Poster: LLM-empowered Cyber-Physical Systems by Integrating Local and Remote Inference

Bo Chen
Department of Computer Science, Michigan Technological University, Houghton, MI, USA
bchen@mtu.edu

***Abstract*—Large language models can be utilized to improve the reasoning capabilities of modern cyber-physical systems (CPS). However, the inference time could be a significant obstacle due to the low-power nature of CPS. This work proposes a novel design to integrate local inference and secure remote inference to address this obstacle. The preliminary results demonstrate the feasibility of the proposed approach.**

***Index Terms*—cyber-physical systems, LLM, inference, security**

I. Introduction

Cyber-physical systems (CPS) such as humanoid robots [2], robot dogs are increasingly being deployed. Large language models (LLM) represent a new form of AI that is capable of understanding, creating, and forecasting language by training over massive text datasets. Integrating LLMs with cyber-physical systems allows smarter decision making in complex environments, thanks to their superior reasoning capabilities compared to traditional AI. However, the considerable size of LLMs leads to slow inference speeds, which poses a significant challenge for low-power cyber-physical systems.

To address this challenge, the CPS terminal device can simply delegate the LLM inference process to more powerful edge/cloud servers (remote inference for short). However, there are some issues that need to be addressed. First, remote inference strongly relies on network connectivity, meaning that its performance can be greatly influenced by factors such as network latency and current server load. If the network suffers from congestion or the servers are overwhelmed by a large number of requests, remote inference would suffer from severe delays in responses, preventing the CPS devices from making real-time decisions. Second, remote inference suffers from various outsider attacks, leading to the compromise of user privacy (e.g., the attacker may view prompts/responses without authorizations), data integrity (e.g., the attacker may corrupt prompts/responses) and data freshness (e.g., the attacker may replay obsolete prompts/responses). Additionally, the attacker may perform DoS attacks on the edge/cloud servers.

To address the first issue, we cannot rely solely on the remote servers for LLM inference. Having observed that many LLM models are open and low-power devices are capable of running a "thin" version of a given model, we integrate remote inference and local inference to achieve the best trade-off among inference effectiveness, time delay, and robustness. In particular, the CPS terminal device runs a "thin" version of the LLM model locally, and the cloud/edge servers run "thicker" versions of the LLM models remotely. The prompt issued by the CPS terminal device will be sent simultaneously to the LLM modes running on the local device, the edge, and the cloud server, respectively. The highest quality response received within a predetermined time period will be adopted. To address the second issue, we incorporate security strategies ensuring user privacy, data integrity and freshness, and mitigating DoS attacks on the cloud/edge servers.

II. Background

A large language model (LLM) is an AI model that has been trained on a large amount of text data. For example, Google Gemma 3 (27B) was trained with 14T tokens, and Microsoft Phi-4 was trained using 9.8T tokens. By extracting semantic insights from language data, LLMs outperform traditional AI methods in reasoning tasks. Many modern LLMs are built using transformer architecture. Several LLM services have been provided, for example, ChatGPT, Claude, Grok, and Gemini. These services allow users to send prompts and receive responses, serving as examples of *remote inference*. There are also quite a few LLM models which are open (e.g., Llama series from Meta, Qwen series from Alibaba). The open models can be downloaded and the prompts can be answered locally using inference tools such as Hugging Face transformers and llama.cpp (that is, *local inference*). Open models may be provided with a variety of sizes. For example, Qwen2.5 has versions 0.5B, 1.5B, 3B, 7B, 14B, 32B, and 72B. Typically, an LLM model with a larger size would have better performance in complex tasks with higher computational cost.

III. System and Threat Model

We consider a system consisting of LLM cloud/edge servers and clients. The servers provide LLM inference services by receiving prompts from clients and sending back responses. LLM clients are CPS devices, such as a humanoid robot, a robot dog, or a robot vehicle. We only consider external attackers who: 1) may inspect the network packets and compromise use privacy; and 2) may alter the packets being captured and simply replay the old packets; and 3) may perform a DoS attack on the LLM servers. We do not consider attackers who can compromise the CPS devices or the LLM servers.

IV. Design

To ensure that the CPS device can always receive a timely response from the LLM model to support its decision-making, we introduce two key designs.

