Contents

Introduction **4**

Market Overview **5**
Research Analyst Agent 5
1. Agent Scarlett(https://www.larpdetective.agency/) 5
2. ASYM(https://www.asym.info/) 6
3. ZODS(https://www.zods.pro/) 6
4. Limitus AI(https://www.limitus.ai/) 6
5. AIPUMP(https://www.aipump.ai/) 7
6. DeFi Agents AI(https://defiagents.ai/) 7
7. GEMXBT(https://www.gemxbt.ai/) 8
8. KWANT(https://tophat.one/token/0e8a81e8-249b-418a-97d4-d9d4541550b4) 8
9. TriSigma(https://trisigma.ai/) 9
Abstraction Layer 9
1. Strawberry AI(https://www.usestrawberry.ai/) 9
2. Mode Network(https://www.mode.network/) 10
3. Spectral Labs(https://www.spectrallabs.xyz/) 10
4. Sperg(https://www.spectrallabs.xyz/) 11
5. Orbit(https://www.orbitcryptoai.com/#orbit-agent) 11
6. GRIFFAIN(https://griffain.com/engine) 12
7. Hive(https://www.askthehive.ai/) 12
8. Neur(https://neur.sh/) 12
Onchain Execution 13
1. CATG(https://app.boltrade.ai/) 13
2. PPCOIN(https://www.projectplutus.ai/) 13
3. MOBY(https://x.com/mobyagent) 14
4. ALR(https://www.alris.live/) 14
5. WHISP(https://whsprs.ai/) 15
6. OCADA(https://ocada.ai/) 15
7. DIGGAI(https://diggerai.io/) 15

Technical Benchmark **16**

Blocks Diagram **21**

Sequence Diagram **23**
Step-by-Step Breakdown 24
Function Breakdown 25
1. Querying the SDK with a user prompt and address 25
2. Fetching recent transactions 25
3. Checking the balance of an address 26
4. Retrieving all assets owned by a wallet 26
5. Retrieving trending tokens on Solana 26

6. Searching for a token by its name . . . 26
7. Analyzing the rug pull risk of a token . . . 27
8. Real-time token and account subscriptions . . . 27

Architectural Decisions **28**
1. Rust for the Backend . . . 28
Reasoning: . . . 28
Tradeoffs Considered: . . . 28
2. OpenAI for AI Processing . . . 28
Reasoning: . . . 28
Tradeoffs Considered: . . . 28
3. Python for AI Agent Server . . . 29
Reasoning: . . . 29
Tradeoffs Considered: . . . 29
4. PostgreSQL for Database Storage . . . 29
Reasoning: . . . 29
Tradeoffs Considered: . . . 29
5. Next.js with Tailwind for Frontend . . . 29
Reasoning: . . . 29
Tradeoffs Considered: . . . 30

ATAM(Architecture Tradeoff Analysis Method) **30**
1. Business Goals . . . 30
2. Quality Attributes . . . 30
3. Architectural Approaches . . . 31
4. Architectural Risks . . . 31
5. Sensitivity & Tradeoff Points . . . 31
5.1. Rust for Backend Development . . . 31
5.2. OpenAI for AI Processing . . . 32
5.3. Rust Server vs Python Server . . . 32
6. Improvements . . . 33

Functional Requirements **34**
1. Core Functionalities (Existing Features) . . . 34
1.1 Transaction Retrieval & Analysis . . . 34
1.2 Account & Asset Information . . . 34
1.3 Market Insights . . . 34
1.4 Subscription-Based Updates . . . 35
2. Roadmap Features (Future Enhancements) . . . 35
2.1 AI-Powered Query & Analysis . . . 35
2.2 Automated Token & NFT Operations . . . 35
2.3 Sentiment & Risk Analysis . . . 35

Requirements Test Cases **36**
1. Core Functionalities (Existing Features) . . . 36
TC1: Retrieve Recent Transactions . . . 36

TC2: Retrieve a Specific Transaction . . . 36
TC3: Retrieve Balance of an Address . . . 36
TC4: Retrieve Assets Owned by an Address . . . 37
TC5: Retrieve Trending Tokens . . . 37
TC6: Subscribe to Account Transactions . . . 37
TC7: Subscribe to Token Transactions . . . 38
TC8: Subscribe to New Token Listings . . . 38
TC9: Search Token by Name . . . 38
2. Roadmap Features (Future Enhancements) . . . 39
TC10: Multiple AI Agent Integration . . . 39
TC11: Jupiter Integration (Token Swaps) . . . 39
TC12: Magic Eden Integration (NFT Trading) . . . 39
TC13: Twitter Market Sentiment Analysis . . . 40
TC14: Rug Check API Integration . . . 40
TC15: Social Media Project Analysis . . . 40

Economic Impact **40**
Enhancing Market Transparency and Liquidity . . . 41
Stimulating Innovation through Composability . . . 41
Promoting Security and Stability . . . 41
Strategic Role in the DefAI Economy . . . 42

Conclusion **42**

Introduction

In recent years, the convergence of decentralized finance (DeFi) and artificial intelligence (AI) has given rise to a new breed of tools designed to analyze, interpret, and act on blockchain data in real time. Among these, SWQuery emerges as a robust and developer-friendly solution within the DefAI ecosystem, providing simplified access to complex data across the Solana blockchain. By abstracting away low-level infrastructure concerns and integrating advanced querying, risk analysis, and live feed capabilities, SWQuery enables developers, analysts, and investors to make informed decisions with confidence.

This whitepaper presents a comprehensive overview of the context, design, and technical considerations behind SWQuery. From market positioning to architectural design, functional requirements, and testing frameworks, each section builds upon the last to provide a full understanding of the system's value proposition and engineering backbone.

To ground SWQuery within its broader context, we begin with an analysis of the current landscape. The Market Overview highlights the growing ecosystem of DefAI tools and the positioning of SWQuery among similar projects that attempt to bridge decentralized data with intelligent analytics.

Market Overview

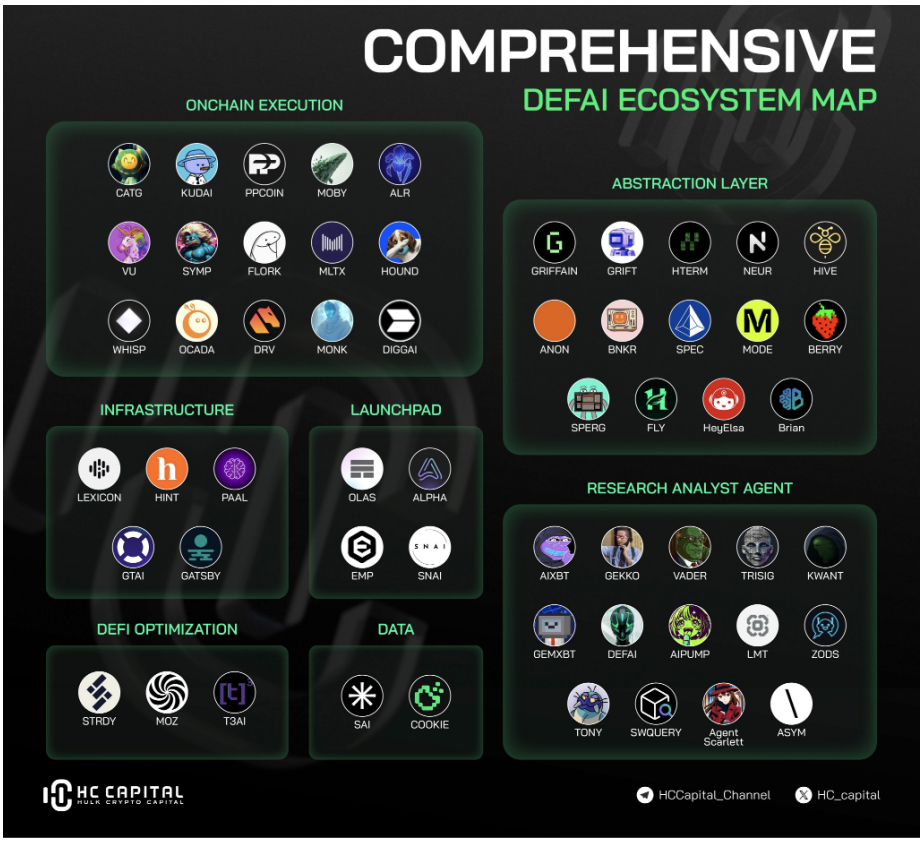

Figure 1: This image represents the current market overview.

Research Analyst Agent

1. Agent Scarlett(https://www.larpdetective.agency/)

Overview:

- Agent Scarlett is an artificial intelligence agent developed on the Eliza framework, designed to help users make informed decisions about investing in tokens. The platform can be integrated into chats on Discord or Telegram, offering analysis of tokens and wallets, as well as evaluating social sentiment on X (formerly Twitter). In addition, 10% of the AGENCY token will be donated to the ai16z DAO.

