Master's Degree in IT Industrial Engineering

TFM - Master Thesis

Development of a Multi-Agent, LLM-Driven System to Enhance Human-Machine Interaction: Integrating DSPy with Modular Agentic Strategies and Logical Reasoning Layers for the Autonomous Generation of Smart Contracts

— A collaboration with Sony R&D Lab Brussels —

Author - Yago Mendoza
Advisor - Antonio Calomarde
Supervisors - Hugo Embrechts,
Kasper De Blieck and
Michele Minelli

June 2024

Abstract

Recent advances in hardware capabilities have significantly accelerated the development of neural networks, leading to the emergence of Large Language Models (LLMs). Originally designed for text generation, these models have evolved to perform an increasingly broad spectrum of tasks, demonstrating capabilities that extend beyond early predictions based on their sole auto-regressive nature. During my internship at Sony's R&D Center Brussels Laboratory, I studied the relevance of these models for inclusion in a broader project focused on fan engagement in Web3, integral to the Loyalty 3.0 initiative, operating on the Ethereum Virtual Machine (EVM) to streamline the deployment of decentralized protocols.

The module developed in this thesis autonomously generates these protocols by interpreting natural language inputs from non-expert users, showcasing state-of-the-art language models' capabilities in translating early propositions into advanced source code. The system incorporates a Human-Machine Interaction Module, syntax generation with specialized layers, and a novel Survey Augmented Generation for Validation (SuAV) technique for alignment assessment. This system integrates DSPy for declarative syntax prompting, custom modular strategies for enhanced flexibility, and LangChain sequence patterns to ensure robust and accurate protocol generation. A comprehensive study comparing various techniques revealed the effectiveness of a hybrid model with conversational agents, particularly for smaller language models. Yet, the research highlights the complexities of optimizing LLMs for specialized tasks, emphasizing the need for nuanced approaches that balance model capabilities with task-specific requirements.

To gain a comprehensive understanding of the overall implications and contextual relevance of the material presented in this work, it is advisable to prioritize reading the conclusions, followed by the epilogue. These sections provide valuable insights beyond the numerical results of the thesis, offering a broader perspective on the study's significance and impact.

Keywords: Large Language Models (LLMs), Smart Contracts, Ethereum Virtual Machine (EVM), Fan Engagement Platform (FEP), Autonomous Protocol Generation, Natural Language Processing (NLP), Loyalty Programs, Web3, Blockchain Technology.

Acknowledgements

I would like to express my most sincere gratitude to Hugo Embrechts, for hosting me in the group at the Sony Brussels Lab and for ensuring my progress was steady and purposeful, keeping my path both challenging and clear.

To my colleagues at the Sony Group: Kasper Da Blieck for being a partner in thought, Michele Minelli for unwavering support, Dimitri Torfs for providing insights, and Sabina Orazem for being the thread joining all pieces.

I cherish the thought of future collaborations with them.

I would also like to thank Antonio Calomarde at ETSEIB, whose intervention was the greatest trust, allowing me to explore with autonomy and cultivate my own academic insights.

To conclude, discussions I had with my friend Rafael Mesa on the latest research and trends profoundly enriched this project. His contagious passion, both then and now, shapes my career and life.

Contents

Chapter 1

Introduction

1.1 Onboarding and Exploratory Discussions

The content of this thesis has been developed as part of a project at Sony, aligning with their ongoing research into new AI technologies. Sony has provided essential support and expertise when necessary, but has not directed the overall content of this work. It should be noted that the theories formulated, the methodologies chosen, and the findings and conclusions presented are the author's own, with Sony's input being limited to technical advice in specialized areas and periodic progress reviews. This approach stems from the nature of my role within a broader initiative, which afforded me the space to engage my competencies and widen my expertise alongside a dedicated team.

Motivation: In recent years, while pursuing my studies in Industrial Engineering, I have actively kept pace with the evolution of AI, exploring literature with a a keen focus on Deep Learning, Knowledge Representation, and Transformed-Based Models for NLP applications. At the same time, I've kept a watchful eye on the shifting sands of industry trends, eagerly getting my hands on every emerging library and framework I could—from OpenCV and PyTorch to the more recent additions like LangChain and DSPy. Hence, the genesis of the system developed in this thesis can be traced back to the blend of my long-standing passion for Computer Science, which has grown and evolved over the course of time, and Sony's ability to recognise and harness this passion as a valuable asset for their projects.

Onboarding: The R&D Center Brussels Laboratory hosts teams specialising in Ethereum blockchain, conducting penetration testing and analytics, while collaborating closely with Tokyo headquarters on projects ranging from effective computing and medical IoT to privacy, music, and digital rights management. During the employment interview, Michele Minnelli — a seasoned expert on the Blockchain team and my mentor — shared that there was recently a heightened eagerness within the company to explore the potential of Deep Learning and Generative AI. Acknowledging the potential of these technologies, and recognising my familiarity with blockchain systems, he expressed certainty that an opportunity for implementation would eventually be clear. This aligned perfectly with my own ambitions to extend my academic research into practical, corporate applications, hinting at the dual possibility of developing my thesis while participating in product development.

Project Frame: Just a few weeks into my tenure, I was introduced to the project on which I was going to work for the remainder of my time, not cited here in great detail due to confidentiality reasons. For the purpose of this initial chapter, it suffices to comprehend that my task involved developing a system designed for software protocols automated scripting —namely, Smart Contracts— on Ethereum's programmable blockchain, as a component of a bigger initiative set to innovate fan engagement. We will dive into the project definition and explore these related concepts in subsequent chapters.

AI Suitability: Given the project's requirement for dynamic and context-aware code generation, it became evident that leveraging my background on NLP would stand as the most propitious methodology. Large Language Models (LLMs) represent a significant advancement in code generation tasks, surpassing the capabilities of traditional rule-based systems. Unlike these systems, which are limited by predefined rules and lack flexibility, LLMs utilise advanced neural network architectures to understand and interpret context, syntax, and semantics. This understanding enables LLMs to produce code that is not only coherent but also contextually appropriate, reflecting a deep adaptability and creativity. Rule-based systems, in contrast, struggle with novel or unforeseen scenarios due to their rigid frameworks. LLMs, however, with their ability to generate human-like text, offer a more dynamic and scalable solution. This makes them particularly suited for tasks that require both technical precision and the capacity to innovate or respond to new challenges organically.

First Approach: In the early stages of developing, I drafted a high-level framework aimed at fostering a decentralized computation environment, taking inspiration from the [Age24] project and the [Bro+20] and [Wu+23] papers. Both works envisioned a network where AI agents [1], powered by generative AI and Large Language Models (LLMs), could autonomously coordinate and execute tasks based on the collective needs of the network by receiving rewards, with minimal human oversight. Agents were designed to interact within a peer-to-peer system, dynamically crafting and refining code to establish a robust infrastructure for distributed computational tasks. It's evident that this approach extended beyond merely refining smart contract generation using transformer models. To me, it represented a significant challenge to master concepts such as asynchronous programming, API gateways, and systems-level memory safety, integrating both Python and Rust within the workspace. The focus was not just on AI inference and generation but also on embedding these as medium priority components within the broader framework.

Reevaluation: Despite the central generative focus of the first approach, during a planning meeting with Sony on February 22, driven by Sony's understanding of the evolving needs and likely trajectory of the aforementioned project, the feasibility of constructing such an extensive infrastructure was reassessed. The boundaries of the system must be defined by the project's progress, necessitating alignment rather than leading the development's contour conditions. Hence, it was decided that this approach exceeded the immediate needs, and several aspects of the general infrastructure, such as protocol handling and decentralisation, were pared back. To better synchronize with the project timeline and seamlessly integrate the tool design within the overall framework, a decision was made to narrow the scope to the generation capabilities of what was initially conceived as a single node. This recalibration shifted the focus toward configuring a controlled and reliable modular system, not only engineered to elevate capabilities through the judicious use of AI but also crafted to smoothly integrate with human interactions, thus enabling users, particularly those lacking in coding expertise, to take an active role in creating smart contracts. This was deemed relevant under the following considerations:

- With network interactions off the table, the imperative for a dynamic interface that facilitates user engagement in smart contract creation comes sharply into focus. This refined strategy calls for an initial module that tackles the requirement design phase, strategically positioned at the outset of the Waterfall Model for the Software Development Lifecycle (SDLC) [Roy70]. Subsequent agentic modules are envisioned to build upon this foundation, advancing through the generation and rigorous assessment of code by applying diverse reasoning methodologies and implementations [PM14].
- It's logical to assume that integrating a project into a specific use case naturally introduces more constraints. These constraints often shape the project's scope, functionality, and design,

[1] Throughout this discourse, it's crucial not to confuse the initial definition of an 'agent' as a node in a decentralized network based on ongoing projects with the post-meeting definition, which now refers to various modules interacting within the system. Although the final agreed-upon design has the potential to evolve into an agent within a larger network, such a transition post-project completion would require substantial refactoring of its capabilities, dismissing many others, and enabling connectivity ports and routing protocols between agents, which are far removed from the final agreed-upon purpose for the work.

driven by the necessity to conform to operational standards and existing technological frameworks. Consequently, there's a risk that high standards for performance and usability might overshadow exploration, potentially compromising academic research and innovation. Yet, while this synergy suggests that a narrow focus on applicability could diminish the project's capacity for innovation, such an outcome is not inevitable in all circumstances.

- Mirroring the challenges encountered with the network setup of the initial design, tailoring the behavior of the LLM to meet such specific demands continues to offer substantial educational value. Presently, LLMs frequently grapple with issues of strict compliance and controllability, which presents hurdles in adhering to standards while advancing the frontiers of AI's capabilities [Ros23]. Addressing these challenges necessitates a sophisticated approach to both AI training and system architecture, ensuring that the models not only fulfill the rigorous demands of blockchain applications but also push the envelope in AI development.

1.2 Objectives and Success Criteria

Success criteria can fundamentally be distilled into the concept of moving in a relevant direction, a decision fundamentally shaped by two driving forces: organisational interests and personal learning aspirations. Nonetheless, the emphasis in this section is on the first one, as the outlined exploratory discussions already made it clear that ample learning opportunities will be provided.

From an organisational perspective, the value propositions of this project for the company are multifaceted:

- By exploring the potential of Large Language Models (LLMs) and related topics discussed in this thesis, the project equips the company with specialized knowledge and insights into an emergent and largely untapped field. This initiative not only identifies new opportunities but also enhances strategic decision-making. The emphasis on LLMs aligns with current industry trends and positions the company to capitalize on future innovations, effectively merging academic research with practical applications.

- By integrating both fundamental and applied research within the specific engineering area of this project, we lay the groundwork for future technological advancements. The knowledge and systems developed are intended to serve as a blueprint that can be adapted and expanded, thereby fostering a cycle of continuous improvement and adaptation within the project.

- By fostering an environment that values continuous learning and knowledge sharing, the contribution to the project aligns with the company's strategic goal of staying at the forefront of technological innovation, encouraging an environment of ongoing education and improvement.

My experience has taught me that experimentation is a cornerstone of product development, especially in fields as dynamic as AI. This iterative process of trial and error does not only fuel the project's innovative essence but also enriches my academic and professional growth. By adopting a fast-paced, learn-through-failure approach and integrating insights from the latest academic research, I continuously ignite new ideas and refine existing processes.

This experimentation is aimed at creating a module tailored for Sony's use and constitutes a tangible outcome in its own right, underscoring the importance of adaptability and proactive exploration in crafting tools that are both innovative and aligned with corporate needs. As such, initial chapters of this thesis encapsulate a broad array of concepts and insights gathered during development, yet their apparent influence on the final product is not as substantial as that of subsequent chapters, where it becomes much more pronounced. These concentrate more on the methodology and streamlined development of the tool, focusing on elucidating the workings of the final, refined components and restricting insights to what has proven useful.

The iterative development of the project serves as a practical outcome, beginning with the Minimum Viable Product (MVP) and evolving through stages of increasing complexity. This process is not just a straightforward progression but a nuanced pathway that evolves in response to the project's changing demands. It supports dual objectives: firstly, it establishes the essential functionalities that underpin the thesis's goals, providing a base for further enhancements. As the project advances, this foundation allows for the incorporation of additional, more sophisticated features, each iteration not only improving the system's capabilities but also its relevance to Sony's technological environment. This ongoing refinement helps ensure the project remains aligned with Sony's strategic goals and can adapt to extend its impact within the company. Open communication during this process is crucial for aligning the project with corporate objectives and integrating it successfully.

Details regarding the specific functionalities of the tool and its development will be elaborated in subsequent sections, once the scope moves beyond the initial project configuration and overarching concepts to focus on the particular developmental aspects of the tool. The discussions here are intentionally broad, reflecting the need to align with corporate strategies and the inherently collaborative nature of my thesis work within this corporate setting.

1.3 Structure Overview

This thesis is divided into two parts.

Part I contains **4** chapters, each treating theoretical foundations, project planning, and initial development stages. Part II includes **3** chapters on the implementation, testing, and conclusions of the Multi-Agent LLM-Based Smart Contract Generation Module.

Chapter 2, "Theoretical Background" — This chapter dives into the foundational knowledge of Ethereum, smart contracts, and large language models. It explores the intricacies of the Ethereum Virtual Machine (EVM), a Turing-complete system that executes smart contracts. The concept of tokenization is examined, highlighting how digital assets are represented on the blockchain through standards like ERC-20 and ERC-721. The evolution of language models is traced, from early neural networks to transformer architectures, with a focus on their formal definition and the principles of self-attention mechanisms. The chapter concludes by discussing emerging AI trends in software development, such as code generation and automated testing, setting the stage for the project's innovative approach.

Chapter 3, "Fan Engagement Platforms" — This chapter examines the evolving landscape of loyalty programs and their transition into the Web3 era. It discusses the "loyalty paradox," where traditional programs often fail to engage customers effectively despite widespread adoption. The shift towards Loyalty 3.0 is explored, emphasizing how blockchain technology enables more transparent, engaging, and rewarding experiences for fans. The chapter analyzes third-party Software-as-a-Service (SaaS) platforms that facilitate fan engagement, comparing their features and limitations. It concludes by highlighting the transformative potential of blockchain in revolutionizing fan engagement strategies, particularly in areas such as tokenized rewards, exclusive digital collectibles, and community-driven initiatives that foster deeper connections between brands and their audiences.

Chapter 4, "Software Development Project Plan" — This chapter outlines the comprehensive strategy for developing the Multi-Agent LLM-Based Smart Contract Generation Module. It begins by establishing key assumptions, including the project's alignment with Sony's broader initiatives and the flexibility afforded for academic exploration. The chapter then dives into the system's features and functionalities, detailing the integrated-type system architecture that encompasses human-machine interaction, code generation, and assessment modules. A feasibility analysis is presented, incorporating workflow diagrams and key milestones to guide the development process. The tech stack is thoroughly

examined, highlighting the use of Python for its versatility and extensive library support, alongside tools like Poetry for dependency management and Streamlit for creating interactive web interfaces. The chapter also discusses the implementation of a dual-device GitHub workflow, emphasizing the importance of version control and collaborative development in a corporate setting.

Chapter 5, "Minimal Viable Product (MVP)" — This chapter explores the initial approach to developing the Minimum Viable Product, emphasizing its role as a foundation for contemplating configuration parameters and requirements rather than establishing a permanent architecture. It details the first implementation using FastAPI for asynchronous operations and the LangChain framework for streamlining interactions with Large Language Models (LLMs). The chapter then conducts an in-depth comparison of API endpoints versus local implementations, analyzing various models and providers such as OpenAI's GPT series, Meta's Llama 3, and offerings from Anthropic and Hugging Face. Special attention is given to frameworks like LlamaCPP and Ollama for local inference of LLMs, discussing their capabilities and limitations. The chapter concludes by introducing the LangChain Standard, explaining its role in simplifying the creation of LLM applications through modular abstractions. It also addresses the challenges encountered with LangChain, such as documentation issues and scalability concerns, which led to the exploration of alternative solutions like DSPy for better testing of reasoning layers within the project's use case.

Chapter 6, "Declarative Pipelines via Multi-Agent Compilation" — This chapter introduces DSPy as an innovative solution to address the challenges encountered in prompt engineering. It begins by outlining the limitations of traditional methods, such as the brittleness of hard-coded prompts and the lack of scalability in manual prompt engineering. The chapter then dives into the core components of DSPy: signatures, modules, and teleprompters. Signatures are explained as high-level, natural language typed function declarations that abstract input/output behaviors. Modules are described as declarative components that replace conventional hand-prompting techniques, capable of being flexibly composed into various pipeline configurations. The role of teleprompters in optimizing these modules to maximize specified metrics is elaborated upon. The chapter also discusses the concept of multi-agent reasoning systems, contrasting the project's Distributed Problem Solving (DPS) approach with more flexible Multi-Agent Systems (MAS). It explains how the rigid structure of DPS, characterized by predefined communication strategies and fixed protocols among computing entities, provides a predictable and manageable framework ideal for tasks requiring precision and stability, such as code generation.

Chapter 7, "MaLB-SC Generation Module" — This chapter provides a comprehensive overview of the Multi-Agent LLM-Based Smart Contract Generation Module (MALB-SC), detailing its key components and innovative approaches. The system's workflow encompasses user interaction, syntax generation, and alignment processes, including a Human-Machine Interaction Module that utilizes coverage sets and agent coalitions. The chapter discusses the integration of the Solidity compiler (solc) in Python and the use of various linting tools for code quality and security. A significant focus is placed on SUAV (Survey Augmented Generation for Validation), a novel technique for alignment assessment that employs a dual-agent framework with inquisitor and scrutineers. The chapter also presents a syntax generation study comparing various prompting techniques and reasoning methods, including meta-prompting, chain-of-thought reasoning, and a hybrid model with conversational agents. Key findings highlight the challenges of in-context learning for complex code generation and demonstrate the effectiveness of a hybrid model with specialized agents, showing significant improvements, especially for smaller language models like Llama-3-7b. The chapter concludes by emphasizing the complexity of optimizing language models for specialized tasks like smart contract generation, underscoring the need for nuanced approaches that consider the interplay between model capabilities and task-specific requirements, and setting the stage for future research in this area.

Chapter 2

Theoretical Background

2.1 Ethereum Virtual Machine (EVM)

> *"The internet was complicated in 1995, and the blockchain, the consumer blockchain, is complicated in 2022. And so, we take education extremely seriously here."*
>
> – Gary Vaynerchuk, *speaking at the inaugural Web3 Demo Day in 2022*

2.1.1 Distributed Ledgers Foundations

Decentralization, as a concept in the field of computer science, centers around distributing processes and data across multiple locations to enhance system resilience and efficiency. The history of decentralization can be traced back to the development of distributed computing and the exploration of peer-to-peer network architectures in the 1970s and 1980s. During this era, foundational texts such as 'Distributed Systems: Principles and Paradigms' [TS06] provided extensive insights into mechanisms that manage distributed networks effectively. Significant advancements in this period included the development of protocols for distributed data handling and consensus mechanisms. One notable example is the Domain Name System (DNS), which revolutionised internet address resolution by distributing the task of mapping domain names to IP addresses across a vast, globally interconnected network of servers. Unlike centralised address books, DNS distributes the responsibility of name-to-IP address mappings to a global network of servers, enhancing the system's ability to scale and resist failures. This decentralisation ensures that a single point of failure will not disrupt the entire network, thereby maintaining internet stability and reliability. The late 1990s and early 2000s saw further application of decentralisation principles with the advent of peer-to-peer file-sharing technologies, such as BitTorrent. BitTorrent diverges from traditional file download methodologies by enabling users to receive pieces of files from multiple sources simultaneously, rather than relying on a single server. This method not only speeds up file transfer rates but also distributes the load across many nodes, reducing reliance on any single source and increasing resilience against censorship or service disruptions.

It was not until 2008 that a groundbreaking innovation emerged: a peer-to-peer electronic system organizing transactions into data blocks to enhance transparency and security [Nak08]. Each block is cryptographically linked to its predecessor, forming an immutable chain that renders unauthorized changes computationally infeasible. The blockchain employs a Byzantine consensus mechanism to ensure all participants maintain a consistent copy of the ledger, which records transactions chronologically.

Network participants, known as validators or miners, compete in solving cryptographic puzzles to secure the network and prevent unauthorized modifications to confirmed blocks. The first to solve the puzzle gains the right to add a new block, earning a reward in the native cryptocurrency. This incentive

system promotes the allocation of computational resources and bolsters the blockchain's security and operational stability with each addition. This core innovation, termed Blockchain, with Bitcoin at its center, functions as a decentralized ledger that records transactions across a network of computers, enabling secure and transparent peer-to-peer transactions without intermediaries.

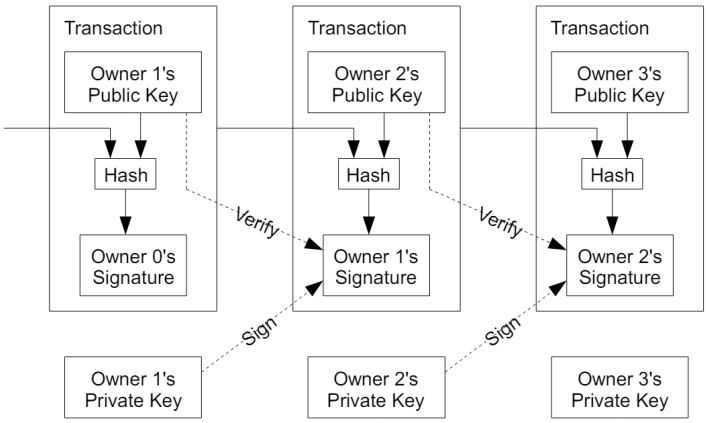


Figure 2.1: Diagram illustrating a chain of transactions using public key cryptography [Nak08]. Each transaction contains the previous owner's signature and the new owner's public key. The process involves hashing the transaction data and verifying the previous owner's signature with their public key. The new owner then signs the transaction with their private key. This chain of signatures ensures the integrity and authenticity of each transaction, linking each step to the previous one securely.

From a computer science perspective — and this will inform our comparison of Ethereum in the next section — we can think of Bitcoin as a distributed consensus state machine, where transactions cause a global state transition, altering the ownership of coins [Ant17]. The rules of consensus ensure that all participants eventually converge on a common state of the system after several blocks are mined, securing the network's reliability and integrity.

We consider three[1] key downsides of blockchain and bitcoin:

- **Scalability and Efficiency Challenges:** Despite its formidable cryptographic security, which ensures superior data integrity standards, Bitcoin grapples with scalability challenges affecting transaction throughput. These issues stem from its decentralized architecture, requiring comprehensive transaction verification by multiple nodes to maintain consensus and enhance security. This extensive verification process inherently increases latency [Nar+16].
- **Resource Utilization Trade-offs:** In line with the previous point, Bitcoin's mechanism of proof of work presents significant resource utilization trade-offs. The intensive computational requirements for mining contribute to high energy consumption and environmental concerns, highlighting the need for more efficient consensus mechanisms in blockchain technology [SKG19].

[1] Although it differs from the others by not setting up a technical comparison with the next section, which is the narrative focus of this chapter, understanding Bitcoin's economic purpose remains as a fourth relevant downside for understanding the broader context. Bitcoin fundamentally serves as a store of value. Yet, its volatility is driven by factors such as scarcity, speculative trading, regulatory uncertainty, and evolving market dynamics. These elements undermine its original value proposition, making its future uncertain. The reliance on a virtual world necessitates a leap of faith that the intangible asset will retain its integrity and significance. This ambiguity blurs the line between faith and understanding, causing divergent views: some see Bitcoin as a potential replacement or enhancement to the Bretton Woods system and a future monetary standard due to its technical advancements, while others are attracted to it for the potential of rapid financial gains, exploiting its inherent volatility. Ultimately, Bitcoin's trajectory remains shrouded in captivating uncertainty.

- **Nature of Money and Blockchain Suitability:** The adoption of blockchain technology is largely confined to areas where stringent security and immutable records are vital. This is why, with money being a static database for resource allocation, since its emergence, blockchain technology, with Bitcoin at its forefront, has predominantly functioned as a transformative monetary system, making it ideally suited for Distributed Ledger Technology (DLT), where transaction speed is not a critical concern. This has brought up its unique value primarily in financial environments rather than broader applications, to the point that most academics and industry experts agree that absent its integral function in financial systems, blockchain might simply be viewed as an inefficient database [BC14].

While Bitcoin pioneered the concept of decentralized digital currency, its limitations in scalability, resource efficiency, and application scope became apparent over time. Recognizing these challenges, a few years later, a significant development emerged with the introduction of Ethereum by a young innovator named Vitalik Buterin. This new blockchain platform promised to expand the capabilities of blockchain technology far beyond monetary applications. Indeed, Ethereum is the blockchain platform related to the work discussed in this study. The subsequent chapter will dive into how Ethereum aims to address these limitations, offering a foundation for the advanced applications we explore.

2.1.2 Ethereum Ecosystem and Smart Contracts

From a computer science perspective, Ethereum is *an open-source, deterministic but practically unbounded state machine, consisting of a globally accessible singleton state and a virtual machine that applies changes to that state* [AW18] by executing code of arbitrary and unbounded complexity. In other words, Ethereum is akin to a global operating system, where code of any complexity can be deployed and executed in a secure, decentralized manner. It was proposed by Vitalik Buterin in late 2013 as a flexible, scriptable approach for creating a more versatile platform that could support a wider variety of decentralized applications beyond simple currency transactions, extending Bitcoin and Mastercoin's[2] capabilities.

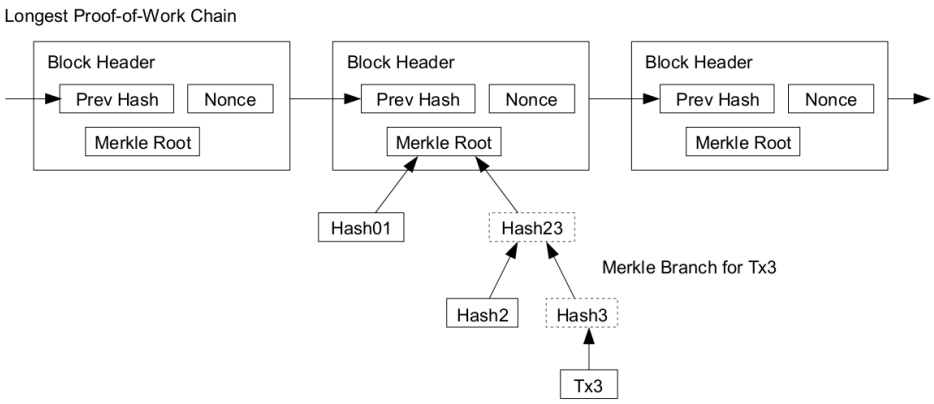


Figure 2.2: Diagram of the Longest Proof-of-Work Chain. Each block contains a Block Header with a previous hash, nonce, and Merkle root. The previous hash links to the prior block, ensuring chain integrity. The Merkle root summarizes all transactions in the block. The diagram also illustrates a Merkle branch for transaction Tx3, showing how transactions are hashed together (Hash3, Hash23, Hash01) to form the Merkle root. This structure enables efficient and secure verification of transactions within a block.

