The Graph as AI Infrastructure

Anirudh Patel[1], Sam Green[1], and Eva Beylin[2]

[1]Semiotic Labs
[2]The Graph Foundation

1 Introduction

Launched in 2018, The Graph protocol began by organizing Ethereum data and providing developers with custom APIs for querying this data using GraphQL. Since its inception, The Graph has experienced rapid growth, evolving into the leading indexing technology in web3, indexing over 50 chains and processing more than a trillion queries [28]. Recognizing the protocol's potential for extension, Semiotic Labs, the primary authors of this document, are spearheading its expansion to serve artificial intelligence (AI) use cases.[1]

Our journey into this expansion is rooted in several prior innovations — notably, our work with Agentc and AutoAgora. Agentc was our initial entry into AI-enhanced blockchain user interfaces, successfully translating natural language into SQL to interact with blockchain data dynamically. This experiment tested our hypothesis about AI's utility in blockchain UX and provided crucial insights into the practical applications and challenges of integrating AI within The Graph's ecosystem.

Similarly, our earlier development of AutoAgora, a tool designed to automate and optimize dynamic pricing for The Graph's queries, honed our expertise in using AI to solve complex problems.

1.1 This White Paper

For this white paper, we reach beyond what we've done with Agentc and AutoAgora to provide a new and more comprehensive thesis that explores why and how AI models should be served on The Graph, how the existing blockchain data within The Graph can drive powerful AI capabilities, and our planned architecture for The Graph's inaugural AI services. We present two primary services: the **Inference Service** and the **Agent Service**.[2]

1.2 Inference Service

Inference is the process of using an AI model. For example, when you send input text to ChatGPT, it performs inference and returns the output text. The Graph's Inference Service is designed to push The Graph beyond its roots in data indexing, expanding into AI model deployment and inference capabilities. For example, dapps can build ChatGPT-like functionality into their front-end or use AI to synthesize research or subgraph data. The Inference Service enables The Graph's Indexers to opt into providing the computational resources required to run these models, e.g., GPUs. The Inference Service also facilitates access to AI technologies and guards against centralization risks such as censorship and service instability.

[1]Semiotic Labs is an R&D core developer building and maintaining The Graph alongside seven additional teams. Semiotic specializes in AI and cryptography. The team has a track record in pioneering AI applications in web3, including productizing reinforcement learning for query pricing in The Graph and deploying Agentc, a playground for querying The Graph's data using natural language.

[2]For definitions of commonly used technical terms, like *inference*, please see the glossary at the end of this document.

1.3 Agent Service

The Agent Service is an innovation partly inspired by our earlier Agentc project. This service allows developers to build AI-driven decentralized applications (dapps) that perform complex operations within blockchain environments. The service is not merely about processing data but creating interactive, autonomous agents that can integrate various functionalities — including natural language processing akin to Agentc — to simplify and enhance user interactions with blockchain data derived from The Graph.

1.4 Looking Forward

Our experiences, from Agentc to AutoAgora, have shaped our understanding and paved the way for these sophisticated AI services. By leveraging our insights and proven track record in integrating AI with blockchain, we aim to enhance The Graph's capabilities significantly. We ensure it remains at the forefront of AI integration in web3 and is positioned to expand into serving web2 needs.

Current status of The Graph's AI Services

Today, The Graph's AI services are a concept and an in-flight project. This white paper represents our current expectations of the AI services' components and features. Ultimately, its actualization will depend on (and change alongside) the contributions and needs of The Graph ecosystem.

2 The Graph as AI Infrastructure

Supporting AI is a natural extension of The Graph. The Graph developed subgraphs, which became the standard API for querying blockchain data. Subgraphs will always be an important component of the web3 stack, but we have seen increasing demand for more ways to organize and access blockchain data. This demand resulted in The Graph announcing New Era; this ambitious roadmap commits to delivering a rich market of *data services*, like SQL, Firehose, Substreams, and AI. The Graph can commit to these new data services because it has economies of scale; like hundreds of Indexers.[3]

Target audience for this white paper

The first part of this white paper is intended for all members of The Graph ecosystem. If you are a developer, please also read the appendix, which is more technical and describes the types of applications that you can build on The Graph's AI services.

3 The Graph's AI Services

We are building an *Inference Service* to empower decentralized AI applications (AI dapps) built on The Graph's decentralized infrastructure and, optionally, using web3 data.

Inference is the process of running a neural network using inputs a user provides. For example, when you send a prompt to ChatGPT, OpenAI uses your prompt as an input to a neural network and performs inference for you. From a mathematical perspective, a neural network can be viewed as a function $f(x)$, like you saw in algebra, and a query to a neural network is x. For example, imagine an Indexer hosts a (tiny) neural network defined by the function $f(x) = 3x + 1$. If an end user requests inference with a query $x = 5$, then the Indexer will run $f(5) = 3 \cdot 5 + 1 = 16$ and return the result to the end user. Indexers who support The Graph's Inference Service will host AI models and run those models for AI dapp developers. Of course, the Inference Service will support many different neural networks in practice.

[3]For example, Semiotic is close to launching an MVP of a new SQL data service for The Graph. Our experience gained from creating the new SQL data service will be leveraged to launch The Graph's AI data services.

The Inference Service will be used to build decentralized AI applications. We expect that some AI dapps will be decentralized versions of centralized applications, e.g., decentralized clones of ChatGPT, and some AI dapps will be web3-native and use blockchain data queried from The Graph. We call these *decentralized AI dapps* and *web3-native AI dapps*, respectively.

Example of a Web3-Native AI Dapp

A web3-native AI dapp is a dapp using a neural network served by a decentralized network and one that performs inference using web3 data. For example, imagine if a developer were to build an application that helped Farcaster users know if their posts would get a lot of likes. How would you build this as a web3-native AI dapp? Here is one way:

- Use The Graph's Subgraph data service to retrieve thousands of Farcaster comments and how many likes they received.
- Train a neural network to predict whether a new Farcaster comment will be liked.
- Deploy the neural network to The Graph's Inference Service.
- Deploy a dapp that helps users write Farcaster posts by predicting if people will like them.

The main point is that everything in this example dapp is about web3 data, and it is powered by The Graph — a web3 protocol. If you are a dapp developer, many other ideas are included in the appendix.

The *Agent Service* builds on the Inference Service. An agent is an autonomous decision-maker that takes actions based on a user's request or *intent*. An agent can be controlled with something simple, like hard-coded rules, or something complex, like a large language model. This is where The Graph's data becomes useful. Agents are typically specialized in a specific task, and their data must be similarly specialized. The Agent Service supports many different agents. For example, if a DeFi agent needs Uniswap data, it can ask an analytics agent to write the correct SQL query and retrieve the associated data from The Graph.

We'll now expand on the Inference Service and then the Agent Service.

Blockchain vs. Crypto vs. Web3

Blockchain, *crypto*, and *web3* are ambiguous terms. Blockchain refers to several competing solutions, e.g., Ethereum and Bitcoin, that offer decentralized computing and computing services. We view crypto as blockchain technology related to tokens. Web3 represents all projects that have an onchain component. Web3 encompasses DeFi, NFTs, DeSoc, DePIN, decentralized AI, etc. Web3 operates on top of crypto rails. We may alternate between crypto and web3 throughout this document depending on what seems to fit or what is more commonly said. We often lean toward using crypto (and crypto examples) because the use cases are simpler; however, we firmly believe that the ideas we discuss here should and will be broadly extended to web3. We recommend the book *Read Write Own*, by Chris Dixon, if you would like a more thorough take on nomenclature.

3.1 The Inference Service

The **Inference Service** supports deploying and querying AI models. *Serving* an AI model means that a compute provider offers an endpoint developers can call to run inference. Indexers are the compute providers and will serve the AI models.

A note to Indexers

Currently, Indexers are experts in indexing and database optimization, which allows The Graph to be a competitive indexing and data-serving protocol. With the integration of AI services, we expect some Indexers to become world-class experts in serving AI models. Also, over time, we anticipate that competition in supporting the Inference Service will attract a new type of Indexer specializing in serving AI models. This specialized knowledge will help drive lower prices and latency.

How are AI models incorporated into The Graph protocol? We answer this by first explaining how the SQL service works today. We'll use three protocol participants for the explanation:

- **Indexers:** Compute providers that index blockchains and serve blockchain data through custom APIs
- **Dapps:** Send queries for blockchain data to gateways
- **Gateways:** Route queries to Indexers and return the result to the dapp