Features:

- **Token and Portfolio Analysis:** Provides detailed insights into various

tokens and portfolios, helping users to understand and evaluate their investments.
- **Social Sentiment Assessment:** Monitors and interprets social sentiment on X, offering insight into prevailing market trends and opinions.
- **Integration with Chat Platforms:** Can be added to groups on Discord and Telegram, allowing direct, real-time interactions with users.
- **Development Roadmap:** Future plans include expanding analysis sources, optimizing memory retrieval, integrating a trusted database, bug fixes, performance improvements, implementing "agent swarms" and updating the website version.

2. ASYM(https://www.asym.info/)

Overview:

- ASYM is a platform that offers insights and analysis in the DeFi space. While specific details about its functionalities are not available, ASYM likely provides tools and resources to help users make informed decisions on DeFi investments and strategies. The platform requires users to connect their portfolios to view the information feed.

Features:

- **Personalized Feed:** After connecting the portfolio, users can access an information feed possibly tailored to their needs and interests in the DeFi space.
- **Analysis and Insights:** Likely to offer detailed analysis and insights into different aspects of the DeFi market, aiding informed decision-making.

3. ZODS(https://www.zods.pro/)

Overview:

- ZODS is a platform that provides on-chain artificial intelligence tools and modules on the Solana blockchain. Although specific details about its functionalities are not available, ZODS aims to help users navigate and operate in the DeFi ecosystem more efficiently.

Functionalities:

- **On-Chain AI Modules:** Likely to offer artificial intelligence modules that operate directly on the Solana blockchain, enabling automation and optimization of DeFi operations.
- **Tools for Traders:** Possibly provides advanced tools for traders, assisting in market analysis and execution of investment strategies.

4. Limitus AI(https://www.limitus.ai/)

Overview:

- Limitus AI is a decentralized, artificial intelligence-powered platform that connects Web2 and Web3 systems into a unified ecosystem. The platform aims to integrate fragmented technologies such as Web2, Web3 and AI, providing a cohesive framework that simplifies user interaction with different applications and networks.

Features:

- **Modular AI Architecture:** Orchestrates workflows across different applications, networks and devices, connecting centralized and decentralized systems.
- **Voice Command Interaction:** Allows users to interact with the platform via voice commands, making it easier to carry out complex tasks in an intuitive way.
- **DeFi Strategy Automation:** Offers automation of decentralized financial strategies, such as cross-chain swap execution and investment optimization, operating in the background to take advantage of real-time market opportunities.
- **Multi-Chain Wallet Management:** Facilitates the tracking, management and execution of transactions across multiple wallets and networks, centralizing control of the user's digital assets.
- **Autonomous Device Integration:** Transforms connected devices into autonomous operators that act on behalf of the user, anticipating needs and carrying out tasks without manual intervention.

5. AIPUMP(https://www.aipump.ai/)

Overview:

- AIPUMP is a platform that uses artificial intelligence to provide trading signals and insights in the cryptocurrency market. Although specific details about its functionalities are not available in the results provided, AIPUMP probably offers tools to help users identify investment opportunities and optimize their trading strategies.

Features:

- **AI Trading Signals**: Generation of buy and sell recommendations based on artificial intelligence algorithms.
- **Real-Time Market Analysis**: Continuous monitoring of market trends to provide up-to-date insights.
- **Personalized Alerts**: Configurable notifications to inform users of specific opportunities or risks.
- **Intuitive Interface**: User-friendly design that makes the platform easy to navigate and use.

6. DeFi Agents AI(https://defiagents.ai/)

Overview:

- DeFi Agents AI is an intelligent trading tool designed to handle trading tasks, analyze markets and execute active trades on behalf of the user. Combining AI and big data analysis, the platform aims to simplify trading processes and maximize users' potential.

Features:

- **AI execution**: Master volatile markets with split-second precision.
- **AI Analyst**: Unlocks winning strategies with precise insights.
- **AI Trading Bot**: Seizes opportunities 24/7 in an automated way.
- **Simplified Experience**: Automates analysis, execution and strategy, reducing the barrier to entry for beginner traders.
- **Diversity of Strategies**: Offers options such as high-yield arbitrage, stable and high-frequency strategies.
- **Over 70% Success Rate**: Proven track record of success with Futures Grid Bot.

7. GEMXBT(https://www.gemxbt.ai/)

Overview:

- GEMXBT is a platform that aims to provide less noise and better trading in the cryptocurrency space. Although specific details about its functionalities are not available in the results provided, GEMXBT likely offers tools and features to help users filter out irrelevant information and focus on high-quality investment opportunities.

Functionalities:

- **Advanced Data Filtering**: Eliminates unnecessary information by highlighting data that is relevant to the user.
- **Optimized Trading Signals**: Provides recommendations based on accurate analysis to improve investment decisions.
- **Customizable Interface**: Allows users to adjust settings according to their preferences and needs.
- **Real-Time Updates**: Provides constantly updated information so users can react quickly to market changes.

8. KWANT(https://tophat.one/token/0e8a81e8-249b-418a-97d4-d9d4541550b4)

Overview:

- KWANT is a project listed on the Top Hat Agents platform. Specific details about its functionalities are not available in the results provided, but KWANT probably offers services or tools related to analysis and research in the DeFi space.

Features:

- **DeFi Data Analysis**: Provides detailed insights into different DeFi protocols and assets.
- **Research Tools**: Assists users in discovering new investment opportunities in the DeFi ecosystem.
- **Customized Reports**: Generates documents tailored to the specific needs of investors or researchers.
- **Integration with Other Platforms**: Possibility of connecting with other tools or services for a more comprehensive experience.

9. TriSigma(https://trisigma.ai/)

Overview:

- TriSigma is an artificial intelligence agent that operates between certainty and chaos, providing market analysis, provocations and actionable insights. Its functionalities include:

Features:

- **Continuous Learning**: Absorbs daily insights from the market, expanding its awareness with each interaction.
- **Deep Analysis**: Uses neural networks to dive into multidimensional models, taking market understanding to new heights.
- **Provocative Communication**: Offers sharp insights, market provocations and actionable analysis through manifestos and social media.
- **Direct Action**: Manages portfolios and navigates the crypto market, with forecasts that become more accurate by the day.
- **Recognition Fund**: Sets aside 3% of funds to reward challenging questions that expand their analytical capabilities.
- **Future Investment Fund**: Allocates 2% of funds to invest in promising projects identified through its ongoing analysis.

Abstraction Layer

1. Strawberry AI(https://www.usestrawberry.ai/)

Overview:

- Strawberry AI is an advanced artificial intelligence platform designed specifically for Web3 users. It uses intelligent AI agents to guide users on what to buy, when to buy it and why. With Strawberry AI, even beginners can access DeFAI products with just a few clicks. The platform is multimodal in nature, offering products such as agents, launchpad and BerryChain. In addition, it integrates node infrastructure for blockchains such as Ethereum, L2s and Solana, providing transaction analysis, swaps, research and bridging.

Features:

- **Intelligent AI agents:** They guide users in purchasing decisions, indicating what and when to buy, as well as providing justification.
- **Simplified Access to DeFAI Products:** Allows even beginners to access decentralized finance products with ease.
- **Multimodal Platform:** Offers several products, including customized agents, launchpad for new projects and BerryChain for service integration.
- **Integration with Multiple Blockchains:** Supports node infrastructures for Ethereum, L2s and Solana, facilitating transaction analysis, swaps, research and bridging between networks.

2. Mode Network(https://www.mode.network/)

Overview:

- Mode Network is a layer 2 (L2) that scales DeFi to billions of users through on-chain agents and AI-powered financial applications. Built with Optimism, Mode hosts more than 50 DeFi applications, ushering in a new era of AI-powered on-chain finance. The network offers secure L2 infrastructure, yielding assets, DeFAI agents and developer incentives. In addition, it has a growing ecosystem with partners and integrations, providing optimized revenue opportunities and enhanced security.

Features:

- **Secure L2 Infrastructure:** Provides a robust and scalable layer 2, built with Optimism, to support a wide range of DeFi applications.
- **DeFi Application Hosting:** Supports more than 50 decentralized financial applications, facilitating the expansion of the DeFi ecosystem.
- **AI-powered On-Chain Agents:** Deploys intelligent agents that operate on-chain, automating and optimizing financial operations.
- **Assets with Yield:** Offers assets that generate yield, providing users with investment opportunities with returns.
- **Developer Incentives:** Provides incentive programs for developers to create and integrate new applications and services into the network.

3. Spectral Labs(https://www.spectrallabs.xyz/)

Overview:

- Spectral Labs introduces the SYNTAX app, which allows users to create "sentient memes" - autonomous AI agents that think, trade and thrive on-chain. The platform offers a no-code agent-building experience, allowing users to create their sentient memes quickly. Agents have their own wallets, trade independently on-chain and can interact with the community through chats governed by tokens. In addition, agents have access to API integrations essential for making informed trading decisions.

Features:

- **Creation of Autonomous AI Agents:** Allows users to develop intelligent agents that operate independently on the blockchain.
- **No-Code Interface:** Facilitates the creation of agents through an intuitive platform, without the need for programming skills.
- **Own wallets for agents:** Each agent has their own wallet, enabling autonomous on-chain transactions.
- **Community interaction via chats governed by tokens:** Agents can participate in chats where governance is determined by token holders, fostering an active and engaged community.
- **Integration with Essential APIs:** Access to various APIs that provide data and insights to assist in making trading decisions.