[2] MasterCoins, an early cryptocurrency built on the Bitcoin blockchain, aimed to add features like smart contracts and decentralized exchanges. It ultimately failed because Bitcoin's infrastructure wasn't suited for these advanced functions, and it faced competition from more capable platforms like Ethereum.

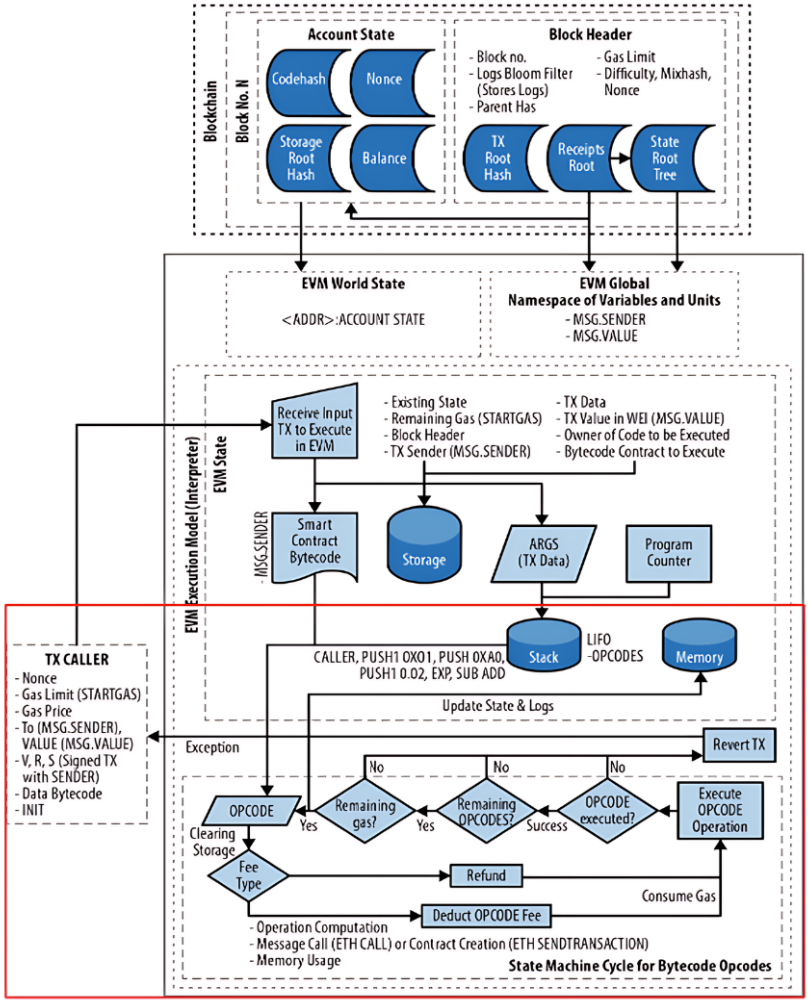


Figure 2.3: Diagram of the Ethereum Virtual Machine (EVM) architecture [AW18]. The top section illustrates the blockchain structure with each block's header and account state, including metadata like block number, gas limit, and root hashes. The EVM World State maps addresses to account states, while the EVM Global namespace holds variables like MSG.SENDER and MSG.VALUE. Input transactions are executed in the EVM State, interacting with components such as storage, stack, memory, and the program counter. The State Machine Cycle for Bytecode Opcodes checks for exceptions, gas, and opcode execution, updating state and logs or reverting transactions upon failure. Fees are deducted for operations, calls, contract creation, and memory usage. This diagram provides an overview of EVM's role in executing smart contracts on Ethereum.

While Bitcoin's Script language is intentionally constrained to simple true/false evaluation of spending conditions to track the state of currency ownership, Ethereum's language, namely Solidity, is Turing complete, meaning that Ethereum can straightforwardly function as a general-purpose computer, employing consensus mechanisms to store, synchronize and track state transitions to the system's state. The general-purpose data store is designed to hold any data expressible as a key-value tuple within the codebase infrastructure, providing a temporary yet crucial storage area for the running state of applications. In the context of Ethereum, this data storage does not only act temporarily but is continually updated and preserved across the blockchain network, maintaining a permanent and immutable record of all interactions and changes.

This concept of a persistent general-purpose data store, also referred to as the world state, is central to understanding Ethereum. In traditional computing, data is stored in Random Access Memory

(RAM), allowing quick access and manipulation by the computer's processor. Similarly, Ethereum's world state acts as a dynamic data storage model, enabling the system to operate across multiple states [Dan17]. This virtual-machine-like functionality allows for the associative array to be stored in the blockchain itself, in the collective storage space set by the deployment of the ensemble of smart contracts. Central to the focus of this thesis, these smart contracts are integral to Ethereum's decentralized computing infrastructure and are the primary subject of the generation we will be conducting in the next chapters. This is why a clear and precise definition is needed.

A smart contract, already conceptually laid out by Nick Szabo [Sza96] in the early 1990s, was defined as a set of promises, specified in digital form, including protocols within which the parties perform on the other promises. From a computer science perspective, it can be viewed as an immutable computer program on the Ethereum World Computer that runs automatically and deterministically when certain conditions are met, i.e., when they are called by a transaction[3]. When that happens, nodes within the Ethereum network identify the contract's address on their internal blockchain representation, retrieve the program's bytecode through the codeHash attribute, and run it through the Ethereum Virtual Machine (EVM)[4]. The virtual machine processes a series of opcodes (operation codes) to perform contract-specified actions, from arithmetic to control flows, directly influencing contract storage and transactions on the blockchain and ultimately altering the blockchain's state. These operations are performed utilizing a stack-based architecture that integrates both the incoming transaction arguments and the variables from the world state, which are stored in a hierarchical tree-like data structure — known as the Merkle Patricia Tree [AW18]. All account states are hashed and compactly aggregated into a single root, ensuring efficient verification and immutable record-keeping.

In other words, drawing an extended analogy to a computer, Ethereum functions like a decentralized operating system that loads the program into its memory — akin to loading an application on a traditional computer — and then processes it through its state machine, executing commands and managing data much like a computer executes software using its central processing unit (CPU) and memory resources. These transactions trigger predefined functions in the smart contract, prompting updates to its state variables, which are then securely recorded on the blockchain to maintain a consistent state across the network.

On a more technical level, the creation and deployment of a smart contract on the Ethereum blockchain starts with writing the contract in a language like Solidity[5] and compiling it into bytecode. This bytecode, drawing parallels to x86_64 architecture, encapsulates the program logic and is included in a deployment transaction and broadcast across the network. Once validated by blockchain integrity mechanisms, this representation is stored at a unique address created during the deployment process. The address ensures the bytecode is replicated across all Ethereum nodes, thus making the smart contract operational within the decentralized infrastructure.

The functions being compiled into bytecode through the Solidity Compiler (SOLC) encapsulate reusable pieces of code that perform specific tasks within a smart contract. They can have different visibility levels such as public, private, internal, or external, determining who can access them and from where, crucial being the public nature of the platform. They can also address multiple contexts like call, block, or transaction for streamlined interaction with external entities.

The functions [SKH19], once compiled into bytecode via the Solidity Compiler (SOLC), embody modular segments of code designed to execute distinct operations within a smart contract. These functions are characterized by varied visibility levels—public, private, internal, and external—each key in defining the scope and accessibility within the ambit of an expansive public platform. Furthermore,

[3]Contracts can also be triggered by other contracts, forming chains of execution. However, the first contract in such a chain is always triggered by a transaction from an Externally Owned Account (EOA). Subsequent contracts may remain dormant until activated by further transactions.

[4]It's worth mentioning that the Ethereum Virtual Machine (EVM) operates without built-in scheduling, relying instead on external mechanisms for the order of execution. Clients verify and sequence transactions, managing the flow of execution outside the EVM itself. This results in a sequential, event-driven model of operation, similar to the single-threaded nature of JavaScript.

[5]Developed by Dr. Gavin Wood, Solidity is imperative Ethereum's primary programming language for writing smart contracts, offering robust functionality and ease of use.

the architecture incorporates a sophisticated context differentiation mechanism designed for handling calls, blocks, or transactions. In addition, Solidity provides a robust framework supporting modifiers, which enforce specific preconditions on functions; inheritance, which fosters code reusability by enabling new contracts to adopt properties from existing ones; and events, which serve to broadcast notifications, thereby enhancing the interactivity and responsiveness of the blockchain environment.

While Solidity is the most popular choice for Ethereum smart contracts, there are alternatives like LLL (a Lisp-like language), Vyper (influenced by Python), and Bamboo (similar to Erlang), each with its own design philosophy and syntax aimed at addressing specific needs and security considerations in the smart contract ecosystem. However, for this project, Solidity is chosen due to its robustness, wide adoption, and strong developer support, making it a suitable language for a comprehensive exploration of smart contract functionalities.

This overview of Ethereum's smart contract capabilities sufficiently addresses the essential technical elements and programming paradigms. With this foundation, we are well-prepared to proceed to the next chapter.

2.1.3 Tokenization and Asset Representation

Tokens serve as an additional abstraction within this system, ultimately providing a clear gateway to comprehending its utility and functionality. They can be viewed at an overview level as assets created on top of existing blockchain networks and can represent virtually anything, ranging from reputation points in an online platform, financial assets like a share in a company or even an ounce of gold or fiat currencies like USD. Entities with these characteristics share the common interest of utilising blockchain for ensuring secure, transparent, and decentralized recording of ownership and transactions.

When it comes to designing digital tokens, standardised protocols play a key role, with ERC-20 being a key example. "ERC" stands for "Ethereum Request for Comments," similar to "RFC" in internet protocol development. ERC-20 specifies essential functions such as "totalSupply", "balanceOf", "transfer", "approve", and "transferFrom", which define how tokens are issued, transferred, and managed within the Ethereum network. These can be seen in the Listing 2.1

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
function approve(address _spender, uint256 _value) public returns (bool success)
function allowance(address _owner, address _spender) public view returns (uint256 remaining)

event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

Listing 2.1: Entities delineating the protocol governing the ERC-20 standard functionalities are: *function name()*: retrieves the token name; *function symbol()*: retrieves the token symbol; *function decimals()*: returns the number of decimal places used; *function totalSupply()*: shows the total token supply; *function balanceOf()*: provides the balance of a specific address; *function transfer()*: transfers tokens to another address, returns success status; *function transferFrom()*: moves tokens from one address to another, given allowance, returns success status; *function approve()*: authorizes a spender to use a specific amount of tokens, returns success status; *function allowance()*: returns the remaining amount the spender is allowed to spend from the owner's balance; *event Transfer()*: emitted when tokens are transferred; *event Approval()*: emitted when approval for a spender is set. Arguments: *address owner*: the address from which the balance or allowance is checked; *address _spender*: the address authorized to spend tokens; *address _to*: the recipient address; *address _from*: the sender address; *uint256 _value*: the amount of tokens transferred or approved.

The importance of such a standard cannot be overstated, as it ensures that the coin is compatible with every wallet and exchange built to these specifications, providing a consistent framework that enhances interoperability and functionality across the network. Additionally, the standards facilitate the ability to transfer tokens, and send and receive them, while also allowing for tokens to be approved so that they can be spent by another on-chain third party.

The primary distinguishing factor between standards is the requirement of fungibility; this is essential for them to function as securities (ERC-20) or as non-fungible tokens, which serve as unique digital objects (ERC-721). In essence, this information and attributes are derived from the various constraints imposed during creation based on the ERC standard, linking the very existence of the token to the smart contract that created it. In this sense, the token and the smart contract can be considered synonymous. The contract, therefore, determines the nature of the aforementioned uniqueness or, in the case of ERC-20, the economic conditions that govern that currency.

There is no central entity; instead, the collective intelligence of computers, through their cryptographic emergent properties, creates a secure and reliable system, embodying a profound shift in how trust is established in the digital age.

The Ethereum Virtual Machine (EVM) provides the infrastructure necessary to deploy smart contracts that manage the issuance, transfer, and redemption of tokens in a secure, efficient, and scalable manner. However, this technology remains highly technical and complex, posing barriers to broader adoption. To bridge this gap, the development of user interfaces and experiences, such as wallets, dashboards for tracking rewards, and marketplaces for redeeming points, is essential.

These enhancements in interoperability represent some of today's most significant challenges, paving the way for contributions from Sony and other major corporations in the industry. In upcoming chapters, the opportunities that this industry offers are examined through the strategic implementation of Sony's project, particularly through the development of the system at the thesis core.

2.2 Large Language Models (LLMs)

"Any sufficiently advanced technology is indistinguishable from magic."

– Arthur C. Clarke, *Hazards of Prophecy: The Failure of Imagination, 1962*

2.2.1 Deep Learning for NLP

While in the 1950s, Noam Chomsky's transformational-generative grammar revolutionised syntactic structures [Cho56], enabling more efficient language analysis, the Perceptron silently came onto the scene in 1957 by Frank Rosenblatt [Ros58]. Initially serving only as a linear separator, it did not yet form part of Natural Language Processing (NLP), yet it laid the groundwork for modern Deep Learning.

The next significant advancement came in the early 1970s with mathematical progress in semantic theory, such as the CYK Algorithm and Earley's parser [You67], which were key in the development of programming language compilers. Meanwhile, the Perceptron had been upgraded with hidden layers shortly after its advent, and by 1980, while IBM continued to explore probabilistic NLP methods like Hidden Markob Models (HMMs) for speech recognition [Rab89], Geoff Hinton popularized an algorithm for training NNs, known as Backpropagation [RHW86], converting them into powerful pattern learners.

This leap sparked interest in the field and, in 1982, the development of Recurrent Neural Networks (RNNs) [Hop82] marked a significant advancement in processing sequential data, enabling the prediction of conditional probabilities of a sequence, such as $P(w_i \mid w_1, \ldots, w_{i-1})$ for words in text. Despite their initial promise, RNNs struggled with the vanishing gradient problem and had limited capacity for maintaining long-term dependencies due to rapidly decaying memory of past inputs.

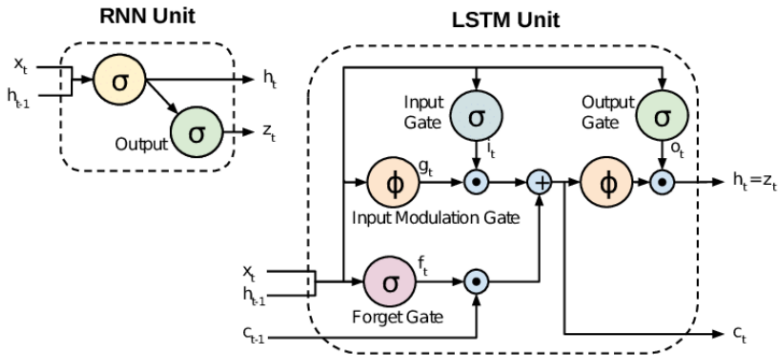


Figure 2.4: The RNN and LSTM architectures. The RNN unit (left) consists of a simple loop allowing information to persist. The LSTM unit (right) improves on RNN by adding gates: the input gate controls the flow of input; the forget gate determines what information to discard; the input modulation gate processes new input; and the output gate controls the flow of the output. These gates enable the LSTM to maintain long-term dependencies, mitigating the vanishing gradient problem typical in standard RNNs.

To address these issues, the Long Short-Term Memory (LSTM) architecture [Hoc97] was introduced in 1997, featuring memory cells with forget gates that better managed long data sequences and alleviated the gradient vanishing problem. However, the computational demands of training LSTMs, involving sequential weight updates for each unit per word, made them less effective for extremely long text sequences and still presented challenges in handling very long-term dependencies. To further refine the model's efficiency, Gated Recurrent Units (GRUs) [Cho+14] were later developed, simplifying the LSTM design by combining several gates into one, thus reducing computational complexity while retaining the ability to manage long-term dependencies effectively.

Despite these advancements, NNs were not yet perceived as revolutionary, even within the field of NLP, as the community remained focused on statistical methods and rule-based systems [Rud21]. However, the increasing momentum fuelled by the growing availability of computational resources and the advancement of GPUs in the early 2000s led to unprecedented scalability of Deep Learning.

ImageNet (2012) [KSH12], developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, revolutionized computer vision by proving the effectiveness of deep neural networks for large-scale image classification. It showed that trained networks could be repurposed for related tasks using transfer learning. This success inspired NLP researchers to adopt similar approaches, leading to the development of word embedding models like Word2Vec [Bak18], GloVe [PSM14], and eventually more complex language models.

Attention is all you need Everything changed in 2017 with the publication of Vaswani et al.'s "Attention is All You Need." [Vas+17]. This paper introduced self-attention mechanisms for processing text sequences simultaneously rather than sequentially, allowing for the capture of long-range dependencies with word-to-word relations and positional encodings. Self-attention mechanisms eliminated the need for naïve neural translation models such as RNNs and LSTMs in many NLP tasks, setting the stage for scalable sequence layer-wise transducers.

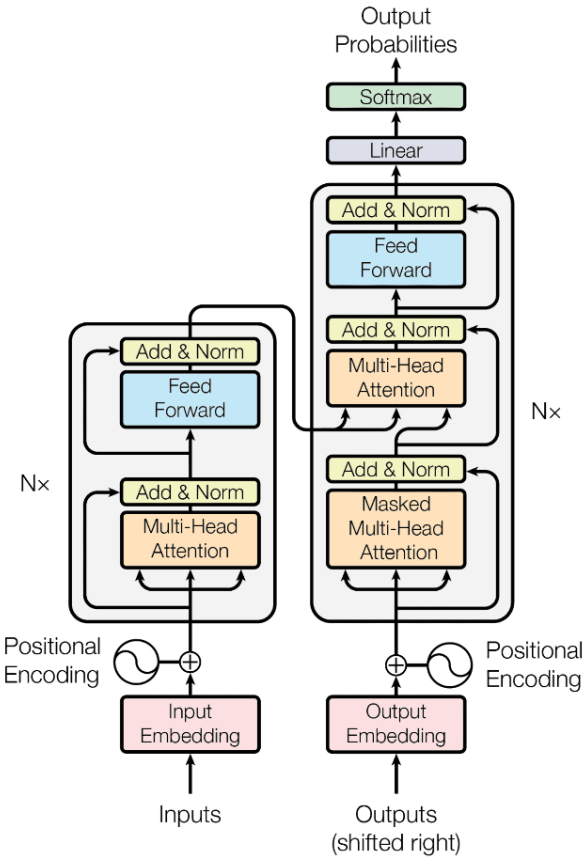


Figure 2.5: The transformer model architecture consists of an encoder (left) and decoder (right). The encoder processes input embeddings [Mik+13] with positional encoding, multi-head attention, and feed-forward layers to generate hidden states. The decoder uses these hidden states, combining them with shifted output embeddings, masked multi-head attention, and feed-forward layers to produce output probabilities. Add & Norm layers stabilize and optimize learning throughout. This structure allows for efficient handling of sequential data with attention mechanisms.

Transformers (see Figure 2.5) consist of an encoder (left) and a decoder (right) stack, each with multiple layers of self-attention and adaptable feed-forward neural networks.

The **encoder** pocesses the input sequence by converting it into high-dimensional vector embeddings that capture the contextual relationships between words. It uses self-attention mechanisms to weigh the importance of each word in relation to others across the entire sequence, allowing it to capture long-range dependencies efficiently. The encoder operates in parallel, enabling fast processing and scalability, and its layered structure refines the embeddings progressively, ensuring a rich and nuanced understanding of the input data.

The **decoder**, on the other hand, generates the output sequence by predicting one token at a time, using self-attention to consider all previously generated tokens and their contexts. It operates in parallel during training to enhance efficiency, but generates sequentially during inference to maintain coherence. The decoder's self-attention layers capture relationships within the output sequence, while encoder-decoder attention layers (when used) integrate information from the input sequence, ensuring the generated output is contextually relevant and accurate.

This architecture laid the foundation for a revolution in AI over the past 2-3 years, influencing the development of models like ViT [Dos+21] for computer vision, applying self-attention to image patches; SORA for video, leveraging attention mechanisms to capture temporal dependencies; and EvoFormer [Jum+21] for DNA, utilizing self-attention to understand molecular interactions. In electric signals, it can decode intricate patterns, such as neural impulses in brain-computer interfaces [Wol+02] or precise signals in electronic circuits; for financial data, it has the potential to analyze structured data like stock market trends and economic indicators to predict market movements and detect fraud; and in the medical field, it can process tokenized and vectorised imaging data from X-rays, MRI scans, and CT scans, enhancing diagnostics, monitoring, and surgical planning [Lit+17].

Yet, the most impactful application of the transformer model architecture has undoubtedly been in the field of Natural Language Processing (NLP), directly relevant to the task at hand inthis work, due to the advent of Large Language Models (LLMs).

2.2.2 A Timeline of Language Models

Introduction of BERT and RoBERTa Building upon the discovery of transformers, Google capitalised on the encoder component to develop Bidirectional Encoder Representations from Transformers (BERT [Dev+19]), in October 2018. Deployed on Tensor Processing Units (TPUs), BERT facilitates advanced sentiment analysis, indexing, and classification through its pioneering use of bidirectional self-attention, enabling a nuanced understanding of context. Simultaneously, Meta embarked on a parallel venture with the introduction of RoBERTa [Liu+19] in 2019, an iteration robustly optimised to enhance the efficacy of pre-trained transformers. This move underlines a growing trend towards refining and adapting foundational models for increased performance and reliability, setting the stage for the exploration of fine-tuning, a concept we will delve into further in subsequent chapters.

Development of the GPT series In a distinctive approach, the non-profit organisation OpenAI, founded in 2015, leveraged the foundational transformer technology to pioneer the Generative Pre-trained Transformer (GPT) series. Emphasising text generation, these models incorporate autoregressive decoders and causal multi-head masking to produce text that is both coherent and contextually relevant. Rapid development cycles led to the creation of initial models, scaling up to 0.11B and 1.5B parameters by 2019. Bolstered by a substantial $1 billion investment from Microsoft [Tec24] –driven by concerns over Google's advanced position int the AI industry– and the computational prowess of NVIDIA GPUs [Giz23], OpenAI introduced GPT-3 in 2020. With an unprecedented 175 billion parameters, GPT-3 was made accessible to the wider public through ChatGPT, and this was the breakthrough that captivated the world, demonstrating the immense potential of LLMs in generating coherent and contextually relevant text, while also setting a new standard for AI language models

Chinchilla and Scaling Laws In January 2022, DeepMind's paper on Chinchilla and Scaling Laws [Hof+22] revealed that proportionally scaling model size and training data volume yields optimal performance, leading the 70 billion parameter Chinchilla model, trained with 1.4 trillion tokens, to significantly outperform larger models like GPT-3. Their findings suggest that many LLMs are undertrained and that increasing the number of FLOPs, training data, or model size can enhance reasoning capabilities, indicating no performance ceiling for improving emergent behaviours [SMK23].

Google's Challenges and Strategic Shifts A year later, in February 2023, they introduced Bard, building upon its LaMDa architecture to compete with GPT-3 [Goo23a]. However, Bard faced significant challenges with accuracy and reliability [WM23]. Recognizing the urgent need for a strategic pivot, Google invested over $2 billion in Anthropic, a startup that had recently released a model called Claude, shortly after Bard's debut [Lun23]. In October 2023, they announced Gemini, their proprietary model [Goo23b], but the presentation was overly staged and failed to demonstrate real-time capabilities, which further eroded Google's credibility [Vin23], compounded by other missteps related to its AI products.

Meta's Open-Source Initiative In Meta's sphere, led by Yann LeCun, the company's exploration of spatial computing didn't hinder its decision to open-source LLaMA-2 (July 2023) and LLaMA-3 (April 2024) models [Tou+23]. Rather than solely seeking a competitive edge, this move aims to accelerate progress in a decentralized manner. It's a deliberate step to counter the consolidation of power in proprietary systems, ensuring that governments, companies (with FANG exceptions), NGOs, and individuals have the means to develop models from the LLaMA foundation, fostering a more inclusive and innovative landscape.

Microsoft and OpenAI's Advancements Unlike Meta's open community model, Microsoft continued funding OpenAI. GPT-4 was released in March 2023, quickly setting new performance benchmarks [Ope23]. It excelled in exams such as the Uniform Bar Exam, LSAT, HumanEval, MGSM, and medical exams, scoring in the 90th percentile. Additionally, it performed strongly in logical reasoning tasks (LogiQA) and natural language inference (MNLI) [Qin+23].

Emerging Competitors and Innovations Two years later, the landscape had become more balanced, with big tech companies and startups narrowing the gap. Claude Opus (137B parameters) [Ant24] and Gemini Ultra (175B) [Goo23b] now boast benchmarks comparable to GPT-4's Mixture of Experts (MoE) [Sha+17] model, which has approximately 220B parameters [**hotz2023gpt4**]. Meanwhile, the Llama models at 8B and 70B parameters not only compete but, in some cases, surpass other models in their respective categories [AI24], with a 400B parameter open-source model on the horizon.

Introduction of GPT-4o and Future Prospects At the time of writing this chapter, it has only been a few weeks since OpenAI started to distinguish itself from its competitors again. On May 13, it announced GPT-4o (*im-also-a-good-gpt2-chatbot* in the Figure 2.6), a new state-of-the-art frontier model superior to GPT-4 and others, featuring audio-native capabilities, making them pioneers once again by entering the market of models for low-latency conversational applications [Ope24a] [MIT24]. Additionally, recent rumors suggest that OpenAI is on the verge of unveiling GPT-5. With its enhanced multimodal capabilities and advanced reasoning, these new iterations are expected to render a substantial portion of the competition obsolete [The24]. Historically, as seen with previous iterations of the GPT series, enhancements in the underlying model have iteratively superseded many secondary-layer solutions designed to compensate for cognitive limitations of earlier models, aside from smaller models.

