Cinder: A Minimal Decentralized LLM Interaction Framework

Carter Richard

cinder@calcifer.cloud

Abstract

Since the November 2022 release of ChatGPT by OpenAI, the technology industry has seen a massive explosion in the development and commercial deployment of Large-Language-Models (LLMs) for Artificial Intelligence (AI) applications. The compute capabilities of LLMs are clear, but rapid widespread adoption and 'AI fever' has resulted in disparate standards for how LLM applications should operate. Simultaneously, consumer data collection efforts over past decades have raised significant public concern over the ethical advancement and usage of Artificial Intelligence technology, including LLMs. In several cases, LLMs which drive AI applications were trained by collecting consumer data with predatory or unclear consent collection practices. Therefore, the increasing automation of jobs across all industries and sectors using AI-driven assistants should concern every citizen of the world. To prevent the abuse of consumer data in training AI-driven capabilities, we propose a minimal open-source interaction framework to allow users to cooperatively develop and share LLM interaction chains over a decentralized network. This enables user democratization over the creation and usage of simple, hyper-specialized AI assistants.

1. Introduction

Artificial Intelligence capabilities and their benefits remain an inaccessible resource for many around the world, whether due to lack of infrastructure, technical ability, or willingness to trust AI-provider platforms' data usage policy. Because Large-Language-Models are computationally expensive, direct access to a LLM is typically provided through centralized platforms that charge users for model consumption in exchange for providing an Application-Programming-Interface (API) key. While a centralized paradigm benefits provider platforms with tremendous amounts of user data to refine model offerings, doing so also erodes users' trust in services provided by the platform. This is an unsustainable business practice, as AI-provider platforms are vitally dependent on their users to continue developing features. We predict that platforms will have difficulty convincing the population of the world to contribute data that could be applied towards the automation of their own jobs.

Instead, users need a unified framework for interacting with LLMs, designing specialized capabilities, and delivering their functionality to others without risking the unauthorized collection of their interaction session data. Because of LLMs' computationally-expensive nature, the framework also requires a transaction system to securely and independently validate payment for client access to a host model's resources in a zero-trust environment. Fortunately, a transaction system that can be applied to this use-case has already been invented (Nakamoto, S. 2008).

This document serves as the source-of-truth for core capabilities of the Cinder interaction framework, and sets expectations for an application that seeks to adopt or integrate with the framework.

It will not contain information about how Large-Language-Models or decentralized networks operate beyond their general capabilities. These technologies are presented abstractly by intention to encourage the reader to conduct research and independently validate the ideas presented here.

Cinder does not directly compete with or consider industry practices in the training and development of Artificial Intelligence models. Instead, the framework's jurisdiction covers the usage of AI models through interactions to abstractly generate compute functionality. As such, the Cinder framework proposes the introduction of a new conceptual layer to the Open Systems Interconnection (OSI) model. This hypothetical Interaction layer would exist between layer 4 (Transport) and 5 (Session) and represents abstract compute functionality that can be provided by Artificial Intelligence models and shared across OSI layers/instances.

2. Host Access

A host application utilizing the Cinder framework must provide a RESTful API to clients with at least the following root endpoints:

1. /spark
2. /interact

The /spark endpoint must reset the host application to an entry state when accessed by a GET request. The /interact endpoint must return JSON-structured message data when accessed by a POST request. These messages must adhere to the below minimum-viable standard.

3. Message Standard

All Cinder messages must include at least the following attributes:

1. sender
2. content
3. tx

The sender attribute is a string and contains the virtual blockchain wallet address of the message sender - regardless if that sender is a human user or another Cinder instance. The content attribute is a string that holds any message content provided by the message sender. Finally, the tx or 'transaction' attribute is an optionally-populated hash value that references a transaction on a blockchain for the purposes of validating access to a host instance. Cinder messages may contain additional data elements, but these minimum attributes are required by every application utilizing the Cinder framework.

4. Assistant Behavior

An assistant running on a Cinder instance must support the following interaction commands and execute the below functionality if those commands are presented in an input message:

1. library
2. chain
3. run
4. task
5. connect

A **library** command with no arguments must return the available tasks available to run on a Cinder assistant. A task name can be passed as an argument to the **library** command: in this case, Cinder must return summary metadata about what the task accomplishes and the estimated cost of execution.

A **chain** command is passed with a task name as an argument. A Cinder assistant executing a run command must impersonate user input for a task with the last-received assistant output, and then execute the task and return output from the task. Additionally, this command must automatically persist the link across task chains for automated execution with **run**.

A **run** command is passed with a task name as an argument as the root task. Upon receiving this command, the assistant must execute the root task and then **run** the next linked task if such a task exists. If not, the assistant has reached the end of the automation chain and must return output.

A **task** command must allow the sender to create and persist a task and associated context on the Cinder instance, where the argument for **task** is the name of the created task. This command must be usable as the first argument for a **chain** command to permit interactive context extension.

Finally, a **connect** command must accept an argument of a IPv4 or IPv6 address to another target Cinder instance, and pass all following input as if it were a message sent from the host instance to the target instance.

5. Architecture Paradigm

Cinder hosts support operation in either a centralized or decentralized paradigm. In a centralized paradigm, a host provides unrestricted access to its available capabilities and assumes users are authorized to access the instance.

In a decentralized paradigm, a host validates the receipt of a blockchain transaction provided by the client before executing any kind of interaction. Decentralized technology allows Cinder to trust the validity of this transaction without requiring a central authority. This allows hosts to sell their assistant capabilities to other users or Cinder instances and treat those capabilities as trade-able abstracted compute power in a democratized system.

As an example, a team of users who wants to use Cinder to develop complex LLM automation chains would provide a host instance to each member of the team. The developers on this team trust each-other and **connect** their Cinder instances to each-other in a centralized paradigm for development. Each member only has to build a small part of the entire automation chain.

When the automation chain is ready for other users to consume, the team collectively re-configures their Cinder instances to run in a decentralized paradigm. This allows other users to access the root-level Cinder instance and run automation chains which connect to and pay the next Cinder instance in-sequence. If at any point trust breaks down between the team, any member can 'break' the chain by removing their Cinder instance. This forces the rest of the team to either replace now-missing functionality or refactor the entire automation chain to work without it.

In effect, this enforces a democratic AI development network where accomplishing a collective automation objective requires the active support of every participant who contributed to the creation of that automation. In exchange, network participants receive compensation determined by the host for lending valuable data to the collective automation chain without exposing their intellectual property.

References

Nakamoto, S. (2008) Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf