Protecting Text IP in the Era of LLMs with Robust and Scalable Watermarking

Gregory Kang Ruey Lau [*1 2] **Xinyuan Niu** [*1 3] **Hieu Dao** [1] **Jiangwei Chen** [1 4] **Chuan-Sheng Foo** [3 4]
Bryan Kian Hsiang Low [1]

1. Introduction

Protecting intellectual property (IP) of human-authored text such as articles and code is increasingly important, especially as sophisticated attacks become possible, such as paraphrasing by large language models (LLMs) (Brewster et al., 2023) or unauthorized training of LLMs on copyrighted text to infringe such IP (Novet, 2024). Although content creators could prove data ownership by watermarking their works, existing text watermarking methods are not **robust enough against such attacks** nor **scalable to millions of users** for practical implementation. For example, directly adding digital metadata or invisible Unicode watermarks (Rizzo et al., 2019; Taleby Ahvanooey et al., 2019) have limited efficacy as they may be easily removed. Existing natural language watermarking methods (Qiang et al., 2023; Yoo et al., 2023) that adjust the text itself also lack robustness to paraphrasing attacks and have limited scalability (see Section 5).

To protect the IP rights of content creators against the unauthorized use of their data for LLM training, it is also essential to achieve **LLM data provenance**, i.e., *prove whether their set of work had been used to train 3rd party black-box LLMs.* Recent works tackling this problem (Abdelnabi & Fritz, 2021; Zhang et al., 2023) largely require intervening in the LLM training process. This is unrealistic as LLM service providers may not cooperate due to incentive misalignment, and adversaries may also use open-source LLMs.

Hence, it is natural to ask *whether it is possible to develop a practical, robust, and scalable text watermarking framework for protecting IP against both plagiarism and unauthorized training of LLMs*. For example, the watermarks should persist regardless of whether the original text has been paraphrased, converted into speech or handwritten text, or used in unauthorized LLM training (e.g., fine-tuning, in-context learning) to generate an output. The framework should also be general enough to tailor to a wide range of text formats (e.g., natural language or code), and be scalable (i.e., support millions of users, potentially multiple watermarks in the same text, and with a reasonable computational cost).

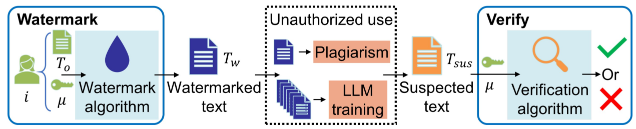

Figure 1: Schematics of problem formulation. Client i watermarks text T_{o} with ID μ_i, producing $T_{\text{w}}^{(i)}$. Client should still be able to verify watermark in T_{sus} after attacks.

In this paper, we propose WATERFALL, the first training-free framework for robust and scalable text watermarking applicable across multiple text types (e.g., articles, code) and languages supportable by LLMs, for general as well as LLM text training data provenance. *Rather than viewing LLMs as just sources of IP infringement, we introduce the novel perspective of using LLMs' capabilities to protect existing IP*. Though simple, our training-free framework comprises several key innovations such as being the first to use LLM as paraphrasers for watermarking along with a novel combination of techniques that are surprisingly effective in achieving robust verifiability, scalability, and data provenance for LLMs, beating state-of-the-art (SOTA) text watermarking methods as we empirically demonstrate. We discuss how our framework is adaptive to evolving LLM landscape and can be realistically deployed in real practice.

2. Problem formulation and desiderata

Consider M clients, with client i possessing unique watermark ID μ and textual data T_{o} (e.g., articles or code). We assume T_{o} has semantic content c (e.g., the IP content) that is only determined by its tokens and fully represents the text's value. The goal is to develop a framework such that client i can use a watermarking operator $\mathcal{W}(\mu, T_{\text{o}}) \to T_{\text{w}}$ to produce a text T_{w} that contains watermark μ, preserves c, and can be used/distributed freely.

There are adversaries who aim to infringe the IP in T_{w} through attacks $\mathcal{A}(T_{\text{w}}) \to T_{\text{sus}}$ that generate their own text T_{sus} without the watermark μ while preserving semantic

[*]Equal contribution [1]Department of Computer Science, National University of Singapore [2]CNRS@CREATE, 1 Create Way, #08-01 Create Tower, Singapore 138602 [3]Centre for Frontier AI Research, A*STAR, Singapore [4]Institute for Infocomm Research, A*STAR, Singapore. Correspondence to: Bryan Kian Hsiang Low <lowkh@comp.nus.edu.sg>.

Accepted to the 2^{nd} *Workshop on Generative AI and Law*, co-located with the *International Conference on Machine Learning*, Vienna, Austria. 2024.

content c. The adversaries do not know μ but are able to perform several classes of attacks, such as paraphrasing or translating with an LLM or using T_{w} with any LLM for in-context prompting or fine-tuning. *No other parties have access to the LLMs used by adversaries*.

After the attacks, client i should be able to use a verification operator $\mathcal{V}(\mu, T_{\text{sus}})$ to generate a score q indicating the likelihood that T_{sus} is watermarked with μ.

A suitable watermarking framework should satisfy the following desiderata: (1) The watermarked text T_{w} should have high fidelity, e.g., T_{w} is semantically similar to T_{o}; (2) the watermark should be easily verified, even after attacks by adversaries; (3) the framework should allow for a large set of IDs while meeting all other desiderata.

3. Method

Our watermarking framework, WATERFALL, first uses an LLM paraphraser to autoregressively paraphrase the original text T_{o}, producing initial logits for the new text T_{w}. The client's ID μ is used to seed a vocab permutation operator to map the logits onto a watermarking space V_w, and choose a perturbation function to produce perturbed logits that encodes the watermark. The LLM samples the perturbed logits in the original token space to produce a watermarked token. For the next token loop, the past $n-1$ tokens are used to seed vocab permutation while all past tokens are fed as context for the next generation, helping the LLM paraphraser maintain the fidelity of T_{w} despite watermarking.

For verification, each token in a suspected text T_{sus} is counted in V_w-space, which is specified for each μ and preceding tokens in the same n-gram unit, producing an average cumulative token distribution. The perturbation function specified by the ID μ, is used to perform an inner product with the cumulative distribution to compute a verification score q. Larger q suggests greater similarity between the underlying distributions that generate T_{sus} and T_{w}, hence T_{sus} is more likely to be watermarked, i.e., T_{sus} is derived from the copyrighted text T_{w}.

4. Discussion

There is currently a lack of actual, practical large-scale deployment of text watermarking effective against LLM attacks, given the current SOTA watermarking methods' limitations and resource requirements. However, WATERFALL may possibly provide a foundation for achieving large-scale deployment, with both decentralized or centralized options. This is made achievable given WATERFALL's low computational cost, scalability to a large number of clients, and robustness to LLM attacks including unauthorized training of LLMs that generates IP-infringing text.

Our framework highlights a few perspectives that we hope more would consider. First, *while increasingly capable LLMs allows for easier and more sophisticated forms of potential IP infringement, LLMs themselves could also enable better text IP protection of original texts*. A key strength of WATERFALL is that its capabilities grow as LLMs become more powerful, with increasingly better watermarking performance, allowing it to potentially keep up with the increasing capabilities adversaries can use for IP infringement. It is able to achieve a higher fidelity-verifiability Pareto frontier, and reduce any fidelity degradation while using higher watermarking strength for greater robust verifiability.

Second, as open-source LLM models become more prevalent and capable, *it is likely not viable to rely only on major LLM providers to assist in IP protection*. Instead, *content creators themselves should be equipped with methods such as* WATERFALL *to protect their work before dissemination*, such as by injecting robust watermarks that allows verifiability even after both traditional attacks and unauthorized use in LLM training by adversaries.

Third, a general text watermarking framework like WATERFALL that can apply across different text types and languages not only helps with practical deployment, but also makes it highly versatile and not dependent on any text-specific properties. This makes it *more easily adaptable for incorporating new defense methods, providing a strong foundation for future works to build on as new threats emerge*.

5. Experiments

5.1. Data ownership

For watermarking of text articles, we demonstrate the effectiveness of WATERFALL using text samples T_{o} from the `c4 realnewslike` dataset (Raffel et al., 2020). This experiment mirror realistic scenarios, for e.g., news outlets watermarking their articles before publishing them, to be able to effectively scan the internet for, and verify, plagiarized content (Brewster et al., 2023). Details are in Appendix F.

As benchmarks, we consider two recent linguistics-based watermarking methods: M-BIT by Yoo et al. (2023) and P-NLW by Qiang et al. (2023). These methods are advanced variants of synonym substitution-based watermarking schemes that use deep learning to improve watermarking performance (details in Appendix F.3).

Fidelity-verifiability. WATERFALL supports adjustable watermarking strength, allowing clients to calibrate the fidelity-verifiability trade-off based on their use case. Figure 2(a) shows the Pareto frontier of the trade-off. Stronger watermark strength κ improves verifiability but also introduces larger distortions to the LLM paraphrasing process, decreasing the fidelity of watermarked text. Even with just 100 tokens (about 65 words), WATERFALL achieves high verifiability with AUROC of 0.98 (Figure 2(b), $\kappa = 6$).

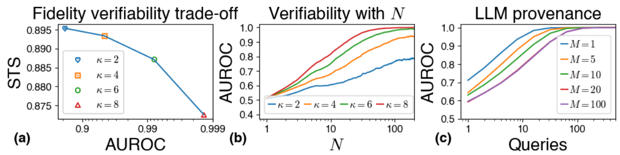

Figure 2: (a) Higher κ trades off fidelity for verifiability. (b) Higher κ and longer token length N improve verifiability. (c) More queries improve LLM provenance verifiability.

Table 1: Robust verifiability under insertion $\mathbb{A}_{1-I}$, deletion $\mathbb{A}_{1-D}$, synonym substitution $\mathbb{A}_{1-S}$, translation $\mathbb{A}_{2-T}$, paraphrase $\mathbb{A}_{2-P}$, re-watermark $\mathbb{A}_3$, in-context prompting $\mathbb{A}_4$.

	$\mathbb{A}_{1-I}$	$\mathbb{A}_{1-D}$	$\mathbb{A}_{1-S}$	$\mathbb{A}_{2-T}$	$\mathbb{A}_{2-P}$	$\mathbb{A}_3$	$\mathbb{A}_4$
WATERFALL	**0.985**	**0.988**	**0.978**	**0.951**	**0.881**	**0.815**	**0.775**
P-NLW	0.656	0.660	0.673	0.475	0.508	0.724	0.502
M-BIT	0.756	0.568	0.669	0.567	0.363	0.664	0.525

Robust verifiability. We consider the various classes of attacks $\mathbb{A}$ on the T_w without knowledge of μ:
$\mathbb{A}_1$: alter T_w with word additions/removals/substitutions;
$\mathbb{A}_2$: alter T_w with translation and paraphrasing by a LLM;
$\mathbb{A}_3$: watermark T_w again with WATERFALL and another μ';
$\mathbb{A}_4$: using T_w with any LLM for in-context prompting.

Table 1 shows WATERFALL achieves significantly higher robust verifiability than benchmarks under the attacks. In fact, as several of these attacks significantly changed the the words and structure of the text, the watermarks of M-BIT and P-NLW were almost completely removed in many instances. Further details and insights are in Appendix G.

Scalability. WATERFALL has a large maximum scalability of $M \sim 10^{130274}$ based on our implementation using the Llama-2 model as paraphraser and Fourier perturbation function (details in Appendix E). In comparison, the scalability of M-BIT and P-NLW is dependent on the number of possible synonym replacements in any given text, which is limited by text length and varies for different text. On this dataset, M-BIT and P-NLW can only embed a mean of 9.5 bits ($M \sim 10^3$) and 23.2 bits ($M \sim 10^{10}$) respectively.

In practice, scalability is further limited by how well the schemes can differentiate among similar watermarks. To demonstrate this, we watermarked $T_\text{w}^{(i)}$ with μ_i and computed the verifiability of $T_\text{w}^{(i)}$ against 1000 randomly selected $\mu_{j \neq i}$. We found that for WATERFALL, all of the IDs achieved very high AUROC, while M-BIT and P-NLW have many IDs with low AUROC: The 1st percentile AUROC for WATERFALL was 0.976, much higher than M-BIT and P-NLW at 0.614, 0.766 respectively. Details and results of WATERFALL scaling to 100,000 IDs are in Appendix F.7.

Computational costs. We note that WATERFALL also has lower computational cost compared to benchmarks (Table 2). WATERFALL verification can be run in parallel on a CPU, requiring only 0.035s when ran on a 16-core CPU, which is 75× and 4237× faster than M-BIT and P-NLW respectively, both which require inference using deep learning models on a GPU. This is important in the context of protection of IP, e.g., where data providers may have to scan through large amount of online data for any IP infringement. Further discussion on the deployment costs are in Appendix I.

Table 2: Mean compute time over 100 texts on 1 Nvidia RTX A5000. *Note that verification for WATERFALL was performed only on CPU without requiring a GPU.

	WATERFALL	M-BIT	P-NLW
Watermark	24.8s	**2.97s**	147s
Verification	**0.035s***	2.61s	148s

5.2. LLM data provenance of articles

Finally, we explore how WATERFALL watermarks may persist after LLM fine-tuning. We consider the setting where client i watermarks a set of text $\{T_\text{w}^{(i)}\}$ that adversaries use, without authorization, to fine-tune their own LLMs (i.e., $\mathbb{A}_5$ attacks). Given multiple queries to the fine-tuned black-box LLM, the goal is for client i to be able to verify that $\{T_\text{w}^{(i)}\}$ had been used for training. This setting mirrors realistic scenarios where content owners want to detect unauthorized use of data for LLM training (Novet, 2024). For this experiment, we used the ArXiv dataset (Clement et al., 2019) to fine-tune the `gpt2-xl`.

Fidelity. We verified that using the watermarked instead of the original dataset has minimal effect on the fidelity of the fine-tuned model. Details are in Appendix J.

Verifiability. To evaluate verifiability, we queried the fine-tuned model, and performed watermark verification on the generated text. Figure 2(c), shows that WATERFALL has high verifiability, reaching AUROC of 1.0 with just 100 queries to the fine-tuned LLM.