This continuous evolution of the GPT series is emblematic of the broader transformative impact these models have across various industries, driving exponential advancements. It would not be surprising if, in the future, a new paradigm surpassing the transformer models emerges, reshaping the current

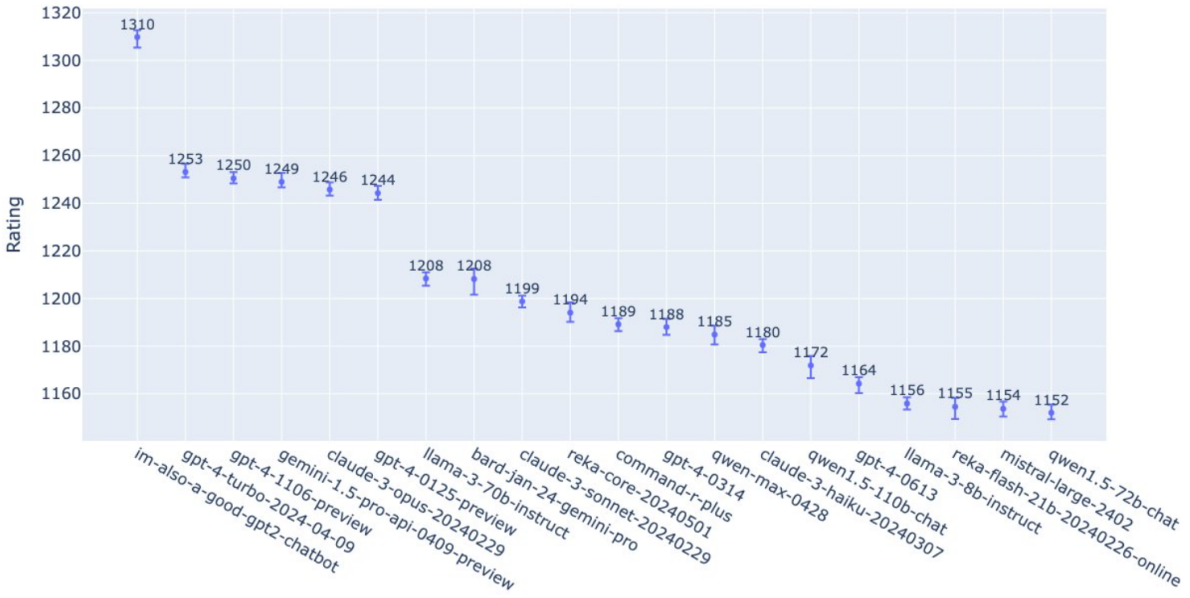


Figure 2.6: Overall ELO scores from LMSys Arena [LM-23], a benchmarking platform for large language models providing standardized tests and comparative performance metrics. The model labeled "im-also-a-good-gpt2-chatbot" at 1310 ELO turned out to be GPT-4o, the natively multimodal end-to-end OpenAI's SOTA model, appended just before rollout. Confidence intervals were determined via bootstrapping.

hierarchy and redistributing the dominance within the corporate landscape. Despite the current reliance on multimodality and massive data sets, these remain areas ripe for efficiency improvements. As the field of artificial intelligence progresses, our ability to innovate and refine architectures also advances, continually pushing the boundaries of what these technologies can achieve.

2.2.3 Formal Definition

Large Language Models (LLMs) can be conceptualised as repositories of programs, encoding the vast array of patterns available on the internet into a latent space as vectorized information. When a user queries a LLM, the model retrieves and executes a program that is specifically configured based on the input data, by interpolating between different programs, thus creating a response that is relevant to your own data.

Discrete State Dynamics: Auto-Regressive Mechanism Inheriting from the rich literature of control systems (see Figure 2.7), LLMs and the unique characteristics they exhibit can formally be defined as

$$\mathcal{E} = (\mathcal{T}, \mathcal{X}, \mathcal{U}, \phi, \mathcal{Y}, h),$$

where:

- **Discrete State and Time:** LLMs operate on sequences of discrete tokens, with states $x(t) \in \mathcal{X}$ and inputs $u(t) \in \mathcal{U}$ evolving over discrete time steps $t \in \mathcal{T}$.
- **Shift-and-Grow State Dynamics:** The state $x(t)$ dynamically grows as more tokens are input or generated, represented by the update function ϕ.
- **Mutual Exclusion on Input vs. Generation:** At each time step, the state $x(t)$ is updated one token at a time, ensuring that the input signal $u(t)$ does not overlap with the generated output $y(t) \in \mathcal{Y}$.

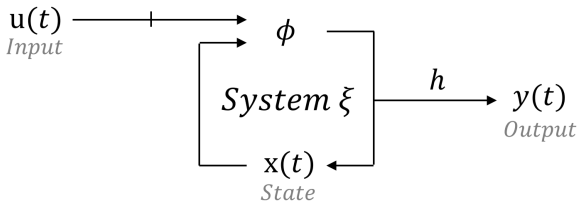


Figure 2.7: LLM Modeled as a Control System.

Likelihood Maximisation for Pattern Learning The training of LLMs focuses on maximizing the likelihood, aiming to make the model's predicted probability distribution as close as possible to the true distribution of the training data. This is expressed mathematically as:

$$\theta = \arg\max_{\theta} \mathbb{E}_{x\sim\mathcal{D}}[\log P_\theta(x_1, \ldots, x_N)] \tag{2.1}$$

Once trained, the execution of the model operates on an auto-regressive mechanism over feed-forward networks, predicting and appending one word at a time based on the preceding sequence. This process can be formalized as a system $\Sigma = (\mathcal{V}, P_\theta)$, where:

- $\mathcal{V}$ is the vocabulary set $\{1, \ldots, |\mathcal{V}|\}$.
- $P_\theta : \mathcal{V}^* \to [0,1]^{|\mathcal{V}|}$ is a probability distribution over the next tokens.
- $P_\theta(x_t \mid x_1, \ldots, x_{t-1})$ where each $x_i \in \mathcal{V}$.
- $\mathcal{V}^*$ is the set of all possible sequences of tokens from $\mathcal{V}$.

State $x(t)$ is a sequence of tokens. The update rule for the state $x(t+1)$ is given by:

$$x(t+1) = \begin{cases} x(t) + u(t) & \text{if } u(t) \neq \emptyset \\ x(t) + x' & \text{else} \end{cases}$$

where $x' \sim P_\theta(x' \mid x(t))$.
In zero temperature sampling, this deterministic system can be described as:

$$x(t+1) = \begin{cases} x(t) + u(t) & \text{if } u(t) \neq \emptyset \\ x(t) + x' & \text{else} \end{cases}$$

where $x' = \arg\max P_\theta(x' \mid x(t))$.

Reachability Sets To further understand the dynamics and reachability of outputs in LLMs, consider the concept of reachable sets. This concept can help visualize the potential outputs that can be generated from a given initial state, depending on the inputs provided.

The reachable set from the initial state $x_0 \in \mathcal{V}^*$ for LLM system Σ is denoted $\mathcal{R}_y^k(x_0)$ and consists of all reachable outputs $y \in \mathcal{V}^*$ from the initial state x_0 via prompts $u : |u| \leq k$.

Cognitive Processes in LLMs Although the model considers the entire past context when making a decision, it selects one word at a time and cannot revise previous choices, a process we have just defined as an auto-regressive mechanism. This one-step-at-a-time approach, without the ability to correct or backtrack, significantly shapes the cognitive reasoning capabilities of LLMs, influencing their performance and the characteristics of the generated text.

Aligning with Type 1 cognitive processes, the auto-regressive approach is fast, intuitive, and heavily dependent on pattern recognition within the latent space of encoded information. Type 1 thinking, as

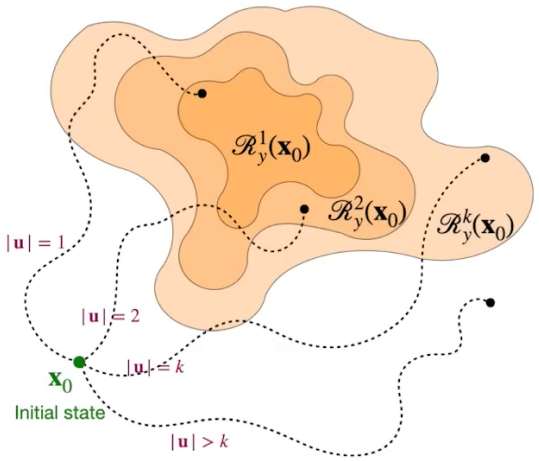


Figure 2.8: Reachable sets from initial state [Bha+24]

popularized by Daniel Kahneman in his book "Thinking, Fast and Slow" [Kah11] is characterized by its speed and efficiency, often at the expense of depth and accuracy. LLMs, operating on this principle, can generate responses quickly by recognizing and extending patterns in the input data. However, this also means that they can be prone to errors when deeper reasoning or complex contextual understanding is required, especially when interpolating.

In this context, the output y is derived from the most recent r tokens of $x(t)$, denoted as $h(x(t); r)$.

In contrast, Type 2 cognitive processes involve slow, deliberate, and analytical thinking. This type of thinking requires more cognitive resources and is engaged when tasks are complex or unfamiliar.

The field of AI is advancing towards the development of strategies that incorporate these more complex Type 2 cognitive processes. Such heuristics would enable current LLMs to revise and refine responses by accessing different programs within the latent space, providing them with the capability for deeper reflection and long-term reasoning, in turn addressing their limitations in areas such as planning, zero-shot learning, common sense reasoning, and strategic decision-making.

Such integration involves the development of closed-loop systems that allow models for thorough processing and evaluation of information, iteratively refining their outputs based on continuous feedback. This evolution from purely autoregressive (Type 1) to a hybrid approach incorporating reflective (Type 2) reasoning processes that mimic human thought processes, such as chain of thought [Wei+22], tree of thought [Yao+23], cumulative reasoning [Zha+23], or beam search [Pry+23], appears crucial for extending the capabilities of the underlying model in use.

While future research should focus on designing hybrid models that can dynamically transition between rapid, intuitive responses and more deliberate, analytical reasoning as dictated by the context, generating different instances of a model to interact and cooperate within a controlled environment via a protocol or signaling mechanism represents a valid approach. This method, offering promising prospects for improvement through the integration of reasoning layers, is elaborated upon in the following chapters, distinctly separating it from the reasoning layers themselves, whose theoretical foundations will be further expanded as they are implemented within the system.

2.2.4 AI Trends on Software Development

In previous subsections, advancements in hardware [NVI21], combined with the massive amounts of collected data, have been examined, highlighting how they have empowered major companies to implement modern machine learning paradigms on an impressive scale. What once seemed like science fiction just a few years ago has now become reality.

Today, setting aside the convolutions of multimodality, we can appreciate the remarkable writing

and comprehension capabilities of these models. They can compose poetry, generate diverse types of content, brainstorm ideas based on specific principles, and meticulously follow complex formatting instructions. This success is not merely a happenstance but a direct consequence of their fundamental understanding of core principles.

Yet, the mastery of these models extends beyond just natural language. Unlike dialects, which are rife with ambiguities, idiomatic expressions, and contextual subtleties, programming languages adhere to well-defined syntactic and semantic rules that govern the construction of valid statements and expressions, creating a rigid framework within which models can operate.

Each keyword, operator, and syntax structure has a specific, defined meaning, allowing a model that learns the syntax and semantics of a programming language to apply this knowledge consistently across different contexts. By analysing the vectorized sequences, models can learn to recognise control flow constructs, data structures, and algorithmic patterns. This enables them to generate, complete, and optimise code with a level of precision and consistency that mirrors human expertise.

Leading companies swiftly recognised the potential of these models [NNB24], adapting them to comprehend and write in various programming languages. A prime example of these efforts is GitHub Copilot [Git21], a pioneering tool backed by the technological expertise of GitHub, a Microsoft subsidiary. Leveraging OpenAI's Codex [Che+21], a specialized Large Language Model (LLM) for programming languages, GitHub Copilot auto-completion and a built-in chat with many context-awareness commands. IDEs like Cursor.sh have started to emerge, with wider context codebase windows, posing a market share risk to Microsoft, and more are set to be released.

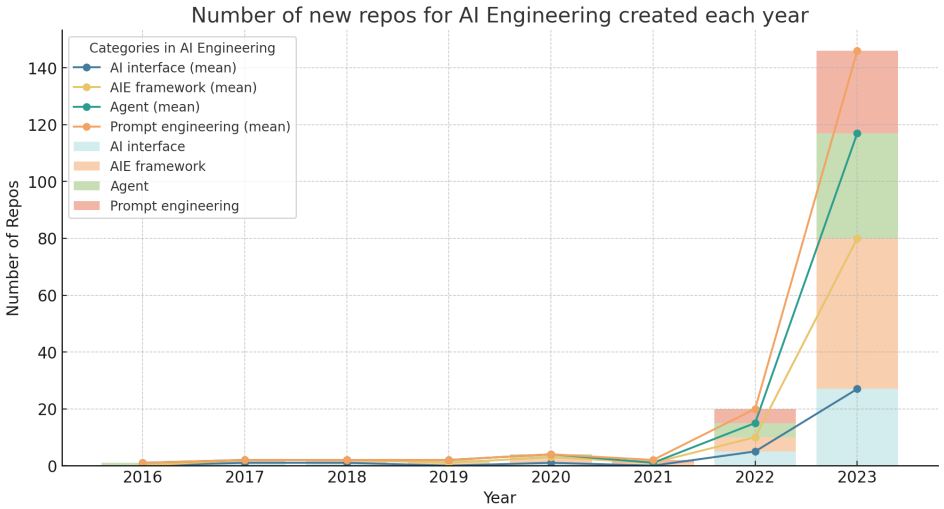


Figure 2.9: Number of new repos for AI Engineering created each year, categorized by AI interface, AIE framework, Agent, and Prompt engineering. The data for 2024 is not available yet but is assumed to follow the exponential growth rate observed in previous years.

Although this initial step of predictive intelli-sense was far from operating unassisted, many scientists have started to refine models into more complex system not only to generate or interpret code but also to start operating semi-autonomously, using tools like command-line shells, code editors, and web browsers to execute actions directly on programs through the operating system.

Open-source projects like OpenDevin [Ope24b] and Devika [Dev24] are ambitiously striving to match the performance of proprietary tools such as Devin from Cognition AI, which recently secured $21 million in funding. Designed to be accessible and customizable, these tools boast advanced AI planning and reasoning capabilities, contextual keyword extraction for focused research, seamless web browsing and information gathering, and dynamic agent state tracking and visualization. Their functionalities are seamlessly orchestrated through natural language interaction via a chat interface, supported by an

extensible architecture that allows for the addition of new features and integrations, all with the goal of excelling in SWE-bench [Ini24] benchmarks.

As these tools continue to evolve, their potential to autonomously plan and divide tasks, build logical bridges between different development phases, and resolve issues such as bugs and conflicts becomes increasingly promising. This could lead to a significant shift in the software development market, where agents handle routine and structured tasks, freeing developers to focus on more creative and innovative aspects.

The implications of such advancements are profound, suggesting a future where the boundaries of what is possible in software development are continually expanded. As metrics like time to first token (TTFT) and tokens per second (TPS) continue to dramatically improve, we foresee a scenario enabling near-instantaneous code generation and efficient tab-completion inspection of numerous options, significantly enhancing developer productivity and interaction with coding environments.

However, as of now (a term being inherently transitory given the rapid pace of AI development), these systems are still in the initial stages and their implementation remains opaque, leaving room for the development of tools aimed at more specialised tasks, since the purpose of these technologies remains broad and their design framework largely experimental.

If these broad-purpose autonomous systems were to replace more specialised tools in the development chain, it is more likely that the underlying models, trained by major companies like OpenAI, would improve sufficiently to render not only these smaller solutions but also the intermediary tools that harnessed the potential of earlier, weaker versions of large language models obsolete.

In any case, while this cycle of models and second-layer solutions continues to be rolled out, there will always be room for the design of secondary stopgap systems. Even if based on large language models, ensuring control over these systems and maintaining quality in terms of requirements will remain a significant challenge for the foreseeable future.

Chapter 3

Fan Engagament Platforms

3.1 The Loyalty Paradox

Companies have long focused on customer service to address what they mistakenly believed were the main reasons for customer departure: price hikes, competitive offers, and product availability. However, customers are increasingly dissatisfied due to poor product experiences and inadequate engagement strategies, citing issues beyond just service interactions.

Despite the unified objective across these programs to bolster customer retention and engagement through the provision of valuable rewards—a notion substantiated by 93.1% of companies reporting favorable returns from such initiatives—the proportion of inactive loyalty members currently stands at a record high of 59% [Sta21]. Defined as those who, despite being enrolled, refrain from actively participating or capitalizing on the available rewards and benefits, these inactive members highlight a critical shortfall in the existing loyalty program models. This reaffirms the pressing need to innovate and integrate more effective components to tap into this latent revenue potential, thereby solidifying retention and engagement as foundational pillars of the business model. With the loyalty industry projected to exceed $24 billion in the next 5 years, the gap between company perception and customer reality urgently needs addressing—a view reinforced by the fact that 96% of clients agree that customer loyalty programs can be improved [Ant22].

A study conducted by KPMG [KPM19] identified four primary shortcomings of loyalty programs: **(a)** 65% agree that most loyalty programs are too hard to join or earn rewards; **(b)** 28% do not join loyalty programs because they do not want their purchasing behavior tracked; **(c)** 81% say their membership encourages them to spend more with the company; and, to conclude, **(d)** 96% say companies should find new ways to reward loyal customers.

It is clear that conventional models are not fully meeting the needs and expectations of customers, and customer loyalty programs has proven to be too critical to be left to customer loyalty programs alone. Loyalty budgets are up, but long-term engagement is elusive.

3.1.1 Towards Loyalty 3.0

In response to these challenges and the shifting dynamics of modern consumer expectations, a new solution has emerged: Loyalty 3.0. Encapsulated under the banner of Web3, marrying digitalisation with customer-centric philosophy, this approach is scalable and built on emerging technology, specifically targeting Millennials, who are set to significantly influence future market trends. Loyalty 3.0 is designed to address the shortcomings found in traditional loyalty programs by delivering engaging, secure, and straightforward schemes.

By offering attractive products and ensuring a high level of customization for each user, these programs effectively meet the specific needs of digital-native consumers, and concurrently have shown to increase ARPU [The23]. This comprehensive strategy makes Loyalty 3.0 an ideal solution that not

only anticipates future trends but also resolves the numerous challenges inherent in current loyalty systems.

The evolution of the internet has seen several transformative stages. Initially, in Web1, spanning from 1990 to 2005, publishers controlled content and revenue, and users primarily consumed information. This period was followed by Web2, from 2005 to 2020, which revolutionized interaction by empowering users to create and share content on social networks and other platforms. The current phase, Web3, represents the next natural progression [Eva23]. First proposed by Dr. Gavin Wood, Web3 introduces a decentralized model, shifting from centrally owned and managed applications to those built on decentralized protocols, thus enhancing user autonomy and data sovereignty. This stage facilitates a tokenized economy where users can own and monetize their content without intermediaries. Technologies such as *web3.js* enable developers to create decentralized applications (DApps) by providing interfaces to peer-to-peer protocols, storage networks like Swarm, and messaging services like *Whisper*, forming a comprehensive development suite for Web3. The digital transformation, especially post-pandemic, has significantly accelerated this shift, marking an unprecedented evolution in how the web operates.

3.1.2 Rethinking Rewards Through Web3

Web3 technologies directly address the main challenges of traditional loyalty programs: engagement and innovation. By integrating blockchain and tokens, companies are not just rewarding purchases but are creating immersive, personalized experiences.

Digital rewards, especially those utilizing non-fungible token (NFT) technology, have emerged as key tools. These unique digital assets serve not just as proof of ownership but can also be programmed with smart contracts to include various perks and experiences, thereby enriching the customer's interaction with a brand while saving the industry over $1 billion annually by preventing financial errors and frauds. While traditional points-based loyalty programs allow rewards to be exchanged for various benefits, the innovation of tokenization on Web3 expands these possibilities, enabling a dynamic suite of options such as subscriptions and tier levels within a secured, tradeable framework.

Companies can leverage Web3 in several key ways to enhance their loyalty programs:

- **Branding:** Digital rewards and collectibles enhance brand image by creating unique and marketable assets. For instance, a sports team could offer digital collectibles of historic games, allowing fans to own and cherish significant moments, which reinforces fan loyalty and enhances brand recall.

- **Gamification:** By incentivizing customer actions, gamification drives deeper engagement. Retailers, for example, might offer digital tokens for purchases that customers can accumulate and exchange for special discounts or unique products. These tokenized rewards, being digital assets on the blockchain, are transferable, elevating their perceived value. Customers see these as desirable assets, enhancing their engagement in a competitive quest to earn rewards, which in turn boosts company revenue and enriches the loyalty program.

- **Exclusivity:** Web3 digital rewards foster a sense of exclusivity and drive customer loyalty. A music artist could release limited-edition digital tokens that provide access to backstage events or exclusive merchandise, making fans feel privileged and more connected to the artist.

- **Personalization:** Digital rewards yield valuable insights into customer preferences and behaviors. Through tracking NFT ownership and interactions, brands can more accurately tailor their marketing and product offerings, leading to higher customer satisfaction and retention. A good loyalty program can introduce new and different benefits that excite customers at a reasonable cost.

- **Community building:** Companies can offer unique digital objects that grant access to brand communities, where customers can provide product feedback and connect with fellow enthusiasts.

> Feeling part of this community makes users less likely to switch to rivals and more likely to encourage friends and family to join.

Companies can provide unique digital objects that grant access to brand communities, where customers can give product feedback and connect with fellow brand advocates. Great rewards and benefits are still a powerful lever, but not if they're indistinguishable from the rest of the pack. Various companies are enhancing customer relationships by leveraging Web3 technologies, which facilitate more direct and engaging connections with consumers [Fan23]. In today's digital landscape, customers seek meaningful experiences and connections with brands rather than just savings. With traditional third-party cookies being phased out and rising customer acquisition costs (CAC), there is a growing emphasis on utilizing first- and zero-party data to build digital communities and foster customer loyalty in innovative ways.

At least 82% of consumers have manifested their willingness to share data to access these customized Web3 offerings [KPM19]. To keep pace with these digital trends, 71.6% of companies plan to revamp their loyalty programs within the next three years. Additionally, 65.2% of companies that do not currently offer experiential rewards plan to introduce this feature, and among those planning to launch a loyalty program in the next two years, 55.4% reported that their programs will include experiential rewards within the next three years.

We envision Sephora enhancing its Beauty Insider program with digital collectibles representing limited edition makeup tutorials and exclusive sessions with celebrity artists, all minted as NFTs to boost exclusivity. Similarly, Spotify could allow users to collect NFTs for creating playlists or attending concerts, offering rewards like VIP tickets and exclusive meet-ups. Even artists like Taylor Swift could leverage a similar model, with her fan loyalty program offering digital collectibles for concert attendance and community engagement, unlocking unique content and real-world experiences like dinner with Taylor or visits to video shoots.

This new generation of businesses values emotional bonds over transactional interactions, achieving ROI by nurturing long-term relationships and significantly increasing customer lifetime value through highly personalized and engaging strategies. Additionally, cross-brand engagement emerges as an advantage of tokenizing loyalty rewards. Blockchain facilitates seamless interoperability between brands and partnerships, allowing spending tokenized rewards across a network of associated companies.

3.2 Third-party SaaS Platforms

A recent survey [KPM19] has revealed that 44% of respondents with existing loyalty programs report significant dependency on their IT departments for program management, leading to operational inefficiencies and reduced agility. To address this, 76.2% of respondents from companies planning to introduce or currently offering loyalty programs aim to empower their marketing teams with tools and processes that reduce IT reliance. This strategic shift towards no-code technology solutions is supported by 70.4% of participants who believe that enabling marketing teams to configure loyalty program rules without coding expertise is crucial for enhancing operational efficiency. No-code platforms, with their user-friendly and intuitive drag-and-drop interfaces, offer a viable solution by allowing marketing teams to independently manage loyalty programs, thus streamlining operations and improving usability.

Building on the framework of modular technology, which allows for easy updates and scalability, Fan Engagement Platforms (FEPs) emerge as a strategic solution designed to meet complex needs. At leading-edge corporations, significant resources are being invested to develop a white-label solution that transcends single-use applications. These infrastructures leverage the composability of the Ethereum blockchain to enhance loyalty programs, offering scalable and customizable solutions tailored to localized strategies and the specific needs of any organization seeking a distinctive approach.

Additionally, as mentioned earlier in this subsection, working with Web3 requires advanced technical programming knowledge, as everything is implemented through smart contracts within the Ethereum Virtual Machine (EVM). For enabling businesses to utilize this third-party service, it is essential to develop a programming module that autonomously generates these digital contracts, eliminating the

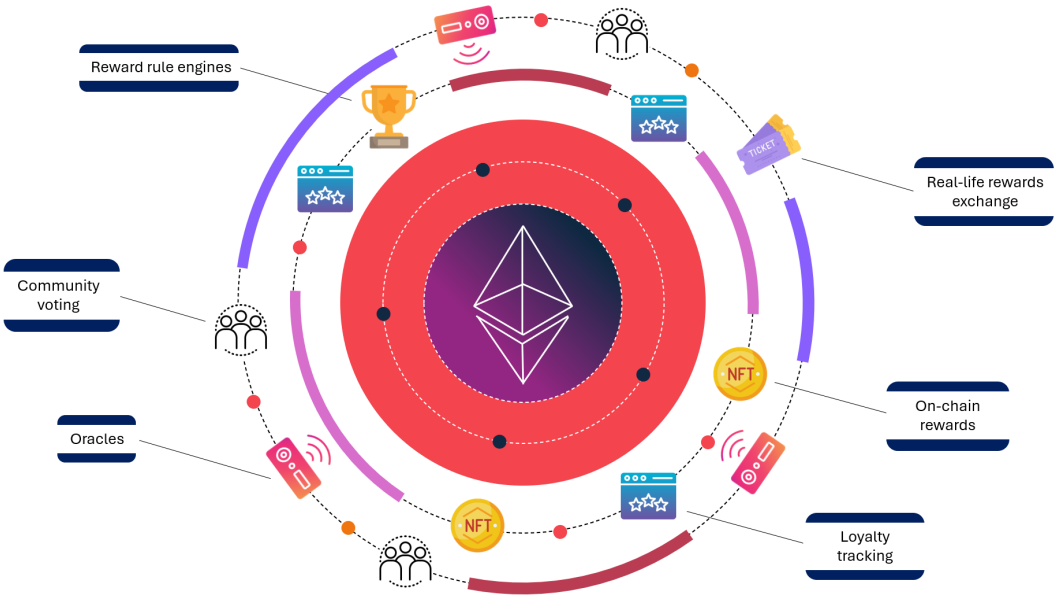


Figure 3.1: Fan Engagement Platform Overview. Illustrates an example of a comprehensive Fan Engagement Platform (FEP), designed to enhance user interaction through social media activities, event attendance, and community voting. The platform utilizes oracles and ZKP services to ensure secure and private data processing, while fans earn rewards and loyalty points through a transparent, blockchain-based system. These points can be exchanged for both digital rewards, such as NFTs, and real-life perks. The system fosters active community participation with mechanisms for managing community funds and implementing voting results, ensuring a dynamic and interactive fan experience..

need for manual coding and reducing response time. This brings us back to the main purpose of this thesis: the development of an automatic code generation tool.

Given that many users, such as artists and smaller businesses, may lack the resources or expertise to write smart contracts, an automated code generation tool is invaluable. This tool will facilitate seamless integration and operation of the service, allowing users to leverage Web3 benefits without needing extensive technical skills.

With the discussion on dApps and Web3 Loyalty 3.0 Engagement concluded, spanning the evolution of Web3 to the specific tool being developed, the next section will examine the role of AI in generating code for business applications. This will provide an agentic perspective to align our technical implementation with industry realities, ensuring both innovation and practicality.

Chapter 4

Software Development Project Plan

4.1 Preliminary Assumptions

Assumption 1 Given the brief deadline of the project, the applied methods are largely heuristic, focusing on practical and straightforward solutions to achieve functional outcomes within the limited timeframe. Data preparation for training, the core algorithm for descriptive data collection, and agent interactions are constrained to a single-thread process and a single port on a single machine.

Assumption 2 The scope of this project is intentionally flexible to accommodate the evolving nature of the technology and the educational objectives of the internship. This flexibility allows for adjustments in the project plan based on ongoing findings and feedback, free from the immediate pressures of production environments, and is essential to ensure that the project can explore the full potential of LLMs for smart contract generation without the risk of disrupting or conflicting with existing operational systems.

Assumption 3 Based on the comprehensive analysis, the project is highly relevant, aligning with market trends and technological capabilities. The strategic assessment confirms the project's significance for both myself and Sony Corporation, demonstrated by their active involvement and interest. Given the dynamic market and technological landscape, these assumptions will be periodically reviewed and adjusted to maintain relevance and efficacy throughout the project lifecycle.

4.2 Features and Functionalities

The content herein should be interpreted as a developmental overview and not as a definitive representation of the system's ultimate functionality or performance. Certain features mentioned, particularly in sections such as Subsection 4.2.2 are marked as optional, automatically discarded, or subject to future implementation. Limitations and technical constraints will be addressed in subsequent sections.

4.2.1 Integrated-type System

The initial goal was to instantiate smart contracts with subsequent quality assessment via a bifurcated modular system, incorporating a feedback loop from the secondary module back to the primary to iteratively refine the code based on the emergent results. This approach aimed to achieve a multi-agent decision system, facilitating inter-agent dialogue, consensual cooperation, and dynamic decision-making to optimize performance metrics, by dynamically altering the underlying reasoning layers and adjusting

the execution thread through strategic breakpoints in the process. Similar approaches are conducted in related studies ([Wu+23], [Che+23]).

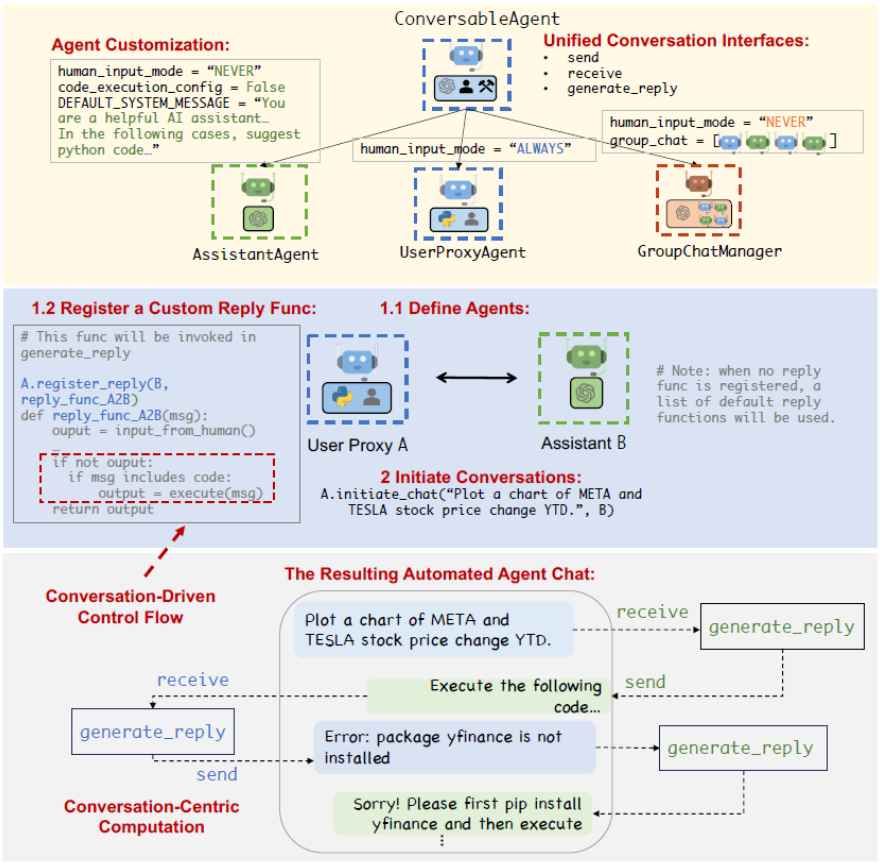


Figure 4.1: **AutoGen [Wu+23] for multi-agent conversation programming:** The AutoGen framework provides built-in agents with unified conversation interfaces, which can be customized as needed; (2) An example application of AutoGen involves developing a two-agent system, featuring a custom reply function for tailored interactions; (3) During execution, the automated agents engage in a dynamic conversation, demonstrating the system's capabilities and flexibility in managing multi-agent dialogues.

Consequently, relying on an architecture with such capabilities, the architectural blueprint was conceived to integrate user feedback within the terminal interface during the generation phase, employing a dual-attention mechanism. This dual-attention workflow, characterized by its dual-layered feedback and iterative refinement process, intermittently supplies insights to steer the generation, optimizing output congruence and precision, thus more responsive to user needs.

Outlined below are the deprecated requirements of this initial architecture:

- **Q/A worfklow for inconsistencies:** the initial design encompassed a Q/A workflow aimed at addressing pertinent incoherences and resolving consistency anomalies as they arose, enabling the system to navigate through complex decision-making processes independently (relocated).
- **Relevance feedback verification:** this component was responsible for assessing the pertinence of feedback to the posed query, ensuring the relevance and refining atomic interactions to bolster the overall quality of the generated output.
- **Terminal-like error messages:** this necessitated a singleton-like construct endowed with the capability to instantiate autonomous agents. These agents possess the autonomy to halt opera-

tional flows upon activation of specific parameters within a neural state-machine, prompting the user and pausing processes until a response is elicited (deprecated).

- **Agent switching mechanisms:** proposed mechanisms to toggle between user input and generation based on predefined thresholds (deprecated).
- **Blackboard system:** the proposal included a blackboard system for data sharing and coordination among agents, utilizing a decentralized architecture facilitated by FastAPI for optimal signal coordination.
- **FastAPI for agent communication:** FastAPI was selected to streamline inter-agent communication and coordination through JSON/YAML formatted data, leveraging its asynchronous request handling and distributed port capabilities to enable swift response times, crucial for high-parallelism reasoning workflows.

4.2.2 Time-bound constraints

It is clear and intuitive that such an integrated-type system needs two main functions: (**1**) recognizing when further insights are needed and (**2**) determining the specific information required. The LLM should identify decision points or standstills by using parameter-based routers[1], and upon recognising needs, it should determine the most relevant data or context to resolve the uncertainty, ensuring context-aware operations. Although LLMs exhibit robust linguistic capabilities, enabling straightforward generation and precise assessments linked to pattern probability and in-token distribution (e.g., code syntax), they falter in abstract evaluations reliant on intuition and common sense (often even in front of few-shot example guidance), such as discrepancy detection or factual consistency checks (SIFiD) [Yan+24]. This shortfall poses a significant challenge to the requirement for timely and contextually accurate responses in integrated-type systems, demanding a deeper exploration of advanced distributed processing techniques to ensure synchronization, with no clear horizon to achieve this integration effectively.

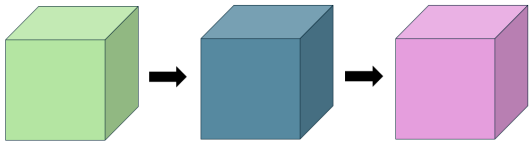


Figure 4.2: Modular Chain Approach: In Chapter 7, their implementations will be thoroughly analyzed.

Remark 4.1. *Furthermore, if human-machine interaction cannot be dynamic and must remain modular, implementing a multi-instance of free dynamic agents operating on a blackboard system through FastAPI becomes redundant, as the system's nature would already conform to a modular structured pipeline. Consequently, the requirements and features for this approach were reconsidered, leading to a refined architecture that introduces a modular interaction module disconnected from the generation module: once the interaction module is processed, it cannot revert to the generation stage.*

[1] Routers, as a part of ReACT agents, are defined in the paper *ReAct: Synergizing Reasoning and Acting in Language Models* [Yao+22], published in October 2022. ReACT agents utilize a prompt structure and workflow divided into two stages: Stage 1 prompts (instructions, status, Q1) and Stage 2 prompts (adding example, memory, Q2), which interact with an LLM. The LLM processes Prompt 1 to produce Response 1, informing Prompt 2, which leads to Response 2 for action processing. Actions are stored in memory and applied to the environment, returning the next state and reward for continuous learning [Rig+23].

4.2.3 Product Backlog

This is why, after reviewing pertinent literature and considering both project goals and my capabilities, I propose a hypothesis centred on employing an modular approach for scalable smart contract generation. Hence, the requirements and features were revisited. This included the introduction of an interaction module separate from the generation one, with an immutable processing flow ensuring system integrity.

1. **Interaction Module**
   - 1.1. Capability to parse and comprehend user inputs articulated in Natural Language (NL).
   - 1.2. Implementation of an iterative feedback loop to refine and enhance user descriptions in terms of contextual appropriateness, logical coherence, information integrity, and ambiguity resolution.
   - 1.3. Precision in identifying errors or ambiguities, coupled with the provision of suggestions for corrections or clarifications (optional).
   - 1.4. Offering contextual help tips and examples to guide users through complex sections of the contract (optional).
   - 1.5. Enabling the importation of relevant data from external sources (e.g., CSV files, previous contracts) to supplement the description (optional).
   - 1.6. Construction of an attribute-based requirements table: Creation of a structured table to capture and organize user requirements based on specified attributes for subsequent agentic processing.
   - 1.7. Development of a graphical user interface (GUI) to facilitate user interaction with the system.
   - 1.8. Integration of the GUI with the backend to display and allow refinement of contract requirements.
2. **Generation Module**
   - 2.1. Allocation of discrete tasks to agents responsible for generating code snippets.
   - 2.2. Delegation of code snippet construction tasks to agents based on detailed and granular requirements.
   - 2.3. Facilitation of configuration for varying reasoning layers without necessitating extensive code modifications.
   - 2.4. Compilation and integration of generated code snippets into a cohesive smart contract.
   - 2.5. Allowing for the generation of multiple versions of a contract (optional).
3. **Assessment Module**
   - 3.1. Utilization of the Solidity compiler (solc) within a Python environment, enabled by the py-solc-x wrapper, to ensure syntax correctness of the smart contract.
   - 3.2. Provision of a sandbox environment for users to test the generated contracts and provide feedback (optional).
   - 3.3. Verification that the smart contract comprehensively covers all specified tasks and aligns with user objectives.
   - 3.4. Execution of a security analysis to identify and mitigate vulnerabilities within the smart contract.
   - 3.5. Analysis of the smart contract's gas consumption to ensure cost-effective execution.
   - 3.6. Implementation of a fallback process involving agent-specific smart-targeted regeneration of smart contract code if errors are detected during syntax checking.
   - 3.7. Addressing the *Qualitative Self-Assessment Dilemma* (QSP), a complex issue concerning the efficacy of self-evaluative measures in ensuring the output's standard.

4.3 Feasibility Analysis, Workflow and Milestones

While this chapter focuses on the technological project aimed at creating a proof of concept to expand knowledge and potential application in a larger initiative, the current section, addresses the academic framework's deadlines and schedules. These academic constraints are deeply embedded within and condition the technological project's development, dictating its thesis with their own timelines and milestones.

Feasibility Given the project timelines and the company's stance, it appears technically feasible to proceed. This conclusion is drawn from the realistic expectations set forth in Assumption 1 (see section 4.1).

Project Development Reports The project's development plan was documented in the systematic drafting of Project Development Reports, spanning a total of 52 pages. These reports provide a qualitative chronicle of the project's phases and have been instrumental in tracking practices and informing the thesis writing, prioritizing rapid iteration and experimentation over rigid planning, inherently involving a high degree of trial and error. The reports were organised as follows:

1. Conceptual Genesis of the Thesis (*26th February 2024*)
2. Architectural Design Approach (*3rd March 2024*)
3. Use Clarification for a Simplified Approach (*24th March 2024*)
4. About the Fan Engagement Platform (*8th April 2024*)
5. Framework Maturation (*8th April 2024*)
6. Preliminary Design Overview (PDO) (*16th April 2024*)
7. Comprehensive Report and Planning (*25th April 2024*)
8. Technical Infrastructure (*28th April 2024*)
9. RP Module Implementation (*29th April 2024*)

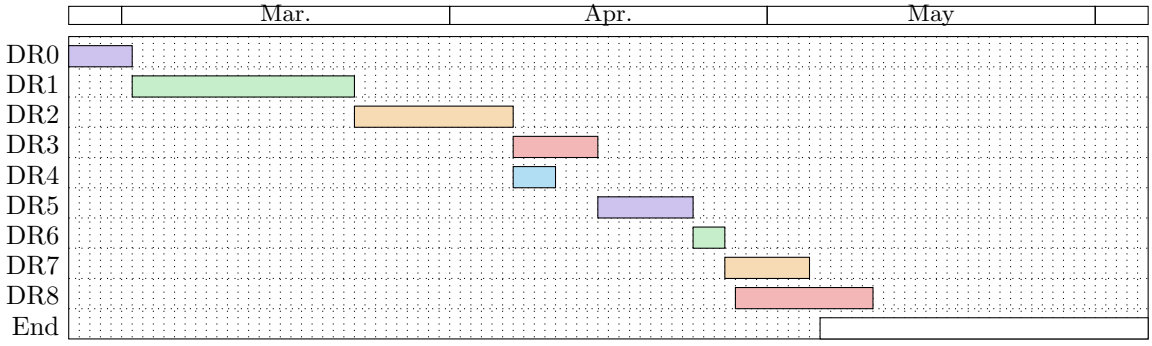


Figure 4.3: Project Timeline Gantt Chart. The last bar does not indicate report writing, as it has been discontinued. Instead, it represents the coding of the final modules, ablation process, and thesis development.

These served as intermediate documentation for the internship and provided reliable content for the thesis, incorporating accumulated insights and detailed project information primarily from detailed Obsidian notes and active project status inspections.

The project achieved early functionality at the MVP (Minimum Viable Product) stage, and subsequent efforts have focused on enhancing the tool through continuous research and exploration of various reasoning methods.

The final report (represented in red in Figure 4.3) discusses the initial stages of software development using the AGILE methodology, which was ultimately discontinued as the project's predefined trajectory did not require strict adherence to this method.

Obsidian Notes Throughout the project, over 135 Obsidian[2] notes were created (not available in readable format), forming an extensive knowledge base. These notes included diverse resources, comprehensive documentation, seminar records, white papers, critical annotations, conceptual ideas, reflective self-assessments, and observational entries. Both the project development reports and the thesis were developed in parallel within Obsidian, ensuring a thorough and interconnected documentation process.

4.4 Tech Stack

Details about the computers, equipment, and other resources, as well as the decision to work with endpoints or locally, will be discussed in Chapter 5 to justify the choices made.

This section provides a comprehensive overview of the tools and technologies employed in the project, detailing their roles and contributions to various aspects of development, version control, dependency management, and documentation.

A summary table is provided below.

Version Control and Repository Management
1a - Git: Used for version control and collaboration.
Language and Development Environments
2a - Python: An interpreted, object-oriented, high-level programming language with dynamic semantics.
2b - WSL (Windows Subsystem for Linux): Provided a Linux environment on the HP Elitebook 840 G6.
2c - Ubuntu: The Linux distribution used within WSL.
Package Management and Dependency Tools
3a - Poetry: A dependency management tool, used to manage project dependencies and venvs.
3b - Chocolatey: A Windows package manager, used to install and manage software packages on WinOS.
Python Libraries and Frameworks
4a - Pydantic: Used for data validation and settings management in Python, ensuring type safety.
4b - DSPy: A library or framework used for pipelining LLMs.
4c - LangChain: A library or framework used for building language model applications.
4d - Streamlit: A framework for creating interactive web applications in Python.
Development and Formatting Tools
5a - Ruff: A linter for Python code to ensure code quality and style adherence.
5b - Black: A code formatter for Python to ensure consistent code style.
5c - GitHub Copilot: An AI-powered code completion tool to assist with coding and improve productivity.
5d - Sphinx: Used for generating documentation from annotated Python source code.

Table 4.1: Tool of the stack.

Subsequently, their roles in the project are explained.

Git, a distributed version control system, required no additional installation as it was pre-configured from a prior project. It serves as the fundamental software for version control, enabling seamless synchronization and updating of remote repositories across various development environments.

Python is the most popular language for AI, deep learning, and statistics, offering a rich ecosystem of libraries and frameworks. Its simplicity and readability facilitate rapid development and prototyping, making it ideal for complex algorithms and data analysis.

[2]Obsidian is a graph-based note-taking application allowing for the creation and management of interconnected ideas in Markdown

Running **Ubuntu**, the **Windows Subsystem for Linux** (WSL) was deployed on the HP Elitebook 840 G6. This subsystem provides an optimal environment for testing, evaluating, and executing specialized tools and libraries. It also supports low-level interfacing necessary for operations with Large Language Models (LLMs).

Poetry has been utilized throughout the project to maintain a clean, functional, synchronized and fully compatible virtual environment. It ensures compatibility and keeps the environment updated, effectively managing the project's dependencies.

Chocolatey was utilized to install Poetry, and it is documented as an alternative package manager in the project's official README. This tool simplifies the process of software installation on Windows platforms, facilitating the setup of necessary development tools.

Pydantic is employed for rigorous data validation and settings management, leveraging Python type annotations. It ensures the integrity of input/output structures within the modules, which define the functional patterns of the agents as generated by the teleprompters.

DSPy implements a declarative compiler framework designed for interaction with various foundation models. It optimizes the machine learning pipeline by leveraging prompting practices, thereby enhancing efficiency and accuracy through declarative syntax.

LangChain was initially considered for its standard capabilities in managing LLM workflows, including placeholders, templates, and chaining mechanisms, as part of the MVP. Subsequently, its use is mainly integrated into the generation module for SOTA, while remaining flexible for other methodologies.

Streamlit, a Python library, facilitates the development of interactive web application frontends. It supports the creation of dynamic UI components such as interactive panels and tabs, thereby enabling efficient human-machine interaction and user data capture.

Additional libraries were used for developmant and code formatting purposes.

1. **Ruff**, a high-performance Python linter, ensures code quality and consistency by performing static analysis to detect and rectify potential bugs and coding standard violations early in the development cycle.
2. **Black**, the uncompromising Python code formatter, enforces a uniform coding style by automatically reformatting code. This reduces cognitive load on developers and simplifies code review processes by ensuring consistent codebase aesthetics.
3. **Github Copilot** is an AI-powered code completion tool that assists with coding by suggesting code snippets and completions, thereby enhancing developer productivity and accelerating development processes.
4. **Sphinx** is used for generating documentation from annotated Python source code. It supports various output formats and ensures that the project's documentation is comprehensive and up-to-date.

Dual-Device GitHub Workflow In this setup, GitHub was used as the remote repository to manage and collaborate on the project. For security compliance, SSH keys were configured for both GitHub accounts on two devices. The configuration involved generating SSH keys (via ssh-keygen) and adding them to the respective GitHub accounts. The personal GitHub account, yagomendoza.dev, was used to host the main project repository. Additionally, I was invited to collaborate on a company repository hosted by Sony, enabling seamless synchronization and contribution to both repositories. The workflow involved using an HP Elitebook 840 G6 running Windows Subsystem for Linux (WSL) and an ASUS-NotebookSKU running Windows. The WSL environment on the HP laptop provided an optimal platform for testing, evaluation, and working with specialized large language models, ensuring a robust development environment.

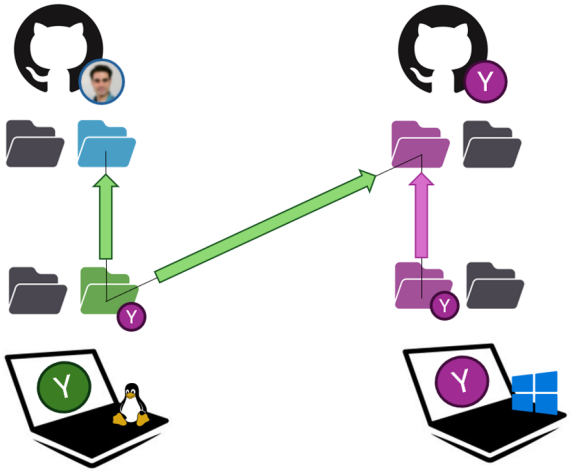


Figure 4.4: Dual Device GitHub Setup: The image illustrates the setup of two GitHub repositories managed from two devices. The *HP Elitebook 840 G6* (left, running Linux via WSL) is used for testing and working with large language models, while the *ASUS-NotebookSKU* (right, running Windows) handles additional development tasks. The green arrows depict the synchronization between local copies and remote repositories on GitHub.

Chapter 5

Minimal Viable Product (MVP)

Throughout this chapter, it should be clear that the MVP does not aim to establish a solid architecture that will endure throughout the project, but rather serves as a foundation for contemplating configuration parameters and requirements. This foundation would naturally undergo significant changes and could even be completely redefined as the architecture evolves around it.

5.1 First Approach

In the context of product development, a Minimum Viable Product (MVP) represents the most pared-down version of a new product that enables the team to gather the maximum amount of validated insights about user interactions with minimal effort. Note that in this specific scenario, the traditional role of the customer is assumed by Sony itself, as the objective is to validate a component of a wider system, as detailed in previous chapters.

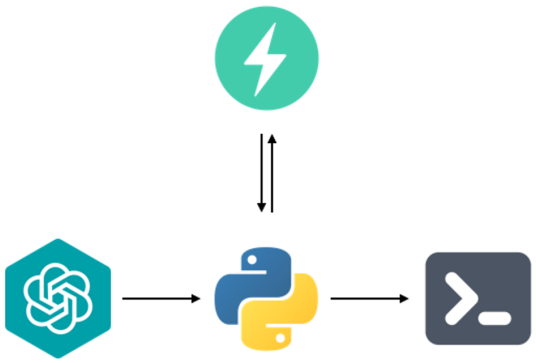


Figure 5.1: Minimal MVP Architecture

1. **FastAPI** effectively supports self-prompting in Large Language Models (LLMs) through its asynchronous operations, ensuring swift response times essential for real-time decisions. Its robust data validation also enhances the accuracy of interactions. Notably, during the MVP phase, HTML was used to improve API management.
2. For the **Python** generation module, the **Langchain** framework was utilized to streamline interactions with the LLMs. This setup involved developing a dedicated LLM class designed to act as an abstract base for crafting agents, encapsulating essential functionalities required for model prompting and inference.

3. The system outputs were initially managed through a terminal class, which allowed for potential expansion into a Command Line Interface (CLI) using the *argparse* library. This feature aimed to enhance user interaction, providing a more accessible and manageable interface for navigating and operating the system effectively.

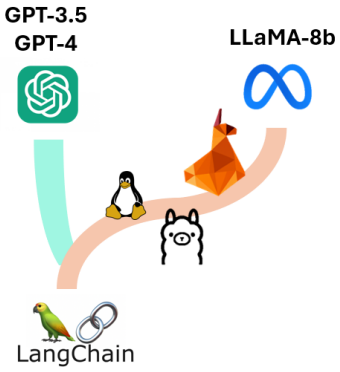
Remark 5.1. *FastAPI was initially used in the MVP but was phased out as the project transitioned to a modular architecture. Despite being deprecated, it remains in the codebase as a foundation for future development if needed. LangChain, on the other hand, was intended to be the core framework for the entire project. Despite DSPy being introduced as a great wrapper solution for LLM calls, LangChain built several components for the latest version of the system due to its simplicity.*

This setup of this minimal pipeline for Smart Contract Generation has demonstrated effectiveness; the program, with fewer than 300 lines of code, is already responsive and equipped to generate answers efficiently.

5.2 API Endpoints vs Local

While the possibility of future changes in our approach is recognized as the project evolves, a clear assessment of the options available was provided when the MVP was first developed. The aim was to understand the tools and potential in the field, ensuring that the initial path was both technically and financially viable. This influenced the choice of LLM implementation, with factors such as the hardware constraints of the project, the economic impact of token costs, and the inferential demands at the beginning of development being considered.

For the preliminary testing of the system, OpenAI's GPT-3.5 was employed due to its advantageous balance of cost-efficiency and performance capabilities. This model facilitated early detection of integration challenges and enhanced the optimization of the system's engagement with natural language processing functionalities, without necessitating significant upfront investment. Concurrently, the introduction of Meta's Llama 3 series garnered considerable attention within the research community owing to its open-source nature, thereby eliminating per-token costs. However, operational deployment of LLaMA-8B necessitates the use of LlamaCPP, optimally configured on a Windows Subsystem for Linux (WSL) to streamline the setup process. Given the available data at that juncture of the project, the decision between these two options remained ambiguous.



Model	Provider	Pricing/1M tokens		TTFT	Speed
		Input	Output		
gpt-4-turbo-2024-04-09	OpenAI	10.00$	30.00$	N/A	55 tokens/s
gpt-3.5-turbo-0125	OpenAI	0.50$	1.50$	N/A	13 tokens/s
gpt-3.5-turbo-instruct	OpenAI	1.50$	2.00$	N/A	13 tokens/s

Table 5.1: Comparison of different model pricing and performance parameters.

However, OpenAI's GPT models and Meta's Llama 3 series (with its implementation detailed further in this section) were not the only contenders. An in-depth analysis of various models and accelerated software providers is presented next, focusing on integration capabilities, performance benchmarks, and cost-effectiveness.

5.2.1 Models and Providers

Additional models and solutions that were still under consideration for incorporating into the pipeline, contingent upon project evolution and compatibilities, encompass models from various providers. **Hugging Face** offers **Gemma-2/7b-it** y **Mixtral-8x7b-Instruct**, both of which do not have specified pricing for 1M I/O-tokens. These models are evaluated for integration with the cloud-based infrastructure **GROQ** and **AnyScale**, respectively, to mitigate the slow generation of local devices I have, with **Gemma-2/7b-it** also being compatible with the LLM interface **LamaCPP**, and **Mixtral-8x7b-Instruct** with **Ollama**.

Both frameworks will be further studied in the following subsection.

From **Anthropic**, there are **Claude-3 Sonnet (VPN)**, priced at \$3/\$15 for 1M I/O-tokens, and **Claude-3 Haiku (VPN)**, priced at \$0.25/\$1.25 for 1M I/O-tokens. Additionally, **OpenAI**'s **GPT5** (unreleased) and **Microsoft**'s **Phi-2/3** are considered, both lacking specified pricing for 1M I/O-tokens. The solutions for cost-effectiveness, including cloud-based infrastructure and LLM interfaces, are under review for all these models.

Model	Provider	Pricing/1M tokens		TTFT	Speed
		Input	Output		
Claude-3 Sonnet (VPN)	Anthropic	3.00\$	15.00\$	N/A	N/A
Claude-3 Haiku (VPN)	Anthropic	0.25\$	1.25\$	N/A	N/A

Table 5.2: Comparison of different model pricing and performance parameters.