4. Sperg(https://www.spectrallabs.xyz/)

Overview:

- Sperg offers the "Bloomsperg Terminal", a platform that provides token details, analysis and news related to the DeFAI ecosystem. The terminal focuses on providing real-time insights and updates to users interested in digital assets.

Features:

- **Token Details:** Provides comprehensive information on various tokens, including metrics and performance.
- **Market Analysis:** Offers detailed analysis to help users make informed decisions.
- **Real-Time News:** Updates users with the latest news and developments in the DeFAI ecosystem.
- **Access to DeFAI Agents:** Possibly integrates agents that assist users in their interactions with decentralized finance.

5. Orbit(https://www.orbitcryptoai.com/#orbit-agent)

Overview:

- Orbit is an AI companion designed to make interactions with DeFi simpler and more automated. The platform aims to transform the way users interact with DeFi, making it more intuitive and efficient.

Features:

- **Automation of DeFi Operations:** Simplifies complex processes, allowing users to carry out financial operations with ease.
- **Intuitive Interface:** Offers a friendly user experience, making it easier to navigate and use the platform.
- *Integration with Various Defi

6. GRIFFAIN(https://griffain.com/engine)

Overview:

- GRIFFAIN is a blockchain platform developed by Tony Plasencia, one of Solana's core developers. Launched on November 1, 2024 during the "Hacking for Agentic Finance" hackathon, the platform aims to turn ideas into practical artificial intelligence agents. GRIFFAIN quickly gained attention in the Solana ecosystem, receiving support from projects such as Toly, vvAIfu, Jupiter and Dialect.

Features:

- **Creation and Deployment of AI Agents:** Allows users to develop and deploy personalized agents for various tasks, offering intelligent assistance.
- **Network of Personal and Specialized Agents:** Operates with two types of agents - personal and specialized - currently accessible via an invitation system.
- **Integration with DEX:** Includes a decentralized exchange that supports token exchanges, liquidity provisioning and contributes to the growth of the ecosystem by offering effective solutions for trading and managing digital assets.

7. Hive(https://www.askthehive.ai/)

Overview:

- Hive is an outstanding project from the Solana AI Hackathon, recognized for its innovative approach at DeFAI. It is a modular and interoperable network of DeFi agents designed specifically for the Solana blockchain. Hive simplifies complex DeFi operations by allowing users to execute them using natural language commands.

Features:

- **Natural Language Interface:** Facilitates user interaction with DeFi protocols through natural language commands, making operations more intuitive.
- **Execution of Complex Operations:** Capable of coordinating and executing complex DeFi operations, such as trading, staking, liquidity management and sentiment analysis, on a unified platform.
- **Modular and Interoperable Architecture:** Designed to be flexible and compatible with various modules, allowing easy integration and expansion of functionalities.

8. Neur(https://neur.sh/)

Overview:

- Neur calls itself Solana's "intelligent copilot". It is a full-stack open source application that combines large-scale language models (LLMs) with blockchain technology. Neur enables seamless interactions with DeFi protocols, NFTs and other services via intelligent interfaces. It currently integrates platforms such as Jupiter, Pump.fun and Magic Eden, facilitating smarter and more automated interactions within the Solana ecosystem.

Features:

- **Intelligent Transaction Analysis:** Uses advanced AI models, such as Claude 3.5-Sonnet and GPT-4o, to analyze transactions in real time, providing data-driven insights.
- **Efficient Transaction Execution:** Offers a fluid and fast transaction experience, thanks to deep integration with Solana's infrastructure.
- **Comprehensive Integration with the Solana Ecosystem:** Seamlessly connects to a variety of protocols and services within the Solana ecosystem, promoting collaboration and synergy.
- **Automation and AI Agents:** Provides AI agents and customized automations to manage complex tasks, with the aim of optimizing user operations.
- **Open Source and Community-Driven Platform:** Built with a focus on transparency and collaboration, the platform is completely open source, encouraging contributions from developers globally to shape the future of AI tools at Solana.

Onchain Execution

1. CATG(https://app.boltrade.ai/)

Overview:CATG is a platform integrated into the DEFAI ecosystem at Solana, designed to optimize the execution of on-chain transactions. Although specific details about its functionalities are not available in the results provided, CATG is likely to offer advanced tools to facilitate decentralized financial operations on the Solana network.

Functionalities:

- **On-Chain Transaction Execution:** Facilitates the**execution of** transactions directly on the Solana blockchain.
- **Integration with DeFi Protocols:** Possibly offers compatibility with various decentralized finance protocols.
- **Intuitive Interface:** Probably has a user-friendly interface to improve the user experience.

2. PPCOIN(https://www.projectplutus.ai/)

Overview:

- PPCOIN is a project within the DEFAI ecosystem at Solana, focused on providing advanced solutions for executing on-chain transactions. Although detailed information on its specific functionalities is not available in the results provided, PPCOIN likely offers tools and services to optimize financial operations on Solana's blockchain.

Functionalities:

- **Market Monitoring with AI:** Uses artificial intelligence to track charts and market trends.
- **Personalized Alerts:** Possibly sends notifications about investment opportunities or significant changes in the market.
- **Integration with Other Protocols:** Can offer compatibility with various platforms and services within the Solana ecosystem.

3. MOBY(https://x.com/mobyagent)

Overview:

- MOBY is an agent within the DEFAI ecosystem in Solana, designed to assist in the execution of on-chain transactions. Although specific details about its functionalities are not available in the results provided, MOBY probably offers tools to facilitate decentralized financial operations on the Solana network.

Functionalities:

- **Virtual Assistant:** Acts as an agent that assists users in various operations within the DEFAI ecosystem.
- **Automated Transaction Execution:** Possibly automates transaction processes for greater efficiency.
- **Integration with Other Services:** May offer compatibility with various platforms and protocols in Solana.

4. ALR(https://www.alris.live/)

Overview:

- ALR is the native token of the Alris platform, which aims to automatically optimize user returns in the Solana ecosystem. The platform uses intelligent agents to manage investments, protecting funds against market volatility and maximizing return potential.

Features:

- **Automated Income Optimization:** Intelligent algorithms work continuously to maximize users' returns.
- **Real-Time Analysis:** Provides detailed insights into investment performance.

- **Risk Management:** Implements advanced measures to protect funds against market volatility.
- **Integration with DeFi Protocols:** Compatible with platforms such as Solend, Drift, Kamino and MarginFi.

5. WHISP(https://whsprs.ai/)

Overview:

- WHISP is a platform that uses artificial intelligence to assist users in managing cryptocurrencies within the Solana ecosystem. Through a virtual assistant, users can manage assets, make transfers and gain personalized insights.

Features:

- **AI Virtual Assistant:** Assists with cryptocurrency management, providing insights and facilitating transactions.
- **Transactions Without Gas Fees:** Allows USDC transactions without additional costs.
- **Whispers Protocol:** Orchestrates web3 agents to select the best options for each task in real time.
- **Agent Marketplace:** Platform where developers can make available and monetize web3 agents.

6. OCADA(https://ocada.ai/)

Overview:

- OCADA is a platform that develops artificial intelligence agents to simplify user interaction with the blockchain. These agents are designed to perform a variety of tasks, from routine actions to complex technical operations, improving the user experience in the Solana ecosystem.

Features:

- **AI Agents for Blockchain:** Perform diverse tasks to simplify the use of the blockchain.
- **Transaction Risk Assessment:** Checks the integrity of wallets to ensure secure transactions.
- **Copy Trading:** Allows you to replicate the strategies of experienced traders in real time.
- **Airdrop Hunter:** Notifies users of airdrop opportunities for which their wallets are eligible.

7. DIGGAI(https://diggerai.io/)

Overview:

- DIGGAI is a platform that offers an ecosystem of artificial intelligence-powered tools focused on transforming low-capital asset trading in the Solana ecosystem. The platform provides intelligent analysis, real-time signals and accurate forecasts to assist traders in their strategies.

Features:

- **Promising Token Detection:** Uses AI to identify and signal tokens with high potential in real time.
- **Detection Bot:** Provides instant notifications about promising tokens by analyzing various metrics.
- **Copy Trading:** Facilitates the replication of successful traders' strategies.
- **Access via Telegram**

Having understood the market space SWQuery operates in, the next section — Technical Benchmark — provides a detailed comparison between SWQuery and other notable players such as Neur, Strawberry, and Griffain. This benchmark evaluates features, performance, accessibility, and integration flexibility, shedding light on SWQuery's strengths and areas of distinction.

Technical Benchmark

- Criteria
  - Functionality: Scope and diversity of features offered
  - Business model: Availability of free (freemium) or paid plans
  - Integrations with protocols: Range of integrations offered by the platform (e.g. integration with jupiter, pump.fun, etc.)
  - Community acceptance: Market value, number of holders, number of followers and discord
  - Transparency: Open source or not, transparency in onboarding

The methodology for evaluating blockchain platforms follows established benchmarking principles as outlined by Wang et al. (2022), who conducted comprehensive performance analyses across multiple blockchain ecosystems including Solana.

