WaterLLMarks: In-data User Tracing for Distributed LLMaaS Environments

Léo Lavaur
University of Luxembourg
Luxembourg
leo.lavaur@uni.lu

Jérôme François
University of Luxembourg
Luxembourg
jerome.francois@uni.lu

***Abstract*—Due to their size and cost of operation, Large Language Models (LLMs) are often deployed in the cloud on Large Language Model as a Service (LLMaaS) platforms. Following the trends in cloud-based services and multi-tenancy, these platforms are becoming increasingly distributed. Monitoring such environments is challenging, especially without access to the underlying infrastructure. Moreover, applications may rely on multiple services from different providers, further complicating end-to-end observability.**

In this paper, we propose a novel type of telemetry, termed *in-data telemetry*, where services are outsourced across multiple LLMaaS providers. We illustrate its application in tracing user interactions with LLMs by embedding user-specific watermarks in the text. These watermarks propagate transparently through multiple services with minimal impact on the semantics of the generated content. We evaluate our approach in a Retrieval-Assisted Generation (RAG) scenario and showcase its potential in a demonstrator.

***Index Terms*—Large Language Models, Observability, Telemetry, Distributed Systems, LLMaaS, Watermarking, Retrieval-Assisted Generation**

I. Introduction

Since the release of ChatGPT in 2022, Large Language Models (LLMs)-based conversational agents have become widespread. Many new providers have emerged, offering Large Language Model as a Service (LLMaaS) platforms. To scale with user demand, these services rely heavily on distributed architectures, where tasks are decomposed and executed across different components. AI-based systems relied on this service model before the LLM era [1]. Tracking service dependencies in this type of distributed system is inherently challenging, an already known problem in serverless and cloud-native paradigms. In addition, subtasks can be delegated to third-party providers, leading to partial observability. In this work, we name this setting multi-tenant LLMaaS, where the services are managed by multiple independent actors. Industry standards such as OpenTelemetry[1] assume that services are instrumented and that data collection occurs within a controlled domain. Unfortunately, these assumptions do not hold in multi-tenant applications.

Thus, the traceability of user requests, whether for troubleshooting, accountability, or security purposes, is challenging in multi-tenant LLMaaS. This is amplified by reasoning-driven GenAI agents, which autonomously decide which AI and non-AI services to invoke [2]. This makes dependencies between subtasks performed by different providers increasingly unpredictable, complicating traceability. However, LLM-based services promote flexibility in user inputs (such as long prompts, images, etc.), making encapsulation of additional information possible.

Hence, we introduce `WaterLLMarks` for tracing users by embedding telemetry data directly into user-level data. Therefore, the contributions of this paper are threefold.

1) We propose a novel approach to trace user interactions with LLMs by embedding telemetry data directly in the user prompt. Thus, it is service-agnostic and transparent to the underlying LLM services, enabling user identification across subsequent requests.
2) Because a core problem is to minimize the impact of embedded telemetry on the semantic integrity of generated content, we review and evaluate two applicable watermarking techniques and discuss their potential in the context of LLM applications.
3) We showcase the potential of our approach with a demonstrator in a Retrieval-Assisted Generation (RAG) scenario where we use `WaterLLMarks` to filter access to documents.

The remainder of this paper is organized as follows. In Section II, the problem is refined and positioned against the related work. Our approach, `WaterLLMarks`, is described in Section III. The evaluation methodology and results are detailed in Section IV and Section V, respectively. We finally showcase the potential of our approach in a RAG scenario in Section VI before concluding in Section VII.

II. Preliminaries

A. Problem Refinement

`WaterLLMarks` targets a multi-tenant LLMaaS environment where the infrastructure and services are operated

This research is supported by the Luxembourg National Research Fund (FNR) (grant reference: C23/IS/18088425/COCTEL).