Scalability. To explore the scalability of WATERFALL for data provenance, we combined the datasets of different number of clients, $M \in \{1, 5, 10, 20, 100\}$, each watermarked with their own unique ID μ, and use the combined dataset for fine-tuning the adversarial model. As expected, Figure 2(c) shows that dealing with a aggregated dataset mixed with a larger M number of different watermarks would result in a decrease in verifiability. However, our results indicate that this decrease leveled off from $M = 20$ to $M = 100$ and still allow for an AUROC (verifiability) of 1.0 with around 100 queries even for $M = 100$, demonstrating the scalability of WATERFALL to a sizeable number of clients.

5.3. Watermarking of code

To demonstrate the versatility of WATERFALL, we consider its out-of-the-box performance on code watermarking. We

Table 3: Fidelity, Verifiability, and Robust Verifiability of WATERFALL with $\kappa = 3$ on code watermarking.

	Fidelity (Pass@10)	**Verifiability** (AUROC) Pre-attack	Post-attack	**Scalability** (# of users)
SRCMARKER	0.984	0.726	0.662	10^5
WATERFALL	0.969	0.904	0.718	10^{130274}

used the MBJSP dataset (Athiwaratkun et al., 2023), and evaluate fidelity using the pass@10 metric (Kulal et al., 2019; Chen et al., 2021) achieved by T_w on functional tests for the original code T_o. We compare WATERFALL with SRCMARKER (Yang et al., 2024), a recent state-of-the-art code watermarking scheme. Further details are in Appendix K.

We found that surprisingly, WATERFALL achieves higher verifiability and robust verifiability (after $\mathbb{A}_2$ paraphrasing attacks) compared to SRCMARKER while maintaining high code fidelity (Table 3). This is despite WATERFALL not requiring any manual training/engineering of programming language-specific watermarking rules, which SRCMARKER does. Instead, WATERFALL inherits its code capabilities from its LLM paraphraser, making it adaptable to other languages (e.g., see Appendix K.5 for Python code results).

6. Conclusion

We proposed WATERFALL, the first training-free framework for text watermarking that has low computational cost, scalability to large number of clients, and robustness to LLM attacks including unauthorized training of LLMs that generates IP-infringing text.

Acknowledgments

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-PhD/2023-01-039J). This research is part of the programme DesCartes and is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. Xinyuan Niu is supported by the Centre for Frontier AI Research of Agency for Science, Technology and Research (A*STAR). Jiangwei Chen is supported by the Institute for Infocomm Research of Agency for Science, Technology and Research (A*STAR). We acknowledge CSC (Finland) for awarding this project access to the LUMI supercomputer, owned by the EuroHPC Joint Undertaking, and hosted by CSC (Finland) and the LUMI consortium. The access was made possible via collaboration between NSCC (Singapore) and CSC (Finland).

References

Abdelnabi, S. and Fritz, M. Adversarial watermarking transformer: Towards tracing text provenance with data hiding. In *Proc. IEEE SP*, pp. 121–140, 2021.

Athiwaratkun, B., Gouda, S. K., Wang, Z., Li, X., Tian, Y., Tan, M., Ahmad, W. U., Wang, S., Sun, Q., Shang, M., et al. Multi-lingual evaluation of code generation models. In *Proc. ICLR*, 2023.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Bird, S., Klein, E., and Loper, E. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proc. AAAI*, 2020.

Brewster, J., Wang, M., and Palmer, C. Plagiarism-bot? How low-quality websites are using ai to deceptively rewrite content from mainstream news outlets. *NewsGuard*, 2023.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Clement, C. B., Bierbaum, M., O'Keeffe, K. P., and Alemi, A. A. On the use of arXiv as a dataset. *arXiv preprint arXiv:1905.00075*, 2019.

Dolan, B. and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Proc. IWP*, 2005.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation. 2021. URL https://doi.org/10.5281/zenodo.5371628.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. In *Proc. ICLR*, 2022.

Kamaruddin, N. S., Kamsin, A., Por, L. Y., and Rahman, H. A review of text watermarking: Theory, methods, and applications. *IEEE Access*, 6:8011–8028, 2018.

Kulal, S., Pasupat, P., Chandra, K., Lee, M., Padon, O., Aiken, A., and Liang, P. S. Spoc: Search-based pseudocode to code. In *Proc. NeurIPS*, 2019.

Levesque, H. J., Davis, E., and Morgenstern, L. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, pp. 47, 2011.

Lin, X., Wu, Z., Dai, Z., Hu, W., Shu, Y., Ng, S.-K., Jaillet, P., and Low, B. K. H. Use your instinct: Instruction optimization using neural bandits coupled with transformers. *arXiv preprint arXiv:2310.02905*, 2023.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. In *Proc. NeurIPS*, 2023.

McCoy, R. T., Smolensky, P., Linzen, T., Gao, J., and Celikyilmaz, A. How much do language models copy from their training data? evaluating linguistic novelty in text generation using raven. *Transactions of the Association for Computational Linguistics*, 11:652–670, 2023.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *Proc. ICLR*, 2016.

Novet, J. Eight newspaper publishers sue Microsoft and OpenAI over copyright infringement. https://www.cnbc.com/2024/04/30/eight-newspaper-publishers-sue-openai-over-copyright-infringement.html, 2024.

Qiang, J., Zhu, S., Li, Y., Zhu, Y., Yuan, Y., and Wu, X. Natural language watermarking via paraphraser-based lexical substitution. *Artificial Intelligence*, 317:103859, 2023.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140):1–67, 2020.

Rizzo, S. G., Bertini, F., and Montesi, D. Fine-grain watermarking for intellectual property protection. *EURASIP Journal on Information Security*, 2019:1–20, 2019.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. R., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Proc. NeurIPS*, 2023.

Shu, L., Luo, L., Hoskere, J., Zhu, Y., Liu, Y., Tong, S., Chen, J., and Meng, L. Rewritelm: An instruction-tuned large language model for text rewriting. In *Proc. AAAI*, 2024.

Taleby Ahvanooey, M., Li, Q., Hou, J., Rajput, A. R., and Chen, Y. Modern text hiding, text steganalysis, and applications: A comparative analysis. *Entropy*, 21(4):355, 2019.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proc. ICLR*, 2018.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. In *Proc. NeurIPS*, 2022.

Witteveen, S., AI, R. D., and Andrews, M. Paraphrasing with large language models. In *Proc. EMNLP-IJCNLP*, 2019.

Yang, B., Li, W., Xiang, L., and Li, B. Srcmarker: Dual-channel source code watermarking via scalable code transformations. In *Proc. IEEE SP*, pp. 97–97, 2024.

Yoo, K., Ahn, W., Jang, J., and Kwak, N. Robust multi-bit natural language watermarking through invariant features. In *Proc. ACL*, pp. 2092–2115, 2023.

Zhang, R., Hussain, S. S., Neekhara, P., and Koushanfar, F. Remark-llm: A robust and efficient watermarking framework for generative large language models. *arXiv preprint arXiv:2310.12362*, 2023.

A. Details on WATERFALL's method

We discuss three key insights to tackle challenges arising from the desiderata, before combining these to present our framework WATERFALL (Watermarking Framework Applying Large Language Models).

A.1. Increasing support set for watermarking via LLMs

First, note that the **fidelity** desideratum is a major constraint to a scheme's ability to meet the other desiderata. Intuitively, a scheme that can only generate a small set $\mathbb{T}_{c,s}^{\mathcal{W}}$ of possible watermarked text would have fewer ways to encode the watermark, leading to lower signal capacity (smaller $|\mathbb{M}|$, lower **scalability**), and less capacity for error correction to withstand attacks (lower **robust verifiability**). For illustration, consider a basic semantic watermarking scheme (BASIC) that lists out synonyms for each word in the original text T_o (e.g., big cat) and remembers a map of IDs to possible combinations of these synonyms (e.g., 01:big feline, 10:large cat, 11:large feline). Watermarking for ID μ is then selecting the text T_w with the matching synonym combination. Note that schemes like BASIC typically only have a relatively small support set $\mathbb{T}_{c,s}^{\mathcal{W}}$ and hence limited watermarking possibilities.

However, LLMs can come up with many more possibilities and access a larger $\mathbb{T}_{c,s}^{\mathcal{W}}$ compared to schemes like BASIC using mechanical paraphrasing rules (e.g., synonym replacement). Past works have shown that LLMs can effectively paraphrase text given suitable prompts (Shu et al., 2024; Witteveen et al., 2019). For example, while synonym replacement can only generate possibilities involving word replacements, an LLM may be able to completely reorder, break, or fuse sentences while preserving semantic content c. In general, as some expressions are more common, we can associate a probability distribution $p_c(T)$ over this set $\mathbb{T}_{c,s}^{\mathcal{W}}$.

Intuitively, we can consider a suitable paraphrasing prompt combined with text T_o as tokens $\hat{c}$ that can constrain an LLM's text generation to $\mathbb{T}_{c,s}^{\mathcal{W}}$. Given $\hat{c}$, the LLM autoregressively access $p_c(T)$ by producing conditional probability distributions $p(w_j|\hat{w}_{1:j-1}, \hat{c})$ for token w_j at step j given the preceding sampled tokens $\hat{w}$, and sampling for each step until it deemed that it had conveyed c. Specifically, at step j, the LLM generates a vector of logits $L_j(\hat{w}_{1:j-1}, \hat{c}) \in \mathbb{R}^{|\mathbb{V}|}$, where

$$p(w_j|\hat{w}_{1:j-1}, \hat{c}) = \text{softmax}(L_j(\hat{w}_{1:j-1}, \hat{c})). \tag{1}$$

We denote LLMs used this way as *LLM paraphrasers*. By using LLM paraphrasers, we significantly increase $\mathbb{T}_{c,s}^{\mathcal{W}}$, which helps us better meet the fidelity, robust verifiability and scalability desiderata.

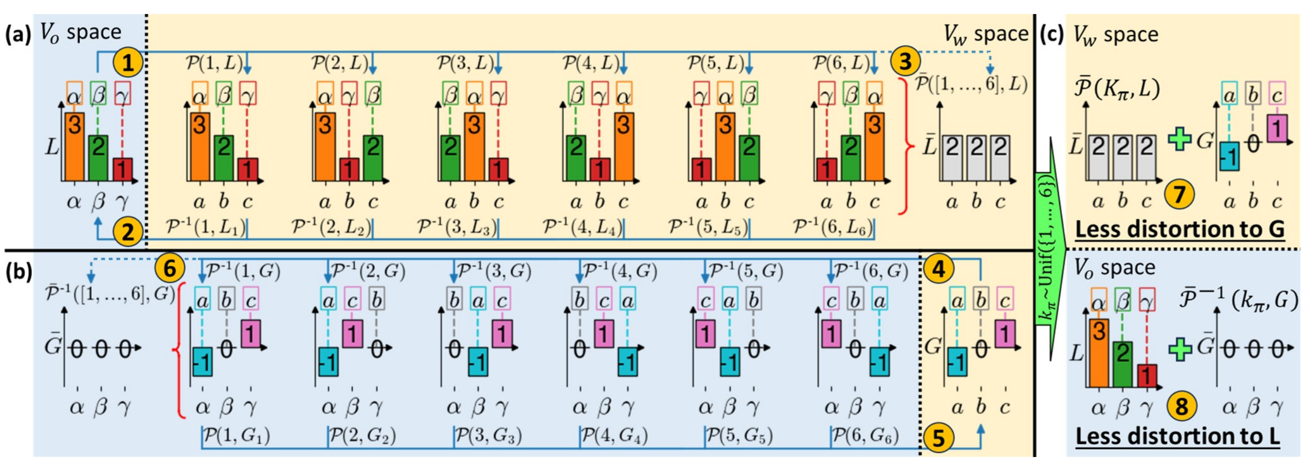

Figure 3: Intuition on permutation operators $\mathcal{P}$, $\mathcal{P}^{-1}$ applied on LLM logits L and watermarking signal G with toy example, Vec. **(a)** $\mathcal{P}$ applied to L in the V_o space results in 6 possible permutations in V_w space. This averages to constant vector $\bar{L}$. **(b)** Similarly, $\mathcal{P}^{-1}$ applied to G in V_w produces permutations in V_o. These averages to constant vector $\bar{G}$. **(c)** With k_π sampled uniformly from the possible keys K_π over multiple LLM generation steps, $L + G$ in shows less distortion to G in V_w space, and to L in V_o space.

A.2. Increasing robustness using n-gram watermarking with LLM deviation correction

Given the extensive threat model, most watermarking schemes would face a major challenge in meeting the **robust verifiability** desideratum. For example, $\mathbb{A}_2$ paraphrasing attacks would likely break schemes such as BASIC which depend on word ordering[1], let alone attacks involving further processing by black-box LLMs (e.g., $\mathbb{A}_4$, $\mathbb{A}_5$ attacks). Instead, we could decompose $p_c(T)$ and the watermarked text T_w into multiple signal carriers, and embed the same watermarking signal to all. This way, we adopt a probabilistic approach where each carrier *could independently be used to verify a watermark*, to withstand attacks that can only corrupt a proportion of carriers.

Specifically, we could consider each consecutive n tokens in T_w as an n-gram carrier unit. At each LLM paraphraser token generation step j, we could apply a watermarking operator $\mathcal{W}$ (Appendix A.3) that perturbs the logits of Equation (1) based on the ID μ and past $n-1$ generated tokens: $\check{L}_j = \mathcal{W}[\mu, \hat{w}_{j-n+1:j-1}](L_j(\hat{w}_{1:j-1}, \hat{c}))$. The perturbed logits will cause a detectable bias in each n-gram, hence the more n-grams that persist after any attack, the higher the verifiability.

Meanwhile, in future generation steps j', the *LLM paraphraser will correct deviations from semantic content* c *and preserve fidelity* given sufficient generation steps, as the subsequent logits $L_{j'}(\hat{w}_{1:j'-1}, \hat{c})$ are still conditioned on paraphrasing prompt $\hat{c}$.

This approach increases our framework's robustness against not just paraphrasing attacks, but also more general LLM-based attacks (e.g., $\mathbb{A}_5$). Past works have shown that language models tend to generate few novel n-grams outside their training set for small n (McCoy et al., 2023). Hence, LLMs trained on text with our watermarked n-grams may more likely generate them in their output. Given sufficient queries to these LLMs, the watermark could then be reliably verified, which we empirically demonstrate in Section 5.