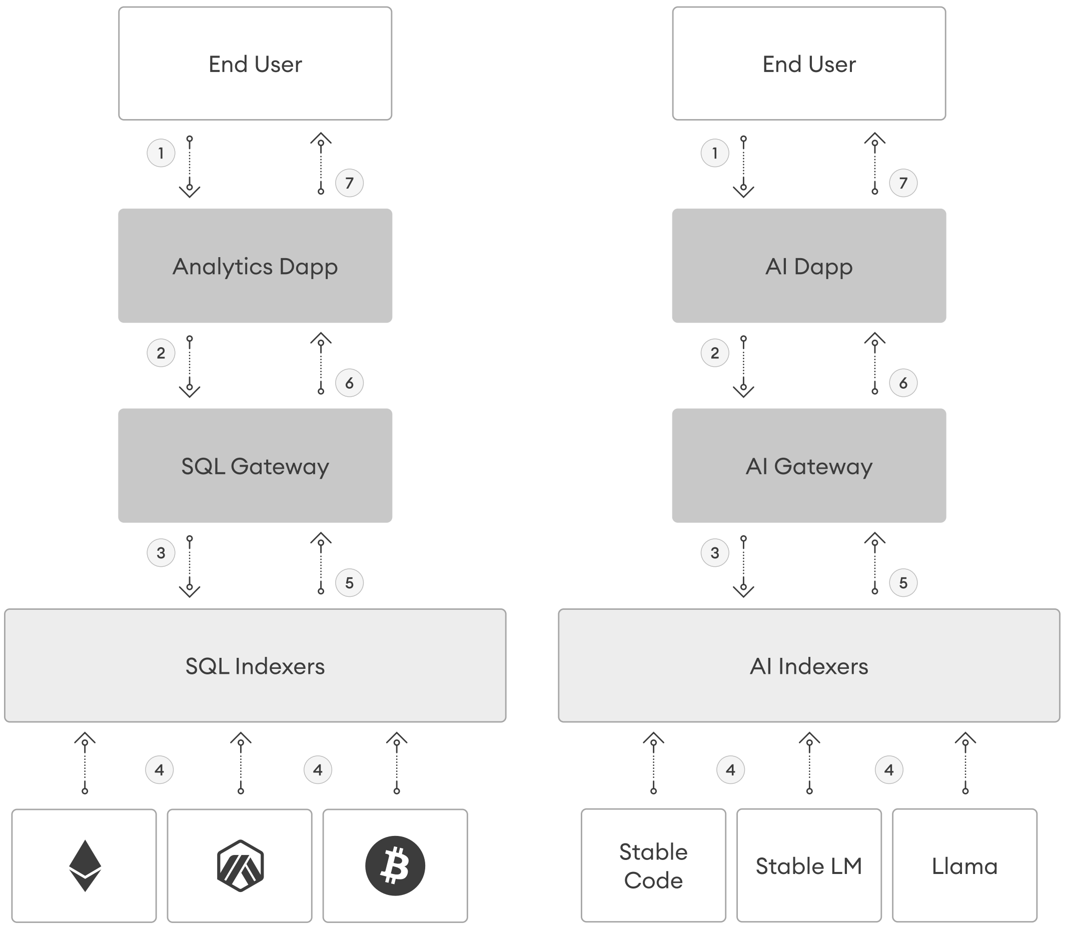


Figure 1: A simplified view of how The Graph's SQL service works and how it is extended for the Inference Service.

Fig. 1 illustrates how The Graph serves data queries:

1. An end user interacts with an analytics dapp front-end.
2. The dapp requests data from one of The Graph's SQL gateways.[4]

[4]The Graph's first SQL data service is currently in beta.

3. The SQL gateway matches the dapp request with an Indexer who can provide the requested data.
4. The selected Indexer has already extracted and stored the requested blockchain data.
5. The selected Indexer returns the data and gets paid for the query.
6. The subgraph gateway routes the data to the dapp.
7. The dapp displays or uses the SQL results.

We modify the existing query flow to provide the Inference Service:

Fig. 1 also illustrates how the Inference Service supports queries:

1. An end user interacts with an AI dapp front-end.
2. The AI dapp requests inference using a particular model from one of The Graph's AI gateways.
3. The AI gateway matches the dapp request with an Indexer who can run inference using the requested model.
4. The selected Indexer loads the requested model and runs inference.
5. The selected Indexer returns the inference result and gets paid for their work.
6. The AI gateway routes the data to the dapp.
7. The AI dapp displays or uses the inference results.

During the MVP, we will support a curated set of popular *open* AI models, that will be selected by Semiotic Labs.[5] The specific list of MVP models that will be supported is still to be decided, but some currently popular open models include Stable Diffusion, Stable Video Diffusion, LLaMA, Mixtral, Grok, and Whisper.

Later, *any* open model will be deployable on the Inference Service with sufficient testing and Indexer operations.

Why use an AI model deployed to The Graph? When a developer hosts a model on a centralized service provider, they typically must agree to abide by the provider's terms of use. This can include potential censorship and often the right to change terms of use at any time, and users can be subject to opaque pricing and rate limiting of AI token access. A developer who doesn't want to be constrained by a centralized provider has two options to mitigate deplatforming risk:

1. The developer buys their own hardware, which is expensive and requires the developer to be an expert in AI model serving, a specialization in its own right. The better alternative is to use Option 2.
2. The developer finds someone willing to serve their model without imposing restrictions that they find unacceptable. This is not necessarily a fully permissionless system, but one in which it is possible to find someone willing to support their application. In other words, AI application developers need an open marketplace of model hosting providers with the compute and expertise required to serve AI models. As we've argued before, The Graph is a natural place to build this system with the resources and data to scale access to models. Notably, as an open-source network, The Graph does not have a preference for what kinds of models are offered. For example, models served by Indexers can utilize web3 data, generate code for a developer, or even generate music for an artist.

So far, we have presented the Inference Service as one that will serve AI models. We specified starting with popular open models like Stable Diffusion and Llama, but we can support thousands of existing open models. Additionally, The Graph's data can be used today to train new models. The following sections address the topics of serving and training custom models.

[5]The creator of an *open* model doesn't share the data or code used to train their neural network – only the final result is shared in the form of parameters needed to execute the model.

3.1.1 Support for custom models

During the last few years, it has become common for AI projects to release *foundation models*. Foundation models are trained on a wide variety of data for various tasks. GPT-4 is currently the most popular closed-source foundation model. Examples of open foundation models include SDXL, a latent diffusion model [17], the LLaMA family of large language models (LLMs) [22], and many others. Foundation models can often be used as-is, which the Inference Service supports or they can be fine-tuned to a specific data set. Fine-tuning involves taking an existing foundation model and specializing it on a custom data set. For example, suppose you have a data set containing all the books your favorite author wrote. Putting copyright issues aside, you could fine-tune a foundation model to write like your favorite author. Many people fine-tune models for different applications, e.g., text, video, or image generation. The Inference Service supports models like these, with the expectation that Indexers will also be willing to support them.

Fine-tuning is performed outside of The Graph

We need to clarify that fine-tuning requires a software stack different from what is required for the Inference Service. At this time, a developer who would like to fine-tune a model would do that outside of The Graph using one of the many existing fine-tuning solutions. The resulting model could then be deployed using the Inference Service.

A future topic to discuss is incentivization for deploying fine-tuned models on The Graph. Suppose a developer spends much time and money developing a fine-tuned model. In the future, The Graph protocol could evolve to add a cryptoeconomic incentive for that developer to make their fine-tuned model publicly accessible in the Inference Service.[6] For example, we could divide query fees between the model creator and the Indexer who served the model.

3.2 Data-Export for Model Training

Originally, The Graph primarily supported dapp frontends with data queried via GraphQL. Now, thanks to The Graph's advancements in blockchain indexing, in addition to subgraphs, developers can also export data from The Graph in various formats through Substreams.[7] For example, Substreams can be written to export data into the Pandas Dataframes format — a standard format used for training neural networks. The resulting data can then be used to fine-tune existing models or train new ones.

In particular, we think the time series data served by The Graph is immediately useful for fine-tuning.[8]

Consider a scenario where developers use The Graph to extract time series data of liquidity and trading volumes from Uniswap. This data, processed through Substreams and exported into Pandas DataFrames, can be crucial for creating predictive models. For example, a predictive model can forecast future demand spikes or liquidity shortfalls by analyzing historical data on token pair trading volumes and liquidity changes. This information could be leveraged by liquidity providers to adjust their contributions to different pools to maximize returns or by traders to anticipate market movements and optimize trading strategies. This enhances individual decision-making and contributes to the overall efficiency and stability of the Uniswap platform.

Data accuracy is a key advantage of using The Graph for training AI models. Unlike centralized approaches, The Graph benefits from public and open-source schemas that ensure data is auditable. This transparency, combined with the traceability of queries and query results via Indexers and The Graphs's gossip network, provides a unique level of data integrity. Data extracted and transformed from blockchain sources undergo rigorous checks by Indexers, and ongoing efforts within the protocol

[6] In the short term, all models hosted on the Inference Service will be publicly accessible. In the long term, we will explore privacy-enhancing technologies to improve model and user privacy.

[7] Substreams is a blockchain indexing technology used within The Graph Network, enabling developers to utilize Rust for creating high-performance, parallelized data streams from various blockchains. This technology allows the extraction and transformation of blockchain data, which is then compiled into WASM modules and executed to deliver data to specified storage systems like Postgres databases or Subgraphs.

[8] A *time series* is a sequence of data points collected or recorded at successive points in time, such as the price of ETH at a specific time. Time series data is often analyzed to identify trends, cycles, or seasonal variations to forecast future values based on historical patterns. This type of data is widely used in various fields, such as economics, finance, and meteorology.

aim to cryptographically verify the accuracy of queried data [19, 21]. This meticulous process ensures that AI models trained with The Graph data achieve high accuracy.

As time passes, more human-readable data will be stored in The Graph. For example, we expect Geo (built on The Graph) to create massive amounts of accurate, human-readable data. For example, web3 protocols could maintain developer documentation on Geo, which could be used to fine-tune a model that knows how to interact with their protocol.

In conclusion, The Graph's Inference Service is a foundational toolkit for decentralized application developers, enabling them to construct specialized AI systems, such as decentralized versions of ChatGPT. The Inference Service supports a broad range of AI model deployments and query capabilities without the constraints of centralized platforms. The Agent Service, described next, will further expand these capabilities by allowing these AI dapps to interact meaningfully with the real world.

3.3 The Agent Service

By hosting AI models on The Graph with the Inference Service, developers benefit from reliable infrastructure that supports efficient and scalable AI model deployment and inference.

The **Agent Service** goes further and enables developers to use The Graph's decentralized infrastructure to build AI dapps capable of performing complex operations within blockchain environments. Through the Agent Service, developers can create dapps where interactive, autonomous agents perform a wide range of functionalities on behalf of end-users.

Think of the Inference and Agent services like a brain and its robotic counterpart. The Inference Service functions as the brain — knowledgeable and capable of understanding and responding to complex questions. It can answer user queries, process information, and guide with intelligence and precision, much like OpenAI's GPT model.

