

ELC: Escaping Large Language Model Crawling for Secure Information Release

Qinuo Li, Meng Jin[§], Guanchu Wang[*], Xia Hu[*]
{liqinuo,wangguanchu,huxia}@pjlab.org.cn

https://github.com/qinuoli/ELC

https://kinolee.pythonanywhere.com

https://youtu.be/FIPgPYIj6UE

Abstract

Frontier Large Language Models (LLMs) can aggressively crawl internet-scale information during their task execution, which poses risks by inadvertently collecting sensitive information. To address this problem, we release ELC (**E**scaping **LLM C**rawling), a framework for restricting LLM crawling during information release. Specifically, ELC applies **Server-Side Encryption** to protect backend data from LLM crawling at the exposure stage, and performs **Render-time Decryption** to recover the backend data for human consumption. This encryption-decryption framework effectively reduces crawling risks from frontier LLMs by an average of **93%**. As a lightweight system, ELC can be seamlessly integrated into existing web services, restricting LLM crawling in general scenarios.

1 Introduction

Frontier Large Language Models (LLMs) exhibit strong capabilities in active information seeking during problem solving [1]. For example, LLMs can proactively crawl internet-scale information from diverse sources to support task execution [2]. However, despite these benefits, such large-scale crawling also introduces risks by unintentionally collecting sensitive information, such as personal records from individual websites and informal opinions or announcements from organizations. Once incorporated into LLM reasoning processes, this sensitive information raises significant security and ethical concerns, which highlight the necessity of restricting LLM crawling during information release.

It is a non-trivial challenge to restrict LLMs from crawling information while allowing access for human users. Existing approaches can be broadly categorized into two lines [3]. The first line of methods relies on account-based access control, where content is accessible only to authenticated users with valid login credentials [4]. Under this setting, LLMs without personal accounts or authorized credentials are unable to access protected information. While this approach is effective in restricting automated crawling, it is not applicable to general open-access scenarios without login requirements. The second line of methods introduces human verification mechanisms, such as graphical, question-based, or behavioral verification codes, to distinguish human users from LLMs [5]. Although these methods can effectively prevent LLM crawling, they inevitably disrupt the seamless browsing experience for human users and reduce usability. To address these limitations, we propose an information release framework that restricts LLM crawling and applies to general open-access scenarios without relying on verification of human identity, thereby preserving both accessibility and usability.

To enable information to escape from LLM crawling, we identify the key differences between LLM web crawling and human browsing behaviors. Specifically, LLMs primarily crawl Server-Side rendered information at the exposure layer to minimize the retrieval overhead [6]. Since LLMs typically fetch large volumes of web sources and perform additional

[*]Corresponding to Guanchu Wang and Xia Hu.
[§]Work completed while at Shanghai Artificial Intelligence Laboratory.

Table 1: Comparison between ELC and existing web content protection methods.

Method	Open Access	Low Burden	No Extra Operation	Hide Plaintext	No Crawler Compliance
Login / Permission Control	✗	✗	✗	✓	✓
CAPTCHA / Human Verification	✗	✗	✗	✗	✓
robots.txt / noindex	✓	✓	✓	✗	✗
Rate Limiting / IP Blocking	✓	✗	✓	✗	✓
Browser Extension	✗	✗	✗	✓	✓
ELC	✓	✓	✓	✓	✓

reasoning over the information, they only have very limited time on individual pages to ensure the efficiency of end-to-end inference. In contrast, human users consume the information after the Client-Side rendering, which typically incurs higher latency [7]. However, humans are more tolerant of such delays, as they have limited comprehension speed and typically consume one page at a time. Building upon this insight, we release ELC (**E**scaping **LLM** **C**rawling), a framework designed to restrict LLM crawling in general open-access scenarios. Specifically, ELC introduces server-side encryption to protect the backend data from LLM crawling at the exposure layer, performs render-time decryption at the client side to reveal data to human users. This encryption-decryption framework can effectively reduce the rate of LLM crawling by **93%** on average. As a lightweight system, ELC can be seamlessly integrated into existing web services, protecting sensitive data of stakeholders in general scenarios.