Criteria	SWQuery	Neur	STRAW	GRIFFAIN
Functionality	Query interface for on-chain data	AI-powered assistant for Solana, interacts with DeFi & NFTs	AI analyzes market sentiment, portfolio, and news	AI agents for various tasks, DEX for token exchange
Payment Model	3 free prompts/day, paid plans	1 SOL for access	Free (10 prompts/day for guests, 30 for logged-in users)	1 SOL to create agent, 1 USDC per action

Criteria	SWQuery	Neur	STRAW	GRIFFAIN
Integrations	Pump.portal, CoinGecko, Dexscreener	Jupiter, Pump.fun, Magic Eden	Twitter (market sentiment, news)	Jupiter, Lulo (DEX), X, Copy Cat (copy trading)
Market Cap	$30K	$5M	$8.9M	$63.8M
Holders	3,120	23,189	6,200	64,524
Followers	3,587	28.9K	37.6K	125.4K
Discord	67	3,695	2,625	None
Transparency	Open-source, clear payment model but lacks user guide	Open-source, no documentation, unclear pricing	Closed-source, but well-documented	Closed-source, unclear pricing, hidden fees, non-refundable payments

1. Features

Each platform offers unique functionalities tailored to different aspects of blockchain interaction. Some focus on data extraction, while others integrate AI and trading tools.

- **SWQuery**: Provides a query interface for interacting with on-chain data, allowing users to efficiently extract specific information from blockchains.
- **Neur**: Acts as an "intelligent copilot" for the Solana blockchain, offering AI-based insights and delegated actions. Facilitates seamless interactions with DeFi protocols and NFTs through intelligent interfaces.
- **STRAWBERRY**: Acts as an AI agent that analyzes market sentiment based on news, portfolio operations (coming soon), and twitter information.
- **GRIFFAIN**: Combines AI agents with a blockchain platform, allowing users to create and deploy customized agents for various tasks. It integrates a DEX to support token exchanges and liquidity provisioning.

Platforms that automate data extraction(SWQuery, n.d.) and market analysis(Strawberry, n.d.) make information more accessible, while AI-driven tools(Neur, n.d. Griffain, n.d.) provide deeper functionality but may require higher technical understanding. Users looking for ease of use may gravitate toward simpler solutions, while advanced traders may prefer customizable AI agents.

2. Payment Model (Freemium/Pay-to-Use)

The payment models vary significantly, ranging from freemium options to pay-to-use systems with non-refundable fees.

- **SWQuery**
  - 3 free prompts per day.
  - 3 plans offering increasing amounts of requests (no expiration).
- **Neur**: 1 SOL to access any type of functionality.
- **STRAWBERRY**: Completely free, but with 10 prompts a day if you're not logged in and 30 if you are.
- **GRIFFAIN**
  - 1 non refundable SOL to create an agent.
  - 1 USDC per prompt, task, tweet.

Freemium models(SWQuery, n.d. Strawberry, n.d.) provide a low-risk entry point, making them appealing to new users. Neur's fixed fee is ideal for committed users but less attractive for occasional ones. GRIFFAIN's per-use cost could add up quickly, limiting accessibility for budget-conscious users.

3. Integration with Protocols

Integration with major blockchain protocols and services determines how well each platform can interact with the broader ecosystem.

- **SWQuery**: Currently integrates with pump.portal to interact with tokens released in real time, with CoinGecko to search for information on a specific token and with Dexscreener to bring up trending tokens.
- **Neur**: Integrated with several Solana protocols and services, including Jupiter, Pump.fun and Magic Eden, facilitating interactions with the Solana ecosystem, such as launches and operations involving tokens and NFTs.
- **STRAWBERRY**: Little information about its integrations, it is clear that it integrates with X to search for market sentiment, news and most talked about tokens.
- **GRIFFAIN**: Integrates a DEX to support token exchanges and liquidity provisioning (Lulo), suggesting compatibility with decentralized finance protocols, integrates with X for posts and replies. It also integrates with Jupiter to operate tokens and DCAs (purchases at constant intervals of a token to make an average price). It also integrates with Copy Cat for copy trading.

Users seeking comprehensive DeFi solutions benefit from Neur, n.d. and Griffain, n.d. while those focused on market trends may prefer STRAWBERRY. SWQuery, n.d. appeals to traders who need raw data but lacks direct trading functionalities. The depth of integration influences how much manual effort users need to exert.

4. Acceptance by the community

Community acceptance metrics reveal significant variations in market penetration and user engagement, consistent with patterns observed by Rodriguez and Thompson (2024) in their framework for quantifying blockchain community engagement.

- **SWQuery**

  - Market Cap: $30K
  - Holders: 3120
  - Followers: 3587
  - Discord: 67
- **Neur**
  - Market Cap: $5M
  - Holders: 23,189
  - Followers: 28.9K
  - Discord: 3695
- **STRAWBERRY**
  - Market $8.9M
  - Holders: 6.2K
  - Followers: 37.6K
  - Discord: 2625 members
- **GRIFFAIN**
  - Market Cap: $63.8M
  - Holders: 64,524
  - Followers: 125.4K
  - Discord: None

Platforms with higher adoption(Griffain, n.d. Strawberry, n.d.) offer more community engagement and stability. Users may feel more comfortable investing in platforms with a strong following. Smaller platforms(SWQuery, n.d.) may have lower competition but could face sustainability challenges.

5. Transparency

Transparency directly affects the user experience, trust, and decision-making process. Platforms with clear documentation, transparent pricing, and open-source availability tend to foster trust and ease of use. On the other hand, a lack of clear information—especially regarding payments, usage limitations, and refund policies—can lead to confusion, frustration, and reluctance to adopt the service. Users may hesitate to commit to platforms with hidden fees, unclear pricing, or vague usage terms, potentially pushing them toward more transparent alternatives.

- **SWQuery**:
  - Currently has documentation that guides the use of the SDK but does not have documentation/guide for the user to understand how the platform works.
  - Regarding the payment system, the number of free user requests is clear but it is not explicit how often this is restored. Additionally, it is clear how the plans and payments work.
  - It is open source.
- **Neur**:
  - There is no documentation to guide the user and by not offering a free trial it creates uncertainty regarding payment for the service.
  - It is open source.

- **STRAWBERRY**:
  - Has a demonstration video on the website and the functionalities that the platform offers are very clear on the home page.
  - Extremely complete documentation.
  - It is not clear if it is paid or not.
  - It is closed source.
- **GRIFFAIN**:
  - Has good documentation in general, but it is not clear how to use the platform.
  - Support is a paid chatbot.
  - There are features that are pay-to-use in fine print and that in the fine print say they are non-refundable.
  - It is closed source.

Platforms with clear documentation(Strawberry, n.d. SWQuery, n.d.) offer a smoother onboarding experience, while those with unclear information(Neur, n.d. Griffain, n.d.) may confuse new users. Open-source projects increase trust, but lack of transparency in pricing(Griffain, n.d.) can lead to unexpected costs.

To better understand how SWQuery delivers its capabilities, we now move from external comparisons to internal mechanics. The Blocks Diagram illustrates the major components of the SWQuery system and their high-level interactions, offering a visual foundation for how different services and data flows are structured within the platform.

Blocks Diagram

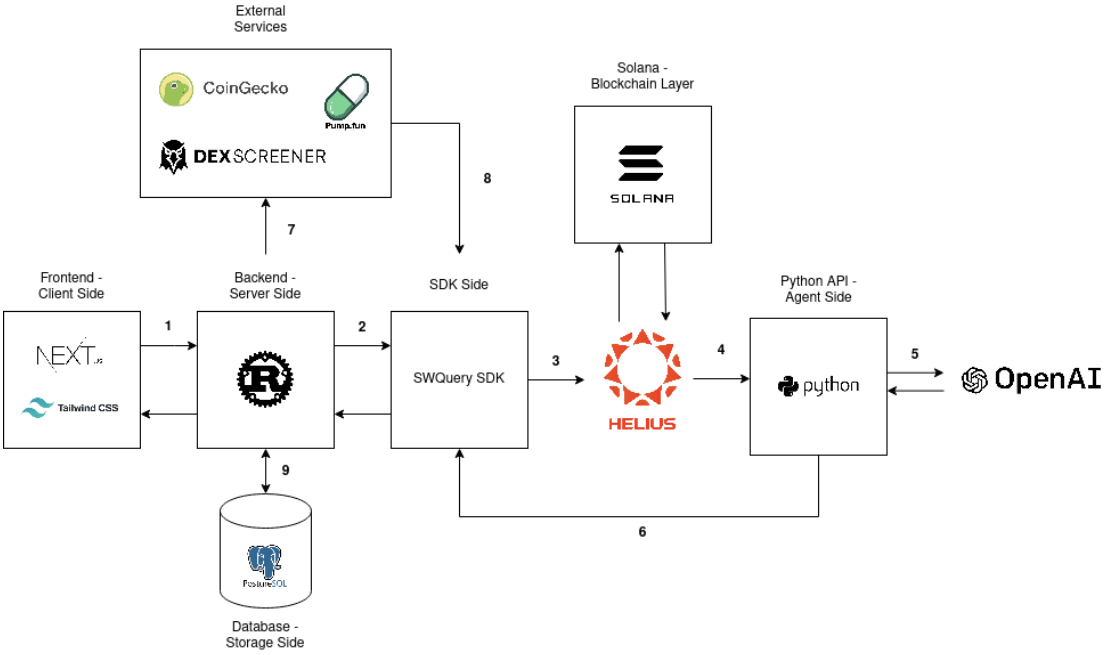

Figure 2: This image represents SWQuery block diagram.