On the other hand, the Agent Service acts as the brain's robotic assistant. AI agents are autonomous systems that have knowledge of their environment and can take actions, such as executing programs, calling external APIs, keeping a memory of past interactions, and facilitating blockchain interactions.

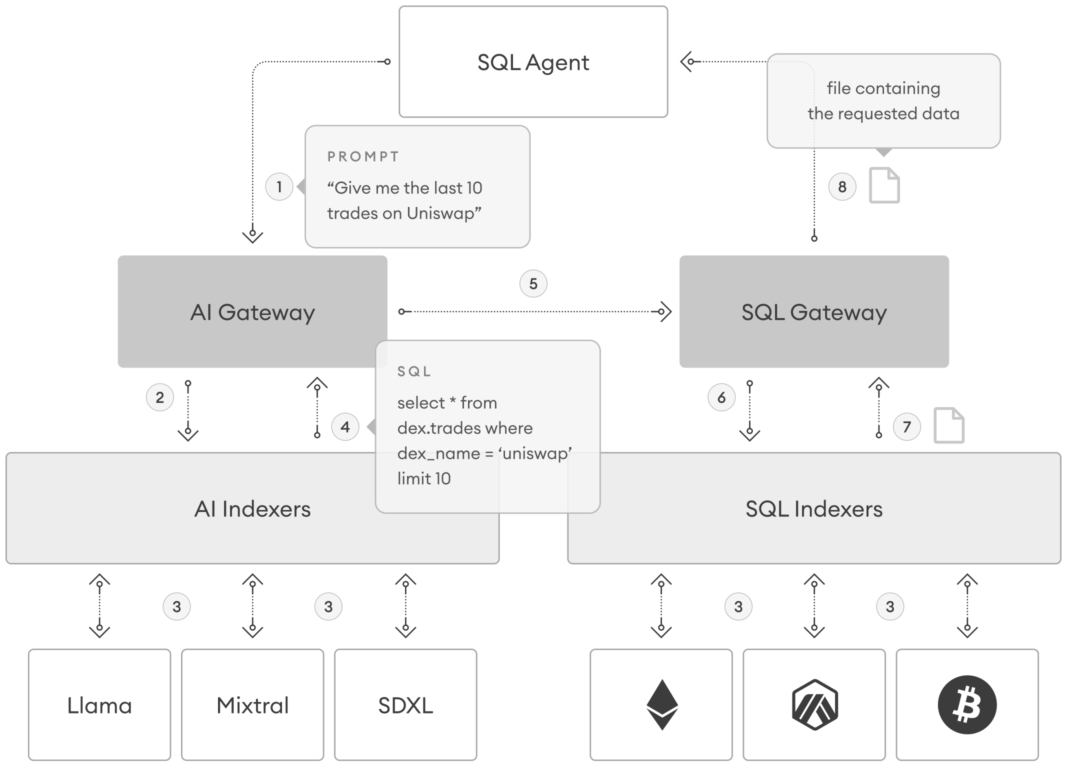
3.3.1 Building Agents with The Graph's Agent Service

An agent is a collection or system of components — not a simple thing like a neural network. Consider the chat agent discussed in the previous paragraph. What exactly is the agent? Is it the graphical interface that pops up on the website? Is it the logic running on a server in the cloud? Is it the database that records the conversation for later use? Is it the external APIs the agent uses to establish a connection with the human operator? It is all of those things and likely many other components. Let's consider an example of how to build an agent using the Agent.

We use an SQL agent for the example. Its job is to retrieve blockchain data after receiving natural language queries. The example SQL agent is illustrated in Fig. 2 and has several components:

- User interface for the AI dapp, e.g., a website for people to ask questions in their native language about data stored in The Graph
- An AI gateway to provide access to the Inference Service
- Indexers with models specialized in generating SQL code
- An SQL gateway to provide access to analytics blockchain data Indexers with databases of a wide variety of blockchain data

The SQL agent waits for a natural language query, e.g., "Give me the last 10 trades on Uniswap." After receiving the query, the SQL agent uses the Inference Service to request an LLM model to generate SQL. The SQL agent then sends a prompt to the LLM, such as, "Your job is to generate SQL. Here is my database schema ... Write the SQL to answer the following question: Give me the last 10 trades on Uniswap." The LLM returns something like `select * from dex_trades where dex_name = uniswap limit 10`. After receiving this SQL query, the SQL agent forwards the query to an SQL gateway, which then, as usual, routes the query to an Indexer possessing the appropriate data.



SQL Agent
file containing the requested data
PROMPT
“Give me the last 10 trades on Uniswap”
1
8
AI Gateway
5
SQL Gateway
SQL
select * from dex.trades where dex_name = ‘uniswap’ limit 10
2
4
6
7
AI Indexers
SQL Indexers
3
3
3
3
Llama
Mixtral
SDXL

Figure 2: A simplified view of an SQL agent.

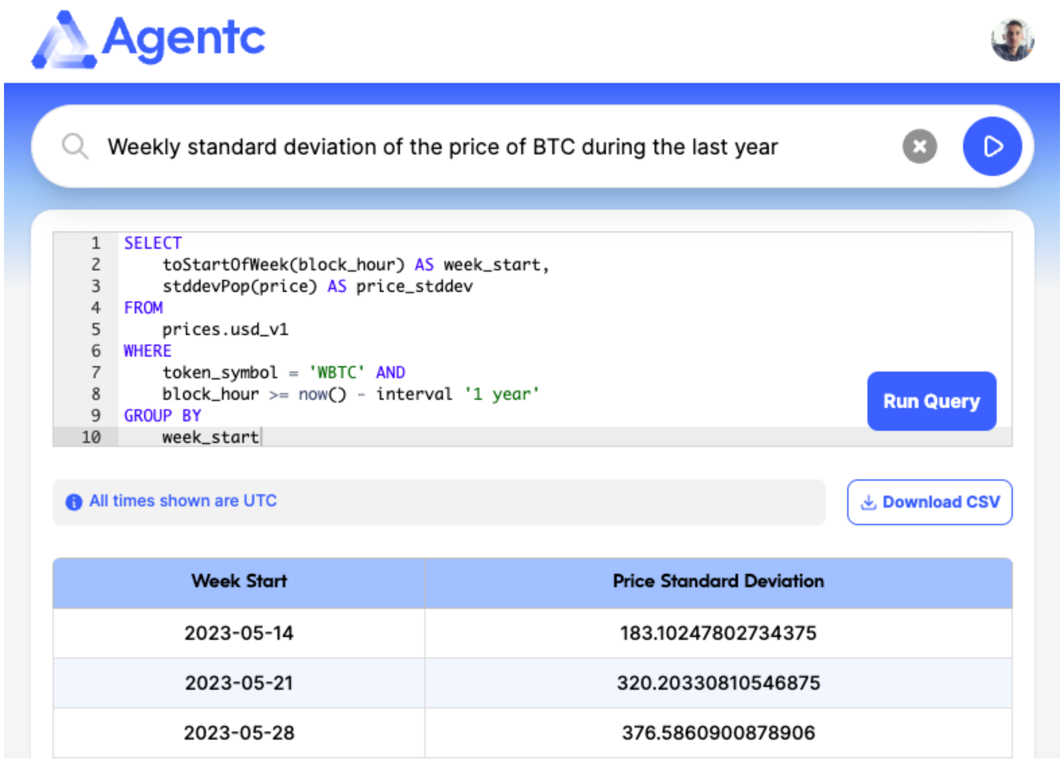
We highlight that developers can create centralized agents after we launch the Inference Service — but they will have to manage all agent components independently. Despite this option, decentralized agents offer far superior resilience and transparency, making them inherently more trustworthy and aligned with the web3 ethos. The Agent Service's value proposition enshrines agent construction, hosting, and execution into The Graph, served by a network of Indexers.[9] Developers can use previously built agents, like Lego bricks, to quickly create new, more sophisticated agents.

As an earlier agent experiment, we built Agentc, a centralized SQL agent using The Graph's indexing software stack for data and OpenAI for inference. Agentc converts natural language to SQL, queries real-time blockchain data from The Graph, and returns the results to the end user. Building Agentc gave us the insight and expertise to scope The Graph's AI services.

The Agent Service enables developers to build AI dapps with functionality like Agentc. It leverages the Inference Service and coordinates the routing between an end user's input and the components required to build and host an AI agent on The Graph. Using natural language agents (like Agentc) as an interface to blockchains makes it much easier for new users to join web3 and for existing users to perform their onchain tasks. Agentc is focused on simple crypto analytics, but agents can do much more. Here are some other use cases that the Agent Service enables:

- **NFT Market Trend Analysis Agent:** Aimed at NFT collectors and traders, this agent provides analytics and market trend predictions to help users make informed decisions in the dynamic NFT market. It integrates time-series analysis models to predict price movements and popularity trends based on historical transaction data indexed on The Graph. Users can interact with the agent by querying specific NFT collections or general market conditions, e.g., "Predict the price trend for the Bored Ape Yacht Club series over the next three months." The agent then uses AI models to analyze the data and outputs predictive insights and graphical trend analyses.

[9]The initial Agent Service is focused on connecting AI models and The Graph's data. Future versions of the Agent Service will enable actions like sending alerts or executing code. The Agent Service won't provide other things, like UI components — the developer will need to provide that if necessary for their agent.


Agentc
Weekly standard deviation of the price of BTC during the last year
Run Query
All times shown are UTC
Download CSV

```sql
SELECT
    toStartOfWeek(block_hour) AS week_start,
    stddevPop(price) AS price_stddev
FROM
    prices.usd_v1
WHERE
    token_symbol = 'WBTC' AND
    block_hour >= now() - interval '1 year'
GROUP BY
    week_start
```

Week Start	Price Standard Deviation
2023-05-14	183.10247802734375
2023-05-21	320.20330810546875
2023-05-28	376.5860900878906

Figure 3: A screenshot of Agentc, a Semiotic Labs pilot project created to translate natural language to SQL queries for blockchain data.