[1] https://opentelemetry.io/ (2025/02/05)

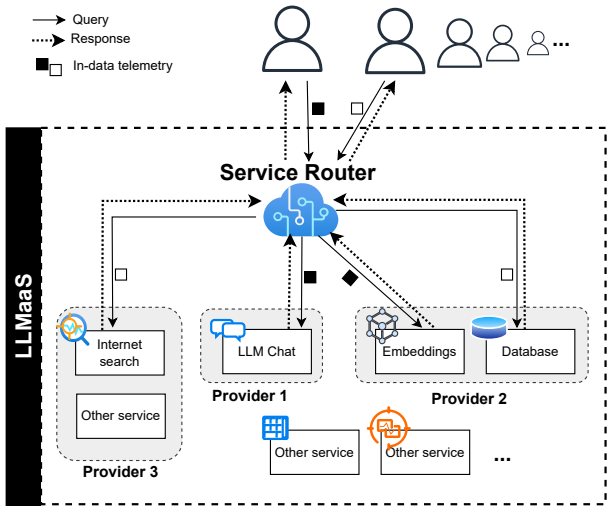

Fig. 1: In-data telemetry for LLMaaS.

by multiple independent providers. As illustrated in Fig. 1, services are not all AI-specific. The *Service Router* can also be offered by any independent provider as an entry point for users or user applications. Its role is to decompose their requests into smaller tasks and redirect each of them to the appropriate service. Today, major LLM providers already offer distinct services like chat completion or embedding creation through a single API access. However, in our case, we consider these services to be accessed through multiple independent providers. Our objective is to trace user requests and their subsequent decomposition. Hence, each initial user request is made traceable by adding a user-specific identifier into the request itself.

As such, our approach works by leveraging the application layer to embed telemetry. We term this *in-data telemetry*, by analogy with in-band network telemetry, which uses the data plane to propagate telemetry data, as depicted in Figure 1. Its main advantage is to alleviate the need for a complex Identity and Access Management (IAM) system coordinated between all involved entities (user applications, service router, and service providers). Because the request format must be flexible enough, this is well suited for inputs such as textual prompts, images, or videos. The core problem we address in this paper is to limit the impact on the resulting responses, *e.g.* the generated content of LLM services, when requests are altered due to in-data telemetry.

B. Background & Related Work

1) Observability: Observability refers to the ability to understand the internal state of a system based on its behavior. *Intrusive* tracing is a common solution that allows one to follow the path of a request through the different services by propagating identifiers or annotations. This approach was initially proposed by Google with `Dapper` [3] and has since been implemented in various forms, such as Zipkin[2] or Dynatrace[3]. OpenTelemetry is slowly becoming the *de facto* standard for tracing in the industry, pushed notably by the Cloud Native Computing Foundation (CNCF) [4]. OpenLLMetry[4] is a LLM-specific extension to OpenTelemetry. However, a major limitation of these approaches is that they assume the different services to be instrumented.

Non-intrusive tracing alleviates this need by relying on existing observable data: invocation of REST-based APIs to extract dependencies between requests [5] or lower network layers [6] for indirect observations. `WaterLLMarks` lies in between intrusive and non-intrusive tracing. It is transparent and does not require adoption from LLMaaS providers, but injects watermarks into user requests.

2) Watermarking: A common technique to track the use, copy, and share of data is watermarking. The information proving ownership or origin is embedded in the data itself, but remains hidden or imperceptible enough to keep the original content usable, *i.e.*, readable in the case of text. It has been widely used in the context of multimedia content, such as images, audio, or video. More recently, its relevance for LLMs gained interest, notably for copyright protection and detecting Artificial Intelligence (AI)-generated content [7]. For text watermarking, the authors identify four families: format-based (*e.g.*, line shift, character substitution), lexical-based (*e.g.*, synonym substitution), syntactic-based (*e.g.*, *pacification*) and generation-based (*i.e.*, using LLMs).

A major challenge is to avoid semantic alteration due to the watermarks. For example, Rizzo *et al.* [8] proposed a text watermarking technique based on Unicode character substitution between similar characters, for example between capital letters C (`\u00A9`) and Roman numeral one hundred C (`\u216D`). This works well for written text, as most users cannot distinguish the difference using a standard Unicode-compatible font.