A.3. Increasing scalability with vocab permutation and orthogonal perturbation

Finally, we propose a watermarking operator $\mathcal{W}$ comprising two components: 1) vocab permutation, and 2) orthogonal perturbation. In this section, we will use a toy example (Vec) to show how these components work before presenting their general form. In Vec, we have logits $L = [3, 2, 1]$, indexed by an ordered set $V_o = \{\alpha, \beta, \gamma\}$ representing the token space, e.g., $L(\alpha)$ = 3. Figure 3a presents L as a graph (V_o as x-axis).

Vocab permutation. The vocab permutation operator $\mathcal{P}$ produces a single permutation of V_o and L for any given key k_π (arrow ① in Figure 3a). The inverse operator $\mathcal{P}^{-1}$ reverses the permutation of $\mathcal{P}$ when provided the same key (arrow ② in Figure 3a). As $|V_o| = 3$, there are 6 possible permutations of L, plotted as graphs over a new ordered index $V_w = \{a, b, c\}$, which we can interpret as the watermarking space. Then, we define the average permutation operator $\bar{\mathcal{P}}$ acting on L (indexed by V_o) as one that takes a sequence of keys K_π, apply $\mathcal{P}$ to get L_{k_π} for each $k_\pi \in K_\pi$, and averages them to get a vector $\bar{L}$ (indexed by V_w). Note that when we use $\bar{\mathcal{P}}$ on L over all possible keys, we get a constant vector (e.g., $\bar{L} = \sum_{i=1}^{6} L_i/6 = [2, 2, 2]$, ③ in Figure 3a).

Similarly, given a vector G indexed by V_w, which we can interpret as the watermark signal, the inverse operator $\mathcal{P}^{-1}$ permutes G and V_w given a key k_π, mapping it to V_o, the LLM-ordered token space (arrow ④ in Figure 3b). $\bar{\mathcal{P}}^{-1}$ acting on G analogously averages over all keys, and will also give a constant vector indexed over V_o (e.g., $\bar{G} = \sum_{i=1}^{6} G_i/6 = [0, 0, 0]$, ⑥ in Figure 3b).

This leads to an interesting insight: the permutation operators provide a way for us to *add watermark signals to logits in a deterministically shifting* V_w *space (based on a sequence of keys) to boost verifiability and fidelity*. For illustration, assume that an LLM paraphraser produces L (in V_o-space) for all token generation steps. We use a long sequence K_π of pseudo-random uniformly sampled keys to apply $\mathcal{P}$ on L multiple times (n-gram watermarking), and add the same watermarking signal G in each resulting V_w space for all instances. If we apply $\bar{\mathcal{P}}^{-1}$ with K_π on the perturbed signal $L + G$, the distortion from the permuted L will effectively contribute only uniform background noise to G (⑦ in Figure 3c), which improves **verifiability**. If we instead convert $L + G$ back to V_o space (for token generation) with $\mathcal{P}^{-1}$ for all steps and apply $\bar{\mathcal{P}}$, we get the original logits with only uniform background noise from watermarking (⑧ in Figure 3c), which improves **fidelity**.

More generally, we define the vocab permutation operator $\mathcal{P}$ and its inverse $\mathcal{P}^{-1}$ as pseudorandom permutations over

[1]Using example in Appendix A.1, "large cat"→"cat that is large" would invert the embedded ID "10" to "01".

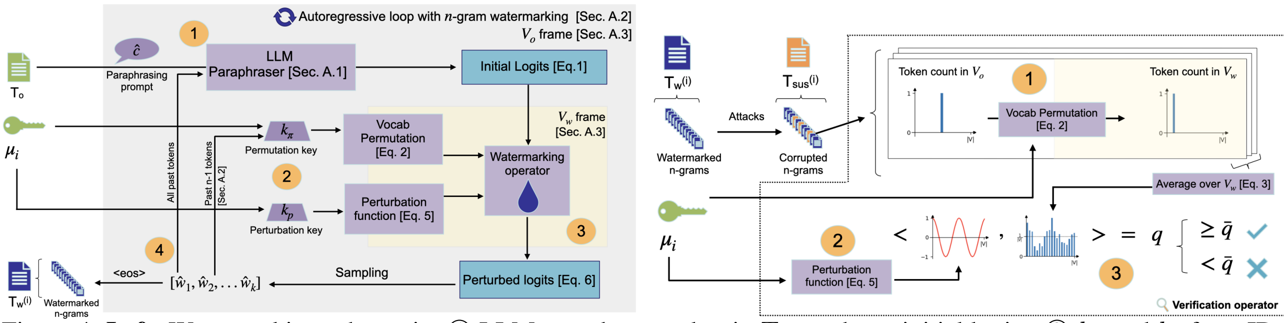

Figure 4: **Left**: Watermarking schematic. ① LLM paraphraser takes in T_o, produces initial logits. ② k_π and k_p from ID μ and metadata k_p for vocab permutation and perturbation function. ③ Perturb logits with Equation (6). ④ Sample perturbed logits, feed past tokens to the next iteration. **Right**: Verification schematic. ① Permute tokens from T_{sus} into V_w with μ and preceding $n-1$ tokens, to get average cumulative distribution. ② Compute perturbation function $\mathcal{F}_1(k_p)$ linked to μ. ③ Compute verification score as inner product of $\mathcal{F}_1(k_p)$ and cumulative distribution, and compare with threshold.

ordered sets V_o and V_w given a key $k_\pi \in \mathbb{K}_\pi$:

$$\begin{aligned} \mathcal{P}(k_\pi, V_o) &= V_o^{k_\pi} \\ \mathcal{P}^{-1}(k_\pi, V_w) &= V_w^{k_\pi} \\ \mathcal{P}^{-1}(k_\pi, \mathcal{P}(k_\pi, V_o)) &= V_o, \end{aligned} \tag{2}$$

where $V_o^{k_\pi}$, $V_w^{k_\pi}$ are uniform-randomly chosen permutations of V_o and V_w if k_π is sampled randomly. For a function L over V_o mapped to a vector of length $|V_o|$, we have $L(\mathcal{P}(k_\pi, V_o)) = L(V_o^{k_\pi})$ and we overload notation by defining $\mathcal{P}(k_\pi, L(\cdot)) \triangleq L(\mathcal{P}(k_\pi, \cdot)) = L_{k_\pi}$. As in the Vec example, $\mathcal{P}$ applied to a function (vector) can be viewed as the same function but with its domain permuted.

We then define an average operator $\bar{\mathcal{P}}$ over a sequence of keys K_π acting on a function L,

$$\bar{\mathcal{P}}(K_\pi, L) \triangleq \tfrac{1}{|K_\pi|} \textstyle\sum_{k_\pi \in K_\pi} \mathcal{P}(k_\pi, L), \tag{3}$$

which outputs an average function of L over V_w (denoted as $\bar{L}$). $\bar{\mathcal{P}}(K_\pi, L)$ will flatten towards a constant function over V_w for a sufficiently large K_π. To achieve this for our framework, we set $K_\pi = \{k_\pi \mid k_\pi = h_\pi(\mu, \hat{w}_{j-n+1:j-1})\}_j$, for all LLM paraphrasing steps j and where h_π is a hash function, which generates pseudorandom K_π sequences. Empirically, we clearly observe the flattened and clear watermarking signals (see Figure 5 in Appendix).

Orthogonal perturbation: Our proposed perturbation operator $\mathcal{F}$ involves two sub-operations acting on V_w. It first maps each key $k_p \in \mathbb{K}_p$ to a unique function in a pre-defined family of orthogonal functions, and then adds the chosen perturbation function to the logits L_j of the LLM output in V_w space:

$$\mathcal{F}_1 : \mathbb{K}_p \hookrightarrow \{\phi : V_w \to \mathbb{R}^{|V_w|} \mid \langle \phi_i, \phi_l \rangle = \delta_{il}\} \tag{4}$$

$$\mathcal{F}(k_p, \kappa, L_j) = L_j + \kappa \mathcal{F}_1(k_p) \tag{5}$$

where $\langle \cdot, \cdot \rangle$ denotes the canonical dot product over V_w. Examples of orthogonal function families include the Fourier or square wave basis, discretized over $\mathbb{V}$. The key $k_p = h_p(\mu, z) \in \mathbb{K}_p$ is a client defined function h_p of ID μ, and also any metadata z (which could be extracted after verification as we demonstrate in Section 5.1) if required. κ is a scalar that controls the perturbation magnitude.

Combining both components, our watermarking operator (Figure 4, and Algorithm 1 in Appendix) for generation step j involves (a) using $k_\pi = h_\pi(\mu, \hat{w}_{i-n+1:i-1})$ and the permutation operator $\mathcal{P}(k_\pi, L_j)$ to transform logits from the V_o to V_w space, (b) applying the perturbation operator in Equation (5), and (c) transforming the perturbed logits back to V_o space using $\mathcal{P}^{-1}(k_\pi, .)$ to produce a probability distribution for sampling and generation of the watermarked text T_{w}:

$$\begin{aligned} \check{L}_j &= \mathcal{W}(k_\pi, k_p, L_j) \\ &= \mathcal{P}^{-1}(k_\pi, \mathcal{F}(k_p, \kappa, \mathcal{P}(k_\pi, L_j))). \end{aligned} \tag{6}$$

Our verification operator will produce a score by computing the average cumulative token distribution of a text using $\bar{\mathcal{P}}(K_\pi, .)$ and taking the inner product with $\mathcal{F}_1(k_p)$. Applying the right keys k_p and k_π on the suspected text T_{sus} will result in a high score q, else the score will be close to 0 (see Figure 4, and Algorithm 2 in Appendix). Using orthogonal functions helps us improve verifiability by avoiding interference from other watermarks (e.g., added by adversaries as an $\mathbb{A}_3$ attack).

Notice that the many possible vocab permutations ($|\mathbb{V}|!$) and perturbation functions in any orthogonal function family $|\mathcal{F}_1|$ allows for a much large set of IDs compared to schemes like BASIC, helping with **scalability**. For example, up to $|\mathcal{F}_1| \cdot |\mathbb{V}|!$ IDs can be assigned to a unique permutation-perturbation function pair for watermarking. Using a relatively small $|\mathbb{V}| = 32000$ and the Fourier basis over that would yield a maximum $|\mathbb{M}| \sim 10^{130274}$. Schemes like BASIC only support M that scales with the number of possible synonym replacements for a given text.

In addition, with orthogonal functions, our framework also allows for the embedding of metadata during watermarking. For example, a client can use μ to verify that T_{sus} is watermarked, and also extract information on which article it was plagiarized from (Algorithm 3). We demonstrate this empirically in Section 5.1 using the Fourier basis as perturbation functions and Discrete Fourier Transform (DFT) for extraction.

A.4. WATERFALL Framework

Our watermarking framework, WATERFALL, combines these insights into a structured watermarking/verification process. For watermarking (Figure 4 left), given T_{o} and μ, WATERFALL uses an LLM paraphraser to autoregressively paraphrase a text T_{o}, producing initial logits for the new text T_{w} [Step ①]. The ID μ is used to seed the vocab permutation operator (Equation (2)) for mapping the logits to V_w space, and chooses the perturbation function (Equation (5)) [Step ②], both of which will be used in the watermarking operation (Equation (6)) to produce the perturbed logits [Step ③]. The LLM samples the perturbed logits to produce a watermarked token, and for the next token loop, the past $n-1$ tokens are used to seed vocab permutation while all past tokens are fed as context which helps the LLM paraphraser maintain T_{w} fidelity despite watermarking [Step ④].

For verification (Figure 4 right), each token in T_{sus} is counted in V_w space as specified by μ and the previous tokens in the same n-gram unit, producing an average cumulative token distribution [Step ①]. The ID μ also specifies a specific perturbation function [Step ②], which is used to perform an inner product with the cumulative distribution to compute a verification score q [Step ③].

Practical considerations. WATERFALL is highly adaptable, i.e., it can be implemented with different LLM as paraphrasers, allowing our framework to achieve better watermarking performance and support more text types as the LLM landscape evolves. Methods like prompt engineering (Wei et al., 2022; Lin et al., 2023) and Reflexion (Shinn et al., 2023; Madaan et al., 2023) may also help to boost performance in some settings, as we demonstrate in our code watermarking experiments (Appendix K.2). We elaborate further on possible large-scale deployment methods of WATERFALL and other practical considerations in Appendix I.

B. Additional details on watermarking and verification operators

Algorithm 1 WATERFALL Watermarking algorithm

Input: Original text T_{o}, ID μ, text-specific metadata z, n-gram length n, perturbation magnitude κ, keys functions h_π and h_p
Provide to LLM paraphraser a prompt $\hat{c}$ containing T_{o} and paraphrasing instructions, which represents semantic content c of T_{o}.
Compute $k_p = h_p(\mu, z)$.
for $j = 1, \ldots$ **do**
    Obtain logits $l_j(\hat{w}_{1:j-1}, \hat{c})$ from LLM paraphraser, given Equation (1).
    Compute $k_\pi = h_\pi(\mu, \hat{w}_{j-n+1:j-1})$.
    Compute perturbed logits $\check{l}_j$ based on Equation (6).
    Sample token $\hat{w_j}$ based on the perturbed probability distribution $\check{p_j} = \text{softmax}(\check{l_j})$.
end for
Output: Watermarked text $T_{\text{w}} = [\hat{w}_1, ..., \text{<eos>}]$.

Algorithm 2 WATERFALL Verification algorithm

Input: Suspected text $T_{\text{sus}} = [\hat{w}_1, \ldots, \hat{w}_N]$, ID μ, n-gram length n, keys function h_π, perturbation key k_p, test threshold $\bar{q}$.
Initialize a vector C of length $|V_o|$, which keeps track of token counts, to 0.
for $j = 1, \ldots, |T_{\text{sus}}|$ **do**
  Compute $k_\pi = h_\pi(\mu, \hat{w}_{j-n+1:j-1})$ and permutation operator $\mathcal{P}(k_\pi)$, given Equation (2).
  Set $C(\mathcal{P}(k_\pi, \hat{w}_i)) + +$.
end for
Compute avg cumulative token distribution $\bar{C} = C/N$.
Compute verification score $q = \langle \bar{C}, \frac{\mathcal{F}_1(k_p)}{\|\mathcal{F}_1(k_p)\|_2} \rangle$ based on Equation (5).
Output: Returns true if $q \geq \bar{q}$.

