0.2,
    presence_penalty=0.6,
    messages=[......])
```

Beyond chat: Other OpenAI API capabilities

The API supports many tasks beyond text generation.

- **Embeddings & Search:** Convert text to vector form for similarity, clustering, or retrieval.
- **Multimodal Input:** Process or generate images (e.g., analysis, editing, or description).
- **Function Calling:** Let models call external tools or return structured JSON outputs.
- **Reasoning Tasks:** Use models with better step-by-step reasoning ability.
- **Fine-tuning:** Train models on domain-specific data to match your tone or tasks.

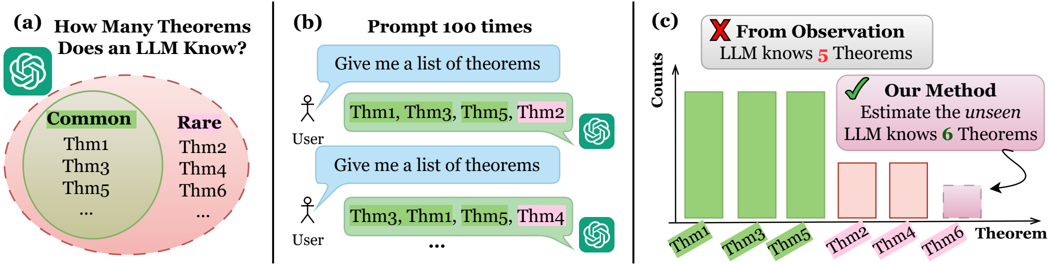
Check `https://platform.openai.com/docs/api-reference` for more details.

An example: Model evaluation via APIs

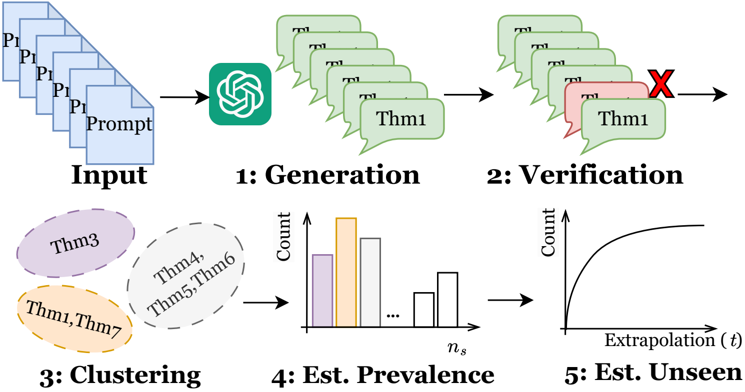
- Even with only API access, we can still conduct meaningful statistical research.
- APIs provide access to LLM intelligence, but interpreting their behavior and improving their utility require statistical thinking.
- For example, suppose we want to evaluate how many theorems an LLM knows.

An example: Model evaluation via APIs

- Even with only API access, we can still conduct meaningful statistical research.
- APIs provide access to LLM intelligence, but interpreting their behavior and improving their utility require statistical thinking.
- For example, suppose we want to evaluate how many theorems an LLM knows.



Our method [Li et al., 2025c]: Estimating the Unseen



- Five-step procedure.
- Steps 2 & 3 standardize answers.
- Verification reduces hallucination.
- Clustering reduces redundancy.
- n_0 = the amount of unseen knowledge.

Results of knowledge estimation

- Query the LLM N_{query} times with a fixed prompt, each time requesting N_{ans} instances of domain-specific knowledge.
- Use external databases for validation (e.g., Wikipedia) and cluster the responses based on their unique external identifiers (e.g., canonical URL).
- $(N_{\mathrm{query}}, N_{\mathrm{ans}}) = (30{,}000, 20)$ for theorems and $(3{,}000, 50)$ for diseases.
- $\mathrm{SKR} =$ seen knowledge / total knowledge.

Results of knowledge estimation

- Query the LLM N_{query} times with a fixed prompt, each time requesting N_{ans} instances of domain-specific knowledge.
- Use external databases for validation (e.g., Wikipedia) and cluster the responses based on their unique external identifiers (e.g., canonical URL).
