Exploring Eigen Layer AVS and Layer AVS as a Service for Context-Based Text Output Generation

Authors

Luiz Vasconcelos, Daniel Yuki, and Arthur Jacobina

November 2024

Introduction

In the blockchain ecosystem, integrating advanced technologies like randomness, predictive models, and decentralized infrastructure is a technical frontier. Our project, “What is the Ticker”

, aimed to explore the potential of combining large language models (LLMs), Verifiable Random Functions (VRFs), and decentralized Actively Validated Services (AVS) to build a seamless, scalable solution. Throughout this journey, we delved into the capabilities of LLMs for creative output generation, the strengths and limitations of WebAssembly (Wasm) for heavy computation, and the challenges of integrating Layer with EigenLayer in a decentralized environment. These explorations shaped our understanding of the technical landscape and inspired a fun use case: dynamically generating unique outputs like meme coin tickers based on user inputs

.

Use of Language Models on Layer

Language models like OpenAI’s GPT or Meta’s LLaMA3 hold vast potential for decentralized applications. We envisioned their integration into Layer AVS as a means to process user inputs creatively and generate quirky meme coin tickers. These models’ generative capabilities would add dynamism and fun, aligning perfectly with the unpredictability of meme coin culture.

However, implementing these models within the current Layer AVS framework revealed some constraints, especially in handling large-scale computational tasks. This was expected on our side, both because we intended to run the model inside a WASI component and because blockchain is not particularly efficient in loading large files.

Starting: wasi_nn

Initially, we explored wasi_nn

as a solution to integrate LLMs within WebAssembly (Wasm). wasi_nn

promised a flexible environment for neural network execution, which we hoped could support LLaMA3 or similar models. However, this approach quickly hit a roadblock due to file handling limitations

. WASI’s infrastructure, mostly used in simpler tasks like image classification and linear regression, struggled to manage large-scale model files.

This challenge rendered wasi_nn

impractical for our use case, forcing us to seek alternative solutions for randomness and creativity.

What We Built on Layer

After encountering limitations with wasi_nn

, we chose to use other models to handle natural language processing. Our chosen approach involved building on top of probability models

, such as Markov chains

and n-grams

.

In short, our code:

1. Build an n-gram model from a predefined text corpus.

2. Generate text using the n-gram model.
3. Extend the corpus with the newly generated text.
4. Construct a Markov chain from the extended corpus.
5. Generate additional text and select a random word as the output.

An n-gram

is a contiguous sequence of n

items (typically words or characters) from a text. These models use statistical information to predict the next item in a sequence. For instance, a bigram (n=2) predicts the next word based on the previous word, while a trigram (n=3) uses the two preceding words. N-gram models are useful because they capture local context and can predict probable word sequences based on observed data.

A Markov chain is a stochastic model that describes a sequence of possible events where the probability of each event depends solely on the state attained in the previous event. In terms of text generation, a Markov chain treats the text as a sequence of states (words or characters), and the next state is determined only by the current state, not the sequence of events that preceded it. This property is known as the Markov property or memorylessness. Markov chains are effective for modeling dependencies in sequences where the future state depends directly on the current state.

In the code, the n-gram model is first constructed using the provided corpus of text (denoted as CORPUS). The corpus is tokenized into words using a regular expression that captures words and punctuation, ensuring that the model can handle various textual elements.

As a corpus of text, we chose a text on Memecoins on Solana, which is our Proof-of-Concept example. The power of the model is a function of the size of the corpus, but in order to make it possible to run on top of the Layer Blockchain we have to load the text in runtime (or wasmtime), which can be have very low performance as the size of the corpus grows.

Also, since our model generates non-deterministic answers to the tasks (because we use randomness), the operators cannot come to a consensus over a task (in the layer chain, as a standard, all tasks are responded to by 3 operators).

Future Enhancements:

- Use of TEEs (Trusted Execution Environments) with signatures from the operators.

: This way, only one operator has to really compute the task response and, since they're on a TEE, other operators can only validate the answer.

- Use of ZK proofs to attest the computation

: This approach is a lot more secure than the previous one but also a lot harder to really achieve in production. With a solution using ZK proofs, only one operator needs to perform the task and can respond to it, providing a ZK proof that it was done in a proper way.

Connecting Layer to EigenLayer

As of now, the Layer AVS as-a-service is not integrated with EigenLayer contracts deployed on Ethereum. Layer is built on the Cosmos SDK, which is efficient and customizable but cannot natively integrate with EVM contracts like EigenLayer's.

Hackathon Solution:

We introduced a middleman

to bridge Layer and EigenLayer:

The tasks are initially sent to the AVS contracts in Ethereum, written in solidity.

These contracts have the function to create tasks (user's interaction) and respond to tasks (operator's interaction). From the point of view of Eigenlayer, the operator is actually the operator, responsible for listening to the events emitted in the AVS contracts and also responding to the tasks with the outputs expected.

When our middleman listens to a task emitted by the avs contract, it gets the input sent by the user and sends it to Layer, effectively creating a new task on the layer chain. Now, our middleman keeps sending requests to the layer chain until the task is completed by Layer's operators, after which it gets the output and forwards it to the AVS contract, responding to the original task.

While this solution works temporarily, it introduces centralization and inefficiencies.

Future Improvements:

- Native WebSocket connections with EVM RPCs

: Eliminate the need for a middleman by enabling Layer operators to listen to EVM events directly.

- Native EVM task responses on Layer

: Allow Layer to respond to EVM tasks without intermediaries.

Developer Feedback

1. Documentation

: Layer's documentation is helpful, allowing the development of services on top of Layer, but at some points it's not clear what is working and what is still under development. It is also not very clear how to properly integrate with ui (front-end or backend in javascript). Although there's an ui example, there was no docs on the api of the task queue, for example.

1. EVM Integration

: We believe integrating Layer AVSs natively with EVM is very important, in order to make it possible to create an end-to-end application without the need of centralization points.

Conclusion

Our project explored integrating technologies like language models, randomness, and decentralized services within the blockchain ecosystem. Faced with limitations in running large models using wasi_nn

, we adapted by employing simpler probability models like n-grams and Markov chains. These allowed us to generate creative outputs, such as meme coin tickers, within the computational limits of the Layer blockchain.

Looking ahead, the integration between Layer and EigenLayer appears promising. While our middleman solution works for now, future efforts should focus on enabling native connections and responses between these layers to reduce centralization and enhance scalability.

Link to project repo:

https://github.com/orgs/What-is-the-Ticker/repositories