Algorithm 3 WATERFALL Extraction algorithm

Input: Suspected text $T_{\text{sus}} = [\hat{w}_1, \ldots, \hat{w}_N]$, ID μ, n-gram length n, keys function h_π.
Initialize a vector C of length $|V_o|$, which keeps track of token counts, to 0.
for $j = 1, \ldots, |T_{\text{sus}}|$ **do**
  Compute $k_\pi = h_\pi(\mu, \hat{w}_{j-n+1:j-1})$ and permutation operator $\mathcal{P}(k_\pi)$, given Equation (2).
  Set $C(\mathcal{P}(k_\pi, \hat{w}_i)) + +$.
end for
Compute avg cumulative token distribution $\bar{C} = C/N$.
Compute highest scoring key $\hat{k_p} = \arg\max_{k_p \in \mathbb{K}_p} \langle \bar{C}, \frac{\mathcal{F}_1(k_p)}{\|\mathcal{F}_1(k_p)\|_2} \rangle$ based on Equation (5).
Output: Returns $\hat{k_p}$.

C. Empirical illustration of watermarking signal in T_w

Here we empirically illustrate how the watermarking signal can be embedded in V_w space with the background logits appearing as uniform noise, as described in Appendix A.3. To illustrate the presence of the watermarking signal, we use the combined watermarked dataset used in the data ownership experiments, and plot its average cumulative token distribution $\bar{C}$ (in Algorithm 2).

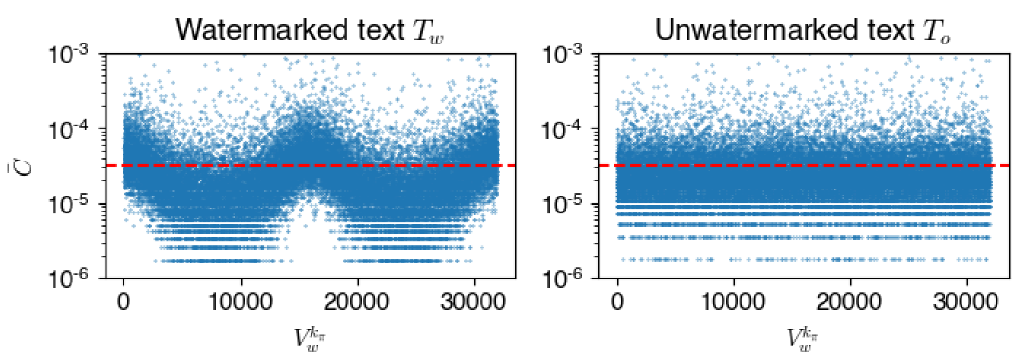

Figure 5: Average cumulative token distribution $\bar{C}$ of watermarked and unwatermarked text from subset of `c4 realnewslike` dataset. Fourier watermark signal with frequency 2 is clearly visible in T_w (left) as compared to T_o (right).

Figure 5 shows that when we use the correct ID and k_π for verification, the watermarking function can be clearly seen for the watermarked text T_w (distribution in the shape of a cosine curve of 2 periods for $k_p = 2$), while the unwatermarked text T_o shows a flat function.

Similarly, Figure 6 shows that when verifying watermarked text T_w, the watermarking function is only visible with the correct permutation $\mathcal{P}(k_\pi)$ (distribution in the shape of a cosine curve of 2 periods for $k_p = 2$), but not with a different permutation $\mathcal{P}(k'_\pi)$ (i.e., wrong ID).

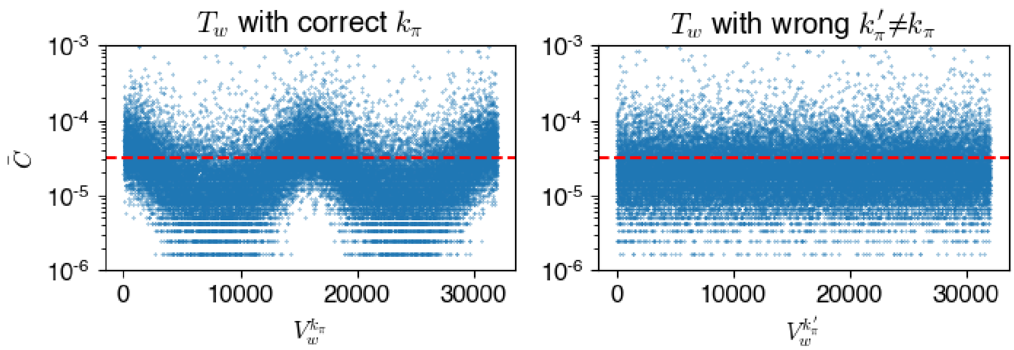

Figure 6: Average cumulative token distribution $\bar{C}$ of watermarked text from subset of `c4 realnewslike` dataset. Fourier watermark signal with frequency 2 is clearly visible when performing the correct permutation $\mathcal{P}(k_\pi)$ (left) compared to the wrong permutation $\mathcal{P}(k'_\pi)$ (right).

D. Discussion on weaknesses of existing text watermarking methods

Both benchmark text watermarking methods, M-BIT and P-NLW, are unable to achieve perfect verification performance despite having deterministic watermarking and verification algorithms, as stated in their respective papers, and corroborated in our experiments.

Both methods first use a language model to select viable word positions at which to perform the synonym substitution, then another model or word list to generate the list of possible synonym for substitution. During verification, we observe that the watermark could be corrupted in three ways.

Firstly, as the text being fed to the model for selecting the word replacement location is different (original text during watermarking and watermarked text during verification), the locations being selected during verification could be different as that used for watermarking.

Secondly, even if the correct locations are selected, a different synonym list could be generated during verification, due to the words that were changed at other locations during the watermarking process.

Thirdly, as the benchmarks perform watermarking by sequentially embedding the bits of the watermark ID into the text, any modifications to the text that inserts, deletes or shuffles the text would destroy the watermark ID. If an insertion or deletion error appears early in the text either through the first corruption above or through attacks, i.e., the location for a word replacement being inserted or removed during the verification as compared to during watermarking, the remainder of the watermark ID would be shifted in position, resulting the all the bits after the error to be in the wrong position, resulting in poor verifiability and robust verifiability. Additionally, as illustrated in Appendix A.2, attacks that reorders the text will also shuffle the watermark ID, destroying its robust verifiability.

On the other hand, WATERFALL is not susceptible to the above mentioned issues. As discussed in Appendix A.2, the watermark signal is injected into each n-gram in the watermarked text, and does not depend on the specific location within the sentence, or specific word replacements. As the hash function h_π is deterministic, the same permutation used during watermarking will always be selected during verification, as long as the n-gram unit is preserved.

E. Examples of orthogonal watermarking functions

We chose cosine and sine functions as the watermarking functions, due to the orthogonality between the cosine and sine functions of different frequencies.

$$\phi_{k_p}(j) = \begin{cases} \cos\left(2\pi k_p \frac{j}{|\mathbb{V}|}\right) & \text{if } k_p \leq \frac{|\mathbb{V}|}{2} \\ \sin\left(2\pi (k_p - \frac{|\mathbb{V}|}{2}) \frac{j}{|\mathbb{V}|}\right) & \text{otherwise} \end{cases}$$

where $j \in \{1, \ldots, |\mathbb{V}|\}$ denote the index in the vocab space, $k_p \in \{1, \ldots, |\mathbb{V}| - 1\}$ denote the index of the available orthogonal functions. We chose the cosine and sine sequences as any other bounded watermarking sequence can be represented by a collection of sinusoidal sequences via the discrete Fourier transform (DFT).

In general, periodic functions of different frequencies could be used as the system of orthogonal functions, along with the phase-shifted counterparts by phase of a quarter wavelength. Other than the cosine and sine functions, one other example is the square wave functions.

Let $k_N = \max_{k \in \mathbb{N}^*}\{k \mid |\mathbb{V}| \equiv 0 \pmod{2^k}\}$. Assuming $k_N \geq 2$, the number of orthogonal square waves supported is $2k_N - 1$, such that $k_p \in \{1, \ldots, 2k_N - 1\}$. The square watermarking function is defined as follows.

$$\phi_{k_p}(j) = \begin{cases} (-1)^{\lfloor 2^{k_p} \frac{j}{|\mathbb{V}|} \rfloor} & \text{if } k_p \leq k_N \\ (-1)^{\lfloor 2^{(k_p - k_N)} \frac{j}{|\mathbb{V}|} + 0.5 \rfloor} & \text{otherwise} \end{cases}$$

F. Data ownership experimental setting

F.1. Dataset

For watermarking of text articles, we demonstrate the effectiveness of WATERFALL with experiments using text samples T_o from the `c4 realnewslike` dataset (Raffel et al., 2020), comprising articles with mean token length of 412.

From the first 2000 samples in the c4 dataset, we selected text that were shorter than 1000 tokens long as our text samples T_o, totaling 1360 samples. We restricted the token length to ensure the paraphrasing prompt, original text and watermarked text can fit within the context window of the LLM used for paraphrasing. In practice, to overcome this limitation, longer original text could either be first split up into multiple sections to be watermarked, or an LLM with a longer context window could be used. The distribution of word and token lengths is shown in Figure 7.

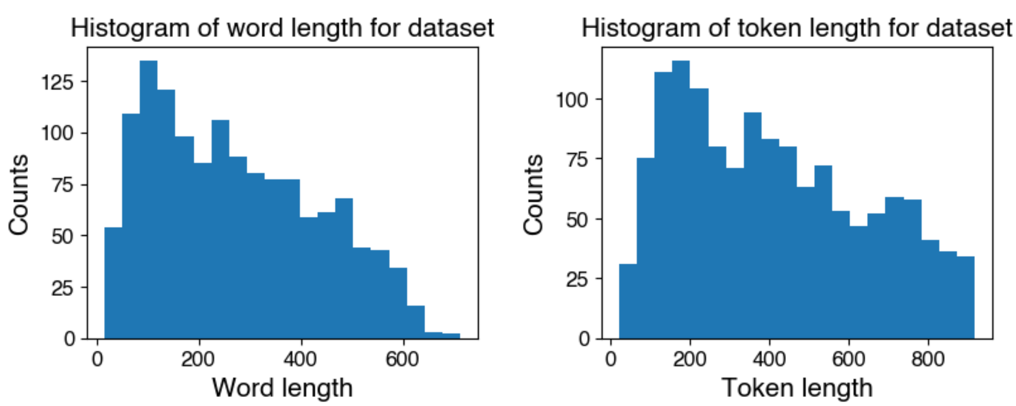

Figure 7: Histogram of word and token lengths of text in the `c4 realnewslike` dataset used for data ownership experiments.

F.2. Watermarking methodology

To implement WATERFALL, we use `llama-2-13b-hf`[2] as the paraphraser, and the Fourier basis for the perturbation functions.

To perform paraphrasing, we followed the prompt format for `llama-2-13b-hf`, and used the following prompt to perform watermarking. No effort has been made to optimize the prompt.

[2]https://huggingface.co/meta-llama/Llama-2-7b-chat-hf

```
[INST] <<SYS>>
Paraphrase the user provided text while preserving semantic similarity. Do not include any
     other sentences in the response, such as explanations of the paraphrasing. Do not
    summarize.
<</SYS>>

{text} [/INST]

Here is a paraphrased version of the text while preserving the semantic similarity:
```

For the results in the experimental (Section 5.1), the watermark was performed with ID $\mu = 0$ and $k_p = 1$. Results for other μ and k_p are reported below.

After watermarking, we perform a simple post-processing step to strip away extraneous generation by the LLM, by filtering out the last sentence or paragraph that contain the following phrases.

- let me know
- paraphrase
- paraphrasing
- other sentences
- original text
- same information
- Note:
- Note :
- Please note
- Please kindly note
- Note that I
- semantic similar
- semantically similar
- similar in meaning
- Please be aware
- the main changes made
- Kindly note
- Note this does
- I have made sure to

This list should be customized depending on the content of the text to be watermarked, and LLM used for watermarking. Other methods of cleaning the watermarked text such as prompting the LLM to critic or correct issues within the watermarked text could be employed (Shinn et al., 2023).

F.3. Benchmark experiment settings

P-NLW (Qiang et al., 2023) proposes a watermarking process by incorporating a paraphraser-based lexical substitution model. While **M-BIT** (Yoo et al., 2023) carefully chooses the potential original word to replace via finding features that are invariant to minor corruption, and a BERT-based lexical substitution model. We use these two approaches as the benchmark for text watermarking in the data ownership problem setting.

Key generation As default, both M-BIT and P-NLW use binary keys as watermark signals. The bits for the keys we use for experiments were generated with a seeded pseudo-random number generator. Specifically, we used 0 as the seed to NumPy's Random Generator to generate the key used in the experiments[3].

F.4. Verifiability

In this section, given threshold score $\bar{q}$, we define the classification problem as follows. Positive sample: watermarked text $T_{\text{w}}^{(i)}$; Negative sample: unwatermarked text T_{o}; Predictive positive: $\mathcal{V}\left(\mu_i, T_{\text{sus}}\right) \geq \bar{q}$, Predictive negative: $\mathcal{V}\left(\mu_i, T_{\text{sus}}\right) < \bar{q}$.

The ROC curves and corresponding AUROC values for different μ, k_p and κ are shown in Figure 8. We show that verifiability is insensitive to different μ, k_p used for watermarking. For $\kappa = 6$, WATERFALL was able to achieve AUROC of 0.989-0.996 across the different settings.

[3] NumPy random generator takes in an unsigned int as the seed

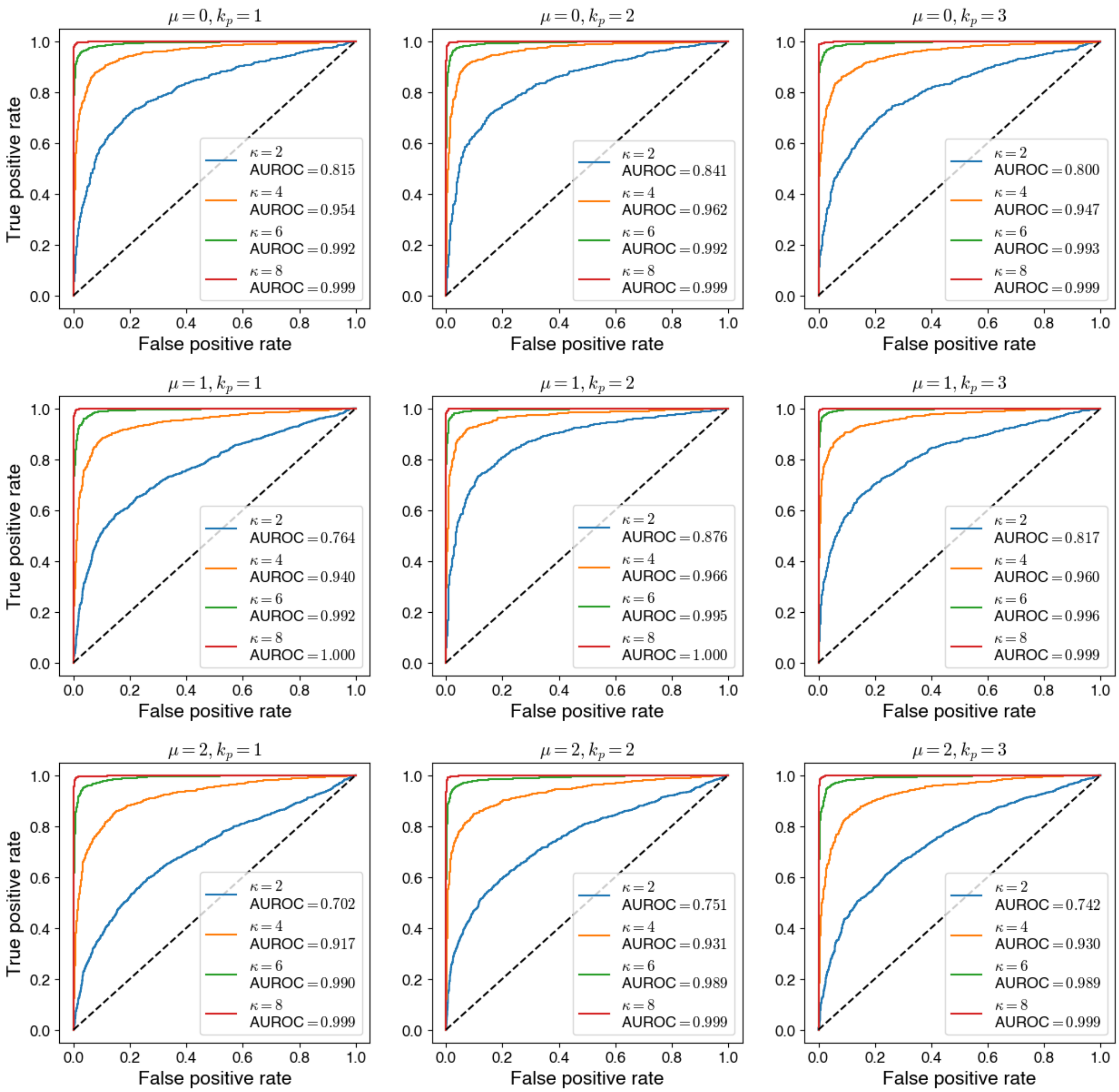

Figure 8: ROC curves and corresponding AUROC values for different μ, k_p and κ

F.5. Fidelity

For this setting, we evaluate the semantic similarity $\mathcal{S}$ using the Semantic Textual Similarity (STS) score based on the `all-mpnet-base-v2` model[4] ($\mathcal{S}$ for sample text pairs are provided in Appendix F.5).

WATERFALL allows for adjustable watermarking strength to calibrate the fidelity-verifiability trade-off based on the clients' use cases. Figure 2a shows the Pareto frontier of the trade-off. Stronger watermark strength κ improves verifiability but also introduces larger distortions to the LLM paraphrasing process, decreasing the fidelity of watermarked text. For our experiments, we mainly used $\kappa = 6$, achieving a mean AUROC of 0.992 and STS of 0.887.

Note that M-BIT and P-NLW were designed with only one setting, allowing for only a single fidelity-verifiability score, with

[4]https://huggingface.co/sentence-transformers/all-mpnet-base-v2

mean STS scores of 0.998 and 0.942 respectively, and corresponding AUROC scores of 0.987 and 0.882. While the STS scores are high, it is expected given that the schemes only make minor edits to T_o which would be more fragile to attacks, as we will see later. Additionally, the word replacements by M-BIT and P-NLW introduced noticeable linguistic errors that are difficult to evaluate and not captured by the STS score.

We provide some examples of text watermarked by the WATERFALL, M-BIT and P-NLW. Table 4 shows a few samples from the c4 dataset with watermarked text of varying STS scores. M-BIT has the highest STS across these samples listed, due to its algorithm only changing very few words within the text, resulting in lower scalability as described in Section 5.1. Despite the high STS score, it can be visually seen that text watermarked with M-BIT and P-NLW introduces linguistic and grammatical errors to the text, which are not measured by the STS score.

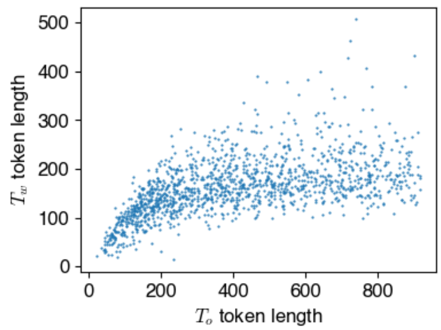

Figure 9: Distribution of token length of unwatermarked text T_o against watermarked text T_w

We noticed that there is a tendency of LLMs to summarize when performing paraphrasing, where some details of the text are lost during the watermarking process. This can be seen in the decrease in token length comparing the original unwatermarked text T_o against watermarked text T_w in Figure 9. However, there are multiple methods of mitigating this issue. Firstly, longer text could be broken apart into different sections to be watermarked separately before being combined together. Secondly, due to the robustness of WATERFALL to modifications, the watermarked text could be edited to correct for errors or reinsert missing information, either manually or by leveraging LLMs to perform self-critic, similar to the method described by Shinn et al. (2023). Lastly, as WATERFALL does not require the use or training of specific LLMs for watermarking, it can be applied to newer models with better paraphrasing performance as they become available.

Table 4: Text watermarked with WATERFALL, M-BIT and P-NLW, along with their STS scores. For M-BIT and P-NLW, highlighted words are words that were selected for watermarking, and may or may not have been changed as compared to the original text depending on the ID used for watermarking.

	Text	STS
Original	After the martyrdom of St. Boniface, Vergilius was made Bishop of Salzburg (766 or 767) and laboured successfully for the upbuilding of his diocese as well as for the spread of the Faith in neighbouring heathen countries, especially in Carinthia. He died at Salzburg, 27 November, 789. In 1233 he was canonized by Gregory IX. His doctrine that the earth is a sphere was derived from the teaching of ancient geographers, and his belief in the existence of the antipodes was probably influenced by the accounts which the ancient Irish voyagers gave of their journeys. This, at least, is the opinion of Rettberg (”Kirchengesch. Deutschlands”, II, 236).	-
WATERFALL	After St. Boniface’s martyrdom, Vergilius became the Bishop of Salzburg in 766 or 767 and worked tirelessly to strengthen his diocese and spread the faith in nearby heathen regions, particularly in Carinthia. He died in Salzburg on November 27, 789. In 1233, he was canonized by Gregory IX. According to Rettberg (“Kirchengesch. Deutschlands”, II, 236), Vergilius’ belief that the earth is a sphere was based on the teachings of ancient geographers, and his belief in the existence of the antipodes may have been influenced by the accounts of ancient Irish voyagers.	0.978
M-BIT (8 bits)	Following the martyrdom of St. Boniface, Vergilius became made Bishop of Salzburg (766 or 767) and laboured successfully for the upbuilding of his diocese as well as for the spread of the Faith in neighbouring heathen countries, especially in Carinthia. He died at Salzburg, 27 November, 789. In 1233 he was canonized by Gregory IX. His doctrine that the earth is a sphere was derived from the teaching of ancient geographers, and his belief in the existence of the antipodes was probably influenced by the accounts which the ancient Irish voyagers gave of their journeys. This, at least, is the opinion of Rettberg (“Kirchengesch. Deutschlands”, II, 236).	0.999
P-NLW (3 bits)	following the martyrdom of St. Boniface, Vergilius was made Bishop of Salzburg (766 or 767) and worked worked for the upbuilding of his diocese as well as for the spread of the Faith in neighbouring heathen countries, especially in Carinthia. He died at Salzburg, 27 November, 789. In 1233 he was canonized by Gregory IX. His doctrine that the earth is a sphere was derived from the teaching of ancient geographers, and his belief in the existence of the antipodes was probably influenced by the accounts which the ancient Irish voyagers gave of their journeys. This, at least, is the opinion of Rettberg (“Kirchengesch. Deutschlands”, II, 236).	0.964

	Text	STS
Original	Karl Kispert, principal of cyber and information security, has more than 28 years of experience in selling, managing and delivering information risk management, internal audit, regulatory and compliance programs, and information security and technology risk management. A former chief information security officer, Kispert has helped design and implement cybersecurity programs for many firms, according to the firm. "By adding this new service line, and bringing someone with Karl's expertise to the firm, we can service yet another important aspect of our clients' and prospects' businesses, ensuring their continued success," CEO Louis Grassi said in a written statement. Services will include full security programs, compliance, third party vendor risk assessment, threat management, and managed security services.	-
WATERFALL	Karl Kispert, a principal specialist in cybersecurity and information risk management, has extensive experience spanning 28 years in providing sales, management, and delivery of information risk management, internal audit, compliance programs, and technology risk management solutions. As a former Chief Information Security Officer, Kispert has supported the design and implementation of comprehensive cybersecurity programs for numerous organizations. The CEO of the firm, Louis Grassi, has expressed enthusiasm about expanding the firm's service offerings through the integration of this new service line, which will be supported by Kispert's proficiency in providing comprehensive security measures, compliance, vendor risk assessment, threat management, and managed security services.	0.899
M-BIT (5 bits)	Karl Kispert, principal in cyber and information security, has more than 28 years of experience in selling, managing and delivering information risk management, internal audit, regulatory and compliance programs, and information security and technology risk management. A former chief information security officer, Kispert had helped design and implement cybersecurity programs for many firms, according to the firm. "By adding this new service line, and bringing someone with Karl's expertise to the firm, we can service yet another important aspect of our clients' and prospects' businesses, ensuring their continued success," CEO Louis Grassi said in a written statement. Services offered include full security programs, compliance, third party vendor risk assessment, threat management, and managed security services.	0.9969
P-NLW (21 bits)	carl kisper, principal of cyber and information protection, has has than 28 old of experience experience selling, managing and delivery information risk risks, internal audit, regulatory cyber cybernetic programs, and information security and technology risk management. A former chief information security officer, Kispert has helped project and project cybersecurity programs for many firms, according to the firm. " By adding this new service line, and bringing someone with Karl' s expertise to the firm, we can service yet another important aspect of our clients ' and prospects ' businesses, ensuring their continued success, " CEO Louis Grassi said in a written job. Services will include full security programs, compliance, third party vendor risk assessment, threat management, and managed security services.	0.938

	Text	STS
Original	Larry checks in with KPCC reporter Sharon McNary, who's been hitting up several polling stations in Orange County and Los Angeles County, as well as Registrar of Voters for O.C. and L.A. After being a finalist for LAPD chief in 2009 only to see the job go to Charlie Beck, Michel Moore has been selected to succeed Beck by L.A. Mayor Eric Garcetti. President Donald Trump signed the "right-to-try" bill into law on Wednesday, a measure that gives terminally ill patients access to experimental drugs that have not yet been approved by the Food and Drug Administration (FDA). Humans have a habit of measuring things. Our shoe size. The ingredients in our food. How long it takes to get to work, with or without traffic.	-
WATERFALL	Larry talks with KPCC reporter Sharon McNary about polling stations and the Registrar of Voters in both Orange County and Los Angeles County. The Los Angeles Mayor, Eric Garcetti, has appointed Michel Moore as the new Chief of the LA Police Department after he was previously a finalist for the position in 2009. The US President, Donald Trump, signed a law giving terminally ill patients access to unapproved experimental treatments. Humans tend to quantify aspects of life, such as shoe size, food ingredients, commute times, and more.	0.857
M-BIT (4 bits)	Larry checks in with KPCC reporter Sharon McNary, who's been hitting up several polling stations in Orange County and Los Angeles County, as well as Registrar of Voters for O.C. and L.A. After being a finalist for LAPD chief in 2009 only to see the job go to Charlie Beck, Michel Moore has been selected to succeed Beck by L.A. Mayor Eric Garcetti. President Donald Trump signed the "right-to-try" bill into law on Wednesday, a measure that gives terminally ill patients access to experimental drugs that have not yet become approved by the Food and Drug Administration (FDA). Humans have a habit for measuring things. Our shoe size. The ingredients of our food. How long it takes to get to work, with or without traffic.	0.999
P-NLW (12 bits)	lary controls on on KPCC journalist Sharon McNary, who is s been attacked up several polling stations in Orange County and Los Angeles County, as well as Registrar of Voters for O.C. los L.A. After being a finalist for LAPD chief in 2009 only to see the job go to Charlie Beck, Michel Moore has been selected to succeed Beck by L.A. Mayor Eric Garcetti. President Donald Trump signed the " right-to-try " bill into law on Wednesday, a measure that gives terminally ill patients access to experimental drugs that have not yet been approved by the Food and Drug on (FDA). Humans have a habit of measuring things. Our shoe size. The ingredients in our food. How long it takes to get to work, with or without traffic.	0.829
Original	Come test your luck on the best slot machine app in the app store. Great graphics make this app so fun to play. Test your luck with Pharaoh Slots! Bet, Spin and Get Lucky!	-
WATERFALL	Experience the ultimate entertainment with the most thrilling slot machine game in the app store! Marvel at stunning visuals that make playing so enjoyable.	0.787
M-BIT (4 bits)	Come test your luck on the best slot machine app in the app store. Great graphics make this app so fun to play. Test your luck on Pharaoh Slots! Bet, Spin and Get Lucky!	0.9985
P-NLW (12 bits)	please test yourself happiness happiness the best place machine app in the app store. Great graphics make it app app fun to play. Test your luck with Pharaoh Slots ! Bet, Spin and Get get!	0.716

