FLock.io

MOVE GPT BEATS GPT-4O

Turbocharging LLMs on Aptos Move in Decentralized Cooperation

TL;DR: We developed novel strategies to improve the capacity of LLM in writing the Aptos Move programming language. The experimental results have shown that our proposed method has superiority over ChatGPT4o when dealing with Aptos Move programming language.

Key Innovations

1. Data + Domain-specific Tuning

Our process began by curating a robust dataset of Aptos Move code, sourced from real-world projects, open repositories, and tutorials. By focusing on the Move language's unique features—like resource-oriented programming and its focus on secure digital asset management—we ensured the model is tuned for domain-specific challenges. This specialised dataset feeds directly into the LLM, sharpening its ability to generate contextually accurate, high-performance Move scripts.

However, not all Move scripts are written in Aptos style, and not all Move scripts have effective comments.

While Aptos Move scripts share common ground with the Move language, not all Move codebases adhere strictly to Aptos-specific conventions. Moreover, many scripts lack effective comments, which makes understanding and enhancing them challenging. To address this, we introduced two key enhancements: a Move script classifier to filter Aptos-specific code and a ChatGPT-4o-assisted comment generator to improve code interpretability.

Mathematical Modeling for Aptos Move Style Classification

Recognizing that not all Move scripts adhere to the stylistic and functional conventions of the Aptos blockchain, we developed an advanced classification model employing deep learning and statistical analysis to distinguish between general Move scripts and those conforming to Aptos-specific standards.

- Aptos Style Classification: We trained a neural network classifier to evaluate whether a given Move script aligns with Aptos guidelines, focusing on efficient transaction handling, resource-oriented logic, and security practices unique to the Aptos ecosystem.
- Positive-Negative Pair Generation: During data preparation, we generated pairs of positive (Aptos-style) and negative (non-Aptos-style) scripts. These pairs help construct a refined, instructed dataset, which is critical for training the LLM to prioritise Aptos conventions over generic Move programming practices.

The use of positive-negative pair generation is a pivotal aspect of our methodology, significantly enhancing the model's ability to understand and prioritize Aptos-specific conventions over generic Move programming practices. Here's an in-depth analysis of why this approach is important and how it contributes to the overall performance of the Large Language Model (LLM):

The primary goal is to ensure the model prioritizes Aptos-specific conventions, and positive-negative pair generation is instrumental in achieving this. By clearly labeling scripts, the model learns explicit preferences, reinforcing the importance of Aptos conventions in its decision-making process. Positive-negative pair generation is rooted in the principles of contrastive learning, a technique where the model learns to differentiate between similar and dissimilar data points.

Objective Function Optimization: By presenting the model with pairs of scripts labelled as either positive (Aptos-style) or negative (non-Aptos-style), we optimise the objective function to minimise the distance in the feature space between similar (positive) examples and maximise it between dissimilar (negative) ones.

Feature Discrimination: This process enhances the model's ability to extract discriminative features that are characteristic of Aptos-style scripts, such as specific syntax patterns, resource management protocols, and security practices.

By training on both positive and negative examples, the model becomes more robust and generalizes better to unseen data. The model is less likely to generate incorrect or non-compliant code because it has learned to recognize and avoid patterns associated with negative examples. And it can adapt to a wider range of inputs, improving its performance in diverse coding scenarios and making it more reliable for developers.

Automated Comment Generation with ChatGPT-4o

One of the challenges we faced was the lack of effective comments in many Move scripts. This made it difficult to understand the purpose and structure of the code, especially for developers who are new to the language. To address this, we used ChatGPT-4 to generate automated comments.

- Move Scripts Analysis: We tasked ChatGPT-4 with analysing the logical flow and structure of Move scripts. By parsing through functions, resources, and modules, it generates meaningful comments that explain what the code is doing.
- Adding Useful Comments: For each section of the code, ChatGPT-4o generates inline comments that clarify its operations and how it follows Aptos or Move conventions. For instance, in a script managing token transfers, it would generate comments explaining key parts like resource movement, permission checks, and asset management.

This approach not only makes the code easier to understand for developers, but it also helps improve the LLM's performance. The comments provide additional context, making it better at generating well-structured and annotated Move scripts.

This approach not only makes the code more understandable for human developers but also serves as an instructional tool for the LLM. With these comments in place, the LLM has an additional layer of context to enhance its learning, making it more adept at generating well-structured, annotated Move scripts.

Train Aptos in Decentralised Manner

An integral component of our methodology involves the adoption of decentralized training techniques to train the Large Language Model (LLM) effectively. This approach addresses the computational and logistical challenges associated with training large-scale models on specialized domains like Aptos Move.

To better utilise the community effort and the private data, we applied a decentralized learning algorithm along with our FLock system [3] that encourages each user to contribute their data and encourages the community to devote to developing a much better model. Before launching the FLock System, we developed an efficient decentralized learning algorithm to improve the overall performance and robustness of training the LLM and mitigate the node bias within the entire network.

We extend the decentralized stochastic gradient descent (D-PSGD) [1] and Alternating Direction Method of Multipliers (ADMM) [2] in LLM for faster convergence concern. We use multiple nodes to simulate the entire network and perform the decentralised training along with the tailored data and train an improved model to archive a ChatGPT-4o standard LLM within our framework.

Experimental Results

We are surprise that, with our hard work, we managed to surpass ChatGPT-4 in open-source LLMs. We were shocked to find that ChatGPT-4 was unable to complete even the simplest tasks. The experimental results are shown in the attached PDF.