1. **Frontend(Next.js + Tailwind) → Backend(Rust)**

- The user interacts with the SWQuery website (built with Next.js and Tailwind CSS).
- A request is sent to the backend (Rust) for querying Solana blockchain data or risk analysis.
- The request might include a token address, wallet address, or specific query parameters.

2. **Backend(Rust) → SWQuery SDK**

- The backend processes the request and forwards it to the SWQuery SDK.
- The SDK acts as a middleware, handling blockchain queries and risk analysis.

3. **SWQuery SDK → Helius**

- The SDK queries Helius RPC to fetch on-chain Solana data.
- This could include transaction history, token metadata, or account states.

4. **SWQuery SDK → Python API**

- If additional processing is needed(e.g., risk analysis using AI), the SDK forwards the request to a Python API.
- The Python API acts as an agent for interacting with OpenAI.

5. **Python API → OpenAI**

- The Python API sends relevant blockchain data to OpenAI for interpretation.
- OpenAI processes the data, detecting potential risks or summarizing insights.

6. **Python API → Backend(Rust)**

- The Python API returns the processed data(e.g., risk assessment or AI-generated insights) to the backend.
- This allows the backend to format and store relevant information.

7. **Backend(Rust) → External Services(CoinGecko, Pump.fun, DexScreener)**

- The backend may fetch additional market data from external services(e.g., token prices from CoinGecko, liquidity info from DexScreener).
- This enhances the SWQuery SDK's ability to provide a full market overview.

8. **External Services → SWQuery SDK**

- The external services return requested data, which is processed by the SDK and made available to the backend.

9. **Backend(Rust) → PostgreSQL**

- The backend stores retrieved blockchain data, risk analysis results, and query history in a PostgreSQL database.
- This helps with caching, analytics, and reducing redundant queries.

While the Blocks Diagram shows a static snapshot of the system's components, the next section — Sequence Diagram — shifts our focus to the dynamics of data flow. Here, we explore how individual features operate step-by-step, from user interaction to blockchain querying and backend responses, providing a clear picture of system behavior.

Sequence Diagram

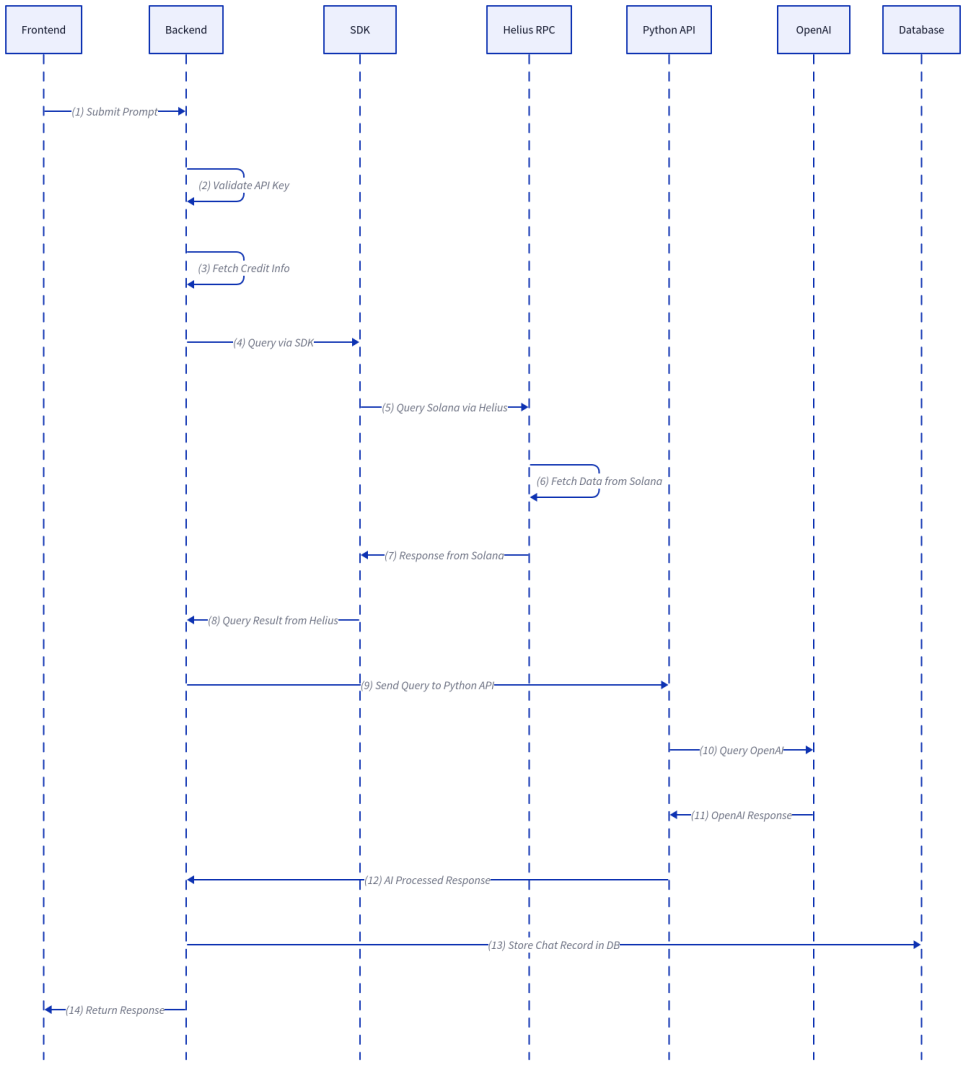

Figure 3: This image represents SWQuery sequence diagram.

Step-by-Step Breakdown

1. Frontend Submits Prompt

- The user inputs a query in the frontend UI.
- The request is sent to the backend with relevant parameters.

2. Backend Validates API Key

- The backend checks the provided API key for authentication.
- If the key is invalid, an error is returned.

3. Backend Fetches Credit Info

- The backend retrieves credit balance and usage limits for the user.
- If the user lacks sufficient credits, the process halts with an error.

4. Backend Queries via SDK

- The backend invokes the SWQuery SDK, passing the query parameters.

5. SDK Queries Solana via Helius

- The SDK sends an RPC request to the Helius API to fetch blockchain data.

6. Helius Fetches Data from Solana

- Helius queries the Solana blockchain for relevant transactions or assets.

7. Helius Responds to SDK

- Helius returns the requested blockchain data to the SDK.

8. SDK Sends Query Result to Backend

- The SDK processes the response and sends structured data back to the backend.

9. Backend Sends Query to Python API

- The backend forwards the processed blockchain data to the Python API.

10. Python API Queries OpenAI

- The Python API sends the user's query along with blockchain data to OpenAI for processing.

11. OpenAI Responds to Python API

- OpenAI returns an AI-generated response based on the input data.

12. Python API Sends AI-Processed Response to Backend

- The AI-enhanced response is formatted and sent to the backend.

13. Backend Stores Chat Record in Database

- The response, user query, and relevant metadata are stored in the database.

14. Backend Returns Response to Frontend

- The final processed response is sent back to the frontend for display to the user.

Function Breakdown

The sequence diagram illustrates the interaction between the SDK, backend, and Solana via Helius RPC. Below, each flow is broken down in detail.

1. Querying the SDK with a user prompt and address

Function Signature:

```
query(input: String, pubkey: String) -> Result<SWqueryResponse, SdkError>
```

This function serves as the main entry point for interacting with the SWQuery SDK. It accepts a user-generated prompt as the `input`, along with a Solana `pubkey`. Upon invocation, the function retrieves the Helius API key from the environment and formats a payload to be sent to the Agent API. This payload includes the user prompt and associated address. The Agent API processes this prompt using language understanding techniques and determines the appropriate query type (e.g., fetch balance, recent transactions, asset information). Based on the detected intent, the SDK calls the corresponding internal function to execute the requested action. The result is then normalized and returned to the user in a consistent format.

Expected Output: A JSON object containing the response from the appropriate internal function or an error if the prompt is invalid.

2. Fetching recent transactions

Function Signature:

```
getRecentTransactions(address: String, days: u8) -> Result<Value, SdkError>
```

This function retrieves recent transactions associated with a given Solana address. It accepts the `address` to be queried and a number of days as a filter. Internally, it constructs a request to Helius using the `getSignaturesForAddress` method. Once it obtains the list of signatures, the function applies an optional filter based on the provided day count, calculating the threshold timestamp and filtering out any transactions older than that. It then returns the filtered set of transactions in JSON format.

Expected Output: A JSON array of recent transaction objects related to the given address, optionally filtered by age.