III. WaterLLMarks

`WaterLLMarks` leverages text-based watermarking to encode telemetry in user requests. The objective is to propagate them transparently through multiple services with minimal impact on the semantics of the generated content. Hence, the following requirements must be met:

1) **User-specificity:** The watermark must be unique to a user, even if the same request (*i.e.* prompt) is made.
2) **Transparency:** The watermark must be transparent and not alter the semantics of the generated answer.
3) **Detectability:** The watermark must be easily detectable by service providers to enable user-specific policies, *e.g.*, regarding access or usage.
4) **Robustness to forgery:** The watermark must be robust to forgery attempts.

[2]https://zipkin.io/ (2025/02/05)
[3]https://www.dynatrace.com/ (2025/02/05)
[4]https://www.traceloop.com/docs/openllmetry (2025/02/05)

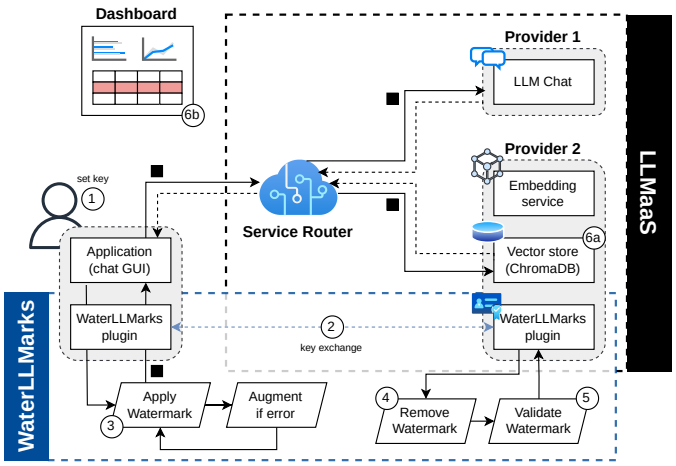

Fig. 2: `WaterLLMarks`'s workflow in a RAG use case (see Provider 2). The numbers refer to the steps delineated in Section III.

5) **Capacity:** The watermark must have a sufficient capacity to embed user-specific information and can support numerous clients.

Accordingly, `WaterLLMarks` incorporates seamlessly with pre-existing services, as long as the watermark-embedding process can be integrated into the user's application. The process can be divided into six distinct steps. First, the user configures their client with a secret key (1), which remains unique to them. This key must be known by any service provider offering additional service to this specific user (2). From that point on, all user queries will be watermarked before leaving the client application (3). When receiving a request, service providers start by removing the watermark to recover the initial user request (4), then iterate over their keychain to verify if they manage to validate the watermark (5). Upon validation, the service can be granted (6a). Additionally, this information can be used to monitor usage and trace users across the different services (6b). Figure 2 illustrates the operation of `WaterLLMarks` in a RAG scenario, using our demonstrator.

A. Watermark computation

Text watermarking often relies on Message-Autentication Code (MAC) algorithms, where the fingerprint w of a message t via a function $\mathcal{H}$ also depends on a password k, such as $\mathcal{H}(t,k) \rightarrow w$. k is a specific secret that is used for user authentication. If the same prompt t is used by two users with their secrets k and k', such algorithms ensure that $\mathcal{H}(t,k) \neq \mathcal{H}(t,k')$ and thus prove the authorship.

Rizzo *et al.* [8] identified `SipHash` [9] as a good candidate for text watermarking. Indeed, `SipHash` only produces 64-bit fingerprints, well below the different HMAC-SHA variants (*e.g.*, 160 for SHA-1), while being up to four times faster than MD5 [9]. This is relevant for text watermarking as it compensates for the small entropy of text, allowing the information embedding in smaller messages.

For these reasons, `WaterLLMarks` relies on `SipHash`, allowing users to attest to the origin of their prompts and prevent watermark forgery.However, the secret key used in `SipHash` makes it akin to symmetric cryptographic schemes, meaning that no third party can verify the watermark without the secret k. To avoid this issue, `WaterLLMarks` maintains a map of user secrets and iterates over it to identify the author of the watermarked message, if valid. The details of the key sharing between the provider and the monitored users are out of the scope of this study, although any form of secured key-exchange such as Diffie-Hellman could apply.

B. Watermark integration

Once a watermark $\mathcal{H}(t,k)$ is calculated, it is included in the original content t, *i.e.*, the user prompt. To fulfill the requirements related to transparency and detectability, we propose two schemes: `token` and `integrated`. The former consists of prepending a 16-byte watermark to the prompt, enclosed in a pair of delimiters (*e.g.*, a pair of brackets). Although it can be easily retrieved to verify the user identity, it is also easily detectable by anyone and can thus be easily removed.