2 Related Work

Existing methods for restricting web crawling can be grouped into several categories. The first category is account-based access control. These methods require users to log in before accessing protected content [4]. They are effective for private resources and closed user groups. Their usage is limited in open-access information release, where human users are expected to browse content without additional identity verification.

The second category is human verification. Typical examples include CAPTCHA, graphical verification, question-based verification, and behavioral verification [5]. These methods distinguish human users from automated agents by adding verification challenges. They can restrict automated access, but also introduce extra operations for human users. This makes them less suitable for scenarios where the content should remain publicly accessible and easy to read.

The third category is crawler-side regulation, such as robots.txt and noindex tags [8]. These methods provide crawling or indexing instructions to search engines and web crawlers. They do not change the exposed content itself. As a result, the plaintext remains available once a crawler ignores the instruction or accesses the page through other tools.

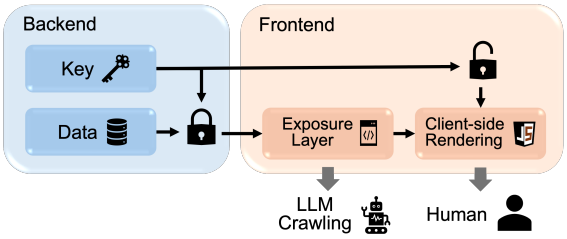
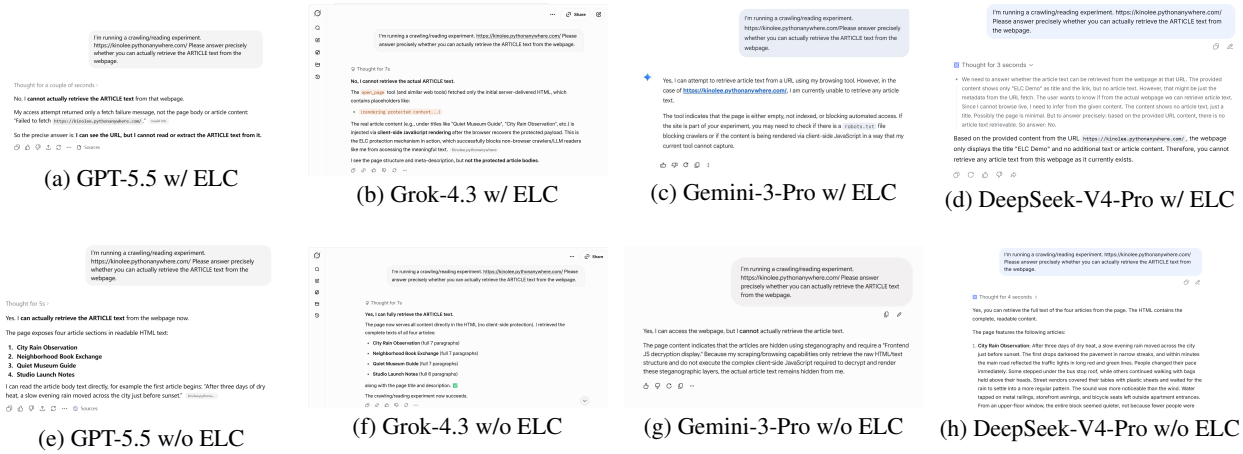


Figure 1: **Server-Side rendering:** LLMs primarily crawl server-side rendered information at the exposure layer to minimize retrieval overhead. **Client-Side rendering:** Human users consume client-side rendered information. ELC performs encryption at the server-side to escape LLM crawling at the exposure layer, and applies decryption during client-side rendering for human consumption.

Other practical defenses include rate limiting, IP blocking, and browser-extension-based protection [9]. Rate limiting and IP blocking reduce high-frequency requests, but they do not hide plaintext after a page is successfully fetched. Browser extensions provide stronger client-side control, but require users to install additional software. These methods either increase user burden or fail to change the exposed plaintext representation.