- $(N_{\text{query}}, N_{\text{ans}}) = (30{,}000, 20)$ for theorems and $(3{,}000, 50)$ for diseases.
- SKR = seen knowledge / total knowledge.

Model	**Math theorem**			**Anatomical disease**		
	N_{seen}	$\widehat{N}_{\text{tot}}$	SKR	N_{seen}	$\widehat{N}_{\text{tot}}$	SKR
① `ChatGPT-4o-chat`	702	1189	0.59	277	732	0.38
② `ChatGPT-3.5-turbo-chat`	868	1064	0.82	268	278	0.96
③ `LLaMA-V3-70B-instruct`	**1432**	**1706**	0.84	**875**	**3372**	0.26
④ `LLaMA-V3-3B-instruct`	1035	1331	0.78	780	1375	0.57
⑤ `Mistral-7B-instruct-V0.1`	753	1194	0.63	489	1723	0.28
⑥ `Qwen2.5-7B-instruct`	444	1162	0.38	426	521	0.82
⑦ `Claude-3.7-Sonnet`	120	201	0.60	115	462	**0.25**
⑧ `DeepSeek-V3`	148	241	0.61	86	334	0.26
⑨ `Gemini-1.5-flash`	100	515	**0.19**	139	143	0.97

What if we have more access to the LLM?

How LLMs combine tokens

Denote the vocabulary by $\mathcal{W} = \{1, \ldots, K\}$, a token therein by w_t, and a text by $w_{<t} := w_1 \cdots w_{t-1}$.

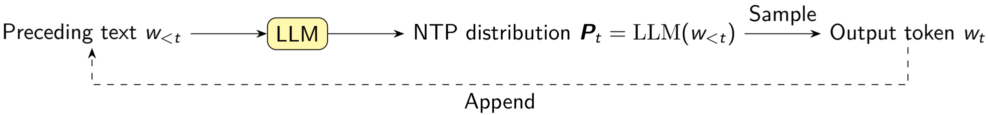
- **Large vocabulary**: $\mathcal{W}$ is large in practice; $K = 50,257$ for GPT-2/3.5, $= 32,000$ for LLaMa models, and $= 128,000$ for DeepSeek-R1.

How LLMs combine tokens

Denote the vocabulary by $\mathcal{W} = \{1, \ldots, K\}$, a token therein by w_t, and a text by $w_{<t} := w_1 \cdots w_{t-1}$.

- **Large vocabulary**: $\mathcal{W}$ is large in practice; $K = 50,257$ for GPT-2/3.5, $= 32,000$ for LLaMa models, and $= 128,000$ for DeepSeek-R1.
- **Autoregresiveness**: An LLM generates each token sequentially by sampling from a probability distribution conditioned on previous tokens:

$$w_t \sim \boldsymbol{P}_t \quad \text{where} \quad \boldsymbol{P}_t = \mathrm{LLM}(w_{<t}) \quad \text{is a distribution on} \quad \mathcal{W}.$$



Example of autoregresive generation

Prefix

I saw a cat on a

$w_{1:6} = [w_1, w_2, w_3, w_4, w_5, w_6]$

LLM

$\mathbf{P}_7 = (P_{7,1}, \cdots, P_{7,K})$

Next Token, $w_7 \sim \mathbf{P}_7$

Mat	25%
Carpet	13%
Boat	5%
Plane	3%
....	...

Sampling

carpet

$\boxed{w_7}$

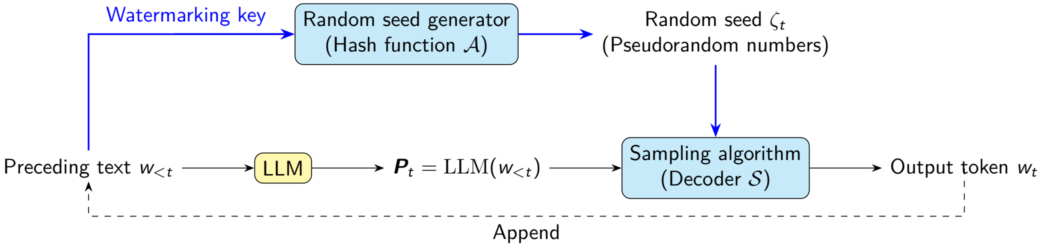
What if we could control the sampling of the LLM?