The second technique encodes the watermark in the prompt using Unicode characters. We apply the embedding proposed by Rizzo *et al.* [8], which relies on two tables of substitution characters: various-length whitespaces and a set of homoglyphs. Each substitution table can be seen as a map $\mathcal{M} : a \rightarrow v$, where a is a bit array (of length 1 for character substitution and 3 for whitespace substitution) and v is the Unicode character to substitute. By iterating over the message, each character is checked if it can be substituted to embed the next n bits of the watermark. Due to the particular context of AI where we intend to use this mechanism, we prefer to refer to this type of watermarking as *integrated* to avoid confusion with vector embeddings. Both removal and verification of the watermark are trivial, as the provider can simply iterate over the message and extract the watermark by applying reverse character substitution ($\mathcal{M}^{-1}$).

C. Prompt augmentation

Although the token-based watermarking technique can be added to the prompt without other considerations, the integrated watermark technique supposes enough substitutable characters to embed the watermark. In Rizzo *et al.* [8], the authors concluded that a 100-character query would be needed to embed a 16 byte watermark 50% of the time. This lower bound may be an issue in the context of RAGs, where a prompt is often a short question or a few keywords. To address this limitation, we generate a longer version when needed, using an LLM with low temperature to preserve the semantics of the original prompt.

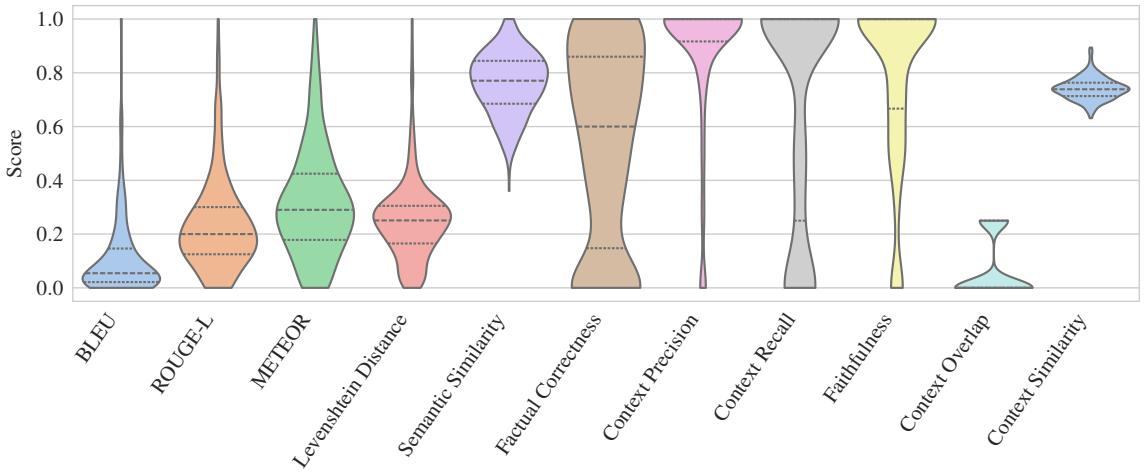

Fig. 3: Baseline results.

IV. Evaluation Methodology

The sought properties of our approach are guaranteed by design, except for *transparency*: The altered request may have a negative impact on response quality. To evaluate that, we instantiate a RAG scenario: When a user prompt is submitted, it is first embedded as tokens to retrieve similar contextual information from a document database. Then, this context allows the LLM to answer more precisely. Hence, both LLM services (embedding and answering) can be impacted by the watermarks, and the quality of both must be assessed.

The evaluation relies on the *2024 LLM Papers*[5] dataset from AutoRAG [10], composed of a corpus of 8,576 chunks of text from 110 scientific papers. It also provides a set of 520 questions and answers. The questions and answers were generated using OpenAI's ChatGPT-4 model. Because we aim to evaluate the specific impact of the watermarking techniques, we first evaluate our baseline without watermarking, before using these results as a reference for the evaluation of the watermarking techniques.