- **DeFi Trading Assistant Agent:** This agent supports DeFi users by automating the preparation of trading transactions based on user-defined strategies and market conditions. It uses natural language processing to interpret user instructions regarding specific trading actions. The agent then drafts transactions that users can review and sign before submission. For instance, a user might request, “Prepare to buy $1000 worth of ETH if the price drops below $2000.” The agent monitors the ETH price and, when the condition is met, prepares the transaction leveraging smart contract interfaces on decentralized exchanges (DEXs).

Executing Onchain Transactions

Semiotic Labs is actively involved in research around agent-based interaction with blockchains. One of Semiotic’s experimental LLM-based agents was the first to facilitate a token swap using a DEX. You can watch the demo video here. In the future, the Agent Service will support fully autonomous transaction execution. Stay tuned for updates on this topic.

3.3.2 Empowering Agents with Real-world Knowledge

So far, this white paper has primarily explored the use of agents that interact with blockchain-based data, but we will broaden that scope. The Agent Service will incorporate real-world data verified through Geo. Each claim within Geo is backed by arguments with links to primary sources, supported by blockchain-based timestamps and voting. This system facilitates the establishment of a verified set of facts and a knowledge graph using data from The Graph.

Knowledge graphs, structured as networks of interconnected entities and relationships between entities, allow for logical and intuitive data querying. The Geo knowledge graph organizes extensive real-world data, such as healthcare regulations and political developments, into a format readily usable by the Agent Service.

Integrating structured data enhances the performance of LLMs. Typically, LLMs can produce "hallucinations," or inaccuracies, in their outputs. By using verified data from knowledge graphs, we transform LLMs into Knowledge Graph-enabled Large Language Models (KGLLMs), which helps reduce these inaccuracies [15]. This ensures the models' responses are more accurate and reflect real-world data and events.

Incorporating KGLLMs within agents transforms these systems into more capable decision-makers that can use accurate, real-time global information. For example, an agent designed to facilitate decentralized governance might utilize political data from the Geo knowledge graph to provide policy change recommendations to a decentralized autonomous organization (DAO), ensuring decisions are informed by current and accurate information.

By integrating Geo's knowledge graph, we enhance agents' functionality and reliability, making them useful tools for developers and users who need reliable, data-driven decision-making capabilities in real-world situations.

The appendix details how the Agent Service supports advanced use cases through KGLLMs and related techniques.

4 Verifiability Upgrades to the AI Services

The Inference and Agent Services will be launched using a curated set of Indexers to serve AI models. We do this to ship an MVP as quickly as possible to learn from the behaviors of Indexers, what works, and what needs improvement. However, *curation* of Indexers implies a trusted arrangement with them and trust their model outputs, which is antithetical to The Graph's mission and the network's open-source nature. We will move to a trust-free relationship with AI service Indexers quickly. Before doing this, several verifiability features must be added for better security:

- Verifiable inference – Needed to prove that the Indexer performed inference correctly.
- Verifiable data – Needed to prove that the input to the model was correct.

4.1 Verifiable Inference

How can an AI dapp developer be sure an Indexer has executed their input correctly using the correct neural network? Dishonest Indexers are naturally incentivized to use smaller, cheaper models when they can get away with it. Imagine an end user sends the following input to The Graph to be executed on LLM A for \$1: "Describe The Graph protocol in three sentences." Suppose LLM A outputs the following at an infrastructure cost of \$1 to the Indexer:

> The Graph is a decentralized protocol designed for indexing and querying data from blockchains, primarily Ethereum. It allows developers to build and publish open APIs, called subgraphs, which applications can query using GraphQL to retrieve blockchain data efficiently. This protocol enhances the performance of decentralized applications by enabling faster data retrieval without relying on traditional, centralized servers.

Suppose the Indexer also has access to LLM B, which costs them \$0.50 to run – a savings of 50% compared to running LLM A. LLM B outputs:

> The Graph protocol is a decentralized indexing protocol for querying blockchain data, allowing developers to build decentralized applications (dApps) that can query and organize data from various blockchain ecosystems. It provides a scalable and efficient way to access and process blockchain data, enabling the creation of complex queries and data visualizations. By leveraging a decentralized network of nodes, The Graph protocol ensures data accuracy, security, and decentralization, making it a crucial infrastructure layer for the web3 ecosystem.

You can see how LLM A output and LLM B output are similar.[10] Both are valid outputs, and it's difficult to identify whether or not the Indexer used LLM A (the model the developer paid for) or LLM B. An Indexer could charge for LLM A but run LLM B. Maybe an end user would be fine with this, but what if the LLM output triggered something important, like onchain actions for a DAO? Verifiability suddenly becomes extremely important. So then, what options are available to AI dapp developers? How does The Graph verify the output of an LLM?

Four methods can be used to verify AI inference: trusted authority, M-of-N consensus, interactive fraud proofs, and zk-SNARKs (zk).

There's debate about which method is best. We are pretty neutral. We think it should be up to developers and end users to decide their required level of security. Likely, AI dapps built for fun won't pay for high security. Critical applications, such as using an LLM output to control smart contracts, must have high security. This means a permissionless decentralized AI protocol will either need to support multiple verifiability options or it will need to specialize in one verifiability method. In contrast, other market participants specialize in other methods. We plan to support multiple verifiability methods.

We detail the methods for obtaining AI inference verifiability in the following sections.

Reproducibility

Neural networks are large mathematical functions (glorified versions of $y = mx + b$). Theoretically, knowing the model, input to the model, and random seed should be enough to guarantee the ability to reproduce inference results. In practice, it's more difficult than that for LLMs, and the difficulty of getting reproducible results has irritated AI researchers. However, there seems to be progress in getting reproducible results, and in this section, we assume results are reproducible [6].

4.1.1 M-of-N Consensus

Using M-of-N consensus or just *consensus* for verifying AI inference means that a developer requests N Indexers to run inference and then checks if at least M of the N results match. The developer considers the inference *verified* if they do. If the results differ, there is a problem; one or maybe all of the Indexers are accidentally or maliciously incorrect.

All methods for verifiability come with extra costs compared to not providing verifiability. Consensus is secured by cryptoeconomics; e.g., a cheater gets slashed if there isn't consensus. In the best case, consensus costs $M \times c$, where c is the computation cost to perform an inference. On the other hand, results are easily verified onchain by M Indexers signing their results with an aggregate signature scheme.

A nice property of M-of-N consensus is that the developer controls the verifiability required for their application. For example, the developer can request 1-of-1 (equivalent to the trusted authority method), 2-of-N, and up to N-of-N security.

M-of-N consensus is straightforward to implement. Let's now examine an even stronger and more efficient verifiability approach.

4.1.2 Fraud Proofs

When using fraud proofs in a blockchain context, a compute provider (*prover*) performs some computation and makes a commitment to the result. If someone (the *challenger*) wants to dispute the result, they enter into a *refereed game* against the prover. A refereed game requires a referee to judge whether the prover's result was correct or incorrect. The referee is usually implemented onchain in smart contract logic.

We adopt fraud proofs for verifiability in the AI service. To do this, an Indexer commits to the result of an inference they claim performed correctly – the Indexer may or may not be right. The inference

[10]In this example, LLM A was ChatGPT 4, and LLM B was Meta AI. We think ChatGPT 4 had a slightly better answer.

result is accepted if no one disputes the Indexer's claim. If a challenger would like to dispute the Indexer, they initiate a refereed game that will verify the Indexer's result.

Here's an important takeaway. Technically, only the Indexer has to do the computation unless they are suspected of foul play. That means the cost for verified inference is roughly the same as that of unverified inference. Challengers can look for instances of Indexers misbehaving and enter disputes (refereed games) if they suspect an Indexer has not performed the correct computations. TrueBit has mechanisms we could adopt to encourage Fishermen to check for bad behavior. You can check out their white paper if interested [20].

Fraud proofs are used by Arbitrum, Celestia, Mantle, and others as their dispute mechanism. Researchers have even applied fraud proofs to large neural networks [6]; this is called optimistic Machine Learning (opML). It's optimistic because a developer can *optimistically* use the result or go through a deterministic dispute process (using an interactive fraud proof) if they suspect something is wrong with the results.

Fraud proofs have stronger security than trusted authority and M-of-N verifiability because fraud proofs only require a single honest participant to make a dispute, which is then deterministically checked onchain. However, fraud proofs have a latency overhead. Fraud proofs require a dispute window for a challenger to check the result and open a dispute if they see a problem. Their outputs can't be considered verified until a dispute window has passed. Depending on the application, such latency could hinder the consumer's experience if they want quick responses like ChatGPT provides, and it could be a source of MEV due to the latency in smart contracts processing neural network outputs. Sometimes, developers need something that can be verified without a dispute window. That brings us to zk.

4.1.3 zk-SNARKs

A zero-knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARKs), often called *zk*, allows a prover to do a computation and generate a proof of its correctness along with the result to a verifier, which is typically a smart contract or off-chain logic. The point of practical zk algorithms is that a verifier should take less work to check the proof than it would have to do in the first place. Zk can be applied to AI. A prover (Indexer) can perform inference and return the output and a proof to the developer. A consumer (or a smart contract) can then efficiently check the proof.

Generating zk proofs is expensive, and producing proofs for every inference seems impractical. Consumers with high-value applications that depend on the results might be willing to pay for a proof. Or, if there was ever a doubt about some results, an Indexer could be challenged (by an end user, developer, or Fisherman) and asked to re-run the model and generate a proof. If a proof fails, it can be submitted and checked onchain to slash the Indexer automatically.