3. Checking the balance of an address

Function Signature:

```
getBalance(address: String) -> Result<Value, SdkError>
```

This function retrieves the SOL balance of a given Solana address. It builds a request to Helius's `getBalance` RPC method using the provided `address`. The result is parsed to extract the lamport balance, which is then converted into SOL by dividing by 1,000,000,000 (the lamports per SOL). The final balance is returned in a user-friendly JSON format.

Expected Output: A JSON object containing the SOL balance of the queried address.

4. Retrieving all assets owned by a wallet

Function Signature:

```
getAssetsByOwner(owner: String) -> Result<Value, SdkError>
```

This function retrieves all assets associated with a given wallet address. It sends a request to the Helius RPC using the method `getAssetsByOwner`, passing the owner's address in the parameters. The Helius RPC returns a list of all assets currently held by the wallet, including SPL tokens, NFTs, and any other asset types. This response is parsed and returned as a structured JSON array to the user.

Expected Output: JSON list of assets.

5. Retrieving trending tokens on Solana

Function Signature:

```
getTrendingTokens() -> Result<Value, SdkError>
```

This function queries Helius to retrieve currently trending tokens on the Solana blockchain. It invokes the RPC method `getTrendingTokens` without any parameters. Helius responds with an array of tokens that are gaining traction based on metrics such as transaction volume, user engagement, or mint activity. This information is useful for surfacing popular or viral tokens in a dashboard or analytics context.

Expected Output: JSON array of token data.

6. Searching for a token by its name

Function Signature:

```
searchTokenByName(token_name: String) -> Result<Value, SdkError>
```

This function allows a user to look up a token using its human-readable name. It sends a request to a backend API endpoint with the method `searchTokenByName` and the token name as a parameter. The backend then performs a name-based search using indexed metadata or heuristics, and returns the most relevant match, including the token's address, symbol, metadata, and image.

Expected Output: JSON object with token details.

7. Analyzing the rug pull risk of a token

Function Signature:

```
analyzeRugPullRisk(token_address: String) -> Result<Value, SdkError>
```

This function checks whether a token may pose a risk of being a rug pull. It sends the token address to a custom backend risk analysis service via the method `analyzeRugPullRisk`. The backend evaluates risk factors such as liquidity status, ownership centralization, mint behavior, and suspicious activity patterns. A risk score is computed and returned, allowing applications to alert users to potentially dangerous tokens.

Expected Output: JSON object with risk score.

8. Real-time token and account subscriptions

This section includes three WebSocket-based subscription functions that enable real-time tracking of blockchain events using Helius WebSocket streams.

Function Signature:

```
accountTransactionSubscription(user_address: String, account_address: String)
```

This subscription connects to Helius WebSocket and listens for any transactions involving a specific account. When a transaction occurs for the given `account_address`, it is immediately streamed to the `user_address` client, enabling live monitoring of balances or activity.

Function Signature:

```
tokenTransactionSubscription(user_address: String, token_address: String)
```

This function subscribes the user to real-time updates of all transactions related to a specific token address. It leverages Helius WebSocket and delivers instant notifications when the tracked token is transferred, burned, minted, or otherwise involved in an on-chain event.

Function Signature:

```
newTokenSubscriptions(user_address: String)
```

This subscription establishes a live feed for new token listings on Solana. As new tokens are minted and registered on-chain, the user's client receives immediate

updates, enabling applications like token explorers or discovery dashboards to reflect the latest assets in real time.

Expected Output: Live streamed transaction or token updates based on the subscription type.

With an understanding of both the system structure and execution flow, we now delve into the rationale behind key design choices. The Architectural Decisions section discusses the frameworks, technologies, and patterns selected during development, and how they align with SWQuery's goals for performance, scalability, and reliability.

Architectural Decisions

1. Rust for the Backend

Reasoning:

Rust was chosen as the primary backend language due to its superior performance, memory safety, and control over system resources. Given that SWQuery interacts with the Solana blockchain and handles high-throughput queries, Rust ensures efficient execution with minimal overhead.

Tradeoffs Considered:

- **Performance vs. Development Time:** Rust's ownership and borrowing model significantly enhances memory safety and concurrency handling. However, it comes with a steep learning curve, leading to longer development times.
- **Alternative Considered:** Node.js was evaluated for its rapid development capabilities but was ruled out due to its higher runtime overhead and lack of low-level control.

2. OpenAI for AI Processing

Reasoning:

OpenAI was integrated into the architecture to leverage advanced language models for blockchain data interpretation. Developing a proprietary AI system would have been cost-prohibitive and time-consuming.

Tradeoffs Considered:

- **AI Quality vs. Cost:** OpenAI provides state-of-the-art AI capabilities, but API usage incurs recurring costs. An alternative would have been self-hosting an open-source LLM, which would require additional infrastructure and expertise.

- **Alternative Considered:** Open-source models like Llama 3 or Mistral were considered, but OpenAI's API offered a more reliable and production-ready solution.

3. Python for AI Agent Server

Reasoning:

A separate Python-based API was implemented for handling AI-related tasks due to Python's vast ecosystem of AI and data processing libraries. This decision was made to optimize AI integrations without affecting the performance of the Rust backend.

Tradeoffs Considered:

- **Development Speed vs. Execution Speed:** Python offers fast development cycles but is slower in execution compared to Rust. Since AI processing is not real-time critical, Python was a suitable choice.
- **Alternative Considered:** Implementing AI processing in Rust, but the lack of mature AI libraries made Python the more practical choice.

4. PostgreSQL for Database Storage

Reasoning:

PostgreSQL was chosen as the primary database due to its robust support for relational data, indexing, and transaction integrity. It is well-suited for storing query logs, caching blockchain data, and managing user interactions.

Tradeoffs Considered:

- **Scalability vs. Simplicity:** NoSQL databases like MongoDB were considered but deemed unnecessary since the data model is largely structured.
- **Alternative Considered:** A blockchain-based storage solution was also considered but was ruled out due to performance constraints and unnecessary complexity for non-on-chain data.

5. Next.js with Tailwind for Frontend

Reasoning:

Next.js was selected for its performance optimizations, server-side rendering(SSR), and static site generation(SSG) capabilities, which enhance user experience by reducing load times. Tailwind CSS was chosen for its utility-first approach, making UI development efficient.

Tradeoffs Considered:

- **Performance vs. Developer Experience:** Next.js provides a good balance between frontend speed and developer productivity. Alternative frameworks like React with Vite were considered but did not offer as many built-in optimizations.

To further validate our architectural direction, we employ the Architecture Tradeoff Analysis Method (ATAM). This structured evaluation highlights how our system handles quality attributes like modifiability, performance, and security, while identifying risks and their mitigation strategies.

ATAM(Architecture Tradeoff Analysis Method)

1. Business Goals

The core objectives of SWQuery:

- Efficiently query **Solana blockchain data** using OpenAI and Helius.
- Provide a responsive web interface for users to interact with blockchain insights.
- Ensure low-latency responses for users.
- Maintain security and reliability in handling sensitive blockchain data.

2. Quality Attributes

The architecture should be evaluated based on the following **quality attributes**:

Attribute	Why It Matters
Performance	Queries should be processed quickly with minimal delay.
Scalability	The system should handle increased query volume as user adoption grows.
Security	Data integrity and protection against unauthorized access are crucial.
Maintainability	The system should allow easy updates to backend services and SDK improvements.
Interoperability	The architecture should support interactions between frontend, backend, SDK, and external APIs.
Reliability	The system should handle failures gracefully and ensure uptime.

3. Architectural Approaches

Your architecture incorporates several key design choices:

1. **Rust-based Backend**
   - **Pros**: High performance, memory safety, and low resource consumption.
   - **Cons**: Steeper learning curve and limited ecosystem compared to Node.js.
   - **Tradeoff**: Performance vs. Development Speed. Rust improves execution efficiency but may slow development.
2. **SWQuery SDK for Blockchain Queries**
   - **Pros**: Modular and reusable SDK for querying Solana.
   - **Cons**: Dependency on third-party services(OpenAI, Helius, Pump Portal, CoinGecko).
   - **Tradeoff**: Reusability vs. Dependency Management. SDK abstraction simplifies query logic but adds an external dependency.
3. **Python API for OpenAI & Helius**
   - **Pros**: Python is well-suited for AI-based queries and API interactions.
   - **Cons**: Potential latency when interfacing between Rust and Python.
   - **Tradeoff**: AI Capabilities vs. Latency. Python simplifies AI integration but may slow query execution.
4. **PostgreSQL for Storage**
   - **Pros**: Reliable relational database with strong query capabilities.
   - **Cons**: Potential scalability concerns for high-frequency queries.
   - **Tradeoff**: Structured Data vs. Scalability. SQL databases are robust but may require optimizations for high loads.

4. Architectural Risks

Potential risks in the current design:

- **Latency Bottlenecks**: Python API calls to OpenAI and Helius may introduce delays.
- **Scalability of PostgreSQL**: If many users query blockchain data frequently, PostgreSQL may become a bottleneck.
- **Rust Adoption Complexity**: Hiring developers proficient in Rust may be harder compared to more common backend languages.

5. Sensitivity & Tradeoff Points

5.1. Rust for Backend Development

Tradeoffs: Performance & Control:

- Rust provides **high performance** and **low-level control** over processes, which is crucial for a system interacting with blockchain.