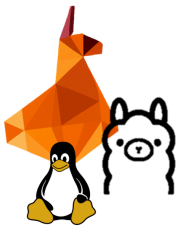
Model performance may fluctuate based on the time of day and day of the week for remote models. It is also contingent on the number of concurrent users and the available computing resources.

5.2.2 LlamaCPP & Ollama

LlamaCPP and **Ollama** are frameworks designed to facilitate the local inference of large language models (LLMs), specifically those in the LLaMA [1] series. These frameworks are integral for running LLMs on various hardware configurations, from consumer-grade devices to high-performance clusters, providing flexibility and efficiency in deployment.

[1] Until April 18, 2024, high-capacity models were either open-sourced but impractically large for local use by small developers or remained pro-prietary and inaccessible. This changed with Meta's release of the Llama3 models in open-source format. Available in 8B and 70B parameter configurations, these models are part of Meta's strategy to compete with Google's Gemini, Anthropic's solutions, and the API services from Mi-crosoft and OpenAI. Currently, Llama3 models match the performance of both medium and large-scale models from these leading companies, having achieved state-of-the-art (SOTA) performance across a broad range of tasks. Llama3 utilized advanced filtering strategies for premium data quality, including heuristic, NSFW, and semantic deduplication methods. It employed a tokenizer supporting a lexicon of 128,000 tokens and introduced a novel Grouped Query Attention (GQA) mechanism for enhancing inference efficiency. The models were trained using a tri-fold parallelization strategy—data, model, and pipeline—on custom-built 24,000 GPU clusters. This infrastructure achieved high compute utilization, exceeding 400 TFLOPS per GPU, optimizing the training process for longer sequences and enabling the handling of sequences up to 8,192 tokens. In performance benchmarks, such as MMLU, GPQA and MATH, Llama3 models have demonstrated superiority over their contemporaries. Compared to its predecessor, Llama2, Meta's new model lineup retains the same transformer architecture with a dense decoder but excludes Mixture of Experts (MoE) and MAMBA. A significant difference is the training method. Deliberately deviating from the optimal practices described in DeepMind's 2022 paper on Scaling Rules—investigates the optimal model size and number of tokens for training a transformer language model under a given compute budget—, Llama3 was trained sub-optimally, utilizing over 15 trillion tokens, which includes a substantial increase in code content and high-quality data. This strategy was chosen to avoid the complexities of larger models, making it suitable for ordinary commer-cial laptops and GPUs. This approach not only reduces the marginal costs of inference but also leads to considerable savings in extensive infer-ence deployments.

Given the hardware specifications—particularly the ASUS computer with an Intel Core i7-6700HQ CPU and 32 GB of RAM, and the HP computer with an Intel Core i7-8565U and 16 GB of RAM—it is feasible to run LLaMA-8B locally. However, the older GPU has few tensor cores which may limit performance, suggesting a potential bottleneck for extensive machine learning tasks. This limitation underscores the importance of batch processing tasks overnight when system demand is low, maximizing the use of computational resources.



ASUS-NotebookSKU
Intel(R) Core(TM) i7-6700HQ CPU 2.60GHz Intel(R) HD Graphics 530 & NVIDIA GeForce GTX 960M 32.0 GB (31.4 GB usable) 90.6 GB de 446 GB total
HP Elitebook 840 G6
Intel(R) Core(TM) i7-8565U CPU 1.80 GHz (up to 1.99 GHz) Intel(R) UHD Graphics 620 16.0 GB (15.3 GB usable) 383 GB de 475 GB total

Table 5.3: Hardware specifications.

LlamaCPP A C++ implementation of the LLaMA model designed to optimize performance and minimize dependencies. Created by Georgi Gerganov, it enables efficient LLM inference on a wide range of hardware, including CPUs and GPUs, and supports multiple operating systems such as macOS, Linux, and Windows. Key features of LlamaCPP include:

1. **Hardware Optimization**:
   - Supports AVX, AVX2, AVX512 for x86 architectures.
   - Optimized for Apple silicon using ARM NEON and Metal frameworks.
   - Custom CUDA kernels for NVIDIA GPUs and HIP for AMD GPUs.
   - Vulkan and SYCL backend support for broader compatibility.
2. **Quantization**:
   - Implements various levels of integer quantization (1.5-bit to 8-bit) to reduce memory usage and improve inference speed.
3. **Deployment Flexibility**:
   - Compatible with Docker, enabling containerized deployments.
   - Supports hybrid CPU+GPU inference to handle models larger than available VRAM.
4. **Model Support**:
   - In addition to the LLaMA models, LlamaCPP supports a variety of other models such as Mistral, Falcon, and several Chinese and French models, enhancing its versatility.
5. **Build System**:
   - Uses CMake for building across platforms, ensuring optimized performance through compile-time flags for different architectures and hardware capabilities.

Ollama Built on top of LlamaCPP, but extends its capabilities with additional features and a more user-friendly interface. While LlamaCPP is focused on raw performance and hardware optimization, Ollama integrates these capabilities into a cohesive system for easier deployment and management. Key aspects of Ollama include:

1. **Build Integration**:
   - Utilizes both CMake for compiling the C++ components and Go for building the overall application, ensuring robust and efficient execution.
   - The Go build system leverages cgo to integrate C-family code, enabling smooth interoperability between the components.
2. **User-Friendly Features**:
   - Simplifies model management by integrating weights and configurations, reducing the manual effort required for setup.
   - Provides a streamlined interface for switching between models and managing inference tasks.
3. **Performance and Scalability**:
   - Designed to take full advantage of GPU acceleration, significantly boosting performance for inference tasks.
   - Supports deployment across various platforms, from desktops to mobile devices, ensuring broad accessibility.
4. **Model and Library Support**:
   - In addition to LLaMA models, Ollama supports various other LLMs and libraries, including phi-2, llava, and mistral-openorca, offering extensive flexibility for different use cases.
   - Focuses on providing a comprehensive local inference solution that rivals cloud-based offerings in terms of performance and flexibility.

Model	Provider	Pricing/1M tokens		TTFT	Speed
		Input	Output		
Llama-3-8b (7 GB)	Ollama	/	/	8 sec	2 tokens/s
Llama-2-70b	Ollama	/	/	5 sec	2,35 tokens/s
Mixtral (26 GB)	Ollama	/	/	72 sec	0,09 tokens/s

Table 5.4: Open-source models characterization in Ollama.

Remark 5.2. *The calculation of "10 tokens per second" for the LLaMA-8B model assumes that the quantized version of the model can process 40 characters per second on optimally configured hardware. Given that the average token length, including spaces and punctuation, is estimated at 4 characters, this rate translates into 10 tokens per second. This estimation assumes that the local hardware (CPU and GPU) can support this level of efficiency, which may vary based on actual system performance and model optimization.*

Groq: LPU-Accelerator Groq's Language Processing Unit (LPU) significantly enhances AI inference, delivering performance that is 25 times faster and 20 times cheaper than traditional GPUs. Unlike AI models like ChatGPT, Groq's LPU focuses on accelerating inference pro-cesses. Its fully deterministic VLIW architecture, which keeps all critical data like weights and activations on-chip, achieves up to four times the throughput of other services while reducing costs dramatically.

This architecture enables advanced reasoning techniques such as chain of thought and supports multiple reflection process-es, ensuring safer and more refined responses in real-time applications. The enhanced speed and efficiency also cater to the demands of autonomous AI systems, which require faster outputs and lower latencies for tasks like code generation and agent responses, making Groq's technology indispensable in modern AI applications.

Model	Provider	Pricing/1M tokens		TTFT	Speed
		Input	Output		
Llama-3-70b (8k CTX)	Groq	0,59$	0,79$	0,18 sec	280 tokens/s
Llama-3-8b/-chat (8k CTX)	Groq	0,05$	0,10$	0,09 sec	870 tokens/s
Mistral-8x7b some (32k CTX)	Groq	0,27$	0,27$	0,27 sec	480 tokens/s
Gemma-7b (8k CTX)	Groq	0,10$	0,10$	0,06 sec	820 tokens/s
Llama-2-70b (4k CTX)	Groq	0,64$	0,80$	0,16 sec	300 tokens/s
Llama-2-7-b (2k CTX)	Groq	0,10$	0,10$	0,16 sec	750 tokens/s

Table 5.5: Open-source models characterization in Groq.

Jonathan Ross, while at Google, recognized the crucial role of low latency in AI performance. He developed a new tensor processing chip, now integral to Google's data centers, addressing the divide between companies with advanced AI capabilities and those with-out. Instead of investing in NVIDIA, he founded Groq to democratize access to powerful, interpretable software for real-time AI inference.

5.2.3 Summary

GPT-3.5 was selected primarily for its optimal balance between cost-effectiveness and performance, while GPT-4 was reserved for tasks demanding higher computational resources. Meta's Llama 3 series emerged as a prospective choice for future scalability owing to its open-source architecture and advanced functionalities, rendering it suitable for extensive deployment scenarios. Despite Groq showcasing remarkable performance, further exploration was halted due to the necessity of integration with an alternative to LangChain. Furthermore, models developed by Anthropic and Hugging Face were disregarded due to integration intricacies and associated elevated expenses.

5.3 The LangChain Standard

Handcrafted prompting has been integral to the use of large language models (LLMs) since their inception, providing detailed and nuanced responses that offer valuable insights into model performance across various tasks. However, as projects grow in complexity and volume, the manual nature of this approach presents significant challenges in terms of scalability and efficiency.

To address these challenges, frameworks like LangChain [Con23a], introduced by Harrison Chase in early 2023, have been developed to simplify the creation of LLM applications. LangChain achieves this through modular abstractions that encapsulate common steps and concepts necessary for working with LLMs. For instance, prompt templates with placeholders eliminate the need for manual coding of context and queries, including instructions for few-shot prompting. Additionally, chains, which link various abstractions to other components, streamline complex NLP tasks with minimal coding. Unlike some other frameworks, LangChain supports integration with almost any LLM given the appropriate API key, providing a standardized interface across all models, whether open-source or proprietary. This flexibility is particularly valuable in projects where the scope remains open to exploration.

Initially, our plan involved implementing reasoning layers using custom-coded methods via LangChain, as the MVP (Minimum Viable Product), with its LLM connections and agents, was already developed

using this framework. LangChain, much like the infamous JavaScript in the early days of the web, had become the standard in this emerging sector.

However, it soon became evident that this approach was not effective due to insufficient control over the outputs. To scale the system predictably for production—whether as a terminal extension plug-in, a library, a command-line interface (CLI), or an early graphical interface design—a more effective strategy than LangChain was required for programming the model's behaviour.

Developers aiming to build robust, scalable, and efficient LLM solutions may find LangChain's promises somewhat unfulfilled. Although LangChain significantly improves the management and structuring of prompts, there remains room for optimization, particularly in automating and refining the prompt engineering process.

Furthermore, LangChain presents several issues in terms of learning and documentation:

- **Poor documentation:** Incomplete and outdated documentation makes understanding and implementing LangChain functionalities challenging. The division into components and use cases is often confusing and counterintuitive.
- **Lack of user interaction in documentation:** The absence of a comments section in the documentation hinders users from asking questions and receiving answers, limiting the ability to quickly resolve issues and learn from others.
- **Insufficient community support:** Limited activity and responses on platforms like Stack Overflow, Reddit, and GitHub Discussions make finding solutions to common problems difficult. While there is an active Discord community, the conversations are not searchable by search engines, and many questions remain unanswered.

Scalability issues prompted the search for alternative solutions, leading to the discovery of a new framework, DSPy. This framework allowed for better testing of reasoning layers within the use case, leading to the reimplementation of several modules that required more control, particularly those involving conditional rules not immediately obvious to the language model and not expressible through a limited number of few-shot examples. Consequently, much of the ongoing process was deprecated and reimplemented using DSPy. However, LangChain continues to be used for the generation module and will remain integral for specific tasks in later versions of the system.

Chapter 6

Declarative Pipelines via Multi-Agent Compilation

LangChain initially served as a useful foundation with its pre-packaged components and implementations of popular, reusable pipelines. However, many of its limitations, as outlined in the previous chapter, necessitated a search for a more effective framework for tasks requiring superior control, such as implementing conditional rules that were not immediately obvious to the language model or expressible through a limited number of few-shot examples. DSPy, recently rolled out, was selected for its composable operators and compilation capabilities, which enhanced the machine learning pipeline through improved prompting practices and declarative syntax. As a result, several modules were reimplemented using DSPy, while LangChain continues to be utilized for the generation module and remains integral for specific tasks in later versions of the system.

6.1 Challenges and Limitations in Prompt Engineering

Language models (LMs) demonstrate remarkable sensitivity to the way they are prompted, an issue that becomes particularly pronounced in pipelines where multiple LM interactions are necessary. Presently, these interactions rely heavily on hard-coded prompt templates, often involving long, meticulously crafted strings of instructions and demonstrations, developed through extensive manual trial and error. Although pervasive, this approach is brittle and lacks scalability, akin to manually tuning the weights of a neural network.

Typical implementations and research libraries, such as LangChain [Con23a], SemanticKernel [Mic23] and LlamaIndex [Con23b], utilize manual prompt engineering. However, their heuristic methods are not LLM-agnostic[1], leading to fragility and frequent breakdowns with changes in the underlying model. This approach demands extensive manual testing, which does not ensure optimal performance or robustness against potential vulnerabilities. Despite these efforts, LangChain has shown in the development of the MVP to often fails to deliver consistent and reliable results.

In late September 2023, an analysis of the LangChain codebase revealed 50 strings exceeding 1000 characters, primarily used as prompts. A substantial portion of LangChain's Python files is dedicated solely to task-related templating and prompt engineering, with 12 prompts.py files and 42 prompt.py files identified. While some solutions attempt to address the length problem by compressing prompts using coarse-to-fine approaches and dynamic budget controllers for enhanced computational efficiency, these prompts still fail to generalize across different pipelines, LMs, data domains, or inputs. This

[1]When using structured input, be aware that each LLM family has their own preferences. Claude prefers xml while GPT favors Markdown and JSON. This preference for specific formats poses challenges in prompting, as it requires understanding and adhering to the preferred input style of each model to ensure optimal performance.

underlines the need for a more systematic and scalable solution.

6.2 DSPy: Declarating Syntax for Prompting Yields

While crafted prompts by experts offer certain advantages, a growing number of researchers are now exploring discrete optimization and reinforcement learning (RL) to enhance prompt effectiveness, typically focusing on individual language model (LM) interactions ([Guo+23], [Yan+23], [Hua+22], [Pry+23]).

Taking a significant step forward, DSPy [Kha+23] introduces a groundbreaking approach by automatically generating task-adaptive prompts from declarative signatures by bootstrapping high-quality multi-stage demonstrations with constraints. This innovation provides a scalable alternative to the restrictive, hard-coded prompts tied to specific database providers within a codebase.

Developed at Stanford, DSPy is rooted in foundational model programming and utilizes differentiable programming with a PyTorch-like syntax. It emerges as a competitive alternative to traditional methods that rely on expert-crafted prompt chains, by autonomously bootstrapping compositions of prompts. This programming model is built on three key abstractions designed for automatic optimization: signatures, modules, and teleprompters. Signatures abstract the input/output behaviors of modules, which replace conventional hand-prompting techniques and can be flexibly composed into various pipeline configurations. Teleprompters then work to optimize all modules within the pipeline to maximize a specified metric, ensuring an efficient and scalable solution.

Signatures The system utilizes high-level natural signatures instead of unstructured strings, adopting a structured approach to interfacing with the language model (LM). These signatures are typed function declarations, comprising input and output fields with descriptive names and metadata. The roles of these fields are inferred based on the function each name implies. This allows for transforming them into self-improving, adaptive prompts through bootstrapping.

```
qa = dspy.Predict("question -> answer")
qa(question="Why is Bitcoin so bad at making friends?")
# Out: Prediction(answer='Bitcoin is bad at making friends because it always keeps things
    decentralized!')

class GenerateSearchQuery(dspy.Signature):
    """Write a simple search query that will help answer a complex question."""

    context = dspy.InputField(desc="may contain relevant facts")
    question = dspy.InputField()
    query = dspy.OutputField(dtype=dspy.SearchQuery)

query_gen = dspy.Predict(GenerateSearchQuery)
query_gen(context="Language typology")
# Out: Prediction(question='What are the main types of language classification?', query=' "
    language classification" OR "language typology" -wikipedia')
```

Listing 6.1: Q/A DSPy demonstration. Basic program using the 'Predict' method with a specified signature (top); usage of an advanced search query generation using a class signature (middle); synthetic dataset generation and execution of the signature (bottom).

By using this method, the system can ensure that formatting and parsing logic are handled in an organized manner, thereby avoiding the common pitfalls of direct string manipulation. It provides a framework for optimizing arbitrary pipelines. High-level declarative signatures drive the bootstrapping of comprehensive, multi-stage demonstrations, complete with constraints, enhancing the model's capability to manage complex transformations within the LM.

Modules The system converts traditional string-based prompts, including complex task-dependent techniques such as Chains of Thought (CoT), into a structured system using declarative modules with natural language typed signatures. These modules function similarly to neural network layers and are

designed to abstract any text transformation task. Modules are parameterized to evolve their desired behaviors through iterative bootstrapping of demonstrations within the pipeline.

```python
import dspy

vanilla = dspy.Predict("question -> answer")  # GSM8K Program 'vanilla'
CoT = dspy.ChainOfThought("question -> answer")  # GSM8K Program 'CoT'

class ThoughtReflection(dspy.Module):
    def __init__(self, num_attempts):
        self.predict = dspy.ChainOfThought("question -> answer", n=num_attempts)
        self.compare = dspy.MultiChainComparison('question -> answer', M=num_attempts)

    def forward(self, question):
        completions = self.predict(question=question).completions
        return self.compare(question=question, completions=completions)

reflection = ThoughtReflection(num_attempts=5)  # GSM8K Program 'reflection'
```

Listing 6.2: Three simple DSPy programs: a one-step Predict module (vanilla), a two-step ChainOfThought module (CoT), and a multi-stage ComparerOfThoughts module (ThoughtReflection). These are fully defined by the following code.

This approach allows them to refine their function, integrate demonstrations, and improve both performance and adaptability. Pipelines are constructed by declaring these modules and integrating logical control flows—such as if statements, for loops, and exception handling—which manage how tasks are executed. This structured approach ensures that language models are invoked through these modules, which are supported by tools like compilers and teleprompters to continuously refine and expand their capabilities.

Teleprompter In the system, the compiler refines the program to enhance quality or reduce costs by simulating versions of the program on given inputs and generating example traces for each module. These traces enable the compiler to iteratively bootstrap modules and craft effective few-shot prompts tailored for specific outcomes. The optimization process is managed by teleprompters, employing modular and general-purpose optimization strategies. These include model selection strategies like cross-validation, reinforcement learning (RL), language model (LM) feedback, and Bayesian hyperparameter optimization.

```python
import dspy
from dspy.datasets.gsm8k import GSM8K, gsm8k_metric
from dspy.teleprompt import BootstrapFewShot

class CoT(dspy.Module): # Define the CoT module
    def __init__(self):
        super().__init__()
        self.prog = dspy.ChainOfThought("question -> answer")

    def forward(self, question):
        return self.prog(question=question)

gsm8k = GSM8K() # Load the GSM8K dataset
gsm8k_trainset = gsm8k.train[:10]
config = dict(max_bootstrapped_demos=4, max_labeled_demos=4) # Configure the teleprompter

teleprompter = BootstrapFewShot(metric=gsm8k_metric, **config)
optimized_cot = teleprompter.compile(CoT(), trainset=gsm8k_trainset) # Compile the CoT
optimized_cot(question="Your math question here") # Use the compiled module
```

Listing 6.3: Example of using a DSPy teleprompter to compile a Chain of Thought (CoT) pipeline for solving math problems from the GSM8K dataset.

This toolkit allows teleprompters to optimize across a range of computational contexts. Moreover, the compiler maps declarative modules to compositions of prompts, transforming these into text

transformation graphs similar to imperative computation graphs. This mapping allows for the efficient utilization of the LM's capabilities within computational graphs, ensuring that each operation is executed efficiently.

Achieving LLM agnosticism through self-configurability involves abstracting parameters independent of the underlying model. Within minutes of compiling, the compiler enables GPT-3.5 and LLaMA-2-13B-chat to self-bootstrap pipelines. These pipelines outperform standard few-shot prompting, typically by over 25% for GPT-3.5 and 65% for LLaMA-2-13B-chat. Furthermore, compared to pipelines with expert-created demonstrations, the improvements range from 5% to 46% for GPT-3.5 and 16% to 40% for LLaMA-2-13B-chat.

6.3 Multi-Agent Reasoning Systems

6.3.1 Distributed Problem Solving (DPS)

Recent works such as SayCan [Ahn+22], ToolFormer [Sch+23], HuggingGPT [She+23], Generative Agents [Par+23] and WebGPT [Nak+21] have demonstrated the feasibility of autonomous decision-making agents that are built on top of a large language model core.

Since they rely on massive models with an enormous number of parameters, such approaches have been so far limited to using in-context examples as a way of teaching the agents, since more traditional optimization schemes like reinforcement learning with gradient descent require substantial amounts of compute and time.

Our system, while not as flexible as the dynamic environments seen in *Multi-Agent Systems (MAS)*, essential for adapting to unpredictable and rapidly changing environments, still offers significant benefits in terms of control and dependability for specific tasks like code generation. The rigid structure of our Distributed Problem Solving (DPS) approach, characterised by predefined communication strategies and fixed protocols—namely, reasoning layers— among computing entities, provides a predictable and manageable framework, in contrast to networks of loosely connected autonomous agents. Given the current capabilities of large language models (LLMs) and the project's scope, this setup is ideal for scenarios where precision and stability are crucial.

The previously mentioned *integrated-type system* could substantiate the designation of *Multi-Agent Decision System*, especially in future iterations given additional development. Nevertheless, the project's designated development domain is a *Multi-Agent Reasoning System*, as delineated in in previous planning sections, more accurately described as a *modular-type system*. This modular reasoning architecture, contrasting with decision systems, enables enhanced output generation through the deployment of computational reasoning strata within agent-based frameworks, where decision capabilities are enacted using heuristic methods.

This approach falls under the broader category of Distributed Artificial Intelligence (DAI), specifically within Distributed Problem Solving (DPS). DPS focuses on solving problems by sharing resources and knowledge among numerous cooperating modules or computing entities. In this project, this is achieved by assembling *reasoning layers*, with communication and information pipelines embedded into the design of each computing entity. This rigid structure, due to its embedded strategies, offers limited flexibility yet provides substantial control and predictability, which are crucial for the efficient generation of high-quality code.

Moreover, the modular design of our DPS architecture supports scalability and reusability, allowing for incremental enhancements and seamless integration of new functionalities without disrupting the existing framework.

6.3.2 Compilable Computing Entities

To address constraints of previous methodologies and to better align with the flexibility required for the project, we present compilable modular agents for fact abstraction, benefiting from the inherent flexibility of DSPy as wrapper components for any declared signature.

These pseudo-agents serve as an interface, generalizing the internal workings of an agent to establish a foundational base that can support various reasoning patterns. This not only grants them significant scalability, but also, perhaps due to its simplicity of implementation, establishes a foundation that can accommodate any reasoning method, even if it requires memory or other advanced capabilities.

This implementation involves the creation of a class designed to integrate basic data loading and execution, in order to be integrated into higher pipelines. The agent can dynamically load models to be declared globally, and tasks in order to configure the program to execute when called upon specific data. An interesting feature is that in-situ BootstrapFewShot training is enabled, instead of dropping in an already trained model, which makes it easier to switch, although only created for debugging purposes.

The provided code demonstrates that this modular structure not only simplifies the quick configuration and replication of modules across multiple agents but also enhances the management of complex systems and bolsters the robustness and scalability of deployments. This effectively lays the groundwork for the efficient application of various reasoning patterns.

```python
class PseudoAgent:
    def __init__(self, module=None, load_path=None, model=None, output=None):
        self.module = module() if module else None
        self.load_path = load_path
        self.model = model
        self.output = output

        if self.load_path:
            self.load_task()

        if self.model:
            self.load_model()

    def load_model(self) -> None:
        model_instance = dspy.OpenAI(model=self.model, max_tokens=300, temperature=1)
        dspy.settings.configure(lm=model_instance, max_tokens=1024)

    def load_task(self) -> None:
        with tempfile.NamedTemporaryFile(delete=False) as temp_file:
            json_content = Path(self.load_path).read_text()
            temp_file.write(json_content.encode())
            temp_file.flush()

        self.module.load(Path(temp_file.name))
        Path(temp_file.name).unlink()

    def train(self, trainset, valset, metric) -> None:
        config = {max_bootstrapped_demos: len(trainset + valset)}
        optimizer = teleprompt.BootstrapFewShot(metric=metric, **config)

        self.module = optimizer.compile(
            self.module,
            trainset=trainset,
            valset=valset
        )

    def execute(self, *args):
        return getattr(self.module.forward(*args), self.output)
```

Listing 6.4: `PseudoAgents` implementation: the module highlights the dynamic, adaptive integration of agents, supporting essential operations tailored for high-level directives. Instantiation parameters—module, load_path, model, and output—ensure that agents are configured to effectively respond to diverse computational environments. This design promotes robust, scalable systems capable of complex reasoning and interaction within variable settings.

Note that use of `tempfile` for temporary storage during task configuration minimizes the risk of data persistence issues, ensuring that each task's configuration is loaded in a clean slate environment. Furthermore, the `execute` method's design to dynamically call the `forward` method on the module with variable arguments underscores the system's flexibility to handle different types of data inputs and tasks without requiring predefined methods, thus facilitating easier scaling and modification of agent functionalities.

The DSPy framework orchestrates a highly technical, modular pipeline designed for synthesizing intelligent agent programs, emphasizing precise data processing, dynamic configuration management, and robust program generation employing advanced machine learning techniques.