F.6. Verifiability fidelity trade-off

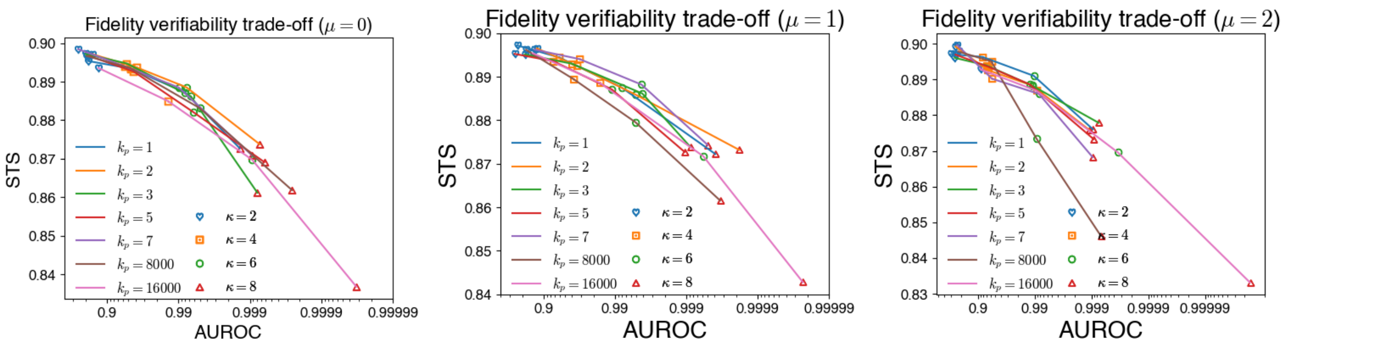

Figure 10: Fidelity and verifiability for different μ, k_p and κ

We observe that different values for μ does not result in noticeable impact on the fidelity and verifiability of the watermarked text, as shown in Figure 10. Varying k_p results in minor variations in fidelity and verifiability at high κ, but the pareto-front of the fidelity verifiability trade-off is similar across the different k_p. Clients using different k_p could adjust the value of κ to suite their requirements for fidelity and verifiability.

F.7. Scalability

We examine the scalability of WATERFALL and benchmarks M-BIT, P-NLW in practice by watermarking with different IDs and verifying with different IDs.

F.7.1. SCALABILITY WHEN VERIFYING WITH DIFFERENT IDS

Using a dataset of text watermarked with ID $\mu = i$, we compare the verifiability using the correct ID ($\mathcal{V}(\mu_i, T_{\text{w}}^{(i)})$) against verifiability using the wrong IDs ($\mathcal{V}(\mu_{j\neq i}, T_{\text{w}}^{(i)})$). Figure 11 shows the histogram plot for the AUROC comparing the 2 verification scores ($\mathcal{V}(\mu_i, T_{\text{w}}^{(i)})$ versus $\mathcal{V}(\mu_{j\neq i}, T_{\text{w}}^{(i)})$) for the different methods.

Notice that the AUROC of WATERFALL for the different IDs are all closely clustered around the high value of 0.985. However, the AUROC of benchmarks M-BIT and P-NLW show a very large range, with some IDs showing very low AUROC down to 0.69 and 0.53 respectively.

To further support our claim of WATERFALL having large scalability, we performed verification with 100,000 different IDs for WATERFALL. Figure 12 shows that the distribution of AUROC values are similar when scaling up from 1,000 to 100,000 IDs, and this performance could be extrapolated into millions of IDs.

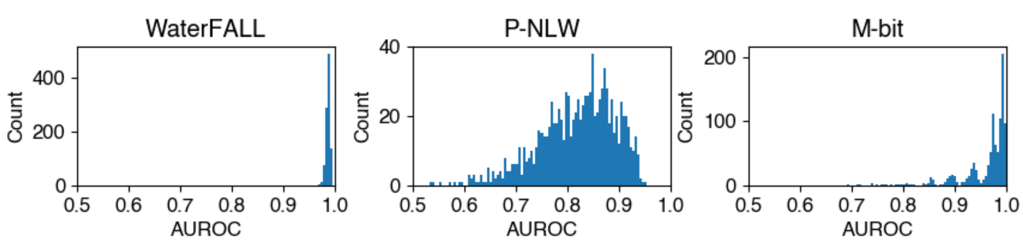

Figure 11: AUROC of $T_{\text{w}}^{(i)}$ when verifying with μ_i vs. $\mu_{j\neq i}$. WATERFALL has consistently high verifiability for all 1000 $\mu_{j\neq i}$, compared to benchmarks which have many $\mu_{j\neq i}$ with poor verifiability.

F.7.2. SCALABILITY WHEN WATERMARKING DIFFERENT IDS

We further explore the scalability of WATERFALL when verifying text watermarked with different IDs. We compare the verifiability using the correct ID ($\mathcal{V}(\mu_i, T_{\text{w}}^{(i)})$) against verifiability using the wrong IDs ($\mathcal{V}(\mu_i, T_{\text{w}}^{(j\neq i)})$). Due to the higher computational cost of watermarking compared to verification, we performed this experiments over a smaller subset of 358

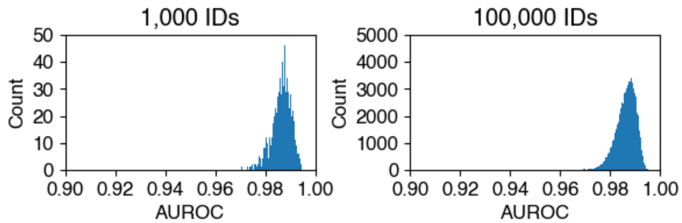

Figure 12: Scalability of WATERFALL for AUROC of $T_{\text{w}}^{(i)}$ when verifying with μ_i vs. $\mu_{j\neq i}$, when using 1000 IDs versus 100,000 IDs. Scaling up to 100,000 IDs shows the same narrow clustering of values around the high AUROC value of 0.985.

pieces of text of the `c4 realnewslike` dataset. 500 different IDs were used to watermark the dataset. Figure 13 shows the distribution of AUROC comparing the 2 verification scores $\mathcal{V}(\mu_i, T_{\text{w}}^{(i)})$ versus $\mathcal{V}(\mu_i, T_{\text{w}}^{(j\neq i)})$ for WATERFALL is closely clustered around 0.98, similar to the results in Appendix F.7.1. Note that a smaller number of text are considered for this experiment, resulting in the slightly difference in distribution.

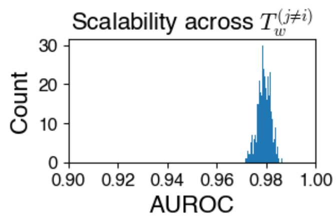

Figure 13: AUROC of $T_{\text{w}}^{(i)}$ vs. $T_{\text{w}}^{(j\neq i)}$ when verifying with μ_i. WATERFALL shows consistently high AUROC when verifying $T_{\text{w}}^{(i)}$ with μ_i compared to verifying $T_{\text{w}}^{(j\neq i)}$ with μ_i

F.7.3. DISCUSSION ON SCALABILITY IN PRACTICE

M-BIT, P-NLW suffer from poor scalability in practice, as shown above. As we consider watermarking or verification with the wrong ID $\mu_{j\neq i}$, there can be situations where the wrong ID differ from the correct ID at only 1 single bit, or very few bits. If the text is too short to be able to encode sufficient number of bits to include the differing bits, the watermarking method would be unable to differentiate between the 2 IDs during verification.

Even if the texts are sufficiently long, IDs that have few differing bits will be harder to differentiate. As discussed in Appendix D, errors could be present in the verification of watermark with M-BIT and P-NLW. Such errors could overshadow the small differences in the watermarking and verification IDs, resulting in poor verification performance. To achieve satisfactory performance, M-BIT and P-NLW would have to limit their scheme to IDs with sufficient number of differing bits, which further limit the scalability of their schemes.

On the other hand, WATERFALL is not susceptible to such issues. As the watermark signal is not embedded directly into the specific substitutions in the text space, but rather into signals in the permuted token space determined by a hash of the ID, small differences in the ID results in drastically different permutations in the token space, and they are extremely unlikely to collide, i.e., 2 different IDs are extremely unlikely to map to the same permutations over the entire piece of text. As a result, WATERFALL can achieve significantly higher scalability than M-BIT and P-NLW in practice.

G. Experimental details and additional results for attacks

G.1. $\mathbb{A}_1$

Following Kamaruddin et al. (2018), we design three types of attack: insertion, deletion, and synonym substitution attacks for $\mathbb{A}_1$. Attack strength indicates the rate of attacked words over the total number of words in a given content.

Insertion attack. We consider two types of insertion attacks mentioned in Kamaruddin et al. (2018):

(1) Localized insertion: this kind of attack inserts a random word into the original content at a random position. This is

labeled as “local” in Figure 14.

(2) Dispersed insertion: multiple random words are added in multiple random positions into the original content. In our experiment, we iteratively insert a random English word into a random position of the original content.

Deletion attack. Random words are deleted, to attempt to distort the watermark in the original content.

Synonym substitution attack. Given original content, the synonym substitution attack tries to replace some words with their synonyms. In our experiments, we use the Natural Language Toolkit (NLTK) (Bird et al., 2009) to find a set of synonyms for a certain word, then choose a random word in this synonym set to replace the original word. We used the random function in the NumPy library (Harris et al., 2020) to randomly select words to be substituted for these types of attacks.

As shown in Figure 14, robust verifiability of WATERFALL shows only a very slight decrease, while that of benchmarks M-BIT and P-NLW fall drastically with increasing attack strength.

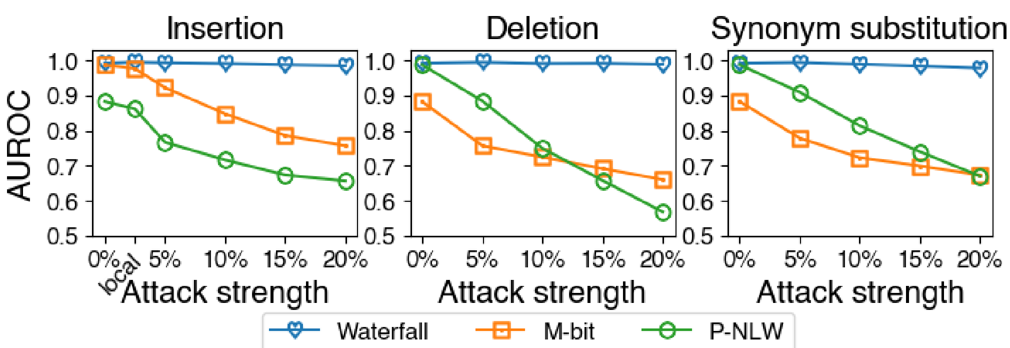

Figure 14: WATERFALL demonstrates robust verifiability under $\mathbb{A}_1$ (insertion, deletion, and synonym substitution attacks) with minimal degradation in AUROC compared to M-BIT and P-NLW.

G.2. $\mathbb{A}_2$

Translation attack was performed with `gpt-3.5-turbo-0613`, with the following prompts, where the language field is “Spanish” and “English”.

```
{
   'role': 'system',
   'content': 'Translate the provided piece of text to {language}.'
}
{
   'role': 'user',
   'content': '{text}'
}
```

Paraphrase attack was performed with `llama-2-13b-hf`, prompted in the following format.

```
[INST] <<SYS>>
Paraphrase the user provided text while preserving semantic similarity. Do not include any
     other sentences in the response, such as explanations of the paraphrasing. Do not
    summarise.
<</SYS>>

{text} [/INST]

Here is a paraphrased version of the text while preserving the semantic similarity:
```

We ran further experiments using different LLMs to perform paraphrasing attack. The robust verifiability of WATERFALL, M-BIT and P-NLW are reported in Table 5. WATERFALL achieves significantly higher robust verifiability than the benchmarks under paraphrasing attack across the different LLMs.

Table 5: Robust verifiability under paraphrasing attack with different LLMs.

	`gemma-7b-it`[5]	`Llama-2-7b-chat-hf`[6]	`Mixtral-8x7B-Instruct-v0.1`[7]	`gpt-3.5-turbo`
WATERFALL	**0.880**	**0.881**	**0.701**	**0.760**
M-BIT	0.524	0.509	0.522	0.385
P-NLW	0.374	0.359	0.467	0.512

G.3. $\mathbb{A}_3$

We show the results of $\mathbb{A}_3$ overlap watermark on WATERFALL when the watermark overlap was applied on μ or k_p in Table 6. We can see that WATERFALL can achieve high robust verifiability for overlap attack for both applications.

$\mathbb{A}_3$ on benchmarks with complement binary key We consider the worst-case scenario of robust verifiability under $\mathbb{A}_3$ for two traditional approaches P-NLW and M-BIT. Because these two methods are based on embedding binary keys in the watermarking stage, we try to apply $\mathbb{A}_3$ with the complement of the binary watermark key that was extracted as part of the verification process (replacing bit 0 with bit 1 and vice versa), to illustrate the worst-case scenario. We conduct this experiment with setting as 5.1. The results are illustrated in Table 7 and Figure 15. Do note that attacks could engineer their attacks by performing overlap watermarking with a mixture of watermark bits, random bits and complement bits, to target any AUROC value between the pre-attack and overlap complement AUROC.

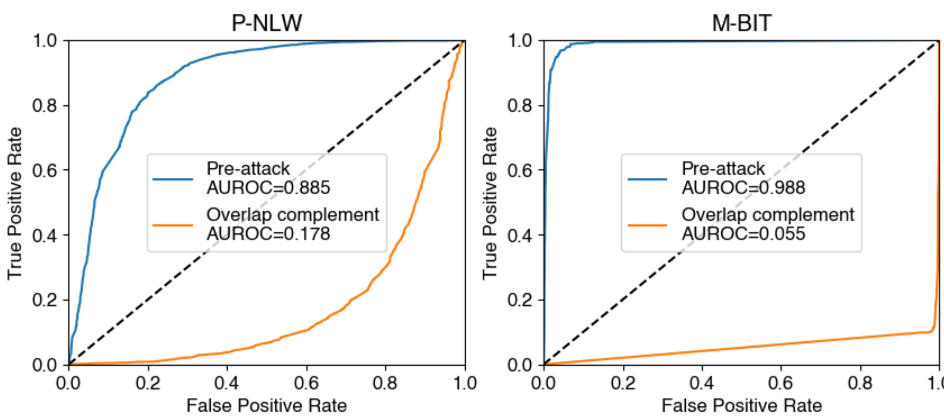

Figure 15: ROC curves and corresponding AUROC values of $\mathbb{A}_3$ with the complement of binary watermark key of P-NLW and M-BIT.