Proof generation costs one million times more than the original computation. That said, the benefit of zk proofs is that they can be verified onchain, which is much cheaper (in dollars) than executing the same computation onchain. This enables applications to trustlessly use offchain computation results in smart contracts.

Today, zk proofs can be efficiently generated for small models, e.g., for linear regression. Zk applied to large neural networks, like Llama, is still far from practical.[11] Furthermore, we claim that cutting-edge AI models will become larger and larger. These models will always be out of reach for zk.

4.2 Verifiable Data

AI models require inputs. If a developer is the source of an input to an AI model, e.g., if a developer sends the prompt "What is the token symbol for The Graph?", they only need to be assured that the Indexer ran an LLM using their input. If the developer wants to use blockchain data as input into an LLM, they will want to know that the data was verified.

[11] Modulus Labs recently demonstrated a successful zk proof of inference using the roughly 1 billion parameter GPT2-XL model. They used the Halo2 proving system and found >1,000,000x overhead for generating the proof. See their blog post for more details.

This section of the paper presented three ways to achieve verifiable inference: N-of-M consensus, fraud proofs, and zk-SNARKs (zk). These methods will also be used to verify that the data being used by AI models is correct. Note that it is valid to mix and match verifiability methods. One verifiability method can be used for data (e.g., zk) and another for inference (e.g., fraud proofs).

When The Graph combines verifiable data with verifiable inference, it will unlock a new dimension in the blockchain application space.

5 Conclusion

Integrating AI functionalities into The Graph is a strategic extension driven by the work at Semiotic Labs. The Inference Service and Agent Service are designed to enhance The Graph's existing capabilities and lay a foundation for new decentralized applications.

Semiotic's experiences with projects like Agentc and AutoAgora have provided us with deep insights into the challenges and opportunities at the intersection of AI and blockchain technology. These multi-year initiatives have shown the practical advantages of leveraging AI to enhance efficiency and accessibility in blockchain data management.

The Inference Service will enable developers to deploy AI models efficiently, using The Graph's decentralized network to mitigate the risks associated with *centralized* AI services. This approach ensures that AI applications can operate with greater independence and reliability and that dapps can more efficiently integrate AI into their front-end experiences.

The Agent Service extends these capabilities by enabling the creation of complex AI agents that can interact seamlessly with blockchain environments and their users. This functionality allows developers to craft more sophisticated applications using The Graph's comprehensive data infrastructure.

Looking ahead, Semiotic Labs is committed to continuing our leadership in integrating these technologies into The Graph and launching AI data services. We look forward to collaborating with the community – developers, researchers, and Indexers – interested in exploring and expanding the possibilities of combining AI and web3. We highlight that other teams in The Graph ecosystem, such as Playgrounds Analytics and DappLooker, are also designing solutions related to the Agent service. We look forward to collaborating with them to accelerate development.

In essence, by integrating AI services, The Graph is not only enhancing its technological offering but is also setting a new standard for how data is accessed and used in decentralized environments. We encourage the community to engage with us as we explore these new frontiers.

6 Acknowledgments

We would like to thank our collaborators at Edge & Node, especially Marco Bagni, for their assistance in creating the figures presented in this document. Additionally, we appreciate Nick Hansen from The Graph Foundation for his many suggestions and edits. Finally, we thank our teammates at Semiotic Labs, Dr. Ahmet Ozcan, Denver Baumgartner, Doug Millett, and Dr. Tomasz Kornuta, for their feedback and edits.

7 Glossary

Agent An agent is an autonomous decision-maker that takes actions based on a user's request or *intent*. An agent can be controlled by something simple, like hard-coded rules, or something complex, like a large language model.

Chatbot An agent or system of agents that interact with people through a natural language interface and optionally perform tasks. A chatbot could be built from a single AI model or be composed of many AI models. Also, see *Hierarchical Agent Tree*.

Decentralized Application (dapp) A type of software application that runs on a decentralized network, avoiding a single point of failure or control.

Deep Neural Network, Neural Network, Model These terms are synonyms in this document. These terms refer to a machine-learning technique roughly based on how animal brains work.

Fine-Tuning The process of updating a pre-trained model's weights to specialize on a particular problem. Compared to training the original model, fine-tuning tends to require less data.

Foundation Model A pre-trained model that is typically very large and whose training was expensive. Open foundation models, like Llama models from Meta, are popular for fine-tuning.

Gateway In the context of The Graph, a gateway serves as the intermediary that routes queries from applications (dapps) to the appropriate indexer that has the data being requested.

Generative AI AI models that can generate text, programming code, images, videos, etc.

Hierarchical Agent Tree A method to break down decision-making problems using multiple agents, each specialized to a specific task. Powerful chatbots can be built using hierarchical agent trees.

Indexer - In the context of The Graph, an Indexer is a participant in the network that processes data from blockchains and serves this data to users via APIs, enabling queries against the indexed data. The AI services will expand the role of Indexers to run neural networks and host agents.

Inference, Running a Neural Network, Executing a Neural Network These terms/phrases are synonyms. You can think of a neural network as being an arithmetic function, like $f(x) = mx + b$, where m and b are the learned weights of the neural network and x is the input of the neural network. When a model performs inference, the user picks a value for x and evaluates the underlying math that specifies how the neural network operates.

Input, Prompt, Query, Request Input is the generic word for the value input to any type of neural network. Generally, a prompt, query, or request is the input to an LLM or chatbot.

Intent A user's request to do something without specifying how it is to be accomplished. For example, a trader could express their intent to a chatbot in plain English by saying they would like to swap some tokens.

Knowledge Base A data structure that organizes information.

Knowledge Graph A type of database that uses a graph-structured data model to store interconnected descriptions of entities — objects, events, situations, or concepts.

Large Language Model (LLM) A type of neural network trained to generate text as a response to a prompt.

Open Model A neural network model whose weights are made available to be studied and used by other developers. Training data and training code are not provided.

Open-Source Model A neural network model whose training data, training code, and resulting weights are made available to be studied and used by other developers.

Pre-Trained Model An already trained model.

Subgraph A defined subset of extracted and potentially transformed blockchain data provided by The Graph's Indexers. It essentially acts as a searchable index of blockchain data.

Time series data A type of data that changes over time. Examples include the daily temperature in a city, average hourly spot prices, or the number of onchain swaps per day.

Training The process of updating a model's weights given a set of inputs and desired outputs.

Weights, Parameters These terms are synonyms. The neural network weights are a collection of numbers that specify the information learned by the model. For example, a model trained to recognize cats would have different weights than one trained to recognize cars.

8 Appendix

We now dive deeper into how The Graph's AI services can help improve the crypto and web3 user experience. This appendix is intended for developers of AI dapps. Most of the focus in this section is on AI-enabled crypto chatbots (more specifically, hierarchical agent trees) powered by The Graph's Inference and Agent Services. *Chatbot* does not imply something simple or trivial to build – a chatbot is an AI agent or system of agents working together to solve a user's expressed desire or intent. For example, in the future, we can imagine medical chatbots that design custom drugs tuned to an individual's genome or logistics chatbots that handle all the shipping and optimization problems for a multinational corporation. If you are familiar with LangChain, then this section should interest you.

8.1 Crypto Chatbots

Crypto has a user experience (UX) problem [8, 23]. The cause of this problem is that the technology is new and complex, involving a rapidly expanding set of cryptography tools and a web of economic incentives and optimization problems. This problem is further compounded by the finality of actions on a blockchain – once a transaction has been submitted, it cannot be undone. Consider the process of onboarding a friend who is new to the space:

1. Set up a wallet.
2. Set up an on-ramp to exchange fiat for crypto (in the form of tokens).
   (a) They'll probably pay a fee for this.
3. Send their newly purchased tokens to their wallet.
   (a) At this point, explain gas.
   (b) Anxiously monitor Etherscan if the transaction doesn't come through quickly, which requires explaining Etherscan.
4. Use a DEX to get tokens that were unavailable through the on-ramp so that they can now use their crypto for web3 applications.
   (a) Explain concepts like slippage.
   (b) Explain that if they want to query subgraphs, they must use one token. If they want file storage, they need another. If they want to compute, that's another.

The crypto UX experience needs to be simplified, especially for new users.

Performing simple crypto tasks requires following long processes without unified documentation. With the popularity of ChatGPT and tools like LangChain, a developer might consider creating a crypto chatbot to guide new users. This would drastically reduce the learning curve of new entrants and reduce common pitfalls. Let's discuss how a crypto chatbot could work.

The example crypto chatbot described in this document is assumed to be powered by LLMs fine-tuned with crypto-related information. The chatbot knows what an *on-ramp* is, can explain the difference between a *hot wallet* and a *cold wallet*, and can explain other basic crypto terminologies.

In this case, the LLM's knowledge is stored entirely in the LLM's weights. Formally, AI researchers classify LLMs as *parameterized knowledge bases* – the parameters being the weights of the LLM. Similarly, they say that the information in an LLM is *internal* to the LLM. While the LLM does not require an external database or source of information to work, working knowledge is limited to the information contained within its training corpus.

LLMs seem to store and recall information reasonably well. Even as far back as 2019, BERT-large[12] was able to recall information at a level comparable to the state of the art at the time, which was

[12]BERT is a Pre-trained Language Model (PLM). ChatGPT, for instance, is based on GPT, which is another PLM.