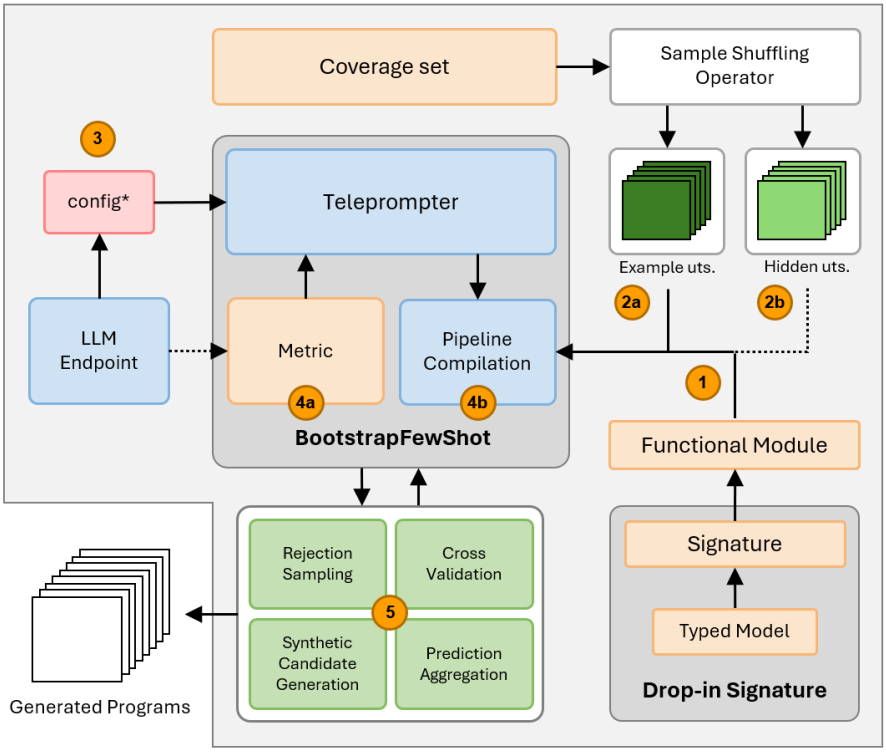


Figure 6.1: **Overview of the sampling approach for program synthesis before inference** (1) The Functional Module processes input using a Typed Model from the Signature block; (2a) Example units and (2b) hidden units are shuffled by the Sample Shuffling Operator, creating a diversified dataset; (3) Configuration settings are sent to the Teleprompter from the config module; (4a) The Metric module evaluates the generated outputs, while (4b) the Pipeline Compilation module integrates various components; (5) Generated programs are validated and aggregated using methods such as Rejection Sampling, Cross Validation, Synthetic Candidate Generation, and Prediction Aggregation. This process is encapsulated in the BootstrapFewShot pipeline.

Data initialization and processing The synthesis process commences with the **functional module**, utilizing a **typed model** grounded in DSPy's signature system. This model ensures accurate data interpretation and validation, aligning input data with predefined operational signatures. The **sample shuffling operator** then enhances dataset diversity by randomly shuffling data entries, thereby safeguarding against overfitting and promoting model robustness across variable data distributions.

Configuration optimization Dynamic configuration within the pipeline is managed by **teleprompters**, which apply real-time adjustments and optimizations based on ongoing performance evaluations. These components fine-tune parameters such as learning rates and hyperparameters, utilizing adaptive algorithms like gradient descent and Bayesian optimization to continuously refine operational settings.

Program synthesis Central to the DSPy synthesis process is the BootstrapFewShot pipeline, which incorporates several advanced machine learning strategies:

- **Rejection sampling** eliminates underperforming iterations through rigorous performance criteria, allowing only optimal solutions to advance.
- **Cross validation** evaluates the model's effectiveness across multiple data segments, ensuring robustness and generalization capabilities.
- **Synthetic candidate generation** augments training data with artificially generated instances, broadening the model's exposure to diverse scenarios.
- **Prediction aggregation** employs ensemble methods to consolidate predictions from various configurations, stabilizing and enhancing output accuracy through methods like weighted averaging or majority voting.

The DSPy pipeline's final stage involves compiling all optimized and validated components into a coherent executable program. This program, capable of performing designated tasks with enhanced efficiency and adaptability, represents the culmination of intensive preprocessing, configuration, and synthesis efforts.

Upon completion, these programs are then seamlessly integrated into the `PseudoAgent` *class*. This integration encapsulates the program within a class framework designed to support diverse reasoning patterns and operational flexibilities, effectively making pseudo-agent a robust platform for deploying the synthesized programs in real-world applications.

Chapter 7

MaLB-SC Generation Module

Language models (LMs) have recently begun to transform the landscape, enabling the development of natural language processing (NLP) systems that facilitate higher levels of abstraction while reducing data requirements [Bom21]. Previous work primarily focused on specialized approaches for improving reasoning [Gev20], [Ran19], [And19]. However, emergent abilities of LLMs have significantly enhanced prompting techniques, allowing for their adaptation to new tasks [Koj22]. Yet, while these advancements have improved systematic reasoning [Wei+22], [Wan+22], many of these techniques have been explored in isolation.

There is now a growing interest in constructing multi-stage pipelines and agents that decompose complex tasks into more manageable queries for LMs, thereby optimizing their performance [Qi+19], [Kha+22], [Doh+22], [Kho22], [Che22], [Pou23], [Shi23]. In this context, the development of MaLB-SC (Multi-Agent LLM-Based Smart Contract Generation Module) exemplifies this trend by experimenting with reasoning layers to facilitate the generation of Solidity protocols, employing a modular architecture to enhance controllability over the generated outputs.

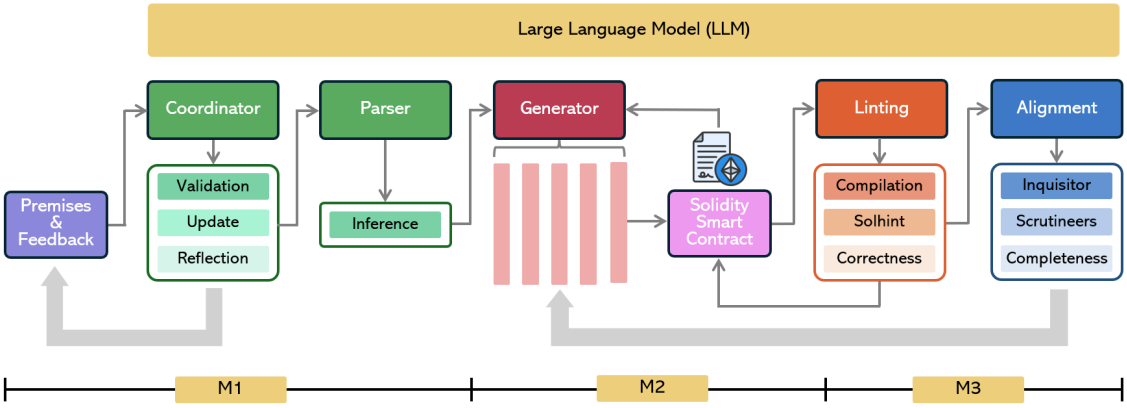


Figure 7.1: MaLB-SC Generation Module Workflow (SuAV-hinting [7.3])

The MaLB-SC system begins with the *Human-Machine Interaction Module*, which actively engages with users to extract detailed specifications for the desired smart contracts, as illustrated in Figure 7.1. This user input is then passed to the *Syntax Generation Module*, where various reasoning methodologies are tested and applied to create the initial contract code. Next, the *Alignment Module* performs a quick *linting* process to ensure the generated code is syntactically correct. Following this, the module uses a feedback loop to provide a control signal, progressively aligning the code with the original user specifications. This iterative process ensures that the final output not only adheres to syntactic correctness but also faithfully represents the user's requirements.

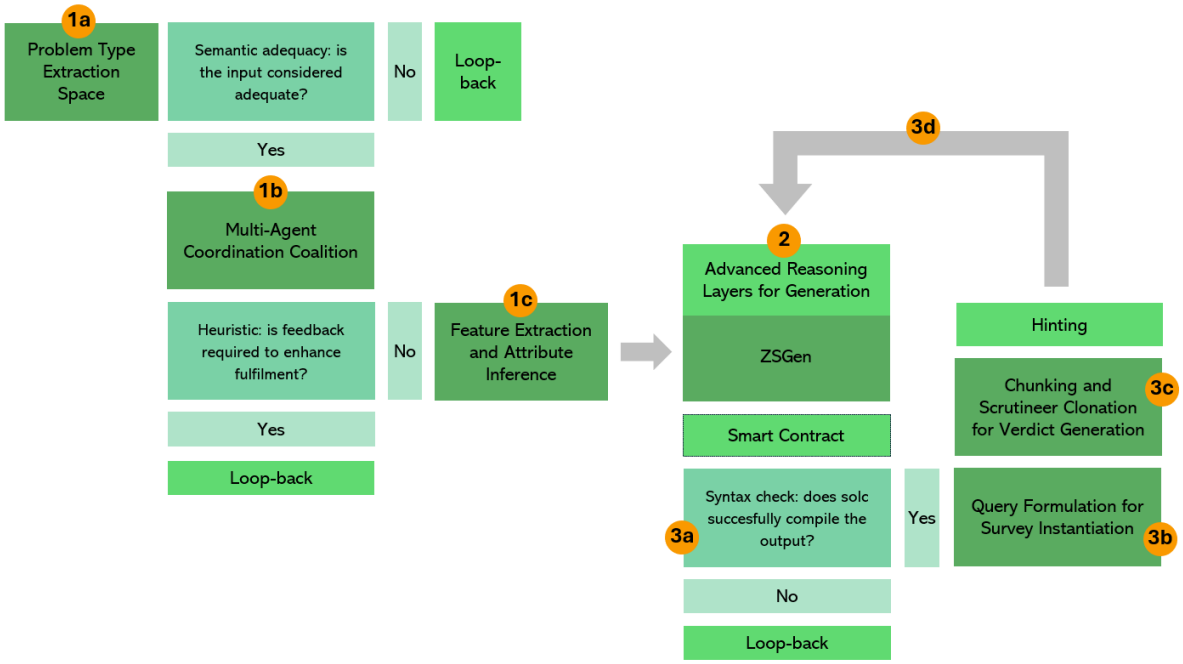


Figure 7.2: Functional Workflow Schema (SuAV-hinting [7.3])

~/InteractionApp The MALB-SC system initiates its process with the development of an attribute-based requirements table in the *Problem Type Extraction Space* (**1a**), where the semantic adequacy of the input is verified. This table encapsulates the smart contract specifications as defined by the user, detailing essential attributes necessary for the contract's functionality. The system incorporates a feedback loop, generating targeted questions and requesting additional clarification from the user, ensuring a coherent and refined features table that enables consistent generation of the required specifications. Once validated, the system transitions to the *Coordination coalition* (**1b**) to determine if heuristic feedback is necessary for further refining user requirements. Subsequently, the *Feature Extraction and Attribute Inference* (**1c**) stage analyzes and infers key attributes, preparing for context-aware code generation (*Parsing Module*).

~/ModGen Once these foundational steps are complete, the system moves to its core functionality within the *Syntax Generation Module* (**2**). Here, advanced techniques are employed to generate the initial smart contract code. This code is then subjected to a syntax check (**3a**) to ensure successful compilation. If any issues arise, a loop-back mechanism initiates re-evaluation and refinement, preserving code integrity. To further validate the generated code, the system utilizes SUAV (see section on alignment), comprising *Query Formulation for Survey Instantiation* (**3b**) and *Chunking and Scrutineer Clonation for Verdict Generation* (**3c**), ensuring the code's correctness and alignment with user specifications. In cases where further refinement is needed, *Hinting* (**3d**) provides feedback and guidance, facilitating iterative improvements and maintaining high fidelity to the user's original expectations. This interconnected, iterative approach ensures comprehensive and precise smart contract generation.

The subsequent sections will outline the functional aspects of the system, including data flow, the relationships between various components, and the specific implementation strategies used for reasoning layers and methods, providing a detailed examination of the system's architecture and operational dynamics.

7.1 Human-Machine Interaction Module

The first module focuses on human-machine interaction, iteratively refining smart contract descriptions based on user feedback. These agents parse the refined descriptions into actionable features, which are then passed to the generation module, all within the confines of a local Streamlit session.

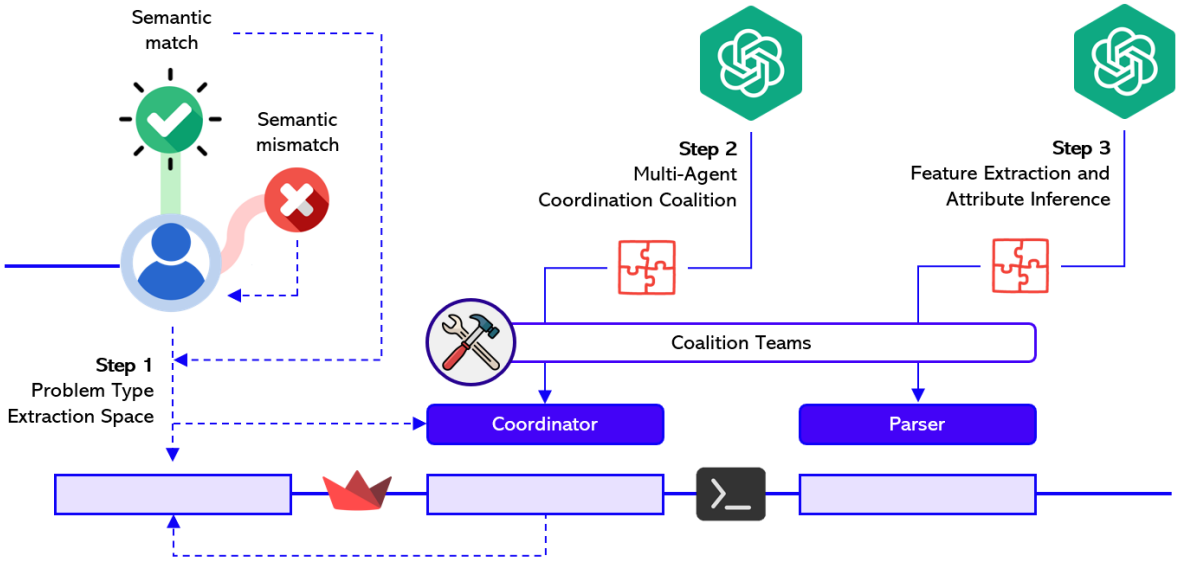


Figure 7.3: Minimal MVP Architecture

7.1.1 Coverage Sets

Some agents needed previous training to better align the output with expected results, as some tasks are marked by a diverse array of possible outcomes and a limited tolerance for deviation from expected results, especially in classification. Thus, JSON coverage sets were produced with Sony's approval to train and validate the behavior of certain agents (as explained in subsection Compilable Computing Entities).

First Set consists of an array of descriptions labeled as invalid due to not meeting the minimum requirements expected from an initial user approach to the problem.

It includes:

- **Commercial descriptions of blockchain projects:** examples of how smart contract technology is applied across various industries to enhance business processes, management, and user or consumer engagement.
- **Isolated features:** well-explained but insufficiently comprehensive to represent an entire contract.
- **Scholarly literature:** theoretical explanations or formal definitions that might be found in academic or specialized literature on blockchain and smart contracts.
- **Nonsensical text:** entries containing senseless repetition or writing errors that do not provide useful or relevant information.
- **Evasion attacks:** deceptive inputs intended to manipulate the model causing misclassification.

Second Set contains valid descriptions, representative of the expected input distribution, and structured breakdowns of their features, each characterized with the following attributes: `id`, `name`, `scope`, `input`, `constraints`, `output`, `primary_scenario`, `alternative_scenario`.

Premise: Managing tiered VIP memberships, allowing fans to upgrade or downgrade their membership based on the tokens they hold. Define membership levels as Bronze, Silver, and Gold, each with associated perks. Upgrading requires the owner to have a sufficient number of additional tokens (100 tokens for Silver, 200 for Gold). Include functionality for members to voluntarily lower their level, refunding tokens proportionally based on their tenure on the platform (70% if >2 years, 50% if >1 year, 30% if >6 months). Once downgraded, a member must wait 12 days before upgrading again.

```
{
    "id": "REQ01",
    "name": "Tiered VIP Membership Management",
    "scope": "Managing tiered VIP memberships using an ERC-721-based smart contract where memberships are categorized into Bronze, Silver, and Gold levels.",
    "input": "Membership upgrade or downgrade request, member's current token count and membership level.",
    "constraints": "Memberships are strictly categorized into three levels with specific token requirements for each.",
    "output": "Member's new membership level following a successful upgrade or downgrade.",
    "primary_scenario": "Member requests an upgrade and has the necessary tokens for the next level, which is then processed successfully.",
    "alternative_scenario": "Member requests an upgrade without sufficient tokens, and the request is denied."
},
{
    "id": "REQ02",
    "name": "Membership Upgrade Requirements",
    "scope": "Defining and enforcing token requirements for upgrading membership levels among Bronze, Silver, and Gold.",
    "input": "Member's current membership level, number of tokens held, upgrade request.",
    "constraints": "100 additional tokens needed for Silver, 200 for Gold. Member must not have downgraded within the last 12 days.",
    "output": "Approval or denial of the membership upgrade based on token count and upgrade cooldown.",
    "primary_scenario": "Member meets the token requirement and cooldown period, successfully upgrades membership.",
    "alternative_scenario": "Member does not meet token requirement or is within cooldown period, and the upgrade is denied."
},
{
    "id": "REQ03",
    "name": "Voluntary Membership Downgrading",
    "scope": "Allowing members to voluntarily downgrade their membership level and receive a token refund based on their tenure.",
    "input": "Member's current level, tenure on the platform, downgrade request.",
    "constraints": "Token refund rates: 70% if tenure >2 years, 50% if >1 year, 30% if >6 months. Downgrade is followed by a 12-day upgrade cooldown.",
    "output": "Number of tokens refunded and new membership level.",
    "primary_scenario": "Member requests a downgrade and is eligible for a refund based on tenure; tokens are refunded accordingly.",
    "alternative_scenario": "Member requests a downgrade but does not meet the minimum tenure for a refund; downgrade proceeds without refund."
},
...
```

Listing 7.1: Second coverage set, at $\sim$/data/smart$_c ontract_d escriptions/data.json$

The previous template has been adapted from the VACCAs [Vac+23] functional suitability model, encompassing requirements for smart contracts. As acknowledged by the paper, there are no specific methodologies or tools in the literature to facilitate the writing of requirements for smart contracts. In our study, this format will be utilized by reasoning agents to construct smart contract code. The requirements delineate the behavior of a software system, its essential properties, and the constraints that the system must satisfy. In software engineering, a template is employed to specify requirements, necessitating the identification of the contract's purpose. For each requirement, the name, purpose,

constraints, input/output values, and potential usage scenarios are meticulously specified.

Sets are not directly used in Python files (.py). During development, the practice of developing the agents' behavior in the modular environment of Jupyter files (.ipynb) was adopted, located in the src/Modules/RP/modules_fabric folder. This approach simplifies testing and allows for more dynamic library loading. Additionally, we created a class to streamline data access, as the Second Set is notably used to train different agents and requires post-processing before being utilized.

7.1.2 Agent Coalitions

In order to craft the description and address the attributes generation, agents in this first module are divided into two different coalitions: `CoordinationCoalition` (at src/Modules/RP/coordinator.py) and `ParsingCoalition` (at src/Modules/RP/parser.py) patterns. Both contain instances of Pseudo agents (see Compilable Computing Entities subsection for definition), initialized with model characteristic and permission keys, which are prompted to perform specific tasks within the team for different interventions upon user feedback.

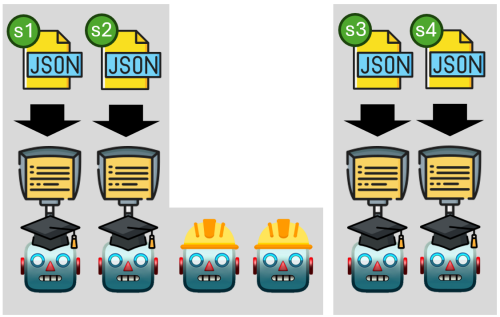


Figure 7.4: Coalition training correlations

`CoordinatorTeam` contains agents for validating the topic, generating questions, updating the descriptions and reflecting, tasks executed in coordination with the sequential logic of the module.

Each agent is assigned a distinct module responsible for a particular task, a load path to load its configuration (if applicable), and an output parameter that defines the type of result it will produce. This setup ensures that agents can leverage the shared model while maintaining unique functional roles within the system

```
    self.ValidateTopic = PseudoAgent(
        module=ValidateTopic,
        load_path="Modules/RP/modules_fabric/M11_ValidateTopic_opt.json",
        model=self.model,
        out="boolean_assessment"
    )

    self.GenerateQuestions = PseudoAgent(
        module=GenerateQuestions,
        load_path="Modules/RP/modules_fabric/M4_GenerateQuestions_opt.json",
        model=self.model,
        out="questions"
    )

    self.UpdateDescription = PseudoAgent(
        module=UpdateDescription,
        model=self.model,
        out="new_description"
    )

    self.Reflect = PseudoAgent(
        module=Reflexion,
        model=self.model,
        out="insights"
    )
```

Listing 7.2: Coordinator agents instantiation: ValidateTopic, GenerateQuestions, UpdateDescriptions & Reflect, from the directory /src/coalitions/coordinator.py (2/4 bootstraping coverage sets)

`ParsingTeam` employs a two-agent configuration: one agent is responsible for deriving requirements from the selected descriptions, while the second agent utilizes these requirements in conjunction with the original descriptions to generate a structured set of attributes. This orchestrated workflow results in a comprehensive blueprint of the features and, ultimately, a detailed blueprint of the designed contract.

```
    self.InferRequirements = PseudoAgent(
        module=InferRequirements,
        load_path="Modules/RP/modules_fabric/M2_InferRequirements_opt.json",  # Assuming a
            config or model path if required
        model="gpt-3.5-turbo-0125",  # Assuming model parameter if required
        out="requirements"  # Output type as expected from the module
    )

    self.GenerateAttributes = PseudoAgent(
        module=GenerateAttributes,
        load_path="Modules/RP/modules_fabric/M31_GenerateAttributes_opt.json",  # Assuming a
            config or model path if required
        model="gpt-3.5-turbo-0125",  # Assuming model parameter if required
        out="structured_requirement"  # Output type as expected from the module
    )
```

Listing 7.3: Parser agents instantiation: InferRequirements & GenerateAttributes, from the directory /src/coalitions/parser.py (2/2 bootstraping coverage sets)

7.1.3 Methodology and Testing

To accurately benchmark the agents' behavior, the aforementioned coverage sets were created, with almost 300 test units: 239 attributes, 30 prompt descriptions, and 20 sampled questions.

The training sessions were conducted using 40% of the data for validation and the remaining 60% for training.

In the case of the `ValidateTopic` agent, various prompts were tested to distinguish valid starting prompts from invalid ones. The results were as follows.

TP	TN	FP	FN
6±2	4±4	18±4	2±2

Table 7.1: Confusion Matrix Metrics

The accuracy of this approach is calculated as $\frac{TP+TN}{TP+TN+FP+FN}$, resulting in a range from 26.67% to 33.33%.

However, when compiling the module with the `DSPy` teleprompter, a surprising replicability of 82.12% was achieved in the first round, and 92.7% to 100% in subsequent rounds, making the most of prompt engineering techniques [Wan+23].

Subsequent training sessions yielded similarly high replicability rates, affirming the robustness of the `DSPy` teleprompter and the effectiveness of the applied methodology, even when employing an LLM-as-judge approach[1] for defining parsing team's metrics.

7.1.4 Feedback Loop & Workflow Integration

When a Streamlit application is executed, it spins up a local server that hosts the web interface, dynamically serving the application to the user via HTTP. The application state is managed dynamically, with automatic re-execution of the script from top to bottom on each user interaction, utilizing WebSocket

[1] LLM-as-Judge uses a strong LLM to evaluate other LLMs' outputs. Despite initial community skepticism, it shows decent correlation with human judgments when well-implemented, like self-calibration [Kad23], reversing chain-of-thought (RCoT) [**xue2023reversing**] or chain of verification (COVE) [Dhu+23]. Indeed, the SuAV system (set to be introduced in Section **??**) falls under this category

connections to maintain a persistent, low-latency communication channel between the client and server for real-tim synchronization.

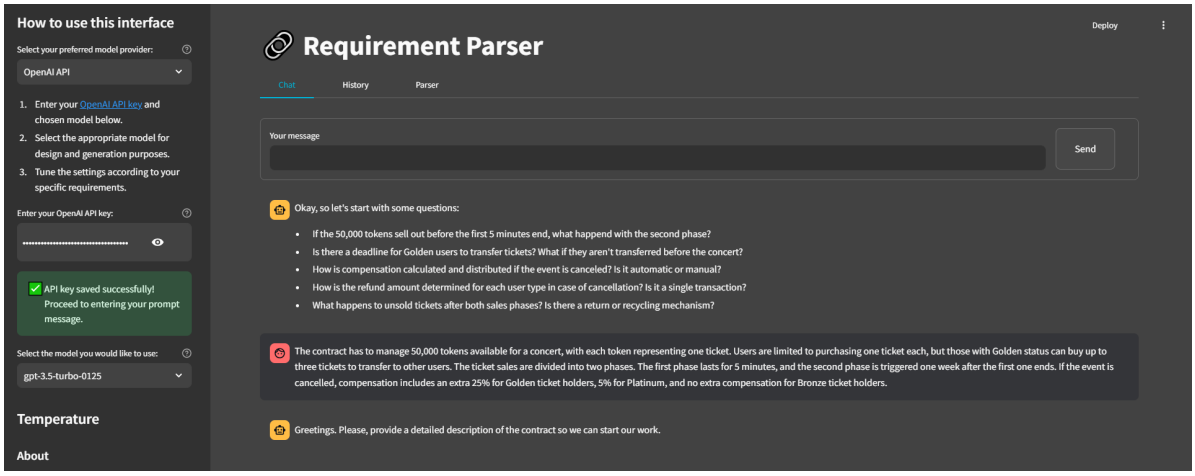


Figure 7.5: Streamlit Session First Tab.

For MALB-SC's application, I developed a sidebar feature to set an API key for OpenAI, sufficient for testing purposes. When the user inputs the API key into the sidebar, it is transmitted from the Streamlit frontend to the server-side via secure HTTPS, ensuring encrypted communication. This API key is then used to authenticate requests against OpenAI's servers. The server-side script includes the API key within the Authorization header of the HTTP request payload, dispatched to the OpenAI API endpoint. Upon receipt, the OpenAI server performs key validation through token verification. If the API key is valid, an authenticated session is established, allowing the application to interact with OpenAI's language models. This enables real-time communication with the LLMs in the interaction area, where the user will be prompted to insert a first approach to the smart contract design.

The topic validation, denoted as *ValidateTopic*, is trained on the initial coverage set and ensures that the initial description D_0 is indeed a technical description of Smart Contracts and not vaguely related to blockchain or some form of evasion attack.

If the description D_0 is deemed invalid, the session will return an error until a valid description is provided. This step is crucial, as it establishes the foundation for the subsequent steps to operate within the appropriate context.