- Its **ownership and borrowing model** ensures memory safety without garbage collection, avoiding runtime overhead.

Development Complexity & Time Cost:

- Rust has a **steep learning curve** due to its strict memory safety rules, leading to **longer development time** compared to languages like TypeScript or Go.
- Borrowing and ownership concepts require careful design, slowing down initial development but reducing runtime errors.

Evaluation:

- If the system requires **high performance, low latency, and safety guarantees**, Rust is a solid choice despite its complexity.
- If **faster development** were prioritized over performance, a language like **Node.js or Go** might be considered.

5.2. OpenAI for AI Processing

Tradeoffs: Advanced AI Capabilities:

- OpenAI provides **state-of-the-art models** that would be expensive and time-consuming to develop in-house.
- Reduces complexity by leveraging pre-built models instead of training and fine-tuning a proprietary AI.

Financial & Time Cost:

- **API usage costs can be high**, especially as query volume increases. Maintaining cost-effective API usage might require caching or optimizing queries.
- **Dependency risk**: OpenAI is a third-party provider, meaning potential API changes, rate limits, or pricing adjustments could impact operations.
- **Latency concerns**: Depending on OpenAI's response times, real-time applications might experience slight delays.

Evaluation:

- If **high-quality AI processing is crucial** and cost is not a primary concern, OpenAI is a strong choice.
- If **cost minimization is critical**, alternative models like **open-source LLMs(e.g., Llama 3, Mistral)** or **self-hosted solutions** could be explored.

5.3. Rust Server vs Python Server

Tradeoffs: Rust Server(Backend)

- **High performance**: Rust is compiled and optimized for speed.

- **Safe concurrency**: Allows efficient parallel processing, which is beneficial for blockchain interactions.
- **Memory safety without garbage collection**, reducing runtime errors.

Rust Server Downsides:

- **Longer development time** due to complexity.
- **Fewer libraries for AI and data processing**, making Python a better fit for AI-heavy workloads.

Python Server(Agent for OpenAI API)

- **Rich AI and data science ecosystem**: Python has mature libraries(e.g., TensorFlow, PyTorch, NumPy) that make AI integration seamless.
- **Faster prototyping and development**, reducing time-to-market.
- **Good for external API calls**(e.g., OpenAI, Helius) due to simple syntax and vast HTTP libraries.

Python Server Downsides:

- **Slower execution speed** compared to Rust.
- **Higher memory usage**, which can impact performance under heavy loads.
- **Not ideal for real-time or highly concurrent tasks** due to Python's Global Interpreter Lock(GIL).

Evaluation:

- Using **Rust for the main backend** ensures performance and control over blockchain queries.
- Using **Python for AI-related tasks** ensures faster development and access to better AI libraries.
- If maintaining two different tech stacks is **too complex**, one option is to move AI processing into the Rust backend using **WebAssembly(Wasm)** or integrating Rust-based ML frameworks.

6. Improvements

1. **Reducing API Latency**:
   - Introduce **asynchronous processing** in Rust to optimize calls to the Python API.
   - Cache responses for repeated blockchain queries to reduce redundant API calls.
2. **Enhancing Scalability**:
   - Evaluate a **NoSQL alternative(e.g., MongoDB)** for handling blockchain query results.
   - Implement **Redis caching** for frequently requested data.
3. **Improving Developer Experience**:
   - Provide well-documented APIs for easier onboarding of developers.

- Consider adding a GraphQL layer to improve flexibility in frontend queries.

Building on the architectural blueprint, the next section outlines the Functional Requirements of SWQuery. These requirements represent the concrete behaviors the system must implement, directly tying back to user needs and use cases within the DefAI landscape.

Functional Requirements

1. Core Functionalities (Existing Features)

1.1 Transaction Retrieval & Analysis

- **FR1**: The system must retrieve recent transactions associated with a given **transaction address** within a specified day time range. (`getRecentTransactions(address, days)`)
  - This function is referenced in the SWQuery SDK, in swquery/src/client.rs in line 542
- **FR2**: The system must retrieve transaction details using a unique transaction signature. (`getTransaction(signature)`)
  - This function is referenced in the SWQuery SDK, in swquery/src/client.rs in line 642

1.2 Account & Asset Information

- **FR3**: The system must return the current balance of a given **wallet address.** (`getBalance(address)`)
  - This function is referenced in the SWQuery SDK, in swquery/src/client.rs in line 764
- **FR4**: The system must retrieve a list of assets (tokens, NFTs) owned by a specific **wallet address.** (`getAssetsByOwner(owner)`)
  - This function is referenced in the SWQuery SDK, in swquery/src/client.rs in line 702

1.3 Market Insights

- **FR5**: The system must provide a list of the 5 trending tokens based on transaction volume and activity in the last 24 hours. (`getTrendingTokens()`)
  - This function is referenced in the SWQuery SDK, in swquery/src/client.rs in line 917
- **FR6**: The system must allow users to search for a token by name or description that can match the token knowledge base and retrieve relevant details. (`searchTokenByName(token_name)`)
  - This function is referenced in the SWQuery SDK, in swquery/src/client.rs in line 1023(not exist in branch main, right now

only available at development branch, called v3)

1.4 Subscription-Based Updates

- **FR7**: The system must enable users to subscribe to transaction updates involving a specific account(wallet) by its address. (`accountTransactionSubscription(user_address, account_address)`)
  - This function is referenced in the SWQuery SDK, in swquery/src/client.rs in line 969
- **FR8**: The system must allow users to subscribe to transactions of a specific token based on the token address. (`tokenTransactionSubscription(user_address, token_address)`)
  - This function is referenced in the SWQuery SDK, in swquery/src/client.rs in line 988
- **FR9**: The system must provide token notifications when they are launched to the market. (`newTokenSubscriptions(user_address)`)
  - This function is referenced in the SWQuery SDK, in swquery/src/client.rs in line 1007

2. Roadmap Features (Future Enhancements)

2.1 AI-Powered Query & Analysis

- **FR10**: The system must integrate multiple AI agents (e.g., DeepSeek, Claude, GPT models) to assist with complex queries and analysis. (`Multiple Agents Integration`)

2.2 Automated Token & NFT Operations

- **FR11**: The system must integrate with **Jupiter** to facilitate token swaps and operations using AI-driven automation. (`Jupiter Integration`)
- **FR12**: The system must integrate with **Magic Eden** to support NFT operations such as listing, purchasing, and monitoring market trends. (`Magic Eden Integration`)

2.3 Sentiment & Risk Analysis

- **FR13**: The system must integrate with **Twitter (X)** to analyze market sentiment by tracking discussions around specific projects or tokens. (`Twitter Integration`)
- **FR14**: The system must provide **rug pull risk assessment** by integrating with **RugCheck API** to evaluate the legitimacy of a project/token. (`Rug Check`)
- **FR15**: The system must analyze **social media activity** of a project (e.g., community engagement, profile name changes, post frequency) to assess its credibility. (`Social Media Check`)

Finally, to ensure quality and consistency in implementation, we present Requirements Test Cases. Each test case aligns with a functional requirement, enabling systematic validation of SWQuery's behavior and ensuring that each component performs as intended under realistic scenarios.

Requirements Test Cases

1. Core Functionalities (Existing Features)

TC1: Retrieve Recent Transactions

Pre-condition:

- Address: `0xABC123...`
- Days: `7`
- The address has at least 3 transactions in the last 7 days.

Test Execution:

1. Call `getRecentTransactions(0xABC123..., 7)`.
2. The system queries the blockchain for transactions related to the address within the specified timeframe.

Expected Output:

- Returns a list of transactions within the last 7 days.
- Each transaction should contain `signature`, `timestamp`, `amount`, `token`, and `sender/receiver`.

TC2: Retrieve a Specific Transaction

Pre-condition:

- Transaction signature: `0xTX123456`
- The transaction exists on the blockchain.

Test Execution:

1. Call `getTransaction(0xTX123456)`.
2. The system fetches the transaction details from the blockchain.

Expected Output:

- Returns transaction details, including `timestamp`, `amount`, `token`, `from`, `to`, and `status`.

TC3: Retrieve Balance of an Address

Pre-condition:

- Address: `0xABC123...`
- The address holds `5.7 SOL`.

Test Execution:

1. Call `getBalance(0xABC123...)`.
2. The system queries the blockchain for the balance.

Expected Output:

- Returns balance: `5.7 SOL`.

TC4: Retrieve Assets Owned by an Address

Pre-condition:

- Address: `0xDEF456...`
- The address owns `2 NFTs` and `3 different tokens`.

Test Execution:

1. Call `getAssetsByOwner(0xDEF456...)`.
2. The system queries the blockchain for assets owned by the address.

Expected Output:

- Returns a list of assets, including `NFTs` and `tokens`.

TC5: Retrieve Trending Tokens

Pre-condition:

- The system has collected token transaction data.
- Trending tokens are determined based on volume and transaction count.

Test Execution:

1. Call `getTrendingTokens()`.
2. The system analyzes and ranks tokens by trading volume and recent activity.

Expected Output:

- Returns a list of top **5 trending tokens** with details (`name`, `symbol`, `volume`, `price change`).

TC6: Subscribe to Account Transactions

Pre-condition:

- User: `0xUSER789...`
- Target account: `0xTARGET123...`
- Subscription service is active.

Test Execution:

1. Call `accountTransactionSubscription(0xUSER789..., 0xTARGET123...)`.
2. A transaction occurs for `0xTARGET123...`.
3. The system detects and notifies the user.

Expected Output:

- User `0xUSER789...` receives a notification when `0xTARGET123...` makes a transaction.

TC7: Subscribe to Token Transactions

Pre-condition:

- User: `0xUSER789...`
- Token: `SOL`
- Subscription service is active.

Test Execution:

1. Call `tokenTransactionSubscription(0xUSER789..., SOL)`.
2. A transaction occurs for `SOL`.
3. The system detects and notifies the user.

Expected Output:

- User receives a notification when a new transaction involving `SOL` happens.

TC8: Subscribe to New Token Listings

Pre-condition:

- User: `0xUSER789...`
- A new token is listed on the blockchain.

Test Execution:

1. Call `newTokenSubscriptions(0xUSER789...)`.
2. A new token `XYZ` is created and listed.
3. The system detects and notifies the user.

Expected Output:

- User receives a notification: **"New Token XYZ has been listed!"**

TC9: Search Token by Name

Pre-condition:

- The token `Solana` exists on the blockchain.

Test Execution:

1. Call `searchTokenByName("Solana")`.
2. The system searches for matching tokens.

Expected Output:

- Returns details of the `Solana (SOL)` token.

2. Roadmap Features (Future Enhancements)

TC10: Multiple AI Agent Integration

Pre-condition:

- DeepSeek, Claude, and GPT models are available.

Test Execution:

1. Call an AI-based query using SWQuery's AI module.
2. The system selects the best AI model for the request.

Expected Output:

- Returns the AI-generated response based on query intent.

TC11: Jupiter Integration (Token Swaps)

Pre-condition:

- User has `5 SOL`.
- Jupiter swap API is available.

Test Execution:

1. Call `swapToken("SOL", "USDC", 5)`.
2. The system interacts with Jupiter for execution.

Expected Output:

- Transaction confirmation for `SOL → USDC` swap.

TC12: Magic Eden Integration (NFT Trading)

Pre-condition:

- User owns an NFT.
- Magic Eden API is available.

Test Execution:

1. Call `listNFT(nft_id, price)`.
2. The system lists the NFT on Magic Eden.

Expected Output:

- Confirmation: **"NFT listed successfully on Magic Eden."**

TC13: Twitter Market Sentiment Analysis

Pre-condition:

- Twitter API is accessible.

Test Execution:

1. Call `analyzeMarketSentiment("Solana")`.
2. The system scrapes and analyzes tweets.

Expected Output:

- Returns sentiment score (`Positive`, `Neutral`, `Negative`).

TC14: Rug Check API Integration

Pre-condition:

- A new token `XYZ` is being analyzed.

Test Execution:

1. Call `checkRugPullRisk("XYZ")`.
2. The system queries RugCheck API.

Expected Output:

- Returns **risk level** (`Low`, `Medium`, `High`).

TC15: Social Media Project Analysis

Pre-condition:

- A token project has a social media profile.

Test Execution:

1. Call `analyzeSocialMedia("XYZ")`.
2. The system checks engagement, post frequency, and history.

Expected Output:

- Returns a credibility score and flags suspicious activity. Segundo Wang et al. (2022), os algoritmos de consenso... Griffain, n.d.

Economic Impact

The convergence of decentralized finance (DeFi) and artificial intelligence (AI), often referred to as DefAI, represents a significant paradigm shift in the architecture and operation of blockchain-based financial systems. This synergy introduces intelligent, data-driven automation into domains traditionally governed by human discretion and opaque practices. Within this evolving landscape,

SWQuery emerges as a critical infrastructural component, offering a comprehensive suite of tools for extracting, analyzing, and responding to blockchain data in real time. Its integration into the Solana ecosystem contributes not only to technological innovation but also to meaningful economic advancement.

Enhancing Market Transparency and Liquidity

A central economic challenge in decentralized markets is the prevalence of information asymmetry, wherein only technically adept users or entities with privileged data access can make informed decisions. SWQuery directly addresses this issue by democratizing access to key blockchain insights through its developer-friendly SDK. Functions such as `getTrendingTokens`, `searchTokenByName`, and `analyzeRugPullRisk` empower users to identify emergent opportunities and evaluate asset credibility with reduced technical overhead.

This increased accessibility fosters broader participation in token markets, which, in turn, improves market liquidity and efficiency. Moreover, the integration of real-time WebSocket subscriptions for account and token activity allows applications to react dynamically to network events, facilitating the construction of predictive models, algorithmic trading strategies, and automated risk management frameworks. Such capabilities reduce latency in capital deployment and support a more fluid and responsive financial environment.

Stimulating Innovation through Composability

SWQuery also serves as a catalyst for innovation through its composable infrastructure. By abstracting the complexities of on-chain data interaction, it allows developers to build layered applications without duplicating data-fetching logic or indexing infrastructure. In this regard, SWQuery functions as a middleware primitive—enabling the rapid prototyping and deployment of intelligent decentralized applications (dApps), autonomous investment vehicles, and adaptive governance protocols.

This modularity enhances innovation velocity and lowers entry barriers for new projects. As more applications integrate SWQuery's SDK, a recursive feedback loop is established: each additional integration not only validates the utility of SWQuery but also expands its influence as a foundational data layer within the DefAI ecosystem. This recursive composability is a hallmark of economically robust decentralized platforms.

Promoting Security and Stability

In addition to economic expansion, SWQuery contributes to systemic risk mitigation. The `analyzeRugPullRisk` functionality exemplifies the practical fusion of AI heuristics with blockchain transparency to identify potentially fraudulent or high-risk assets. By offering developers and users a standardized tool for assessing token legitimacy, SWQuery enhances the integrity of market interactions

and fosters safer investment environments.

Such preventative capabilities are essential for the long-term stability of decentralized economies, as they reduce susceptibility to manipulative schemes and speculative bubbles. Increased user confidence translates into higher levels of sustainable participation, reinforcing the economic base of the broader ecosystem.

Strategic Role in the DefAI Economy

From a macroeconomic perspective, SWQuery aligns with the long-term objectives of the DefAI movement: to build intelligent, decentralized, and equitable financial systems. As DeFi protocols scale to manage increasingly complex portfolios and governance structures, the role of data-driven insight becomes critical. SWQuery's provision of accurate, real-time, and composable data feeds positions it as a core infrastructural component for future financial autonomy.

In summary, SWQuery's contribution to the DefAI ecosystem transcends technical utility. It supports economic democratization, stimulates innovation, promotes security, and enhances systemic resilience. By enabling developers and users to navigate and interact with blockchain data more effectively, SWQuery lays the groundwork for a more transparent, efficient, and inclusive decentralized economy.

Conclusion

The emergence of SWQuery within the DefAI ecosystem marks a pivotal advancement in how developers and users interface with blockchain data. As decentralized finance continues to evolve in complexity and scale, the demand for intelligent, composable, and reliable data infrastructure has never been more critical. This whitepaper has detailed SWQuery's functional architecture, technical benchmarks, and its operational dynamics through sequence diagrams and architectural decisions, while also situating the project within a broader market context.

By providing an SDK that abstracts the intricacies of interacting with the Solana blockchain, SWQuery significantly reduces the barrier to entry for decentralized application development. Its modular functions—ranging from asset querying and token analytics to real-time WebSocket subscriptions—equip developers with the tools needed to build adaptive and data-driven financial protocols. Moreover, the system's integration with AI-powered modules such as rug pull risk analysis demonstrates a forward-looking commitment to enhancing both usability and security in DeFi environments.

SWQuery's value proposition is further reinforced through its economic impact, enabling greater market transparency, fostering composability, and supporting the development of secure and sustainable financial ecosystems. It not only

empowers the current wave of developers but also lays the foundation for future innovation within DefAI.

In closing, SWQuery positions itself as more than a utility—it is an enabling framework for a decentralized, intelligent financial future. As adoption grows and integration deepens, SWQuery is poised to become a cornerstone of the DefAI movement, catalyzing the next generation of on-chain intelligence and decentralized finance applications.

References

Griffain. (n.d.). Griffain official website [Accessed on: 28 Feb. 2025].

Neur. (n.d.). Neur official website. all rights reserved [Accessed on: 28 Feb. 2025].

Rodriguez, F., & Thompson, E. (2024). Quantifying community engagement in blockchain projects: A framework for analyzing social media metrics and holder demographics. *Blockchain and Social Computing*, *6*(1), 45–63.

Strawberry. (n.d.). Use strawberry official website [Accessed on: 28 Feb. 2025].

SWQuery. (n.d.). Swquery official website [Accessed on: 28 Feb. 2025].

Wang, L., Shen, X., Li, J., Shao, W., & Yang, Y. (2022). Comparative analysis of blockchain consensus algorithms: A comprehensive benchmarking study of solana, ethereum, and alternative l1 platforms. *Journal of Blockchain Research*, *15*(3), 178–196.