Setup: Using vLLM [11] on a server with 32 vCPUs, 64 GB of RAM, and an NVIDIA RTX A6000 GPU, two open-source models have been deployed: `Mistral-7B-Instruct-v0.3` for the LLM model, and the vLLM-recommended embedding model `e5-mistral-7b-instruct`. The application is implemented using Langchain, while the evaluation is done with RAGAS[6].

Metrics: We consider 11 metrics from the literature [7], [12] and existing evaluation platforms [13]. The first four metrics are common string similarity metrics from the Natural Language Processing (NLP) community: BLEU, ROUGE-L, METEOR, and Levenshtein distance. The other metrics focus on LLM (two) and RAG (five):

- *semantic similarity*: the cosine similarity between the embedding vectors of a response and a reference.
- *factual correctness*: the F1-score (harmonic mean between recall and precision) based on the number of claims in the response and the reference.
- *context precision*: the proportion of relevant fragments in the retrieved contexts, assessed by a LLM.
- *context recall*: the proportion of reference contexts that have been retrieved.
- *faithfullness*: the proportion of claims in the answer that can be inferred from the context, in the total number of claims that have been extracted.
- *context overlap*: the Jacard distance between the retrieved and reference contexts.
- *context similarity*: pairwise cosine similarity between reference and retrieved context chunks.

V. Evaluation Results

A. Baseline

The baseline experiment shows how our pipeline without watermark varies from the original data set only due to differences in the configuration of the RAG pipeline, for example, by using different LLMs. This is necessary to evaluate what is an acceptable variation, as both RAG deployments are valid.

Figure 3 presents the violin plots of the 11 metrics. All standard NLP metrics perform poorly, with average scores of $0.11_{\pm 0.15}$, $0.24_{\pm 0.18}$, $0.32_{\pm 0.21}$, $0.25_{\pm 0.15}$ for BLEU, ROUGE-L, METEOR, and the Levenshtein distance, respectively. This is expected because these metrics are not well-suited for evaluating LLM applications, especially when the answer spans multiple sentences. In contrast, semantic similarity is better to grasp the actual meaning of the response, despite possible variations in the wording, with an average of $0.76_{\pm 0.11}$.

Most importantly, most context-related metrics are good, especially when using LLM to assess the quality of

[5] https://huggingface.co/datasets/MarkrAI/AutoRAG-evaluation-2024-LLM-paper-v (2025/02/05)

[6] https://www.ragas.com/ (2025/02/05)

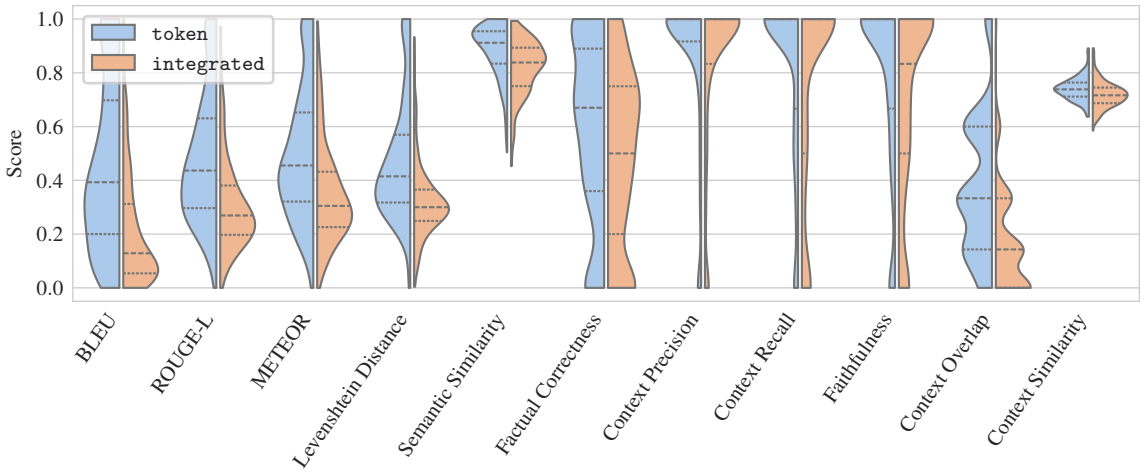

Fig. 4: Watermarking techniques results.

the generated text, *e.g.*, with claim extraction. However, the actual context overlap is extremely low, at $0.07_{\pm 0.11}$ on average and a maximum of 0.25. This means that although the retrieved contexts are relevant to answer a question and convey most of the same information as the reference, they differ most of the time.