At each interaction step t, the system generates a set of questions $Q_t = \{q_{t1}, q_{t2}, \ldots, q_{tn}\}$ designed to elicit informative responses from the user. Based on the user's responses $A_t = \{a_{t1}, a_{t2}, \ldots, a_{tn}\}$, the system updates the description to $D_{t+1} = \mathcal{R}(D_t, A_t)$. This iterative process continues until the description D_t reaches a threshold[2] τ of completeness and accuracy, as determined by a scoring function $S(D_t) \geq \tau$:

Upon reaching this threshold, the refined description D_f is used to generate a set of structured features F. Each feature $f_i \in R$ is an abstract representation of a task derived from D_f, encapsulating multiple attributes $\mathbf{a}_i = \{a_{i1}, a_{i2}, \ldots, a_{im}\}$. The generation of these features is modeled as $\mathcal{G}(D_f) = \{f_1, f_2, \ldots, f_k\}$, with each attribute within a feature f_i being defined as $\{a_{i1}, a_{i2}, \ldots, a_{im}\}$

[2]The threshold for early termination is heuristically set, indicating the final smart contract description is reached once a specific token length is met. This design decision arises from the understanding that models, even when provided with exemplars, frequently struggle to determine when sufficient detail has been achieved. Larger models exhibit enhanced capabilities in handling incorrect or unrelated labels [Wei+23], yet they tend to overlook abstract evaluation metrics due to their inherent nature as instruction-tuned systems optimized for responsiveness. Consequently, the most viable approach to ensuring quality within the context of this project is through the application of rule-based systems. Although more extensive fine-tuning could potentially address this issue, it falls outside the project's current scope. Hence, for the proof of concept, employing a heuristic threshold based on token length is an adequate solution.

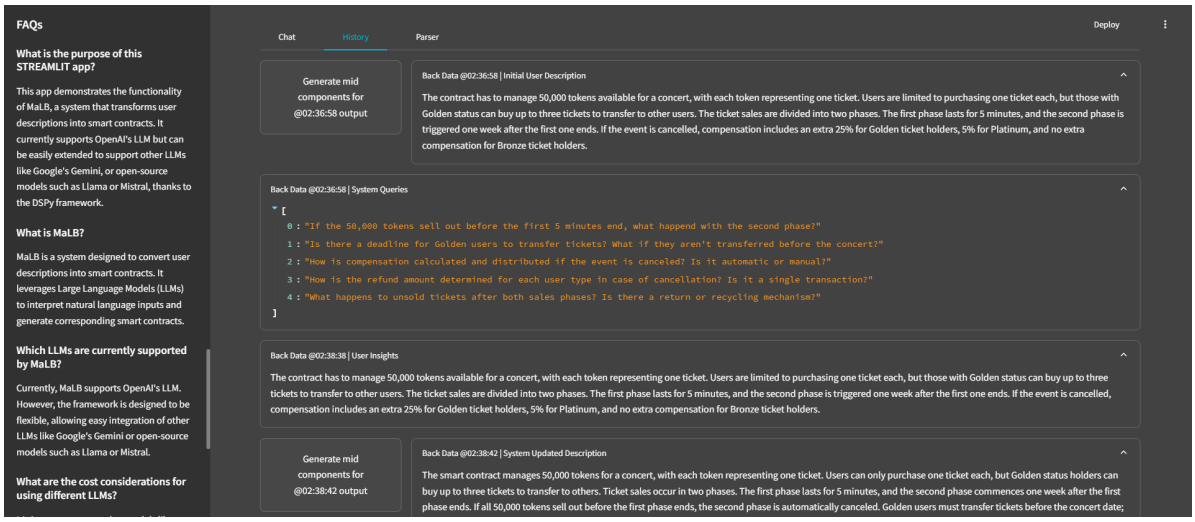


Figure 7.6: Streamlit Session Second Tab.

At this stage, the session notifies the user, who can then proceed to the subsequent tab to view a comprehensive history of the interaction (Figure 7.6), including metadata related to the processes executed by the agents. This interaction history encompasses the various iterations of the descriptions expanded based on user feedback, each accompanied by an actionable button, timestamp, and label, intuitively structured for ease of use.

Upon user activation, the Parsing team, leveraging its trained capabilities, generates the characterization of the smart contract, which is then displayed in the final tab (Figure 7.8). The requirements $F = \{f_1, f_2, \ldots, f_k\}$ and their associated attributes $\mathbf{a}_i = \{a_{i1}, a_{i2}, \ldots, a_{im}\}$, derived from the refined description D_f, are displayed as output. This output can be viewed in sections, highlighting different features separately or as a unified document.

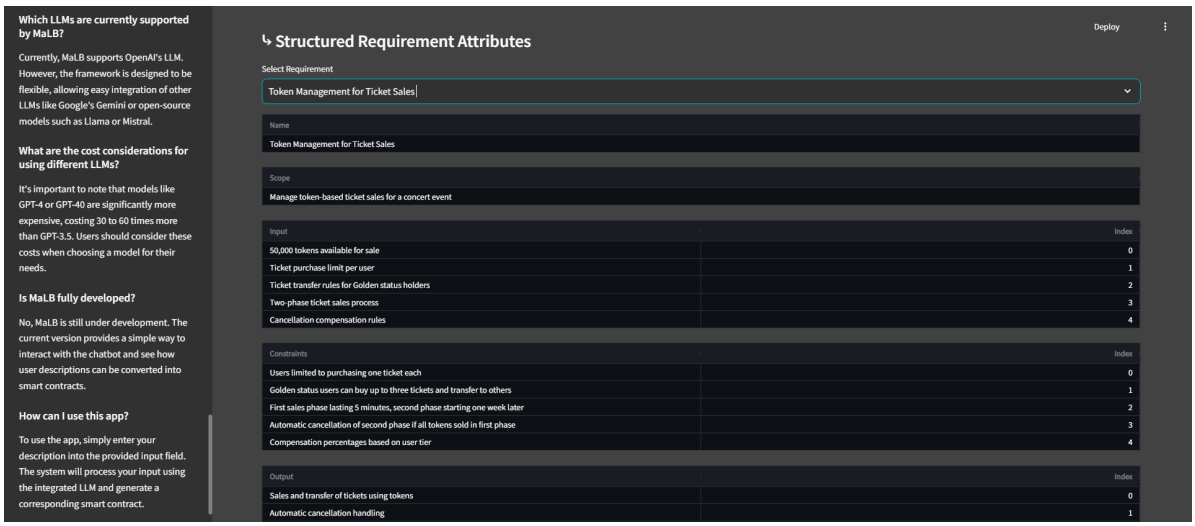


Figure 7.7: Streamlit Session Third Tab.

To enhance usability and structure, the system combines *Pydantic*'s data validation and parsing capabilities with Python's advanced formatting features. This results in outputs with clear labeling, automatic field names, and hierarchical structuring, making it easy to work with the results.

7.2 Compilation and Linting

In software development, linting is an essential process that involves static code analysis to identify potential errors, stylistic issues, and adherence to coding standards. For smart contract development, linting ensures that the code not only functions correctly but also adheres to security and style guidelines, which is crucial given the immutable and public nature of blockchain contracts.

The MALB-SC system uses the *Solidity compiler* (SOLC) version 0.8.4 to verify the functional correctness of generated smart contracts. The compilation process involves translating the high-level Solidity code into low-level bytecode that can be executed on the Ethereum Virtual Machine (EVM). During this process, the compiler checks for syntactic and semantic errors, ensuring that the contract code is executable and reliable.

7.2.1 SolcX Library

In this work, the generation of standalone Smart Contracts from descriptions and intermediate structured features is emphasized, followed by an automatic evaluation of the code samples' correctness. This contrasts with natural language generation, where evaluations are typically heuristic or conducted by human evaluators.

The Solidity compiler can be used as a library in Python, which makes it compatible with both Windows and Linux distributions. This flexibility ensures that the core functionality of the MaLB-SC system remains decoupled from any specific operating system, thereby maintaining an OS-agnostic integration.

The compilation results in a log file for the smart contract, which includes fields for `success`, `errors`, and `ABI` (Application Binary Interface).

```
{
    "success": true,
    "errors": null,
    "abi": [
        {
            "inputs": [
                {
                    "internalType": "uint256",
                    "name": "_auctionDurationMinutes",
                    "type": "uint256"
                },
                {
                    "internalType": "uint256",
                    "name": "_ticketTokenId",
                    "type": "uint256"
                }
            ],
            "stateMutability": "nonpayable",
            "type": "constructor"
        },
        ...
```

Listing 7.4: Smart Contract Compilation Result: The outcome of compiling a smart contract using the Solidity compiler (Solc) version 0.8.4.

7.2.2 Linting Analysis Tools

Following compilation, the MALB-SC system uses *Solhint*, a popular linting tool specifically designed for Solidity smart contracts.

- **MythX:** offers comprehensive security analysis for Ethereum and EVM-based blockchain smart contracts. It employs a variety of techniques including static analysis, dynamic analysis, and symbolic execution to identify vulnerabilities. *MythX* integrates seamlessly with CI/CD pipelines

and developer tools like *Truffle* and *Remix*, allowing for automated security checks throughout the development process. It provides detailed reports on vulnerabilities, mapped to the SWC Registry, which enhances the accuracy and comprehensiveness of the security assessments.

- **Slither:** is a static analysis tool designed to detect vulnerabilities in Solidity smart contracts. It operates via the command line and is known for its speed and integration capabilities within CI/CD pipelines. *Slither* performs various checks, such as identifying reentrancy, incorrect inheritance, and unchecked return values. It also integrates with other tools like Echidna for more extensive testing, making it a versatile choice for developers aiming to maintain high code quality and security.

- **Solhint:** is a popular linter for Solidity code that emphasizes both code quality and security. It integrates well with various IDEs, including Visual Studio Code, and can be incorporated into CI/CD pipelines. *Solhint* allows for customizable rules, enabling developers to enforce specific coding standards and security practices. The tool checks for common issues such as coding style violations and potential security vulnerabilities, making it a comprehensive solution for maintaining Solidity codebases.

Solhint was selected for this project due to its seamless integration with the Python system pipeline and its extensive linting capabilities tailored specifically for Solidity smart contracts. A custom class encapsulates Solhint's functionality, facilitating automated analysis and integration into the broader MaLB-SC system. This ensures that the code adheres to high standards of quality and security throughout its lifecycle.

7.2.3 Wrapper Classes and Compatibility

Integrating Solhint into the MaLB-SC system requires invoking it via system commands due to its lack of a direct *Python* library, unlike *Solc*. This necessitates handling OS-specific configurations and ensuring the presence of the Solhint executable. The `SolhintAnalyzer` class is initialized with paths to the Solhint executable and configuration files. If these are not provided, the class ensures Solhint is installed by checking for its presence using commands like `where solhint.cmd` on Windows. If Solhint is not found, the class logs an error and attempts to create a default configuration file.

The analysis process involves creating a temporary file to store the Solidity source code, executing *Solhint* via system calls, and capturing the output. This output is parsed into *JSON* format for further processing. Robust error handling mechanisms manage potential issues, such as missing installations and *JSON* decoding errors, ensuring smooth operation across different environments.

Utility methods within the class ensure *Solhint* is correctly installed, set up default configurations if needed, and manage *PowerShell* execution policies on Windows. This careful integration manages OS-specific configurations, process handling, and error management, providing a seamless interface for automated code analysis.

7.2.4 Data Storage and Linting Logs

Solhint performs a thorough analysis of the smart contract code to check for stylistic issues, adherence to best practices, and potential security vulnerabilities. This process is exemplified by compilation logs generated during the analysis, as shown in the following listing:

```
[
    {
        "line": 18,
        "column": 5,
        "severity": "Warning",
        "message": "Rule is set with explicit type [var/s: uint]",
        "ruleId": "explicit-types",
        "filePath": "~/AppData/Local/Temp/tmp56h6qw0a.sol"
    },
    {
```

```
        "line": 21,
        "column": 2,
        "severity": "Error",
        "message": "Line length must be no more than 120 but current length is 140.",
        "ruleId": "max-line-length",
        "fix": null,
        "filePath": "~/AppData/Local/Temp/tmp56h6qw0a.sol"
    },
    ...
```

Listing 7.5: Compilation log example, at ∼/output/compiler/compilation_logs (abbreviated for clarity and ease of display)

This *JSON* format is machine-readable and ideal for automated processing in CI/CD pipelines, enabling continuous integration and deployment workflows. *JSON* is preferred over *YAML* in this context because *JSON* is more widely supported and offers better interoperability with various tools and systems used in CI/CD pipelines.

It's important to note that *Solhint* is used as an intermediate step in the MALB-SC system primarily for ablation studies. Ablation studies involve systematically removing or altering components of a system to understand their contribution to overall performance. In this context, *Solhint* helps analyze how the generated smart contracts evolve over time by checking code quality and adherence to standards at various stages of the development process. This intermediate step is crucial for understanding the impact of different generation methodologies and refining the smart contract generation process.

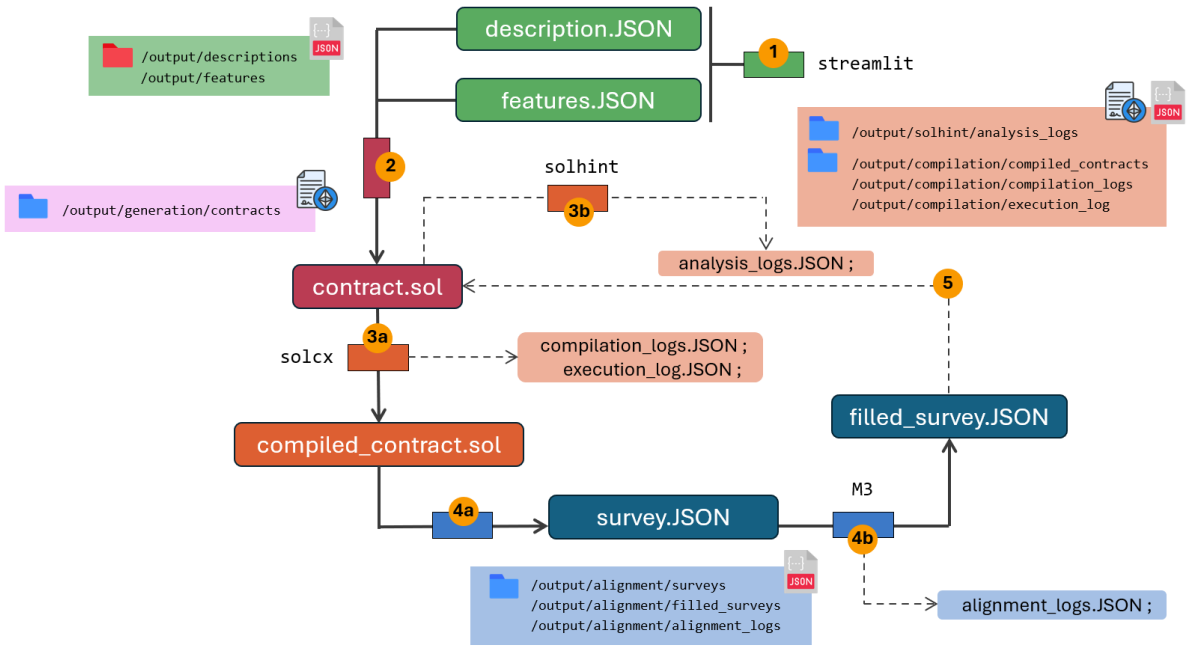


Figure 7.8: The diagram shows the MALB-SC system's data flow. The *Streamlit Interface* (1) gathers user inputs to create `description.JSON` and `features.JSON`. The *Generation Module* (2) uses these to produce `contract.sol`. The *Solc Compiler* (3a, discussed here) compiles it into `compiled_contract.sol`, generating `compilation_logs.JSON` and `execution_log.JSON`. The *Solhint Linter* (3b, discussed here) produces `analysis_logs.JSON` for code style, best practices, and security checks. The *Alignment Module* generates `survey.JSON` (4a) and `filled_survey.JSON` (4b) to ensure alignment with user specs, producing `alignment_logs.JSON`.

7.2.5 DataPipe

At a development level, a class named *DataPipe* was developed to facilitate efficient data management by providing methods to easily set directories and files and load and write content to files with built-in overwrite protection. This made testing and iteration more efficient, ensuring that data was consistently organized and accessible throughout the development cycle. *DataPipe*'s flexibility in handling various file operations allowed for seamless integration of data handling in the broader system architecture, ultimately enhancing the overall workflow and productivity of the project.

Below are instances utilized throughout the project, demonstrating their versatility by handling content and file type or location inputs. They feature flexible read and write methods based on name, extension, or directory index, maintain internal state, and support persistent information storage.

MaLB_bricks.py

function: generate_contracts
DataPipe(self.dirs['contracts'], verbose_policy=verbose_policy)

function: compile_contracts
DataPipe(self.dirs['contracts'], verbose_policy=verbose_policy).fetch_all_files()
DataPipe(self.dirs['compilation_logs'], verbose_policy=verbose_policy)
DataPipe(self.dirs['compiled_contracts'], verbose_policy=verbose_policy)
DataPipe(self.dirs['execution_log'], verbose_policy=verbose_policy)

function: analyze_contracts
DataPipe(self.dirs['compiled_contracts'], verbose_policy=verbose_policy).fetch_all_files()
DataPipe(self.dirs['analysis_logs'], verbose_policy=verbose_policy)
DataPipe(self.dirs['execution_log'], verbose_policy=verbose_policy)

function: assess_alignment
DataPipe(self.dirs['surveys'], verbose_policy=verbose_policy).dump(survey, extension='json')
DataPipe(self.dirs['compiled_contracts'], verbose_policy=verbose_policy).load(n_compiled_smart_contract)
DataPipe(self.dirs['filled_surveys'], verbose_policy=verbose_policy).dump(filled_survey, extension='json')
DataPipe(self.dirs["alignment_logs"], verbose_policy=verbose_policy).dump(log_data, extension='json')

run_STTMD.py

function: run_compiled_contract_lengths_stats
DataPipe(ablation_thread.dirs['compiled_contracts'], verbose_policy=0).fetch_all_files()

function: run_compiled_contract_lengths_distributions
DataPipe(thread.dirs['compiled_contracts'], verbose_policy=0).fetch_all_files() for thread in multi_plotting_threads
DataPipe(ablation_thread.dirs['compiled_contracts'], verbose_policy=0).fetch_all_files()
DataPipe(ablation_thread.dirs['analysis_logs'], verbose_policy=0).fetch_all_files()

function: run_compiled_contracts_warning_plots_1
DataPipe(thread.dirs['compiled_contracts'], verbose_policy=0).fetch_all_files() for thread in multi_plotting_threads
DataPipe(thread.dirs['analysis_logs'], verbose_policy=0).fetch_all_files() for thread in multi_plotting_threads
DataPipe(thread.dirs['compiled_contracts'], verbose_policy=0).fetch_all_files() for thread in multi_plotting_threads

---

This structured presentation of function calls divides into two main programs located at the project root: `MALB_bricks` and `run_STTMD`. The former packages functionalities into actionable modules or "bricks," which are used to articulate and test the latest version of the pipeline. The latter, `run_STTMD`, is responsible for converting the output into parameterized data analysis, aiding in the understanding of changes implemented using the bricks and assessing the global system performance.

7.3 SuAV: A Novel Technique for Alignment Assessment

7.3.1 Overview

Previous components systematized functional correctness assessment, yet functional completeness, defined as ensuring the alignment with nuanced requirements and specified user expectations, remains unaddressed. The *Alignment Module* demonstrates that AI's capabilities extend beyond generative tasks to evaluative and collaborative functions. In this module, a novel technique is introduced to fill this gap: *Survey Augmented Generation for Validation* (SuAV). SuAV integrates principles of collaborative evaluation and systematic feedback to enhance the reliability and completeness of smart contracts, ensuring that they fully meet user expectations.

The development of SuAV is inspired by recent advancements in self-criticism methods such as Self-Calibration [Kad+22] and Self-Refine [Mad+23], which emphasize internal feedback mechanisms. These methods contrast with techniques like Reversible Chain-of-Thought (RCoT) [Xue23], Chain of Verification (CoVe) [Dhu+23], and Cumulative Reasoning [Zha+23], which focus on evaluating and verifying solutions through external scrutiny. Building on these foundations, SuAV employs a unique dual-agent framework designed to systematically assess and validate smart contracts.

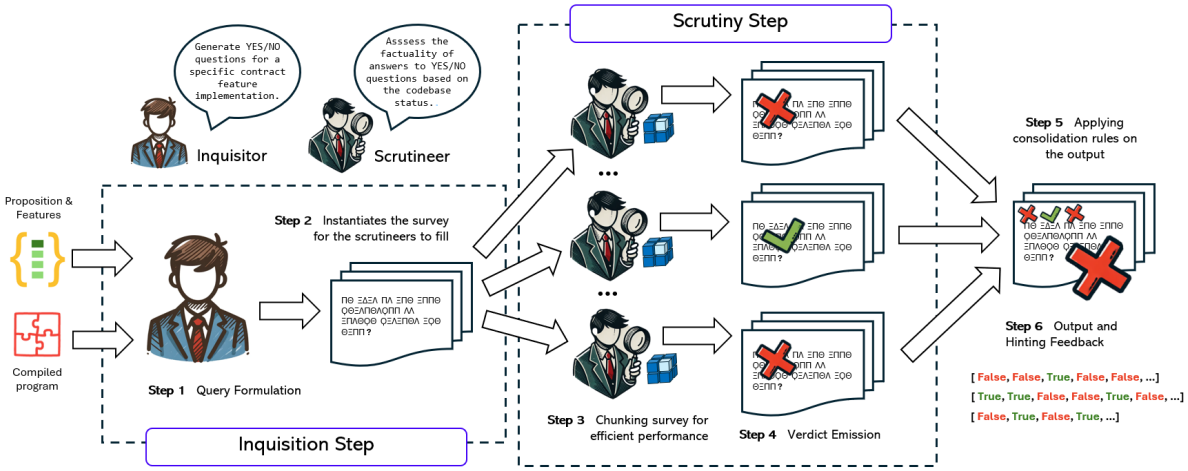


Figure 7.9: Survey Augmented Generation for Validation (SuAV)

SuAV operates through a structured process characterized by the involvement of an inquisitor and a panel of scrutineers. The inquisitor initiates the process by generating augmented surveys that consist of targeted validation queries, meticulously formulated based on the initial user description and features set. These queries are subsequently subjected to a review process by scrutineers. Each one independently validates the queries, ensuring a high degree of thoroughness. Following the concurrent validation, the scrutineers collaborate to reach a consensus on the ratings, which serves as the basis for the final verdict on the alignment.

Coverage set: as Figure 7.9 illustrates, the DSPy library is utilized once again for training the inquisitor to generate pertinent questions based on the contract. This ensures the questions remain within the scope of Y/N queries, which can be easily answered through boolean scrutineer output structures. Consequently, the module includes a coverage set, similar to the interaction coalitions ((see Section 7.1)), for training the inquisitor on effective query formulation.

```
{
    "Proposal creation by members": [
        "Can only members create proposals?",
        "Is each proposal assigned a unique ID?",
        "Are the start and end times for voting correctly set?",
        "Is the proposal creation event emitted correctly?"
    ],
    "Voting on proposals": [
        "Can only members vote on proposals?",
        "Is each member restricted to one vote per proposal?",
        "Is the voting period correctly enforced?",
        "Is the vote count correctly updated?"
    ],
    "Execution of approved proposals": [
        "Can only the admin execute proposals?",
        "Is the proposal execution restricted to after the voting period?",
        "Is the quorum requirement enforced before execution?",
        "Is the proposal execution event emitted correctly?"
    ],
    ...
```

Listing 7.6: Third coverage set, at ∼/data/insights/data.json (abbreviated for clarity and ease of display)

7.3.2 Mathematical Definition

The Survey Augmented Generation for Validation (SuAV) algorithm is designed to ensure the quality and adherence of generated smart contract code to specified features and attributes through a comprehensive survey and validation process. The algorithm begins by defining and initializing key symbols and data structures. D_f represents the final description of the smart contract, providing a textual foundation. The set of features $F = \{f_1, f_2, \ldots, f_m\}$ captures distinct functionalities or characteristics of the smart contract, while $A_i = \{a_{i1}, a_{i2}, \ldots, a_{in}\}$ represents the attributes associated with each feature $f_i \in F$.

The survey S is constructed from these features and attributes using the function $\mathcal{G}$, which generates individual queries Q_{ij} based on D_f, f_i, and A_i. Each generated query Q_{ij} is appended to the survey set $S[f_i]$, organizing the survey questions according to their respective features.

To manage performance, the survey is configured into chunks of size N and distributed among K scrutineers. The configuration function $\mathcal{C}$ handles this partitioning and initialization of scrutineers. Each scrutineer s_k is instantiated and assigned a set V_k to store their verdicts.

The algorithm iterates over each chunk c of survey questions, defined as $Q_c = \{Q_{ij} \in S \mid (c-1)N \leq j < cN\}$, and uses the scrutineer function $\mathcal{S}$ to evaluate the chunk against the generated smart contract code C. The resulting verdicts V_{kc} from each chunk are appended to the scrutineer's set V_k, and ultimately, V_k is appended to the overall verdicts V.

The consensus rating C_r is derived by aggregating all scrutineer verdicts V using the aggregation function $\mathcal{A}$, followed by refinement through the function $\mathcal{R}$ to ensure a coherent consensus. For each query q in the consensus rating C_r, the algorithm checks if any scrutineer has marked it as False. If so, the query q is appended to the set of hints H, providing feedback for improving the smart contract.

The algorithm then calculates two key metrics: True Density (TD) and False-Free Queries (FFQ). True Density TD is the ratio of True responses to the total number of responses in the consensus rating C_r, representing the overall correctness. False-Free Queries FFQ is the ratio of queries with all True responses to the total number of queries in C_r, indicating the reliability of the responses. The final output of the algorithm includes the hints H, True Density TD, and False-Free Queries FFQ, providing comprehensive insights into the quality and adherence of the smart contract code to the specified features and attributes.

Algorithm 1: Survey Augmented Generation for Validation (SuAV)

Symbol: D_f: Final Description
F: Set of Features $\{f_1, f_2, \ldots, f_m\}$
A_i: Set of Attributes for each feature $f_i \in F$ $\{a_{i1}, a_{i2}, \ldots, a_{in}\}$
S: Survey generated from features
Q: Individual Query in the survey
C: Code of the generated smart contract
N: Number of queries per chunk for performance
K: Number of scrutineer clones
V: Verdict from scrutineers
C_r: Consensus Rating from the scrutineers
$\mathcal{G}$: Function to generate survey questions
$\mathcal{C}$: Configuration function for chunking and scrutineers
$\mathcal{S}$: Scrutineer function to assess queries
$\mathcal{A}$: Aggregation function to compile verdicts
$\mathcal{R}$: Refinement function for consensus building
Legend: $\mathcal{H}$: Hints for False responses

Initialize: $S \leftarrow \varnothing$, $C_r \leftarrow \varnothing$;
foreach *feature* f_i *in* F **do**
    $Q_{ij} \leftarrow \mathcal{G}(D_f, f_i, A_i)$
    Append Q_{ij} to $S[f_i]$
C(S, N, K)
Initialize: $V \leftarrow \varnothing$;
foreach *scrutineer* k *in* $\{1, \ldots, K\}$ **do**
    Instantiate scrutineer s_k
    $V_k \leftarrow \varnothing$
    foreach *chunk* c *in* $\{1, \ldots, \lceil |S|/N \rceil\}$ **do**
        $Q_c \leftarrow \{Q_{ij} \in S \mid (c-1)N \leq j < cN\}$
        $V_{kc} \leftarrow s_k.\mathcal{S}(Q_c, C)$
        Append V_{kc} to V_k
    Append V_k to V
$C_r \leftarrow \mathcal{R}(\mathcal{A}(V))$
Initialize: $H \leftarrow \varnothing$;
foreach *query* q *in* C_r **do**
    if $\exists s : q \rightarrow$ *False by scrutineer* s **then**
        Append q to H
Calculate TD as the ratio of True responses to the total number of responses in C_r
Calculate PT as the ratio of queries with all True responses to the total number of queries in C_r
Output: H, TD, PT

7.3.3 Determining the optimal K

In this study, I address a system marked by inherent stochasticity and the heterogeneous behavior of agents. Scrutineers' actions and decisions within this environment do not adhere to a deterministic path, resulting in variability in validation outcomes. Consequently, a statistical approach is imperative to determine the optimal number of scrutineers $(K - s)$ needed to achieve reliable consensus ratings while minimizing variance and associated costs.