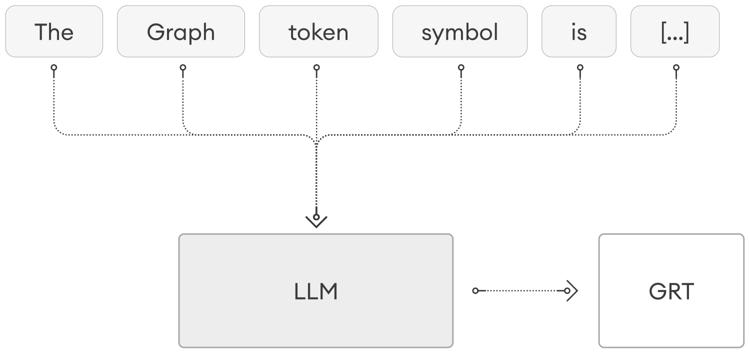


Figure 4: An LLM is a parameterized knowledge base. When it performs inference, an LLM tries to predict the next word in the sentence using only the information stored in its weights and previously input or generated words.

non-neural and supervised [16]. Pre-trained Language Models (PLM) are now beginning to act as knowledge bases [3], with developers going so far as to utilize them to seed and structure information within knowledge graphs [25] (More on this later).[13]

So, why have we yet to see the emergence of a widely adopted crypto chatbot? LLMs are prone to hallucination, generating incorrect information, and are not capable of recognizing the limits of their internal knowledge. This is particularly an issue in scenarios where perfection matters, e.g., when making a blockchain transaction. For example, when we asked ChatGPT 4 (2024-04-09), "What is the address matching samgreen.eth?" ChatGPT responded *incorrectly* with "The address matching 'samgreen.eth' is '0x0ce446255506e92df41614c46f1d6df9cc969183'." This was a hallucination.

This could cause serious issues. Wang et al. found that BART, a PLM, struggles not only to recall training facts at inference time but that it also struggles with closed-book question-answering[14], even when it recalls the relevant information[15] [24]. More recent work even questions whether PLMs can be considered knowledge bases, claiming that previous work used biased prompts [5].

Furthermore, when researchers evaluated ChatGPT (2022-12-15) across various reasoning tasks, they found it only achieved an accuracy of 63.41%[2]. LLMs struggle with domain-specific questions and tend to hallucinate when trained on information that is outside of the context of their training corpus. Simply put, they struggle to reason. [11, 30]. Also, as with any neural network, an LLM's weights are frozen after training, meaning they can't learn anything new. An LLM can't provide up-to-date information [30] by themselves – an *external knowledge base* is required. This turns the crypto chatbot into a *knowledge-enhanced LLM*. Traditionally, this means using either Retrieval-Augmented Generation (RAG) or knowledge graphs (KGs).

8.2 Retrieval-Augmented Generation

The RAG[16] enables an LLM to use an *unstructured external knowledge base*, such as a vector database [13]. Here's how it works.

First, a user inputs text into an embedding model and receives a vector in return. A vector is a list of numbers that represent a piece of text, it is often called an embedding. For example, $[0, 1, 2]$ is a vector. An embedding model maps text inputs into vectors whose positions and values are numeric descriptions of the concepts in the text. For the sake of this discussion, details of the embedding process don't matter. The key point is that distances can be calculated between vectors. If the distance

[13]A knowledge graph is a structured representation of a knowledge base that further details the relationship between information within.

[14]Closed-book question-answering is a task in which a model has to answer questions without access to an external knowledge store.

[15]They tested if the fine-tuned model remembered the relevant information by making it fill-in-the-blanks, what they call *reciting*. They would pick a block of text the model was trained on, mask out certain words, and see if the model could replace the missing words.

[16]The lead author of the paper that coined the term *RAG* bemoaned the name, saying that, "[we] definitely would have put more thought into the name had we known our work would become so widespread" [14].

between vector A and vector B is small, text A is similar to text B. When calculating embeddings of a long document, the document is split into chunks, e.g., sentences, and the embedding of each chunk is calculated. A vector database stores embeddings as keys and the original text chunks as values. For example, a developer could calculate the embeddings of the natural language documentation of a code repository and store the results in a vector database. If a user wanted to query the documentation, the embedding of the user's query would be calculated; then, the vector database would be queried for documentation whose embeddings were *close* to the user's embedding.

Suppose a user asks, "What's The Graph's token?" A developer would first generate the embedding representation of the user's query. Then, the developer could use a distance metric to search their vector database for similar vectors. Maybe the search returns a vector corresponding to the text "Subgraph users can use The Graph Token (or GRT) to pay for queries on The Graph Network" derived from The Graph's documentation. Finally, the developer could include the embedding's text and the user's question in their LLM prompt. This prompt goes into the LLM. You can see this process in fig. 5.

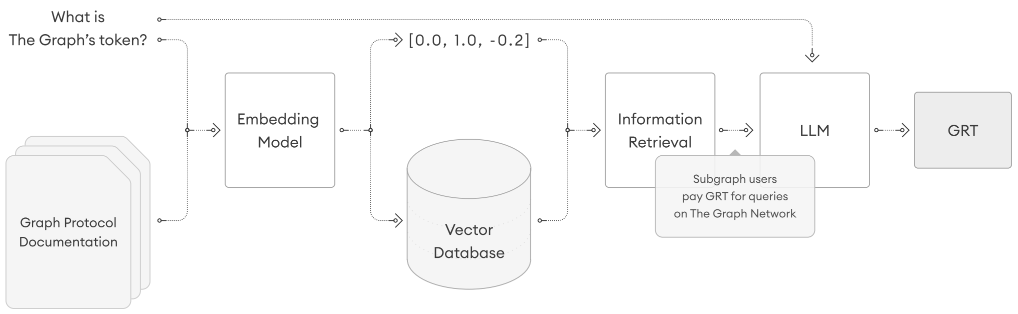


Figure 5: Retrieval-Augmented Generation (RAG) enables information injection into an LLM prompt to reduce or prevent hallucinations. To perform RAG, 1) the text information from a knowledge base is converted to embeddings and stored in a vector database. 2) A user's query is converted to an embedding at query time, and relevant text information is retrieved from the vector database. This information is included with the user's query to generate a more accurate and relevant response. Any text data stored in The Graph could also be used in a RAG pipeline. Text data queried from The Graph could be converted to embedding and stored in a vector database for use with RAG. Semiotic or another developer team may create a vector database service in the future.

Suppose The Graph's documentation is updated. The developer can embed the new documentation and add it to their vector database, thus providing their LLM with this new information. With this technique, the LLM can start to answer questions about data it wasn't trained on, allowing it to stay up-to-date. Better still, it can cite the information from the vector DB, inspiring more user confidence. Since this section aims to make the crypto chatbot more trustworthy, this is an important improvement.

The description of RAG we gave here is the most common description of RAG, but we would be remiss if we left this section without mentioning that this is RAG in its simplest form – sometimes called *Naive RAG* [9].

Focusing specifically on the retrieval process, a common issue with Naive RAG is that the quality of the retrieved information can depend on tuning parameters in a way that's more of an art than a science. Advanced RAG techniques address this weakness in various ways [10]. For example, algorithms like ITER-RETGEN tie retrieval and generation together more tightly and use an iterative technique to improve both [18]. Demonstrate-Search-Predict tries to improve performance by splitting the original question into smaller pieces, decomposing questions that require multiple steps into more straightforward questions to answer [12]. The full spectrum from Naive to Advanced to Modular RAG is surveyed by Gao et al. [9].

With RAG, the crypto chatbot can keep up with the latest news and information. It can also start citing its sources, giving users greater confidence. Where RAG still struggles is with relational data. That's where knowledge graphs come in.

8.3 Knowledge Graphs

Knowledge graph-enabled Large Language Models (KGLLMs) are especially well-suited to handling relational data. Where RAG uses an unstructured, external knowledge base, KGs are a *structured, external knowledge base*.[17] The information in a KG is structured in a graph that models entities, relations, and properties. The relationships between entities are metadata on directional edges between entity nodes. An example crypto knowledge graph is shown in fig. 6.

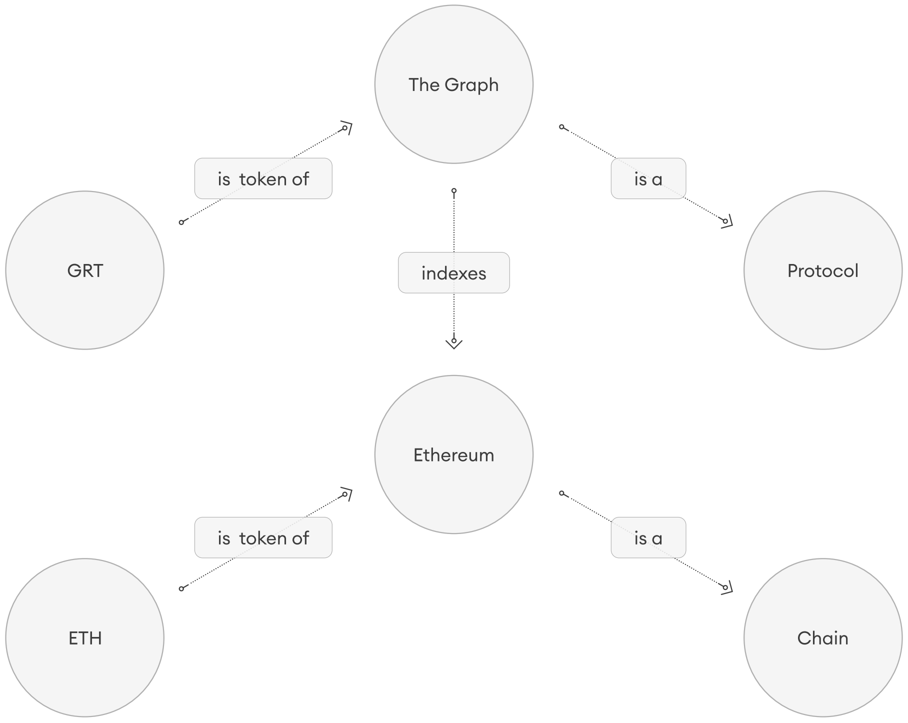


Figure 6: An example crypto knowledge graph. A KG consists of entities (nodes), relations (edges), and properties. Properties can be metadata, but here we choose to show them as nodes via an *isA* relationship.