Reference

1. Lian, Xiangru, et al. "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent." Advances in neural information processing systems 30 (2017).
2. Boyd, Stephen, et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers." Foundations and Trends® in Machine learning 3.1 (2011): 1-122.
3. Dong, Nanqing, et al. "FLock: Defending malicious behaviors in federated learning with blockchain." arXiv preprint arXiv:2211.04344 (2022).

Our POC already beats GPT-4o in Move code generation

	Hello World	Pendle
GPT-4	❌ Even Hello World is wrong	❌ Hallucination
Move GPT by FLock.io	✅ Correct Answer	✅ Generalize to Open Domain Successfully

Case 1: GPT-4o got Hello World wrong (output in Rust)

Move GPT by FLock.io

Chatbot

Write a hello world with move language.

Sure, here is how you could write a "Hello, World!" program in Move language:

```
module helloworld::hello  {
    use std::string::String;

    public fun main(): String acquires HelloWorld{
        let result = b"Hello, World!".to_string();
        return result;
    }
}
```

In the above code, we define a module called `helloworld` with a function named `main` that returns a `String`. The `b""` syntax is used to create a byte string which can be converted into a `String` using `to_string()`. This program will return "Hello, World!" when executed.

Only Move GPT by FLock.io used the String function

GPT-4o

ChatGPT 4o >

Write a hello world in move.

To write a “Hello, World!” program in the Move programming language, you need to create a simple script that will print the message. Below is an example of how you can do this:

move 复制

```
script {
    fun main() {
        // Print "Hello, World!" to the console
        debug::print("Hello, World!");
    }
}
```

Explanation:

- `script` declares a Move script, which is the main entry point of the program.
- `fun main()` defines the main function where the code execution begins.
- `debug::print("Hello, World!");` is the command that prints the “Hello, World!” message to the console. The `debug::print` function is often used in Move for logging or printing to the console.

This script, when executed in a Move environment, will output “Hello, World!” to the console.

给 ChatGPT 发送消息

Case 2: GPT-4o cannot write yield tokenisation (hallucination)

Move GPT by FLock.io (see Notion full response)

Case 4.

Query: Use Aptos language to write a yield-trading protocol that tokenises yield and allow both yield token and principle token to be traded in an AMM. Write it in two parts. First, the protocol should wrap any yield-bearing tokens into SY(standardized yield tokens), which is a wrapped version of the underlying yield-bearing token. SY is then split into its principal and yield components, PT (principal token) and YT (yield token) respectively, this process is termed as yield-tokenization, where the yield is tokenized into a separate token. Second, write an AMM where PT and YT can be traded.

Ours:

Sure, here are two parts of your protocol:

1. **Yield Tokenization**: This part involves wrapping yield-bearing tokens into SY (standardized yield tokens) and then tokenizing the yield component.

```
module standard_yield::token {
    use std::signer;
    use aptos_framework::coin;
    use aptos_framework::aptos_coin::AptosCoin;

    struct SY<phantom CoinType> has key {
        base_coin: coin::Coin<CoinType>,
        yield_rate: u64, // This is a placeholder for the actual yield rate calculation.
                         // You need to calculate it based on your protocol logic.
    }

    struct PT has key {
        base_coin: coin::Coin<SY<CoinType>>,
    }

    struct YT has key {
        yield_rate_coin: coin::Coin<SY<CoinType>>,
    }

    public fun initialize(account: &signer) {
        move_to(account, SY { base_coin: coin::zero<CoinType>(), yield_rate: 0 });
        // Initialize PT and YT with zero balance for the account.
    }

    public fun wrap(account: &signer, amount: u64) acquires SY {
        let sy = borrow_global_mut<SY>
```

Only Move GPT by FLock.io recreated Pendle (not in any of Move's official docs) successfully

GPT-4o

Use Aptos language to write a yield-trading protocol that tokenises yield and allow both yield token and principle token to be traded in an AMM. Write it in two parts. First, the protocol should wrap any yield-bearing tokens into SY(standardized yield tokens), which is a wrapped version of the underlying yield-bearing token. SY is then split into its principal and yield components, PT (principal token) and YT (yield token) respectively, this process is termed as yield-tokenization, where the yield is tokenized into a separate token. Second, write an AMM where PT and YT can be traded.

Here is an implementation of a yield-trading protocol using the Aptos Move language. The code is broken into two parts:

1. **Yield Tokenization**: Wrapping yield-bearing tokens into `SY`, and then splitting `SY` into `PT` (principal token) and `YT` (yield token).
2. **Automated Market Maker (AMM)**: Allowing `PT` and `YT` to be traded.

Part 1: Yield Tokenization

move 复制代码

```move
module YieldTokenization {

    use std::signer;
    use std::vector;
    use std::option::{self, Option};
    use std::coin;
    use std::token::{Token, TokenId};

    // Define SY, PT, and YT types.
    struct SY<TokenType> has key, store {
        wrapped_token: TokenType,
    }

    struct PT has key, store {
```

Next steps

1. Enrich a significantly larger amount of **training data** by creating detailed comments and instructions using both LLM and domain experts

2. Design and implement a quantifiable **benchmark** for LLM performance on the Move language

3. Leverage more computing resources to do comprehensive experiments on various **training strategies and model selections**

4. Write **technical report** on benchmarking

5. Rollout a **production-ready model deployment** for Aptos community with inference optimization