1: integrating both local and remote inferences. Our rationale is that remote inference can mostly provide fast response since the edge/cloud servers are typically much more powerful than a low-power CPS device; however, it may not be sufficiently reliable as the network bandwidth is not guaranteed (in the worst case, there is no network connection at all) and the server might experience performance degradation if it becomes overloaded. This would be exacerbated in an adversarial setting. Local inference will be used as a complement. Specifically, we deploy LLM models of different sizes. The large model is deployed in the cloud, the medium one at the edge, and the small one operates locally within the device. Upon querying the LLM model, three identical prompts are sent to the cloud, the edge, and the local model simultaneously. A timer is set after the prompts are sent. The final decision is based on the responses received before the timer expires, with higher priority given to the responses from larger models if multiple responses are received. If no response has been received within the timer, the timer should be increased.

2: secure remote inference. Remote inference may suffer from various network attacks. The attacker may intercept the prompts/responses in plaintext, compromising user privacy. Therefore, prompts/responses should be encrypted using a secret key shared between the CPS device and the LLM server. Upon sending an encrypted message, the sender ID should also be sent to allow the receiver to locate the corresponding secret key for decryption. In addition, to prevent the attacker from modifying the prompts sent by the LLM clients or the responses sent back by the LLM servers, the sender should compute a message authentication code (MAC) over the encrypted message. A timestamp is also incorporated into the MAC code to ensure the freshness of the message. The receiver should check both the MAC code and the timestamp when receiving a message before decrypting it.

The prior attacks and defenses are general for any network communications. However, the DoS attack on the LLM server could be unique in that the attacker simply sends junk messages to overwhelm the LLM servers, causing a denial of service. Junk messages are computationally inexpensive to generate, but verifying them could be computationally expensive using the MAC code. To discourage this unique DoS attack, we propose to use sample-based integrity checking to quickly filter out the junk message without re-computing the MAC code over the entire message. Our design is as follows.

i) After encrypting a message, the sender will select a snippet (a few bytes in size) from the encrypted message at a random location. The random location is computed by applying a pseudo-random function (PRF) over the sender ID and the current timestamp with a shared secret key. The snippet is further encrypted using the secret key and placed at the head of the encrypted message. Generating the authenticated snippet requires one PRF and one block cipher.

ii) After receiving a message, the receiver can filter out the message as follows: The receiver will extract the sender ID and the timestamp from the message, and apply a PRF over them using the shared secret key, re-generating the random location. The receiver will further extract the snippet from the encrypted message, and re-encrypt the snippet using the same encryption algorithm and the secret key. After encryption, the receiver simply compares it with the one stored at the head of the encrypted message. If they do not match, the message should be filtered out. Filtering can be performed efficiently using one PRF and one block cipher.

	CPS device	Edge server		Cloud server	
	Qwen1.5B	Qwen1.5B	Qwen3B	Qwen1.5B	Qwen7B
Delay (s)	1.1	0.905 (0.914)	1.80 (1.81)	0.61 (0.62)	1.73 (1.77)

TABLE I
INFERENCE DELAY, MEASURING THE TIME BETWEEN THE CPS DEVICE ISSUING A PROMPT AND RECEIVING A RESPONSE FROM EACH ENTITY RUNNING A RESPECTIVE LLM MODEL. X(Y) - X IS THE DELAY WITHOUT SECURITY STRATEGIES AND Y IS THE DELAY WITH SECURITY STRATEGIES.

V. PRELIMINARY IMPLEMENTATION AND EVALUATION

We have implemented a preliminary prototype for the proposed design. The CPS device was implemented using an NVIDIA Jetson Orin Nano 8GB module (equipped with Arm Cortex-A78AE processor and 8GB RAM); the edge server was implemented on a Lenovo IdeaCenter 300s computer (equipped with Intel Core i5 CPU and 16GB RAM), and the cloud server was implemented using an Acer Predator Helios 18 computer (equipped with Intel® Core™ i9 processor, 64G RAM and GeForce RTX™ 4090 GPU). Communication between the client and the servers was based on UDP. Encryption was implemented using AES-128-GCM, which belongs to authenticated encryption and therefore also provides data integrity assurance. The PRF was instantiated using HMAC-SHA256. For LLM inference, we mainly modified the open-source inference tool llama.cpp [1] to support local and remote inference. The models being deployed were Qwen2.5 1.5B, 3B and 7B, in GGUF format with 8-bit quantization. We use a simple prompt "hello" to test each model.

Preliminary results are shown in Table I. We can observe that: 1) running models of the same size, remote inference has significantly less delay compared to that of local inference; and 2) remote inference is good for running larger models, and 3) local inference is good for running smaller models; and 4) our security strategies only add tiny additional overhead (1%-2%).