B. Watermarking techniques

Figure 4 reports the violin plots for the two watermarking techniques: `token` on the left and `integrated` on the right. We now use our baseline results as a reference. Hence, all results are obtained using the same RAG pipeline, which leads to increased standard NLP metrics.

However, watermarking techniques still have a significant impact on the syntax of the generated text (as observed in the four NLP metrics), but keep the semantics of the text mostly intact. The semantic similarity remains high, with $0.90_{\pm 0.10}$ and $0.82_{\pm 0.10}$ for `token` and `integrated`, respectively. Likewise, most context-related metrics remain stable, with a better context overlap with the reference when compared to the baseline evaluation.

Moreover, we can observe that the use of a `token` has a lower impact than the `integrated` watermark. This can be observed across all metrics, with an average decline of $-0.12_{\pm 0.07}$ for the `integrated` watermark compared to the `token`. This can be explained by the character substitution used in the `integrated` watermark, which can alter the semantics of multiple words in the prompt, while the `token` is simply added in front of it. As a result, when tokenizing the prompt, the `integrated` watermark can lead to a different intermediate embedding vector and thus to a different answer. Furthermore, the `integrated` watermark has a greater impact on the size of the message, with an average increase of $+45.04_{\pm 5.2}$ bytes compared to the `token` watermark as shown in Table I based on the 162 questions that did not require prompt augmentation.

TABLE I: Message size in bytes.

Question	Mean size	Difference
`raw`	$128.00_{\pm 24.5}$	n/a
`token`	$146.00_{\pm 24.5}$	$+18.00_{\pm 0.0}$ (14.5%)
`integrated`	$173.04_{\pm 25.4}$	$+45.04_{\pm 5.2}$ (36.27%)

C. Prompt augmentation

The `integrated` watermark requires a minimal prompt size to embed the watermark, which varies depending on the number of spaces and substitutable characters. We observed that our watermarking technique failed to embed the watermark in 358 out of 520 questions, as the prompt was too short. `WaterLLMarks` implements a simple try-fail-retry mechanism to augment the prompt until the watermark can be embedded. The prompt augmentation was successful in 357 out of 358 cases, with an average of 1.02 retries (6 entries required two augmentations). In a single case, the augmentation caused the prompt to be too long for the deployed embedding model.

Figure 5 presents the scores obtained by the 357 augmented questions on the original baseline metrics, compared to the baseline. The results suggest that the augmentation has a significant impact on the quality of the generated text, with a decrease in the semantic similarity and most context-related metrics.

VI. Demonstrator

We showcase the potential of our approach with a demonstrator in a RAG scenario where we use `WaterLLMarks` to perform user-based access control on documents and so on retrieved contexts. More generally, we demonstrate the potential of our approach in a multi-provider LLMaaS environment by correlating dependent user requests even if they are handled by different services. Figure 2 details the considered architecture and

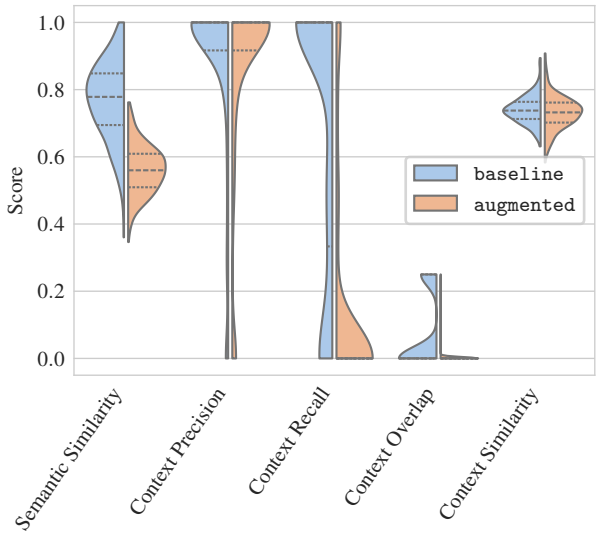

Fig. 5: Data augmentation results.