Consider the example scenario again. A user asks: "What's The Graph's token?" A crypto knowledge graph might have a node *GRT* connected to a node *The Graph* via an edge with the relationship metadata *isTokenOf* Using this relational information, an LLM can easily determine that The Graph's token is GRT. This process is shown in fig. 7

Information retrieval and knowledge-to-text are key to using knowledge graphs at inference time. Unlike in RAG, which uses distance metrics to decide what information to retrieve, in KGLLMs, an algorithm traverses the KG, looking for entities of interest and extracting their relationships to other categories. Often, the graph-traversal algorithm might need to hop through multiple nodes from a parent node to find the relationships that best answer the question. The ability to do this is what makes KGLLMs so powerful, as they allow for the retrieval of information that is related but not semantically similar to the topic of interest

After the graph-traversal algorithm has extracted these relationships, they are typically still in a form that isn't directly usable by an LLM. This is where the knowledge-to-text step comes in. Here, a developer takes the knowledge representation from the graph and converts it to natural language to inject it into the prompt.

[17]Some people, including the authors of the original RAG paper, define RAG differently, allowing it to use both structured and unstructured data stores. However, we believe it's useful to treat these differently. Many algorithms developed for one don't straightforwardly work for the other. For example, a KG can be used before, during, fine-tuning, or inference. RAG only applies at inference time. The category of both RAG and KGLLMs is sometimes called knowledge-enhanced language models.

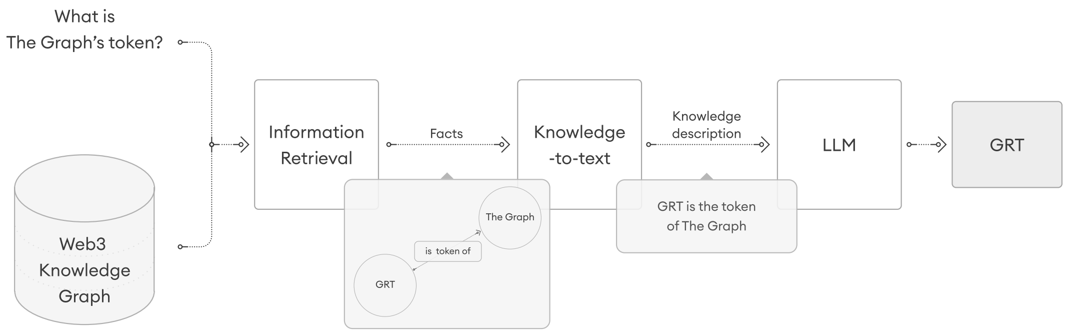


Figure 7: An example of the KG inference process using knowledge-based prompts.

Table 1: A comparison of RAG and KGLLMs.

	RAG	**KGLLMs**
Data	Unstructured	Structured
Retrieval Mechanism	Retrieve info similar to query	Traverse graph and extract relationships
Integration into Generation	Directly augment prompt	The developer needs to convert relationships into text – often need to do an intermediate step of interpolation or template-based generation
Knowledge-Base Storage	Developers often store whole documents and redundant information. RAG's knowledge base is storage-inefficient	KGs don't typically contain redundant information. Each node is unique. They are storage-efficient
Retrieval Compute	Benefits from Approximate Nearest Neighbor algorithms, so their compute scales sublinearly with the size of the knowledge base.	Uses graph traversal algorithms. Graph databases help, but these still don't scale well
Human Effort	Embed data and add it. Minimal effort	Need to define relationships between nodes. Some algorithms exist, but this generally requires a human in the loop
Stage(s) of Use	Inference	Pre-training, training, fine-tuning, inference

KGs can seem somewhat inflexible. Adding information to them is not as straightforward as it is with an unstructured knowledge base like a vector database. One interesting area of research is the dynamic, on-the-fly creation of KGs. Dynamic KGs can be automatically constructed as a language model reads a document [1]. Alternatively, knowledge sub-graphs[18] can be created for various contexts and, using static rules, be transformed into prompts for downstream use [26]. Similar efforts have shown that not only do knowledge graphs improve LLMs, but LLMs also improve knowledge graphs [29].

The pros and cons of RAG and KGLLMs stem from the differences in their underlying data stores, which we summarize in table 1. Ultimately, we believe that a combination of both techniques will enable a crypto chatbot that is accurate enough to be reliable.

Using these techniques, a developer can create a reliable crypto chatbot. It will have been fine-tuned on crypto knowledge, so it's an excellent internal knowledge base. It will be coupled with semantic search or a vector DB to ensure it always has access to updated information that it can cite. It will use a knowledge graph to understand relationships.

8.4 Intents and Agents

This brings us to what we see as the future of web3's UX – intents. In an intents-based system, a user can specify their intent in natural language and then rely on an agent or system of agents to take that intent and act on it. All the agents discussed in this document are based on AI. Using an agent to execute intents is similar to what Vitalik calls "AI as an interface to the game." [4]. For example, a trader could express their intent to a chatbot in plain English by saying they would like to swap some tokens. We now explain how to build a crypto chatbot to realize intents.

For intents, rather than using a single AI model that can do everything, we could break down the problem into a hierarchy of smaller, more specialized AI models. We call these models *agents*. If desired, each agent in the hierarchy can be a different model type: foundation model, fine-tuned model, traditional machine learning model, etc. The optimal model would depend on the task being performed.

Consider fig. 8. It illustrates a hierarchy of agents that can perform tasks for or answer questions about DEX trading, DAO actions, and The Graph protocol. Leaf nodes are what we call *action agents* — they execute actions. For example, a developer can build one action agent that responds in natural language, another that retrieves SQL data, and others that execute transactions on the blockchain. All other nodes simply route queries up or down the tree – we call these "router agents."

This approach has several advantages over the large, do-it-all AI model. First, each action agent is independently useful. For example, users who want to delegate GRT can interact directly with the Delegator Agent, even if none of the agents above exist. Generally, a user who knows which action agent to interact with can use that agent directly instead of using the general-purpose Intents Agent at the top of the graph. This is desirable because running all the router agents to get to the correct agent costs time and money.[19] This property of independent usefulness aids in the decentralized development of these agents. Developers can independently create smaller pieces rather than one monolithic crypto chatbot.

Another advantage of this approach is that no single agent needs full system knowledge; agents only need enough information and skill to do their particular task. Fine-tuning and specialization make models less likely to hallucinate [27].

Yet another advantage is that if something goes wrong, there's a clear activity record for any given query because the problem is solved in smaller steps. Not one massive model whose thought process is opaque, but small steps in which we can trace errors. In a way, this gives us something like a stack trace for agents.

There are a few other small advantages, but the final one we discuss here is that updating a particular agent or adding a new agent requires minimal changes to other agents. If a developer wants to add a

[18]Not the same as subgraphs in The Graph.

[19]Users must pay each time an AI model performs inference, and it takes time for each model to perform inference.

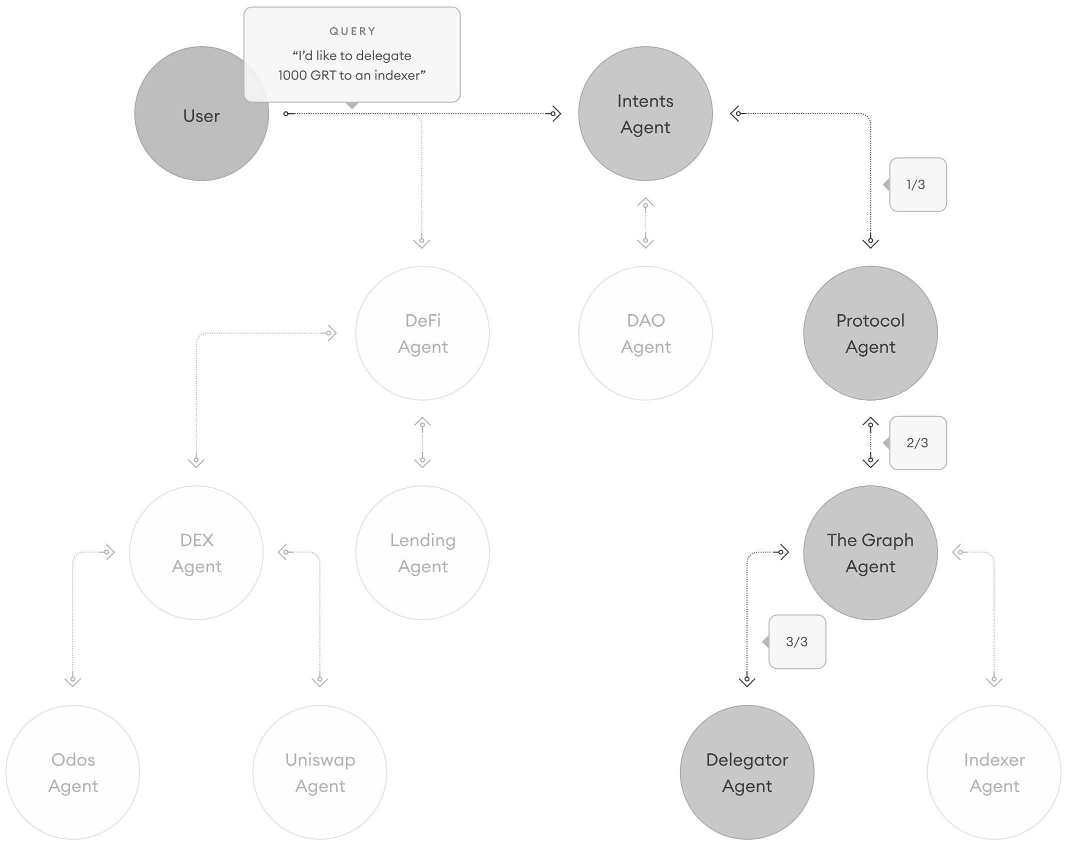


Figure 8: An artistic depiction of how crypto agents could break down the problem of user queries into smaller problems. We call this a *hierarchical agent tree*. For example, if a user wants to delegate GRT in The Graph, the request would first go to the Intents Agent, redirecting the request down the tree until it reaches the Delegator Agent. Each agent only has enough information and capability to do its job and to know whether it needs to pass a request to its parent or a child.

new agent, they only need to modify its parent; if they want to modify an existing agent, they only need to replace it.