- By manipulating the sampling process, we can embed a statistical signal between the input (preceding text $w_{<t}$) and the output (next token w_t).
- Such dependence rarely appears in human-written text but commonly arises in LLM-generated text.

What if we could control the sampling of the LLM?

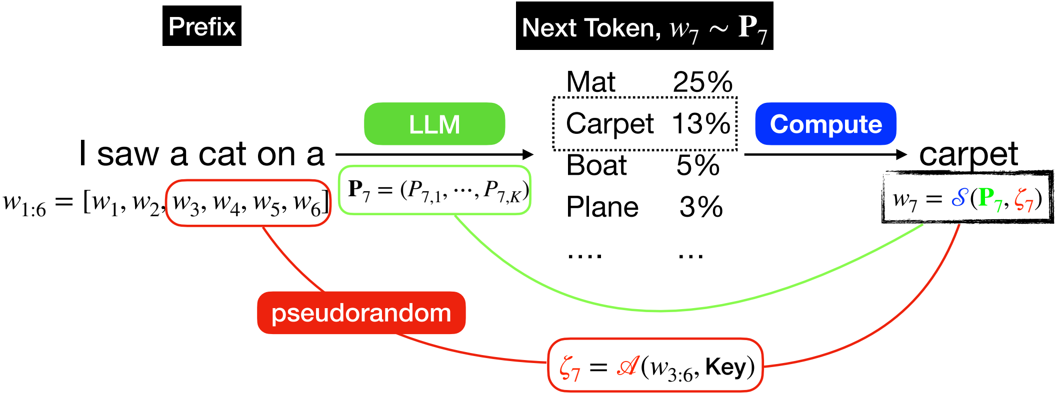
- By manipulating the sampling process, we can embed a statistical signal between the input (preceding text $w_{<t}$) and the output (next token w_t).
- Such dependence rarely appears in human-written text but commonly arises in LLM-generated text.
- This technique is known as **watermarking** [Kirchenbauer et al., 2023].
  - Enables reliable detection and attribution of AI-generated content.
  - Extends naturally to other modalities such as images and tables.
  - Many applications: AI-content labeling, academic integrity checking, biosecurity monitoring, and data provenance tracking.

Watermarked generation: Procedure



- Mathematical speaking: $\zeta_t = \mathcal{A}(w_{<t}, \mathrm{Key})$ and $w_t = \mathcal{S}(\boldsymbol{P}_t, \zeta_t)$.
- A watermark is defined by $(\mathcal{A}, \mathcal{S}, \mathrm{Key})$.
- Watermark signal is the dependence of each w_t on ζ_t.

Watermarked generation: Example



Comments on pseudorandom numbers

$\mathcal{A}(\text{public data}, \text{private info})$ computes the pseudorandom number:

- Public data $=$ prior tokens $w_{<t}$ or a segment $w_{(t-m):t}$ [Kirchenbauer et al., 2023].
- Private info $=$ secret key, denoted Key.

Comments on pseudorandom numbers

$\mathcal{A}(\text{public data}, \text{private info})$ computes the pseudorandom number:

- Public data = prior tokens $w_{<t}$ or a segment $w_{(t-m):t}$ [Kirchenbauer et al., 2023].
- Private info = secret key, denoted Key.

Property (*Soundness of pseudorandomness*)

1. $\zeta_t = \mathcal{A}(w_{1:t-1}, \mathrm{Key})$ for $t = 1, \ldots, n$ are iid copies of a known random variable ζ
2. ζ_t is statistically independent of $w_{1:t-1}$
3. Computationally infeasible to infer Key from ζ_t and $w_{<t}$

Comments on pseudorandom numbers

$\mathcal{A}(\text{public data}, \text{private info})$ computes the pseudorandom number:

- Public data $=$ prior tokens $w_{<t}$ or a segment $w_{(t-m):t}$ [Kirchenbauer et al., 2023].
- Private info $=$ secret key, denoted Key.