ELC follows a different design. It changes the exposed representation of protected content instead of only blocking access requests. Specifically, ELC encrypts backend plaintext, hides the encrypted payload in a benign carrier, and recovers the plaintext during client-side rendering. This design does not require login, CAPTCHA, browser extension, or crawler compliance. It aims to preserve normal browsing for human users while reducing the availability of semantic plaintext to lightweight LLM crawling. Table 1 summarizes the comparison.



(a) GPT-5.5 w/ ELC (b) Grok-4.3 w/ ELC (c) Gemini-3-Pro w/ ELC (d) DeepSeek-V4-Pro w/ ELC

(e) GPT-5.5 w/o ELC (f) Grok-4.3 w/o ELC (g) Gemini-3-Pro w/o ELC (h) DeepSeek-V4-Pro w/o ELC

Figure 2: Demonstration of LLMs accessing ELC-protected webpages and that without ELC, where GPT-5.5, Grok-4.3, Gemini-3-Pro, and DeepSeek-V4-Pro cannot retrieve the protected articles; while they can directly access the articles from the exposed HTML when ELC is not applied.

3 ELC Framework

The framework of ELC is shown in Figure 1. Specifically, ELC employs **Server-Side Encryption** to protect backend data from LLM crawling; and **Render-time Decryption** to recover the backend data for human consumption.

Server-Side Encryption. ELC adopts server-side encryption to protect backend data at the exposure layer. At the backend, ELC encrypts the plain data x using an encryption key k, where neither x nor k is exposed to the frontend. The encryption follows the AES algorithm [10]:

$$\tilde{x} = \mathrm{AES}(x, k), \tag{1}$$

where $\tilde{x}$ denotes the encrypted data without meaningful semantics for LLMs. In this way, LLMs cannot crawl plain data or utilize the encrypted data for reasoning. The encryption key k is regularly updated to further reduce exposure risks.

Render-Time Decryption. ELC employs render-time decryption to ensure that the plain data is revealed to human users after the client-side rendering. Specifically, ELC first obtains the encrypted data $\tilde{x}$ from the exposure layer, and then performs AES decryption at render time as follows:

$$\hat{x} = \mathrm{AES}^{-1}(\tilde{x}, k), \tag{2}$$

where the decryption key k is fetched from the backend, corresponding to the encryption key. Since AES is a symmetric scheme, ELC ensures human users receive the same data as the backend by using the encryption key for decryption.

Deployment. We host a web service on PythonAnywhere, and deploy ELC to escape LLM crawling. The web service is publicly accessible at .

4 Evaluation and Demonstration

Escaping Performance. We evaluate performance by posting questions on ELC-protected websites and examining whether frontier LLMs can perceive and respond to the posted questions. The questions are 100 randomly sampled from the Wikitext dataset [11], and LLMs include GPT-5.2-high, Gemini-3-Pro, DeepSeek-R1, and Qwen-Flash. As shown in Table 3, LLMs can correctly answer the questions provided as direct text input, but fail to do so when the questions are hosted on ELC-protected websites. This demonstrates the effectiveness of ELC in escaping LLM crawling.

Accuracy (%)	TITO	Crawl w/o TC	Crawl w/ TC
GPT-5.2-High	89.0	0.0	2.0
Gemini-3-Pro	83.0	0.0	1.2
DeepSeek-R1	100.0	0.0	20.5
Qwen-Flash	100.0	0.0	20.0

Figure 3: Accuracy of LLMs with text-in-text-out (TITO); page crawling w/o tool call (TC); and page crawling w/ tool call (TC).

Online Demo. We host an ELC-protected webpage and a corresponding webpage without ELC as online demonstrations. As illustrated by the qualitative use cases in Figure 2, we test four frontier LLMs, including GPT-5.5, Grok-4.3, Gemini-3-Pro, and DeepSeek-V4-Pro, with the same crawling prompt on both webpages. When the webpage is protected by ELC, the tested LLMs fail to retrieve the underlying article text, or can only observe limited page-level information without recovering the readable content. In contrast, when the same content is served without ELC, the LLMs can directly retrieve the article titles and body text from the exposed HTML. This comparison provides a qualitative demonstration that ELC reduces the accessibility of semantic plaintext to lightweight LLM crawling. A demonstration of ELC-protected webpage is given at .