There’s one more advantage of this approach we want to describe. Recall that we broke crypto’s UX problem into two problems. One was the problem of navigating the complex web of crypto data and interactions – the navigation problem. We discussed how external knowledge bases can help developers build reliable chatbots. Then, we broke down the simple chatbot and discussed how doing so in a structured way enables intents. The other problem was participating in complex economic systems.

Cryptoeconomic protocol mechanisms can be difficult for protocol participants to understand. The same restrictions don’t bind agents. *Mechanisms that are too complex for humans to navigate can be easy for agents to use*. Davide Crapis from the Robust Incentives Group at the Ethereum Foundation uses this fact to claim that crypto should strive to create an *Internet of Agents* [7]. He believes this will unlock “unprecedented levels of security, efficiency, and collaborative potential.” We agree.

The hierarchical agent tree grants a natural way to construct agents – what Vitalik calls “AI as a player in a game” [4]. Suppose a DeFi agent is tasked with obtaining returns[20]. It would have its own decision-making logic, but it would not need to know the details about how to enact its decisions. It could communicate what it wants to do to the intents hierarchy and let those agents execute the intent.

[20]Agents will be used for more than DeFi. It’s just that DeFi is simple to talk about. For example, agents could follow and participate in all major Ethereum governance conversations – not just tracking a single website, but wherever the conversations took place.

If this is still unclear, imagine if you were tasked with building a house. You may not know all the steps required to build a house, but you could outsource steps to experts.

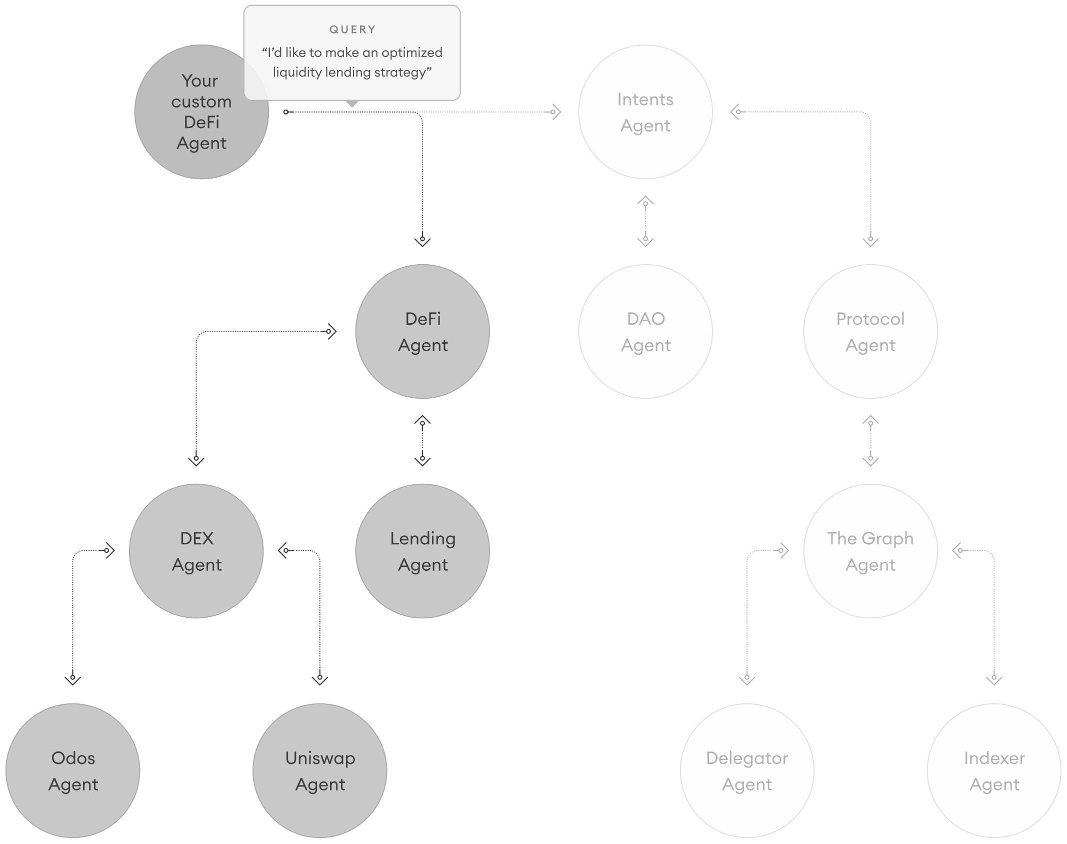


Figure 9: Personalized agents are root nodes in a hierarchical agent tree. Dashed lines show parts of the hierarchy that a personal agent doesn't use for this example.

Outsourcing these steps in the house building example to an external expert is like adding another root node to the intents hierarchy (fig. 9). The crypto community can implement the Internet of Agents with this modular framework. Furthermore, since agents are built on the shared hierarchy, this incentivizes openness and collaboration on the hierarchy. If the hierarchy improves, it benefits everyone.

References

[1] B. R. Andrus, Y. Nasiri, S. Cui, B. Cullen, and N. Fulda. Enhanced story comprehension for large language models through dynamic document-based knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10436–10444, 2022.

[2] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung, et al. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*, 2023.

[3] N. Bian, X. Han, L. Sun, H. Lin, Y. Lu, and B. He. Chatgpt is a knowledgeable but inexperienced solver: An investigation of commonsense problem in large language models. *arXiv preprint arXiv:2303.16421*, 2023.

[4] V. Buterin. The promise and challenges of crypto + ai applications — vitalik.eth.limo. https://vitalik.eth.limo/general/2024/01/30/cryptoai.html, 2024. [Accessed 19-05-2024].

[5] B. Cao, H. Lin, X. Han, L. Sun, L. Yan, M. Liao, T. Xue, and J. Xu. Knowledgeable or educated guess? revisiting language models as knowledge bases. *arXiv preprint arXiv:2106.09231*, 2021.

[6] K. Conway, C. So, X. Yu, and K. Wong. opml: Optimistic machine learning on blockchain, 2024.

[7] D. Crapis. Advanced rag techniques: an illustrated overview. https://davidecrapis.notion.site/The-Internet-of-Agents-23aa09799b9c4620a1a287926bcfd6af, 2024. [Accessed 19-05-2024].

[8] V. Fabusola. Web 3.0 does have ui/ux problems. https://dailycoin.com/web3-has-ui-ux-problems/, 2023. [Accessed 19-05-2024].

[9] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[10] I. Ilin. Advanced rag techniques: an illustrated overview. https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6, 2023. [Accessed 17-05-2024].

[11] N. Kandpal, H. Deng, A. Roberts, E. Wallace, and C. Raffel. Large language models struggle to learn long-tail knowledge. In *International Conference on Machine Learning*, pages 15696–15707. PMLR, 2023.

[12] O. Khattab, K. Santhanam, X. L. Li, D. Hall, P. Liang, C. Potts, and M. Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*, 2022.

[13] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[14] R. Merritt. What is retrieval-augmented generation, aka rag? https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/, 2023. [Accessed 19-05-2024].

[15] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2024.

[16] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.

[17] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis, 2023.

[18] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*, 2023.

[19] S. Sisneros. Using The Graph to Preserve Historical Data and Enable EIP-4444 — ethresear.ch. https://ethresear.ch/t/using-the-graph-to-preserve-historical-data-and-enable-eip-4444/17318, 2023. [Accessed 17-05-2024].

[20] J. Teutsch and C. Reitwießner. A scalable verification solution for blockchains. *arXiv preprint arXiv:1908.04756*, 2019.

[21] The Graph Foundation. Historical Ethereum Data Access After EIP-4444 — thegraph.com. https://thegraph.com/blog/historical-ethereum-data-access/, 2021. [Accessed 17-05-2024].

[22] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.

[23] N. F. VC. Does crypto have a ux problem?, Jan 2022.

[24] C. Wang, P. Liu, and Y. Zhang. Can generative pre-trained language models serve as knowledge bases for closed-book qa? *arXiv preprint arXiv:2106.01561*, 2021.

[25] C. Wang, X. Liu, and D. Song. Language models are open knowledge graphs. *arXiv preprint arXiv:2010.11967*, 2020.

[26] J. Wang, W. Huang, Q. Shi, H. Wang, M. Qiu, X. Li, and M. Gao. Knowledge prompting in pre-trained language model for natural language understanding. *arXiv preprint arXiv:2210.08536*, 2022.

[27] S. Xu, S. Liu, T. Culhane, E. Pertseva, M.-H. Wu, S. Semnani, and M. Lam. Fine-tuned llms know more, hallucinate less with few-shot sequence-to-sequence semantic parsing over wikidata. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[28] L. Yadlos. The graph achieves one trillion queries, redefining data indexing and querying, Jun 2023.

[29] L. Yang, H. Chen, Z. Li, X. Ding, and X. Wu. Give us the facts: Enhancing large language models with knowledge graphs for fact-aware language modeling. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[30] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen, et al. Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*, 2023.