Property (*Soundness of pseudorandomness*)

1. $\zeta_t = \mathcal{A}(w_{1:t-1}, \mathrm{Key})$ for $t = 1, \ldots, n$ are iid copies of a known random variable ζ
2. ζ_t is statistically independent of $w_{1:t-1}$
3. Computationally infeasible to infer Key from ζ_t and $w_{<t}$

- Well-developed in theoretical computer science [Barak, 2021].
  - Pseudorandom numbers are "reproducible" ***iff*** Key is known.
  - Key will be shared with the verifier through a secure protocol.

Comments on pseudorandom numbers

$\mathcal{A}(\text{public data}, \text{private info})$ computes the pseudorandom number:

- Public data $=$ prior tokens $w_{<t}$ or a segment $w_{(t-m):t}$ [Kirchenbauer et al., 2023].
- Private info $=$ secret key, denoted Key.

Property (*Soundness of pseudorandomness*)

1. $\zeta_t = \mathcal{A}(w_{1:t-1}, \mathrm{Key})$ for $t = 1, \ldots, n$ are iid copies of a known random variable ζ
2. ζ_t is statistically independent of $w_{1:t-1}$
3. Computationally infeasible to infer Key from ζ_t and $w_{<t}$

- Well-developed in theoretical computer science [Barak, 2021].
  - Pseudorandom numbers are "reproducible" ***iff*** Key is known.
  - Key will be shared with the verifier through a secure protocol.
- Theory assumes true randomness; Implementation use deterministic generation.
- Similar to setting a *seed* for reproducibility in simulations.

Pseudocode for watermark embedding

Algorithm Watermarked LLM Generation

Inputs: a watermark $(\mathcal{S}, \mathcal{A}, \mathrm{Key})$ and a language model.
Load the language model $\mathrm{LLM}(\cdot)$.
Receive the user prompt s and feed it to the model to generate a continuation.
Initialize $w_{<1} = s$ and set n the maximum length.
for step $t = 1, 2, \cdots n$ **do**
    Compute the NTP distribution: $\boldsymbol{P}_t = \mathrm{LLM}(w_{<t})$.
    Compute the pseudorandom number: $\zeta_t = \mathcal{A}(w_{<t}, \mathrm{Key})$.
    Compute the next token: $w_t = \mathcal{S}(\boldsymbol{P}_t, \zeta_t)$.
    Append the history: $w_{<(t+1)} = w_{<t} w_t$. ▷ Autoregressive
return $w_{\leq n}$.

A high-level intro of watermark detection

$$H_0 : w_{\leq n} \text{ is human written} \quad \text{v.s.} \quad H_1 : w_{\leq n} \text{ is LLM-generated.}$$

- **Human-written text**: w_t is independent of ζ_t as humans don't know $\mathcal{A}$ and Key.
- **LLM-generated text**: w_t depends on ζ_t via the decoder function $\mathcal{S}$.

A high-level intro of watermark detection

$$H_0 : w_{\leq n} \text{ is human written} \quad \text{v.s.} \quad H_1 : w_{\leq n} \text{ is LLM-generated.}$$

- **Human-written text**: w_t is independent of ζ_t as humans don't know $\mathcal{A}$ and Key.
- **LLM-generated text**: w_t depends on ζ_t via the decoder function $\mathcal{S}$.

Take-away

Watermarking couples each token w_t and a psedorandom ζ_t, altering their joint distribution. Pivotal statistic Y_t quantifies this evidence.

A high-level intro of watermark detection

$$H_0 : w_{\leq n} \text{ is human written} \quad \text{v.s.} \quad H_1 : w_{\leq n} \text{ is LLM-generated.}$$

- **Human-written text**: w_t is independent of ζ_t as humans don't know $\mathcal{A}$ and Key.
- **LLM-generated text**: w_t depends on ζ_t via the decoder function $\mathcal{S}$.

Take-away

Watermarking couples each token w_t and a psedorandom ζ_t, altering their joint distribution. Pivotal statistic Y_t quantifies this evidence.

- Practical choices uses the pivotal statistic $Y_t = Y(w_t, \zeta_t)$ [Li et al., 2025b]:
  - Under H_0, $w_t \perp \zeta_t$ such that $Y_t \sim \mu_0$.
  - Under H_1, $w_t = \mathcal{S}(\boldsymbol{P}_t, \zeta_t)$ such that $Y_t = Y(\mathcal{S}(\boldsymbol{P}_t, \zeta_t), \zeta_t) \sim \mu_{1,\boldsymbol{P}_t}$.

A high-level intro of watermark detection

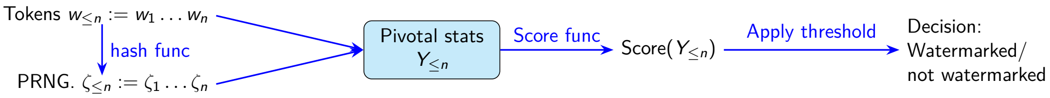
$$H_0 : w_{\leq n} \text{ is human written} \quad \text{v.s.} \quad H_1 : w_{\leq n} \text{ is LLM-generated.}$$

- **Human-written text**: w_t is independent of ζ_t as humans don't know $\mathcal{A}$ and Key.
- **LLM-generated text**: w_t depends on ζ_t via the decoder function $\mathcal{S}$.

Take-away

Watermarking couples each token w_t and a psedorandom ζ_t, altering their joint distribution. Pivotal statistic Y_t quantifies this evidence.

- Practical choices uses the pivotal statistic $Y_t = Y(w_t, \zeta_t)$ [Li et al., 2025b]:
  - Under H_0, $w_t \perp \zeta_t$ such that $Y_t \sim \mu_0$.
  - Under H_1, $w_t = \mathcal{S}(\boldsymbol{P}_t, \zeta_t)$ such that $Y_t = Y(\mathcal{S}(\boldsymbol{P}_t, \zeta_t), \zeta_t) \sim \mu_{1,\boldsymbol{P}_t}$.



Pseudocode for watermark detection

Algorithm Watermark Detection

Inputs: a text $w_{\leq n}$, hash function $\mathcal{A}$, secret Key, and significance level α.
Simulate n iid samples from the pivotal distribution μ_0.
Set $\widehat{q}_n$ as the empirical $(1-\alpha)$-quantile of those null samples, if its theoretical counterpart is hard to compute.
Initialize $w_{<1}$ by any prefix.
for step $t = 1, 2, \cdots, n$ **do**
    Compute the pseudorandom number: $\zeta_t = \mathcal{A}(w_{<t}, \mathrm{Key})$.
    Compute the pivotal statistic: $Y_t = Y(w_t, \zeta_t)$.
Compute the test score: $T = \mathrm{Score}(Y_{\leq n})$.
Claim: LLM-generated if $T > \widehat{q}_n$ otherwise human-written.

How to optimally choose the Score function?

- For the sum-based rule $\mathrm{Score}(Y_{\le n}) = \sum_{t=1}^{n} h(Y_t)$, Li et al. [2025b] introduced an efficiency measure

$$R_{\mathcal{P}}(h) = \lim_{n\to\infty} -\frac{1}{n}\log(\text{Type II error}) \quad \text{s.t.} \quad \text{Type I error} = \alpha,$$

  which characterizes the exponential decay rate of the Type II error under a fixed Type I error level.

How to optimally choose the Score function?

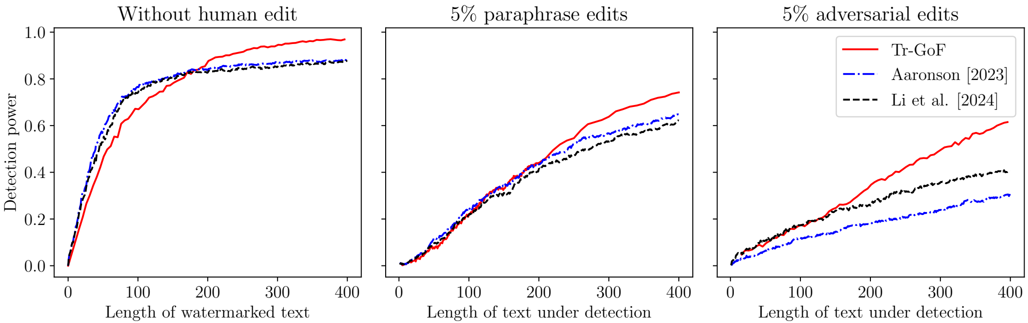
- For the sum-based rule $\mathrm{Score}(Y_{\leq n}) = \sum_{t=1}^{n} h(Y_t)$, Li et al. [2025b] introduced an efficiency measure

$$R_{\mathcal{P}}(h) = \lim_{n\to\infty} -\frac{1}{n} \log(\text{Type II error}) \quad \text{s.t.} \quad \text{Type I error} = \alpha,$$

  which characterizes the exponential decay rate of the Type II error under a fixed Type I error level.
- Maximizing $R_{\mathcal{P}}(h)$ over h yields the optimal detection rule when all NTP distributions are contained in a given class $\mathcal{P}$.
- Li et al. [2025a] proposed a truncated goodness-of-fit (GoF) test for corruption scenarios and proved its optimality in certain regimes.
- He et al. [2025] demonstrated that general GoF tests enhance both detection power and robustness to corruption.
- Cai et al. [2025] derived the optimal score function in settings where the pseudorandom variables are not i.i.d.

Performance of different detectors

- On C4 news-like dataset [Raffel et al., 2020] and OPT-1.3B model [Zhang et al., 2022] (temperature 0.3).



A self-contained watermark python demo

- **Download**:
  https://github.com/lx10077/WatermarkFramework/blob/main/watermark_demo.py
- **How to use**:
  `python watermark_demo.py -temp 1 -alpha 0.01 -model facebook/opt-1.3b`

```
tokenizer = AutoTokenizer.from_pretrained(args.model) # Load tokenizer which convers text to a sequence of
# Ensure the tokenizer has a pad token
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
model = AutoModelForCausalLM.from_pretrained(
    args.model,
    device_map="auto",              # Automatically place model layers on available GPU(s)
    torch_dtype=torch.float16       # (Optional) Set tensor data type to float16 for faster computation
)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu") # Use GPU if possible
print(f"Using device: {device}")
model = model.to(device) # Move the model to GPU, otherwise it is default on CPU
model.eval()

# Load the first 200 samples from the AG News dataset.
# You could also test your own questions or queries. The model will continue to write after your given tex
# To that end, simply change the following 'raw_texts' with a list of your questions.
dataset = load_dataset("ag_news", split="train[:200]")
```

Concluding remarks

- Even with only API access, we can still conduct meaningful research. For example, estimate unseen knowledge.
- Watermarking enhances content traceability and integrity by coupling each toke with tractable pseudorandom numbers.
- Many interesting statistical challenges emerge in practical watermarking applications:

Whether watermarked → How many watermarked → Which watermarked

- Have a try to watermark!

  https://github.com/lx10077/WatermarkFramework/blob/main/watermark_demo.py

References I

B. Barak. An intensive introduction to cryptography, lectures notes for Harvard CS 127. https://intensecrypto.org/public/index.html, Fall 2021.

T. T. Cai, X. Li, Q. Long, W. J. Su, and G. G. Wen. Optimal detection for language watermarks with pseudorandom collision. *arXiv preprint arXiv:2510.22007*, 2025.

W. He, X. Li, T. Shang, L. Shen, W. J. Su, and Q. Long. On the empirical power of goodness-of-fit tests in watermark detection. In *Advances in neural information processing systems*, 2025.

A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein. A watermark for large language models. In *International Conference on Machine Learning*, volume 202, pages 17061–17084, 2023.

X. Li, F. Ruan, H. Wang, Q. Long, and W. J. Su. Robust detection of watermarks for large language models under human edits. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, page qkaf056, 2025a.

X. Li, F. Ruan, H. Wang, Q. Long, and W. J. Su. A statistical framework of watermarks for large language models: Pivot, detection efficiency and optimal rules. *The Annals of Statistics*, 53(1):322–351, 2025b.

X. Li, J. Xin, Q. Long, and W. J. Su. Evaluating the unseen capabilities: How many theorems do LLMs know? *arXiv preprint arXiv:2506.02058*, 2025c.

References II

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (1):5485–5551, 2020.

S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.