Verifiable & Private Inference

Methods beyond ZK & FHE

Erhan Tezcan

Lead Developer @ Dria

4.9.2025

Introduction

whoami

- Lead Developer at **Dria**
- We are building a peer-to-peer network of

State of AI & LLMs

Artificial Intelligence (AI), and Large Language Models (LLMs) in particular, are revolutionizing the world. Close to %10 of the entire world population is using ChatGPT alone[1].

New models are coming out every week, smashing the existing records on numerous benchmarks, with an ever increasing performance demand.[2].

We are actually progressing faster than we though we were, as noted by powerhouse's such as OpenAI, Anthropic, and Google DeepMind[3].

[1] https://backlinko.com/chatgpt-stats

[2] https://hai.stanford.edu/ai-index/2025-ai-index-report

[3] https://80000hours.org/agi/guide/when-will-agi-arrive/

State of AI & LLMs

Within this talk, we are specifically interested in the **inference** part of the AI/LLM stack. Inference is the process of running a trained model to make predictions or generate outputs based on new input data, as shown in Figure 1.

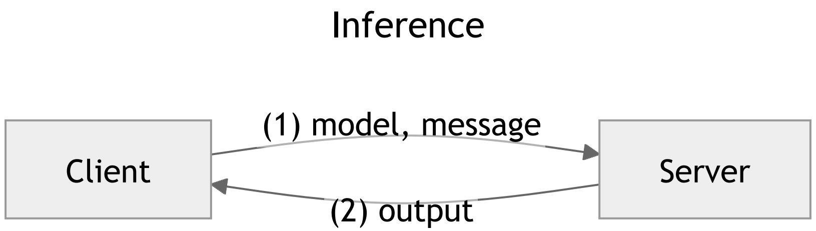

Figure 1: Inference

State of AI & LLMs

There are two problems with such an inference:

- Is the **Server** really using `model` and `message` to generate the `output`?
- Is the **Server** peeking into user `message`?

These problems are denoted as **verifiable inference** and **private inference**, respectively.

Problem Setting

We would like to focus on **consumer-grade** model providers in particular, as they can:

- Locally serve models on their own hardware, utilizing their idle-compute on open-source models
- Join a permissionless network & earn from their services
- Decentralize the inference market, which is currently dominated by a few big players

ZK & FHE & TEE

Verifiable Computation

we will talk about how ZK solves verifiable inference, and how FHE solves private inference.

also mention TEE

Transformers

Attention is All You Need

Talk about LLM transformers, inference in particular. Mention encoder / decoder architecture.

Vanilla Verification

If you have a model provider that you would like to check for simple compliance, a better-than-nothing is to use **vanilla verification**.

Every few requests, you can send a procedurally generated prompt with a known output, and check if the provider returns the correct result.

- For mathematical reasoning, there are static analysis tools like Math Verify by HuggingFace.
- For code generation, you can use a sandboxed unit-test to check if the generated code works as expected.
- For text generation, you can use a set of known question-answer pairs; or simply do a string inclusion check.

Vanilla Verification

The problem with this approach:

- Tends to report false negatives, as the model might not always return the correct output, even if it is compliant.
- Static analysis & sandbox may not work as intended.

You should provide a **margin of error**, and not ban a provider on the first failure.

story time

Split Inference

talk about split and denoise stuff

- verisplit
- split-and-denoise
- privacy partitioning

STIP

STIP (Secure Transformer Inference Protocol)

TOPLOC: A Locality Sensitive Hashing Scheme for Trustless Verifiable Inference [1]

Bibliography

[1] J. M. Ong *et al.*, "TOPLOC: A Locality Sensitive Hashing Scheme for Trustless Verifiable Inference." [Online]. Available: https://arxiv.org/abs/2501.16007

Thank You!