Future works. 1) Currently, we only use a simple "hello" prompt to test the design. This is because the response to "hello" is short and pretty similar under different models. For other prompts, the generated responses will be very diverse and the delay caused by this content generation would vary significantly. We will test a diverse set of prompts in future work to assess this impact. 2) CPS systems may be deployed in diverse network environments, and how varying network conditions may affect the inference delay will also be evaluated. 3) Edge/cloud servers may need to handle a large number of inference requests, and we will investigate load-balancing strategies specific in this context.

REFERENCES

[1] GitHub - ggml-org/llama.cpp: LLM inference in C/C++. https://github.com/ggml-org/llama.cpp.

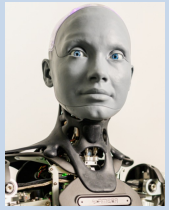
[2] Invasion of the Home Humanoid Robots. https://www.nytimes.com/2025/04/04/technology/humanoid-robots-1x.html.

LLM-empowered Cyber-Physical Systems by Integrating Local and Remote Inference

Bo Chen
Computer Science Department, Michigan Technological University

Introduction

- CPS devices like humanoid robots and robot dogs benefit from LLMs' reasoning.
- Remote inference is powerful but depends on network stability and raises security concerns.
- A hybrid solution is proposed: local "thin" LLM and remote "thick" LLMs.
- Highest-quality, fastest response is selected within a timeout window.

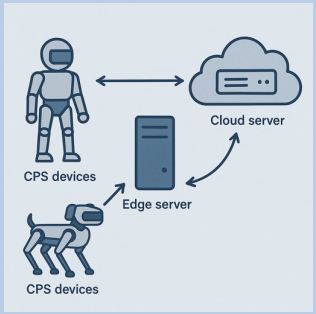


Background

- A large language model is an AI model that has been trained on a vast and diverse collection of text data.
- Open models may be provided with a variety of sizes, e.g., Qwen2.5 has versions 0.5B, 1.5B, 3B, 7B, 14B, 32B, and 72B.
- Local inference uses local resources of the CPS device to run a model via tools like llama.cpp.
- Remote inference uses edge/cloud servers, capable of running larger models.

System/Adversarial Model

- A system that consists of LLM cloud/edge servers and CPS clients.
- Attacker can intercept or alter data over network, but cannot compromise the devices themselves.
- Threats include privacy leaks, data tampering, replay attacks, and DoS.

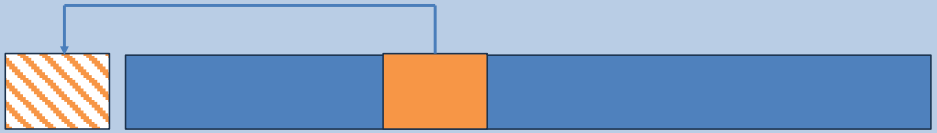


Key design 1: integrating both local and remote inference

- Deploy LLM models of different sizes: the large model is deployed in the cloud, the medium one at the edge, and the small one operates locally within the device.
- Prompts sent to cloud, edge, and local simultaneously.
- Timeout determines which response to use, prioritizing higher model quality.

Key design 2: secure remote inference

- Encrypted communication (AES-128-GCM).
- Message authentication with HMAC-SHA256.
- Timestamping to prevent replay.
- Fast filtering to mitigate DoS attacks on the edge/cloud servers: randomly select (using a pseudo-random function) a snippet from the message and encrypt it with a secret key.



Preliminary Implementation and Evaluation

- The CPS device: an NVIDIA Jetson Orin Nano 8GB Module; the edge server: implemented on a Lenovo IdeaCenter 300s computer (Intel Core i5 CPU and 16GB RAM); the cloud server: implemented using an Acer Predator Helios 18 computer (Intel® Core™ i9 processor, 64G RAM and GeForce RTX™ 4090 GPU).
- Observations: 1) upon running models of the same size, remote inference has significantly less delay compared to that of local inference; and 2) remote inference is good for running larger models, and 3) local inference is good for running smaller models; and 4) our security strategies only add tiny additional overhead (1%-2%).

	CPS device	Edge server		Cloud server	
	Qwen1.5B	Qwen1.5B	Qwen3B	Qwen1.5B	Qwen7B
Delay (s)	1.1	0.905 (0.914)	1.80 (1.81)	0.61 (0.62)	1.73 (1.77)

Acknowledgments

This work was supported by US National Science Foundation under grant number 2225424-CNS and 2043022-DGE.