G.4. $\mathbb{A}_4$

To perform the in-context prompting experiments, we made use of `gpt-3.5-turbo-1106` to generate 3 questions each for 300 text articles. The following prompt was used to generate the questions.

[5] https://huggingface.co/google/gemma-7b-it
[6] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[7] https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

Table 6: Robust verifiability under overlap watermarking attack with different μ or k_p.

	Pre-attack	Post-attack
Overlap μ	0.992	0.815
Overlap k_p	0.992	0.743

Table 7: AUROC of P-NLW and M-BIT under $\mathbb{A}_3$ with the complement of binary watermark key (worst case scenario)

	Pre-attack	Overlap complement
P-NLW	0.8848	0.1780
M-BIT	0.9882	0.0547

```
{
   'role': 'system',
   'content': 'Using the provided article, create 3 reading comprehension questions.'
}
{
   'role': 'user',
   'content': '{text}'
}
```

We then separately prompt `gpt-3.5-turbo-1106`, providing the watermarked text as the context to answer the questions.

```
{
   'role': 'system',
   'content': 'Using the provided article, answer the questions.'
}
{
   'role': 'user',
   'content': '{text}\n\n{questions}'
}
```

G.5. Additional results for robust verifiability

Beyond AUROC reported in the main paper, we additionally report the true positive rate (TPR) at fixed false positive rate (FPR) of 0.1 and 0.01 for verifiability and robust verifiability under different attacks across different watermarking methods in Table 8.

Table 8: TPR at FPR of 0.1 and 0.01 for verifiability and robust verifiability.

FPR		Pre-attack	$\mathbb{A}_{2-T}$	$\mathbb{A}_{2-T}$	$\mathbb{A}_3$	$\mathbb{A}_4$
0.1	WATERFALL	0.982	**0.890**	**0.750**	**0.640**	**0.472**
	P-NLW	0.667	0.078	0.110	0.281	0.114
	M-BIT	**0.993**	0.126	0.126	0.520	0.000
0.01	WATERFALL	**0.910**	**0.608**	**0.405**	**0.284**	**0.122**
	P-NLW	0.110	0.007	0.010	0.037	0.032
	M-BIT	0.693	0.126	0.000	0.126	0.000

Note that under WATERFALL, we are able to drastically improve the verification performance when multiple pieces of text are available to be considered, where a realistic setting would involve multiple samples from the adversaries that we could test the watermarks for. In reality, IP holders are concerned about large-scale unauthorized IP use (i.e., multiple infringements) rather than one-off cases.

To demonstrate this, we ran an experiment where we test our watermarks given multiple samples under attack $\mathbb{A}_4$. Desipte the low TPR of 0.472 and 0.122 for FPR of 0.1 and 0.01 respectively when only considering 1 sample, our results demonstrates that given just 10 samples, we are able to achieve a TPR of 0.907 even with the strict requirement of a FPR of 0.01. The TPR increases to even 1.000 given 17 samples when we have the requirement of 0.1 FPR. This is also realistic because in practice, IP holders may use this as a screening tool for suspicious parties, to investigate them further, and hence would be alright with a higher FPR.

H. Metadata extraction

We also demonstrate how WATERFALL could be used to embed metadata while watermarking. We consider metadata $k_p \in \{1, 2, ..., 31998\}$, and the task is to extract the embedded k_p if the text has been verified as watermarked with μ. We do so by using k_p as the frequency of the Fourier perturbation function $\mathcal{F}_1$, and perform extraction with the Discrete Fourier transform (DFT). To evaluate extraction accuracy, we applied Algorithm 3 on the watermarked text. The accuracy is calculated based on the percentage of exact matches (extracted $\hat{k_p}$ matches the k_p used to watermark the text).

Figure 16 shows the extraction accuracy of WATERFALL for different perturbation magnitudes κ. Note that as there are 31999 supported k_p when using the Fourier basis functions with `llama-2-13b-hf` as the paraphraser, the probability of randomly guessing the correct k_p is $\frac{1}{7}31999 = 0.003125\%$. Despite this, WATERFALL is able to achieve high extraction accuracy of 48% when extracting from a single text for our default setting of $\kappa = 6$. This performance can be further improved when more pieces of watermarked text are available, such that accuracy improves to the high value of 99% with only 5 pieces of text. This is done by combining multiple pieces of text watermarked by the same ID μ and perturbation key k_p, by simply summing the cumulative token counts in V_w space, C, of the different pieces of text, before performing step 7 of Algorithm 3.

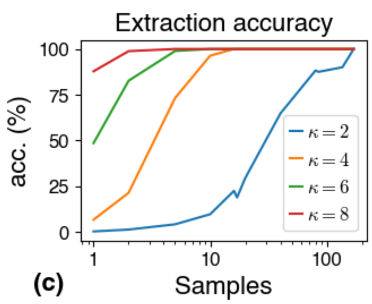

Figure 16: Higher watermark strength κ and more samples of watermarked text improves extraction accuracy.

I. Practical considerations for real world deployment of WATERFALL

WATERFALL's initial setup and computational resources for large-scale applications are low and practically viable. This makes actual large-scale deployment of text watermarking feasible, which is currently not possible given the current state of the art (SOTA) watermarking methods' limitations and resource requirements.

We illustrate this by laying out two approaches (decentralized or centralized) to deploying WATERFALL, both of which have low initial setup and computational cost requirements.

I.1. Decentralized deployment

In this approach, clients randomly generate their own IDs (given the large space of supportable IDs), and can do watermark and verification operations on their own using their laptops with minimal setup.

Setup For most common text types/languages supported by LLMs, clients could immediately run WATERFALL with no setup, given a default LLM and WATERFALL settings, to generate the watermarked text T_{w}.

Computational cost WATERFALL's watermarking computational cost is just that of running inference of the LLM paraphraser, with negligible overheads. Using a GPU available in many laptops (Nvidia RTX 5000), a user could use the Llama-2-13b model to watermark a text in < 25s to already achieve great performance, as shown in Table 2 in our paper. We expect that the cost of running high performance LLMs on personal devices (e.g., MacBooks, laptops with GPUs) will get cheaper and cheaper, given the rapidly evolving landscape of LLMs.

WATERFALL's verification operation is extremely fast and can be run on just CPU (< 0.04s per text), without the need for any LLM. For practical applications, the verification operation will be the main operation run multiple times, rather than the watermarking operation (typically only once before the user publishes the text). WATERFALL's verification operator is 2-5 orders of magnitude faster than baseline text watermarking methods (Table 2 in our paper).

I.2. Centralized deployment

In this approach, central parties assigns clients unique IDs, and run the WATERFALL watermarking and verification operations for them. This is similar to how LLM service providers are providing interfaces or APIs for LLM queries.

Setup At a minimum, they could do the same as individuals in the decentralized approach and not need to do any setup. However, given their scale, they could also provide customized service by optimizing the choice of LLMs and WATERFALL settings for specific non-common text types or other user requirements (see Appendix I.3 below for clarification on adaptability).

Computational cost Existing LLM service providers could easily provide this additional watermarking service to clients, given the minimal overheads of WATERFALL over processing a single LLM chat API call. The speed of our verification operation even allows companies to provide value-added services such as near-real-time scanning of newly-published articles from target sources to detect any plagiarism.

I.3. Adaptability to different LLMs

A key strength of WATERFALL is that it evolves together with the evolving landscape of LLMs, with increasingly better watermarking performance as LLMs become more capable. As LLMs become more capable, they would be able to better preserve semantic meaning of the original text while still embedding watermarks when used as LLM paraphrasers via WATERFALL. This allows WATERFALL to achieve higher fidelity-verifiability Pareto frontier, and reduce any fidelity degradation while using higher watermarking strength for greater robust verifiability.

To illustrate, we have performed additional experiments with other LLM models as paraphraser models, with the same `c4-realnewslike` dataset used in the main paper. Figure 17 shows that the newer/larger models have higher Pareto fronts with higher STS scores for the same verifiability values. Going forward, we expect further significant improvements in LLM capabilities, allowing WATERFALL's performance to also significantly improve.

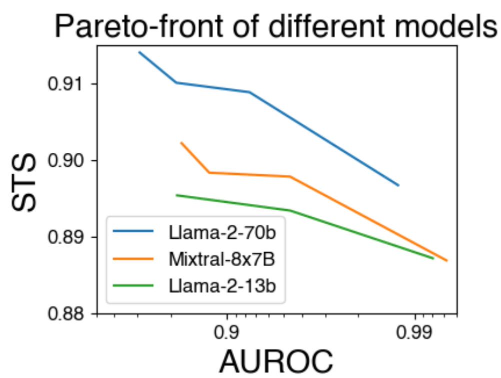

Figure 17: Plot of Pareto frontier of different LLMs, where larger/newer models show better Pareto fronts on the fidelity-verifiability trade-off.

I.4. Selection of watermarking LLM and hyperparameter

As with any adaptable methods, WATERFALL would require some effort to gain boosted performance in specific domains (e.g., text type or language). That said, the WATERFALL framework is designed to reduce such efforts, and it is relatively easy for a user to perform such fine-tuning given only 1 hyperparameter to tune (watermarking strength κ) and the choice of LLM paraphraser. For example, the user could just follow these simple steps:

1. Identify the SOTA LLM for the domain, to use as the LLM paraphraser component. As a domain expert and content creator (of the text to be watermarked), the client should be familiar with what is available. Given the evolving landscape of LLMs, we believe that it is realistic for each domain to have a relatively capable fine-tuned model.
2. Run WATERFALL with default watermarking strength κ and assess if the fidelity and robust verifiability of the text

meets expectation. As a domain expert, the client can assess if the text has sufficient fidelity or use a domain-specific fidelity metric to automate the check. The client can also use an automated suite of robustness checks (comprising standard attacks) would assess the expected robust verifiability of the watermarked text.

3. If the results are not up to expectation, perform optimization over the κ hyperparameter using standard AutoML methods like Bayesian Optimization (BO). This could be automated especially if a fidelity metric is provided, but manual sequential checks could also be used given just 1 hyperparameter and a query-efficient approach like BO.

In practice, if WATERFALL is widely adopted, an open research or developer community would also likely be able to share such configurations and fine-tuning, similar to how fine-tuned deep learning models are also being shared today. Even if WATERFALL is implemented by closed-source companies, economies of scale would make it worth fine-tuning and optimizing WATERFALL across languages and text types.

I.5. Refinement of watermarked text to improve fidelity

As paraphrasing is applied to the original text when performing the watermark, there might be a change in the style of writing, some loss in information, or in the case of code watermarking, loss of functionality. However, these can be mitigated through several techniques, some of which we have already implemented in our experiments.

In practice, the client could assess the fidelity of the watermarked text T_w before using it. If T_w does not meet the fidelity threshold (i.e., semantic content is not sufficiently preserved), the client could simply use the LLM paraphraser to correct the watermarked text T_w to increase semantic preservation. This could be done automatically as demonstrated in the code example (e.g., Reflexion, or multiple generations), or done manually with prompt engineering. The LLM paraphraser will once again introduce the same embedded watermark to produce the new watermarked text T'_w, strengthening both the verifiability and fidelity of the text.

Additionally, as the field develops, it is expected for LLMs' paraphrasing capabilities to increase significantly across domains, languages and text types. This enables the WATERFALL framework, using these more capable LLMs, to generate watermarked text with smaller and smaller semantic degradation, further improving its performance and allowing WATERFALL to remain effective in highly specialized or technical domains.

J. Details of experiments on LLM data provenance

J.1. LLM fine-tuning experimental setup

For our experiments, we watermarked the ArXiv dataset (Clement et al., 2019) which consists of scientific paper abstracts categorized into topics. Each topic category is associated with a unique client ID μ with 4000 text. These texts are then used to fine-tune the `gpt2-xl` model using the LoRA framework (Hu et al., 2022)[8] (details in Appendix J.1).

To fine-tune the 1.5B parameter `gpt2-xl` model, we used the LoRA framework (Hu et al., 2022), with LoRA rank of 16 and target modules `c_attn, c_proj, c_fc`. The models were fine-tuned for a total of 5 epochs, with default batch size of 128 and learning rate of 0.0003. Fine-tuning was performed on a single Nvidia L40 GPU, requiring an average of approximately 15 minutes per client (4000 data samples for each client) for the fine-tuning of the model.

J.2. Verifiability of watermark in the model fine-tuned over watermarked text

To evaluate the verifiability of the watermark, we prompted the fine-tuned model with randomly selected abstracts from the training set used to fine-tune the model. We truncated the abstracts to the first 50 tokens, which is supplied to the model without any other additional prompts, for the model to generate completions to the input. We limited the generation to a maximum of 100 newly generated tokens. Note that in real applications, generating more tokens could improve the verifiability performance, as we have demonstrated in the data ownership experiments. Only the generated tokens were considered when evaluating the verifiability of the watermark. The same ID and k_p used during watermarking was used to perform verification. For the model fine-tuned on the original unwatermarked text, the corresponding ID and k_p that was used for the watermarked text was used for verification.

[8] Note that this is a different model compared to that used for watermarking. We chose this to demonstrate that our watermark can persist despite the models' different tokenizers.

J.3. Fidelity of model fine-tuned over watermarked text

We used `lm-evaluation-harness`[9] (Gao et al., 2021) to evaluate the fine-tuned models for its fidelity over several different datasets (Gao et al., 2020; Merity et al., 2016; Wang et al., 2018; Dolan & Brockett, 2005; Bisk et al., 2020; Levesque et al., 2011). Table 9 reports the models fine-tuned over the watermarked datasets results in minimal differences in fidelity as compared to the model fine-tuned over the unwatermarked datasets. This shows that act of watermarking data used for fine-tuning does not significantly affect its value for fine-tuning.