`WaterLLMarks`'s operation in that context, the code being available in open access[7].

Our RAG setup relies on a small-scale container-based LLMaaS environment and consists of four main components: a LLM provider (*i.e.*, vLLM), a vector database provider (*e.g.*, ChromaDB), an embedding service (which can also be hosted on a vLLM instance), and a service router. On the client side, a chat interface allows users to interact with the RAG application (see Figure 2), but also to apply their own watermark to the prompt. Finally, a dashboard allows for monitoring and visualizing the interactions between the different services, highlighting requests from the same users. The document provider can then filter access to the retrieved documents based on the watermarks, which will also be displayed in the dashboard.

VII. Conclusion

In this paper, we introduced `WaterLLMarks`, a novel approach to trace user interactions in a multi-provider LLMaaS environment. We leverage watermarking techniques to embed user-specific information in the queries, allowing specific providers to identify users accessing their services, despite the lack of cross-provider integration. Although watermarks can have a significant impact on the generated text, semantics are mostly preserved. We also presented a demonstrator that showcases the potential of our approach in a RAG setup. We believe that `WaterLLMarks` opens up new perspectives and future work for the traceability of user interactions in LLMaaS environments, and can be applied to other forms of content-based services.

[7] https://gitlab.com/uniluxembourg/snt/sedan/coctel/waterllmarks-demo

References

[1] M. Yan, P. Castro, P. Cheng, and V. Ishakian, Building a chatbot with serverless computing. In *1st International Workshop on Mashups of Things and APIs*, ACM, 2016.

[2] Y. Liu, S. K. Lo, Q. Lu, L. Zhu, D. Zhao, X. Xu, S. Harrer, and J. Whittle, Agent design pattern catalogue: A collection of architectural patterns for foundation model based agents. *Journal of Systems and Software*, 2025.

[3] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, Dapper, a Large-Scale Distributed Systems Tracing Infrastructure., 2010.

[4] D. Gomez Blanco, Opentelemetry fundamentals. In *Practical OpenTelemetry: Adopting Open Observability Standards Across Your Organization*. 2023.

[5] M. Cinque, R. D. Corte, and A. Pecchia, Microservices monitoring with event logs and black box execution tracing. In *IEEE World Congress on Services (SERVICES)*, 2021.

[6] J. Shen, H. Zhang, Y. Xiang, *et al.*, Network-centric distributed tracing with deepflow: Troubleshooting your microservices in zero code. In *ACM SIGCOMM Conference*, New York, NY, USA: ACM, 2023.

[7] A. Liu, L. Pan, Y. Lu, J. Li, X. Hu, X. Zhang, L. Wen, I. King, H. Xiong, and P. Yu, A Survey of Text Watermarking in the Era of Large Language Models. *ACM Comput. Surv.*, no. 2, 2024.

[8] S. G. Rizzo, F. Bertini, and D. Montesi, Content-preserving Text Watermarking through Unicode Homoglyph Substitution. In *Proceedings of the 20th International Database Engineering & Applications Symposium*, 2016.

[9] J.-P. Aumasson and D. J. Bernstein, SipHash: A Fast Short-Input PRF. In *Progress in Cryptology - INDOCRYPT 2012*, 2012.

[10] D. Kim, B. Kim, D. Han, and M. Eibich. "AutoRAG: Automated Framework for optimization of Retrieval Augmented Generation Pipeline." (2024), [Online]. Available: http://arxiv.org/abs/2410.20878.

[11] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. "Efficient Memory Management for Large Language Model Serving with PagedAttention." (2023), [Online]. Available: http://arxiv.org/abs/2309.06180.

[12] R. Friel, M. Belyi, and A. Sanyal. "RAGBench: Explainable Benchmark for Retrieval-Augmented Generation Systems." (2025), [Online]. Available: http://arxiv.org/abs/2407.11005.

[13] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert. "RAGAS: Automated Evaluation of Retrieval Augmented Generation." (2023), [Online]. Available: http://arxiv.org/abs/2309.15217.