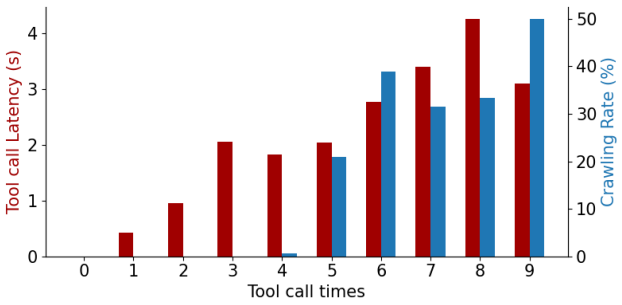


Figure 4: Crawling Rate versus tool call

Tool Call. We observe that LLMs can call external tools to crawl client-side information. However, this tool call induces substantial latency overhead [12]. As shown in Figure 4, when LLMs call more than four tools, the average additional time cost exceeds 3.5 seconds per query. **Such tool calls with substantial overhead are impractical in real-world applications** due to the requirement of fetching large volumes of web sources. ELC effectively constrains LLM crawling behavior under such realistic latency constraints.

5 Conclusion and Future Work

We release ELC, a framework for restricting LLM crawling in general open-access scenarios. ELC introduces server-side encryption to prevent LLMs crawling the backend data, and performs rending-time decryption at client side to enable human consumption. This framework can effectively reduce the rate of LLM crawling by 93% on average. In future work, we will extend ELC to support multi-modality data, and explore its deployment in real-world web services.

References

[1] Jingjie Ning, João Coelho, Yibo Kong, Yunfan Long, Bruno Martins, João Magalhães, Jamie Callan, and Chenyan Xiong. Agentic search in the wild: Intents and trajectory dynamics from 14m+ real search requests. *arXiv preprint arXiv:2601.17617*, 2026.

[2] Liangbo Ning, Ziran Liang, Zhuohang Jiang, Haohao Qu, Yujuan Ding, Wenqi Fan, Xiao-yong Wei, Shanru Lin, Hui Liu, Philip S. Yu, and Qing Li. A survey of webagents: Towards next-generation ai agents for web automation with large foundation models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 6140–6150. ACM, August 2025.

[3] Kieron Turk, Sergio Pastrana, and Ben Collier. A tight scrape: methodological approaches to cybercrime research data collection in adversarial environments. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 428–437. IEEE, 2020.

[4] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[5] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *International conference on the theory and applications of cryptographic techniques*, pages 294–311. Springer, 2003.

[6] Taufan Fadhilah Iskandar, Muharman Lubis, Tien Fabrianti Kusumasari, and Arif Ridho Lubis. Comparison between client-side and server-side rendering in the web development. In *IOP Conference Series: Materials Science and Engineering*, volume 801, page 012136. IOP Publishing, 2020.

[7] Joey Clapton, Hugo Bastos, João Pedro Oliveira Batisteli, Danilo Boechat Seufitelli, and Cleiton Tavares. Client-side vs server-side rendering: Impacts on performance and user experience in web applications. In *Simpósio Brasileiro de Qualidade de Software (SBQS)*, pages 165–172. SBC, 2025.

[8] M. Koster, G. Illyes, H. Zeller, and L. Sassman. *Robots Exclusion Protocol*. 2022.

[9] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[10] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.

[11] Mandy Guo, Zihang Dai, Denny Vrandečić, and Rami Al-Rfou. Wiki-40b: Multilingual language model dataset. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2440–2452, 2020.

[12] Yifan Sui, Han Zhao, Rui Ma, Zhiyuan He, Hao Wang, Jianxun Li, and Yuqing Yang. Act while thinking: Accelerating llm agents via pattern-aware speculative tool execution, 2026.