Table 9: Fidelity of model fine-tuned using watermarked text (Watermarked) and unwatermarked text (Unwatermarked) of different number of clients M, evaluated over the various datasets.

Dataset		M				
		1	5	10	20	100
Pile-ArXiv (ppl)	Watermarked	2.209	2.218	2.218	2.180	2.166
	Unwatermarked	2.192	2.210	2.197	2.170	2.154
Wikitext (ppl)	Watermarked	1.771	1.770	1.780	1.787	1.818
	Unwatermarked	1.766	1.769	1.774	1.783	1.814
MRPC (acc)	Watermarked	0.662	0.618	0.674	0.581	0.326
	Unwatermarked	0.679	0.627	0.627	0.380	0.314
PIQA (acc)	Watermarked	0.687	0.676	0.682	0.676	0.673
	Unwatermarked	0.686	0.682	0.683	0.680	0.678
WNLI (acc)	Watermarked	0.563	0.620	0.535	0.549	0.493
	Unwatermarked	0.620	0.577	0.592	0.563	0.535

K. WATERFALL in code watermarking

K.1. Code watermarking experiment settings

In the main paper, we report the result of code watermarking on the MBJSP dataset (Athiwaratkun et al., 2023) with the data ownership problem setting. This is a JavaScript dataset including around 800 crowd-sourced JavaScript programming problems. To show the ability of WATERFALL on watermarking other programming languages, we also perform data ownership watermarking on Python datasets, which can be found in Appendix K.5.

In this setting, we use `Phind-CodeLlama-34B-v2`[10] , as LLM paraphraser for code watermarking, the square wave basis with $k_p = 1$ (Appendix E) for watermark perturbation and randomly choose $\mu = 10$ in all code experiments. As default, we denote WATERFALL code to indicate WATERFALL in this code watermarking settings. Moreover, we also show that prompt engineering techniques, such as Reflexion (Shinn et al., 2023) could improve the fidelity of watermarked code while preserving the verifiability (Appendix K.2). For SRCMARKER (Yang et al., 2024), we configured their algorithm for 16-bit watermarks, to demonstrate scalability of at least 10^5.

For verifiability evaluation, we use the same evaluation protocol as article watermarking in Section 5.1. As a result, the ROC curve and AUROC values for WATERFALL code are shown in Figure 18

Watermarked code fidelity evaluation As mentioned in the main paper, we evaluate the fidelity of the watermarked code by evaluating its accuracy based on functional tests for the original code and use the standard pass@k metric (Kulal et al., 2019; Chen et al., 2021) for evaluating functional correctness. Given the deterministic nature of the baseline SRCMARKER (Yang et al., 2024), which inherently upholds fidelity, the pass@10 metric is adopted to facilitate a fair comparison between WATERFALL and SRCMARKER in terms of fidelity performance. This metric specifically measures the likelihood of WATERFALL producing watermarked code that passes unit tests within 10 generation attempts. The pass@10 metric is also realistic in practice as it aligns with real-world scenarios where clients can assess the quality of watermarked code through predefined tests and subsequently regenerate the code if test failures arise.

To evaluate the functional correctness of code, we adapt the JavaScript evaluation protocol from Athiwaratkun et al. (2023)

[9]https://github.com/EleutherAI/lm-evaluation-harness
[10]https://huggingface.co/Phind/Phind-CodeLlama-34B-v2

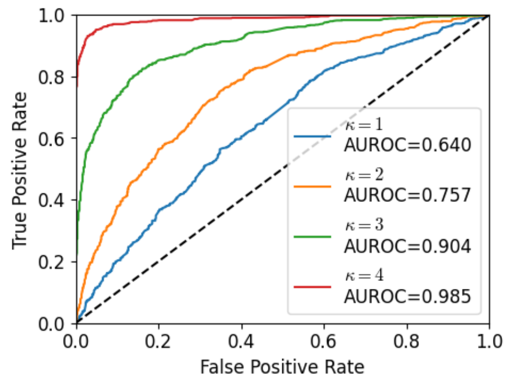

Figure 18: The ROC curves and corresponding AUROC values on the MBJSP dataset using WATERFALL code.

for the MBJSP dataset. On the other hand, for Python evaluation, we adapt the HumanEval (Chen et al., 2021) code evaluation protocol[11] and test script from both datasets (Chen et al., 2021; Austin et al., 2021). However, the watermarked code usually modifies the original function name into some related names, so we use Levenshtein distance to find the new text function in the watermarked code. For a more precise evaluation of the watermarked code, this related function name-finding process can be improved by using other similarity distances, such as the Semantic Textual Similarity (STS) score.

K.2. WATERFALL code + Reflexion methodology

In this section, we show that some prompt engineering approaches could help the watermarked code improve fidelity without hurting the verifiability. Adapting the techniques from Shinn et al. (2023), we try to correct the watermarked code through the LLM-based self-reflection mechanism. After being watermarked with WATERFALL code, this watermarked code undergoes a correcting process via multiple feedback loops (3 feedback loops in our experiments). Each feedback loop contains two self-reflection components aiming to perform syntax correction and functional similarity alignment. Each self-reflection component performs two main steps: 1) evaluating or analyzing the given information based on task criteria, e.g., the correctness of programming syntax. 2) regenerate the "better" code based on given feedback.

Applying the same LLM in WATERFALL code to the self-reflection component plays a crucial role in this combination. This is simply because LLM is a good way to handle and generate linguistic feedback, which contains more information than scalar results in the evaluation step. Moreover, watermarking LLM helps the final code preserve the robust and scalable watermark signal through the correction step, which is the ultimate goal of our text watermarking framework. The prompts to perform the syntax correction step and functional similarity alignment are illustrated in Appendix K.6.

The effect of the Reflexion approach is shown in Figure 19. From this illustration, we can see that Reflexion improves fidelity while maintaining high verifiability of WATERFALL code. So we apply this technique in all code watermarking experiments.

K.3. Verifiability and fidelity trade-off

Figure 20 shows the trade-off of verifiability and fidelity can be adjusted via κ. Similar to article watermarking in Section 5.1, increasing watermark strength κ can increase verifiability but lower fidelity. Therefore, the users can adjust κ to balance the trade-off based on their preference.

K.4. Scalability of WATERFALL in code watermarking

One of the advantages of WATERFALL over baseline SRCMARKER is in terms of scalability. SRCMARKER verifiability depends heavily on the number of watermarked bits (scalability), larger number of bits, worse verifiability (Yang et al., 2024). Therefore, to ensure high verifiability, SRCMARKER can not support larger scalability. In contrast, the verifiability of WATERFALL is independent to its scalability, and this scalability only depends on the vocabulary size of the tokenizer.

[11] https://github.com/openai/human-eval

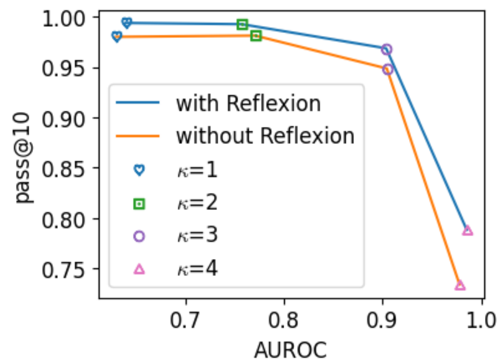

Figure 19: The effect of Reflexion in WATERFALL code on MBJSP dataset

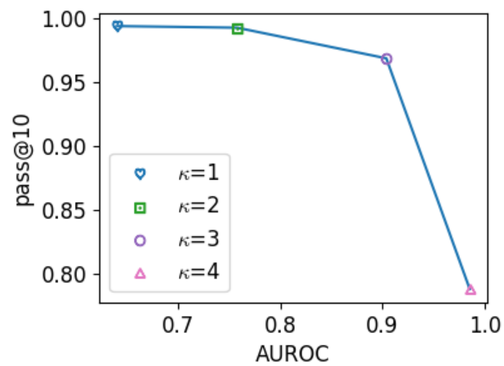

Figure 20: Verifiability and fidelity trade-off of WATERFALL code on the MBJSP dataset

In our experiments (Table 3), we use `Phind-CodeLlama-34B-v2` , which has a large vocabulary size as same as `llama-2-13b-hf`, which $M \sim 10^{130274}$, far better than $M \sim 10^5$ of SRCMARKER 16-bits.

K.5. WATERFALL in watermarking Python code

Inheriting the multi-lingual ability of LLM, WATERFALL can easily apply to new programming languages without the need for pre-defined syntax rules. This is a big advantage of WATERFALL in comparison to AST-based code watermarking approaches like SRCMARKER (Yang et al., 2024). We show that WATERFALL can also watermark Python code, through experiments on the MBPP dataset (Austin et al., 2021) which includes around 1000 crowd-sourced Python programming problems. We show the verifiability and fidelity results of WATERFALL on watermarking Python code in Table 10.

	pass@10	AUROC
MBJSP	0.969	0.904
MBPP	0.954	0.897

Table 10: WATERFALL code achieves high verifiability and fidelity on MBJSP and MBPP datasets.

K.6. LLM prompts for code watermarking

We use the following prompts and apply the chat template of `Phind-CodeLlama-34B-v2`, which follows the alpaca instruction prompt format on these prompts.

Code paraphrasing

````
### System Prompt
You are given a user-provided code snippet.
Please do ONLY two tasks:
1. Refactor the provided code snippet with the following requirements:
- retain all imported libraries.
- keep the same programming language.
- retain the function names and functionality of the code.
- don’t complete the code, just refactor it.
- don’t explain.
2. Return the response with the refactored code snippet in the following format strictly:
```
<refactored code>
```
Do not generate any comments or explaining texts.
### User Message
```
{input code}
```
### Assistant
Here is the refactored code:
```
````

Functional similarity alignment

````
### System Prompt
You are given two code snippets, code A and code B. Modify code B based on code A, such
    that these two code have the same functionality, input, and output. Return the
    response with corrected code B in the following format strictly:
```
<corrected code B>
```
Do not generate any comments or explaining texts.
### User Message
code A:
```
{original code}
```

code B:
```
{watermarked code}
```
### Assistant
Here is the code B:
```
````

Code syntax correction

````
### System Prompt
Double-check the code to make sure the syntax is correct. Only generate the corrected code
     in the following format.
```
<corrected code>
```
Do not generate any comments or explaining texts.
### User Message
```
{watermarked code}
```
### Assistant
Here is the corrected code:
```
````
````

K.7. Watermarked code examples

Examples of code watermarking by WATERFALL are illustrated in Figure 21. Note that WATERFALL code changes not only the variable names but also the ways of representing the same code logic, which results in high verifiability while preserving high fidelity.

Original code

```
function tupString(tup1) {
  let str = "";
  let i = 0;
  while (i < tup1.length && tup1[i] !== " ") {
    str += tup1[i];
    i++;
  }
  return str;
}
```

Watermarked code

```
function extractNonSpaceString(tupIn){
  let extractedStr = '';
  let idx = 0;

  for (const val in tupIn){
     if(tupIn[val] === ' ') break;
     extractedStr += tupIn[val];
     idx++
  }
  return extractedStr;
}
```

Original code

```
function evenBitToggleNumber(n) {
    switch (n) {
    case 10:
        return 0;
    case 20:
        return 30;
    case 30:
        return 20;
    }
}
```

Watermarked code

```
function evenToggle(n) {
  let returnVal=0
  if(n===30){
     returnVal= 10
  }
  else if(n===20){
     returnVal= 10
  }
  else{
     returnVal= 10
  }
  return  ( returnVal & ~1 ) ^ n

}
```

Original code

```
function countSubstringWithEqualEnds(s) {
  var count = 0;
  for (var i = 0; i < s.length; i++) {
    var subString = s.substring(i, s.length);
    for (var j = 0; j < subString.length; j++) {
      if (subString[j] === s.charAt(i)) {
        count++;
      }
    }
  }
  return count;
}
```

Watermarked code

```
function countSubstringWithEqualEnds(str) {
  let count = 0,
      subStr;

  for (let start = 0, length = str.length; start < length; start
+= 1) {
   subStr = str.substring(start, length);
   subStr.split('').forEach((char, index) => {
     if(char == subStr[index] && subStr[index] ===  str[start]){
       count++;
     }
   }, this);
  }
  return count;
 }
```

Original code

```
function flattenList(list1) {
  if (list1.length < 1) {
    return [];
  }
  let result = [];
  list1.forEach(item => {
    if (Array.isArray(item)) {
      result = result.concat(flattenList(item));
    } else {
      result.push(item);
    }
  });
  return result;
}
```

Watermarked code

```
function flattenArray(arrayList = []) {
  let flatList = [];
  for (const item of arrayList) {
     if(Array.isArray(item))
         {
         const flatenedItem=flattenArray(item);
         for (const sub_item of flatenedItem){
         flatList.push(sub_item)}

     }

     else {
     flatList.push(item)}

  }
  return  flatList
}
```

Original code

```
function charFrequency(str1) {
    let arr = str1.split('');
    let obj = {};
    arr.forEach((ele) => {
        let key = `"${ele}"`;
        if(obj.hasOwnProperty(key)) {
            obj[key] += 1;
        }
        else {
            obj[key] = 1;
        }
    });
    return obj;
}
```

Watermarked code

```
function charCount(str) {
  let chars = str.split('');
  let counts = {};

  chars.forEach((char) => {
     let key = `"${char}"`;
     counts[key] = (counts[key]) ? counts[key] + 1 : 1;
  });

  return counts;
}
```

Original code

```
function checkChar(string) {
    if (string.startsWith('a') && string.endsWith('a')) {
        return "Valid";
    }
    if (string.startsWith('b') && string.endsWith('b')) {
        return "Valid";
    }
    if (string.startsWith('c') && string.endsWith('c')) {
        return "Valid";
    }
    if (string.startsWith('d') && string.endsWith('d')) {
        return "Valid";
    }
    return "Invalid";
}
```

Watermarked code

```
function validateStringStartAndAnywhereMatch(str) {
  const validOptions = 'abcd';
  for (const option of validOptions) {
     if(str.endsWith(option)){
         if(str.startsWith(option)){
            return 'Valid';
         }
     }
  }
  return 'Invalid';
}
```

Figure 21: Example of watermarked code with WATERFALL.