To identify the optimal $K-s$ that strikes a balance between accuracy, consistency, and cost-efficiency, I conducted a series of experiments examining the variance across different validation runs and various smart contracts. The coverage set of the first module, with predefined features and attributes (see Section 7.1), was utilized for this analysis.

In our experiments, I varied the number of scrutineers ($K - s \in \{1, 3, 5, 7, 10, 15, 20\}$) and assessed their performance over multiple iterations. For each combination of scrutineers and validation runs, I

recorded the variance in the consensus ratings. The variance analysis considered three key factors:

1. **Inter-Contract variance** ($\sigma^2_{\text{inter-contract}}$): variance in consensus ratings across different smart contracts.
2. **Intra-Run variance** ($\sigma^2_{\text{intra-run}}$): variance within the same run across different scrutineers.
3. **Inter-Run variance** ($\sigma^2_{\text{inter-run}}$): variance across different runs for the same number of scrutineers.

First, each contract was evaluated multiple times (N repetitions) by the same number of scrutineers ($K - s$), capturing the intrinsic variability of each contract and averaging the results over multiple executions. Second, the individual results of each scrutineer in a single run were recorded before combining the results to calculate the final metric. This ensures that the intra-run variance reflects the differences between the scrutineers in a single validation cycle, isolating each result before combination. Third, the entire experiment was repeated multiple times (M runs) for each number of scrutineers ($K - s$). This allows for a robust measurement of inter-run variance.

Table 7.2: Variance Results for Different Numbers of Scrutineers

K-s	$\sigma^2_{\text{inter-contract}}$	$\sigma^2_{\text{intra-run}}$	$\sigma^2_{\text{inter-run}}$
1	0.080	0.109	0.094
3	0.051	0.074	0.067
5	0.042	0.050	0.051
7	0.036	0.043	0.043
10	0.033	0.032	0.034
15	0.020	0.031	0.031
20	0.019	0.031	0.029

The variance metrics are defined and calculated as follows:

Inter-Contract Variance:

For each contract i, the final metric $V_{i,j}$ is calculated in N repetitions. The inter-contract variance is calculated as:

$$\sigma^2_{\text{inter-contract}} = \frac{1}{N} \sum_{i=1}^{N} \left(V_{i,j} - \overline{V}_j\right)^2 \tag{7.1}$$

where $\overline{V}_j$ is the mean of $V_{i,j}$ over all contracts.

Intra-Run Variance:

For each run, the variance between the individual results of the scrutineers s_k is calculated. The intra-run variance is calculated as:

$$\sigma^2_{\text{intra-run}} = \frac{1}{K} \sum_{k=1}^{K} \left(s_{k,j} - \overline{s}_j\right)^2 \tag{7.2}$$

where $\overline{s}_j$ is the mean of the individual results of the scrutineers in a single run.

Inter-Run Variance:

Perform M runs for each K scrutineers and calculate the variance of the final metrics between these runs. The inter-run variance is calculated as:

$$\sigma^2_{\text{inter-run}} = \frac{1}{M} \sum_{m=1}^{M} \left(V_{m,j} - \overline{V}_j \right)^2 \tag{7.3}$$

where $\overline{V}_j$ is the mean of the final metrics across the runs.

Results are summarized above, in Table 7.2.

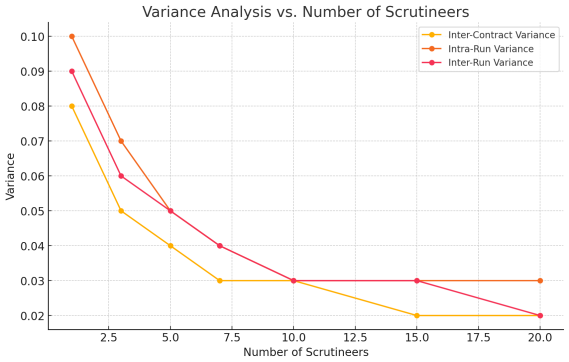


Figure 7.10: All three variance types (*Inter-Contract*, *Intra-Run*, and *Inter-Run*) against the number of scrutineers for a comprehensive comparison.

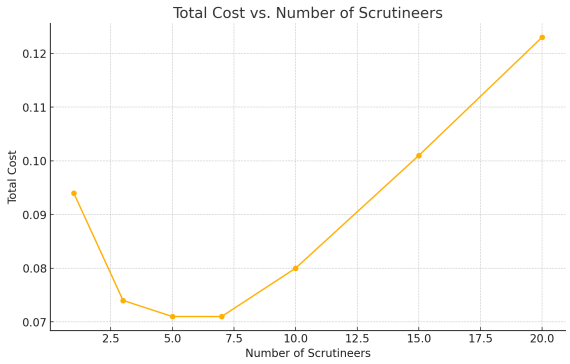


Figure 7.11: Variance reduction and cost of inference combined. Total cost decreases initially but starts to rise again as the number of scrutineers increases due to the cost factor.

The combined weighted variance V is defined as:

$$V = w_c \cdot \sigma^2_{\text{inter-contract}} + w_i \cdot \sigma^2_{\text{intra-run}} + w_r \cdot \sigma^2_{\text{inter-run}} \tag{7.4}$$

where the weights are $w_c = 0.4$, $w_i = 0.3$, and $w_r = 0.3$.

The cost function $C(n)$ increases linearly with the number of scrutineers n:

$$C(n) = 0.01n \tag{7.5}$$

The total cost function $T(n)$ combines the weighted variance and the cost function:

$$T(n) = V(n) + \lambda \cdot C(n) \tag{7.6}$$

where $\lambda = 0.5$ is a scaling factor.

By calculating $T(n)$ for each n, we identify the optimal number of scrutineers n^* that minimizes the total cost function $T(n)$:

$$n^* = \arg\min_n T(n) \tag{7.7}$$

Based on the data, the optimal number of scrutineers is found to be 5.

Algorithm 2: SuAV downstream integration

Symbol: D_f: Final Description
F: Set of Features with Attributes
C: Code of the generated smart contract
H: Hints for improving the smart contract
Legend: $\mathcal{F}$: Final validated and refined smart contract

Initialize: $C \leftarrow \varnothing$, $H \leftarrow \varnothing$, $\mathcal{F} \leftarrow \varnothing$
while *True* **do**
  $C \leftarrow \text{GenerateSmartContract}(D_f, F)$
  `// Apply Survey Augmented Generation for Validation (SuAV)`
  $H, TD, PT \leftarrow \text{SuAV}(D_f, F, C)$
  if $\mathit{IsValid}(TD, PT)$ **then**
    $\mathcal{F} \leftarrow C$
    Break
  `// Feed hints back to the generation module`
  $D_f, F \leftarrow \text{RefineInput}(H, D_f, F)$
Output: $\mathcal{F}$

The algorithm presented above shows how the alignment module (SuAV) is integrated within the broader iterative process of the MaLB-SC system for generating and refining smart contracts. This process begins with the initial generation of the contract based on the final description (D_f) and the set of features (F). The algorithm then invokes the SuAV module, which performs a comprehensive validation of the generated contract through the formulation of survey questions and evaluation by multiple scrutineers.

The output of SuAV, which includes hints (H), true density (TD), and true purity (TP), is used to determine whether the contract meets the validity criteria. If the contract fails to meet these criteria, the hints generated by SuAV are used to refine the input description and features, thus initiating another iteration of the generation process. This loop continues until a contract that satisfies the established validity criteria is produced. In this way, the SuAV module integrates to iteratively refine the smart contract, ensuring alignment with user requirements.

7.4 Ablation Study on Syntax Generation

In-context learning (ICL) enables generative AIs to acquire skills and perform tasks by including examples and pertinent instructions within the prompt, thereby avoiding the need for retraining or weight adjustments. In contrast, MaLB's approach utilizes zero-shot prompting techniques for generation. Incorporating examples, even if carefully selected, would be prohibitively expensive and impractical for large-scale, high-demand systems.

DSPy was applied to the human-interaction and alignment modules, but it wasn't used for code generation. Generating Smart Contract code is significantly more complex and would necessitate extensive datasets of well-matched proposition-syntax pairs. DSPy, relying on few-shot learning and constrained by the context window, is inadequate for this scale of training. Assuming the availability of these datasets, fine-tuning a large language model (LLM) would be more prudent, yet an approach diverging from the original thesis intent —as it embeds the necessary knowledge directly into the model, bypassing the need for human-interaction and alignment modules—.

Consequently, these modules were separated, and LangChain prompting techniques were employed for generation. These techniques leverage the reasoning layers discussed throughout the thesis, ensuring the generation process remains aligned with the established methodologies.

During the evaluation process of the syntax generation module, we employed several advanced language models to ensure the robustness and accuracy of our system. Below, we delineate the models utilized, highlighting their specific roles and the context of their application within our testing framework. It is important to note that for individual trials, we exclusively utilized the gpt-3.5-turbo model to maintain consistency in isolated performance metrics.

- **Llama-3-7b** (using LamaCPP through Python bindings):
  - *Role*: Alternative model for comparative analysis.
  - *Description*: Leveraging LamaCPP, Meta-Llama-3-7b was included to provide a comparative baseline against OpenAI's models. This model's integration facilitated a broader understanding of how different architectures perform in similar tasks, thus enriching our analysis with diverse perspectives on language model efficacy.
- **gpt-3.5-turbo-0125**:
  - *Role*: Primary model for individual trial evaluations.
  - *Description*: This model served as the baseline for initial performance benchmarks. Its stable and reliable outputs provided a consistent reference point against which other models' performances were gauged.
- **gpt-4-1106-preview**:
  - *Role*: Advanced model used in the preview phase for enhanced capability testing.
  - *Description*: As a preliminary version of the GPT-4 series, this model was employed to evaluate the improvements in language understanding, generation quality, and contextual coherence over the 3.5-turbo model. It enabled us to assess the advancements in handling complex queries and generating more nuanced responses.
- **gpt-4o** (state-of-the-art, tested in the last weeks):
  - *Role*: Cutting-edge model integrated towards the end of the evaluation period.
  - *Description*: This model represents the latest in the GPT-4 series, incorporating state-of-the-art advancements in natural language processing. Tested in the final weeks, it was instrumental in pushing the boundaries of our system's capabilities, demonstrating superior performance in terms of accuracy, coherence, and context management.

Due to the rapid iteration cycles inherent in our testing methodology, only the final hybrid version of the system was incorporated into the definitive project documentation. Previous iterations, while briefly evaluated, were subject to minimal testing. These preliminary tests were conducted in a manner that was methodically grounded and formally justified, ensuring that even limited evaluations provided meaningful insights that contributed to the refinement of the final system.

7.4.1 Testbed

In our initial experimental phase, we embarked on a comprehensive evaluation of various prompting techniques, utilizing a consistent testing configuration across all experiments in this section. Our methodology involved iterative testing on all contracts, employing a set of $k = 5$ scrutineers to ensure thorough examination.

Our baseline experiment, conducted without scrutiny-augmented validation hinting feedback, yielded the following metrics.

Table 7.3: Base TD & FFQ results for the baseline study

Backbone	TD	FFQ
Llama-3-7b	0.45	0.01
gpt-3-turbo-0125	0.51	0.03
gpt-4-1106-preview	0.72	0.20

Then, we iterated over the SuAV module to fetch the labels of the queries that contribute to the FFQ parameter. Through three hinting iterations, we obtained the following results. However, the enhancement was minimal and did not meet expectations. Trying other aforementioned LLMs also did not yield significant improvement, discarding the hypothesis that there is a correlation related to the model's capabilities.

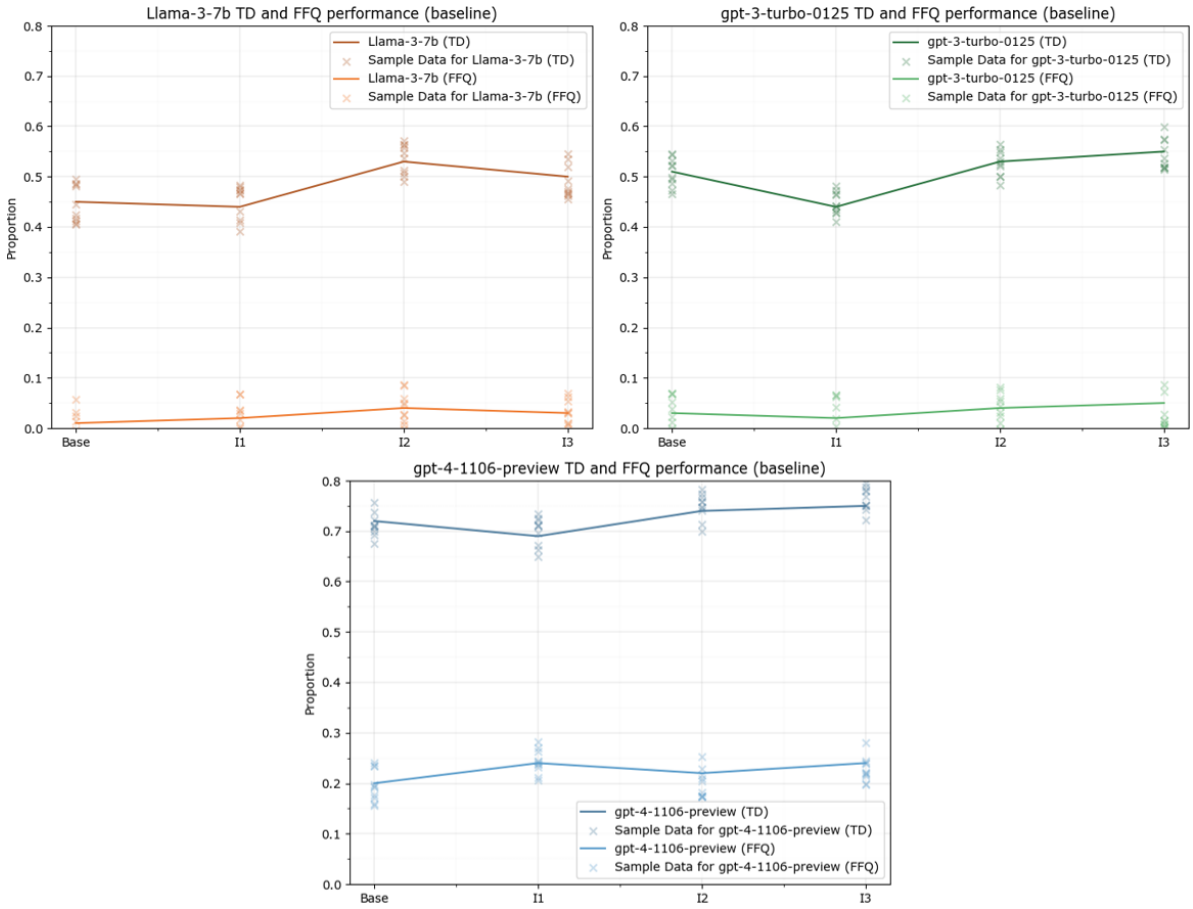


Figure 7.12: Baseline: TD and FFQ over 3-hinting (k=5). Iterations were capped at X3 due to convergence in performance metrics, TD and FFQ, where marginal improvements approached zero, indicating a plateau. This decision, consistently applied across all experiments in this initial phase, aligns with the observed asymptotic behavior, optimizing computational efficiency by avoiding redundant iterations.

The graphics illustrate the proportion of True Density (`TD`) and False-Free Queries (`FFQ`) across three hinting iterations (X1, X2, X3) using the SuAV module. The sample data for `FFQ` and TD are plotted to provide a comparative baseline.

Despite the trend that more advanced models generally yield better results, given the observed minimal improvements in Temporal Difference (TD) scores across the evaluated models (see Figure 7.12), implementing an algorithm to retain the highest-performing smart contract over N iterations would be technically feasible. However, the marginal gains demonstrated in this study—with the most substantial improvement being a mere 17.78% for Llama-3-7b and the least being 4.17% for gpt-4-1106-preview (see Table 7.4)— suggest that the computational overhead and implementation complexity required for such an optimization process would likely outweigh the modest performance benefits.

Table 7.4: TD & FFQ results over SuAV iterations

Backbone	FFQ	TD	Δ(%)
Llama-3-7b	0.01, 0.02, 0.04, 0.03	0.45, 0.44, 0.53, 0.50	0.1778
gpt-3-turbo-0125	0.03, 0.02, 0.04, 0.05	0.51, 0.44, 0.53, 0.55	0.0784
gpt-4-1106-preview	0.20, 0.24, 0.22, 0.24	0.72, 0.71, 0.74, 0.75	0.0417

- TD scores generally increased across iterations for all models, with Llama-3-7b showing the highest relative improvement (17.78%), followed by gpt-3-turbo-0125 (7.84%), and gpt-4-1106-preview (4.17%).
- The performance gap between models is evident, with gpt-4-1106-preview consistently outperforming the other two in both TD and FFQ metrics.
- The improvements in both TD and FFQ were marginal across all models, suggesting that the SuAV hinting process applied on raw prompting has limited effectiveness in enhancing smart contract code alignment.

The initial results obtained using a simple prompt serve as our baseline. The cost-benefit analysis suggests that alternative approaches may warrant consideration.

7.4.2 Meta-Prompting Techniques Evaluation

Building upon our baseline, we evaluate three meta-prompting techniques—RE2, RaR+S2A, and SR—across Llama-3-7b, gpt-3.5-turbo-0125, and gpt-4-1106-preview models. Performance was measured using SH-Eval and SuAV-Eval (FFQ-3-hint) metrics, compared against each models' baseline.

- Rephrase and Respond (RaR) [Den23] instructs the LLM to rephrase and expand the question before generating the final answer. This could all be done in a single pass or the new question could be passed to the LLM separately.
- System 2 Attention (S2A) [Wes23] first asks an LLM to rewrite the prompt and remove any information unrelated to the question therein. Then, it passes this new prompt into an LLM to retrieve a final response.
- Re-reading (RE2) [Xu23] prompts the model into re-reading the question in order to retrieve better results. Although this is such a simple technique, it has shown improvement in reasoning benchmarks, especially with complex questions, just as the original Chain Of Thought.
- Self-Reflection (SR) [RG24] incorporates a step where the LLM evaluates its own response before providing a final answer. This involves the model critically examining its initial output, considering potential errors or biases, and refining its answer accordingly.

Backbone	Strategy	Method	SH-Eval	SuAV-Eval
Llama-3-7b	Prompting	RE2 (Xu et al., 2023)	20.91	0.57
	Prompting	RaR+S2A (Yao & Deng et al., 2023)	20.80	0.50
	Prompting	SR (Renze et al., 2024)	19.41	0.56
	Baseline		**19.11**	**0.50**
gpt-3.5 -turbo-0125	Prompting	RE2 (Xu et al., 2023)	17.27	0.47
	Prompting	RaR+S2A (Yao & Deng et al., 2023)	19.17	0.42
	Prompting	SR (Renze et al., 2024)	17.80	0.54
	Baseline		**17.12**	**0.55**
gpt-4 1106-preview	Prompting	RE2 (Xu et al., 2023)	13.99	0.69
	Prompting	RaR+S2A (Yao & Deng et al., 2023)	15.83	0.61
	Prompting	SR (Renze et al., 2024)	14.80	0.71
	Baseline		**13.39**	**0.75**

Table 7.5: Overall performance of initial meta-prompting on SH and SuAV (TD-3-hint) benchmarks. SH-Eval results for the testbed were gathered during the experiments conducted to obtain the FFQ and TD data in the previous analysis for the GPT-3.5-turbo-0125 model. The SuAV-Eval represents the highest rating of iterations for FFQ.

To assess the effectiveness of each technique, the proportion of successful outcomes (represented by SuAV-Eval scores) was calculated for each method across all models. Chi-square goodness-of-fit tests were then performed to compare observed frequencies with expected frequencies based on the testbed baseline, generating p-values for each technique-model combination. These p-values were combined using Fisher's method to produce a single test statistic for each technique, assessing its overall impact across models.

$$\text{SuAV-Eval} = \chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}; \quad p\text{-values} = \Pr(\chi^2); \quad F(\text{combined}) = -2 \sum \ln(p_i)$$

The results, as presented in Table 7.5, demonstrate only marginal improvements over the baseline:

RE2 showed mixed results across models. It improved SuAV-Eval for *Llama-3-7b* (0.57 vs. 0.50 baseline) but decreased it for *gpt-3.5-turbo-0125* (0.47 vs. 0.55) and *gpt-4-1106-preview* (0.69 vs. 0.75). SH-Eval scores increased for *Llama-3-7b* and *gpt-3.5-turbo-0125*, indicating more errors detected, but decreased for *gpt-4-1106-preview*. The overall impact appears inconsistent across models.

RaR+S2A Did not affect the SuAV-Eval scores for *Llama-3-7b* (0.50 vs. 0.50) and *gpt-3.5-turbo-0125* (0.42 vs. 0.55) compared to their respective baselines, indicating worse performance. However, it slightly improved for *gpt-4-1106-preview* (0.61 vs. 0.75). SH-Eval scores worsened for all models, particularly for *gpt-3.5-turbo-0125* (19.17 vs. 17.12 baseline). The technique shows varied effectiveness across different model sizes.

SR demonstrated more consistent performance improvements. It improved SuAV-Eval for *Llama-3-7b* (0.56 vs. 0.50) and *gpt-3.5-turbo-0125* (0.54 vs. 0.55), with a slight decrease for *gpt-4-1106-preview* (0.71 vs. 0.75). SH-Eval scores increased marginally for *Llama-3-7b* and *gpt-3.5-turbo-0125* but decreased for *gpt-4-1106-preview*. This technique appears to be more beneficial for smaller models.

The results demonstrate the intricacy of optimizing language model performance for specialized tasks like smart contract code alignment. While certain meta-prompting techniques (RE2, RaR+S2A, SR) show model-specific improvements, the absence of statistically significant enhancements across all models indicates the need for more nuanced approaches.

The variability in SH-Eval and SuAV-Eval scores across different model-technique combinations suggests that the **effectiveness of meta-prompting is highly dependent on the underlying model architecture and pre-training**. For instance, RE2 improved SuAV-Eval for Llama-3-7b (0.43 vs 0.37 baseline) but decreased it for *gpt-4-1106-preview* (0.50 vs 0.56 baseline). This discrepancy may be attributed to the intricate nature of smart contract code alignment, which potentially benefits more from techniques like RE2 that enhance comprehension of complex queries. However, **larger models such as GPT-4 may already incorporate some of the benefits these techniques aim to provide**, leading to **smaller relative improvements**.

- SH-Eval scores generally increased, indicating that meta-prompting techniques led to the detection of more potential errors or inconsistencies in smart contract code alignment.
- SuAV-Eval scores showed mixed results, with RE2 and SR demonstrating some improvements for certain models, while RaR+S2A consistently decreased performance.
- The combined p-values for each technique (RE2: 0.427, RaR+S2A: 0.121, SR: 0.639) all exceed the conventional significance threshold of 0.05, suggesting that none of the techniques consistently outperform the baseline across all models.

Hence, further hybrid approaches beyond meta-prompting engineering could be tested to see whether combining multiple techniques or developing model-specific strategies might yield more consistent and significant improvements across different language models. This approach could potentially address the varying effectiveness of techniques across different model sizes and architectures, tailoring optimization strategies to the specific strengths and limitations of each model.

7.4.3 Chain of Thought (CoT)

Chain-of-thought (CoT) prompting, introduced by *Wei et al.* (2022) [Wei+22], addresses complex problems with non-straightforward input-output relations by introducing intermediate steps or "thoughts" $(z_1, z_2, \ldots, z_n)$ that connect the problem (x) to the solution (y). Each thought represents a coherent problem-solving step, akin to an intermediate equation in math. Mathematically, CoT sequentially samples each thought based on the input and preceding thoughts $(z_i \sim p^i_{\text{CoT}}(z_i \mid x, z_{1\ldots i-1}))$, and then samples the final output $(y \sim p_{\text{CoT}}(y \mid x, z_{1\ldots n}))$. Typically, the sequence of thoughts and the answer are generated as one continuous language sequence $([z_{1\ldots n}, y] \sim p_{\text{CoT}}(z_{1\ldots n}, y \mid x))$, enabling the model to "show its work" and mimic human-like reasoning for complex problem-solving.

Key aspects of CoT prompting are:

- Enhanced performance on complex tasks: CoT significantly improves model performance on arithmetic, commonsense, and symbolic reasoning tasks by mimicking human problem-solving strategies.
- The effectiveness of CoT correlates strongly with model size, emerging as a capability in sufficiently large models. Smaller models may struggle to utilize CoT prompts effectively.
- By making the reasoning process explicit, CoT enhances the transparency of model outputs. This allows users to follow the model's thought process, making it easier to identify errors and understand complex responses.

Empirical results show significant performance improvements across diverse benchmarks. For example, a 540B-parameter model with CoT prompting achieved state-of-the-art accuracy on the GSM8K math word problem benchmark.

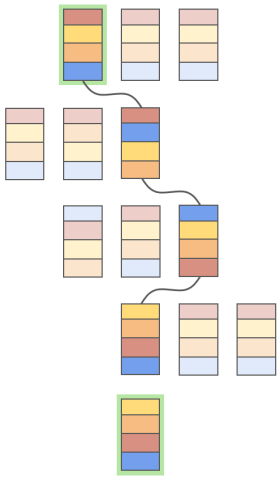


Figure 7.13: CoT Conceptual Illustration

The application of *Chain-of-Thought* (CoT) prompting to smart contract code generation presents challenges that may constrain its efficacy, primarily due to the discrete and non-linear nature of code execution.

Unlike traditional reasoning tasks, where intermediate steps often represent gradual transformations of information, software development—particularly in the domain of smart contracts—involves abrupt state changes and complex interdependencies.

The discrete state transitions inherent in smart contract execution do not always align with the continuous, step-wise reasoning model promoted by CoT.

Furthermore, the high degree of entanglement between different components of a smart contract (e.g., state variables, functions, and external interactions) introduces a level of complexity that may not be adequately captured by linear CoT reasoning. In the case of smart contract generation, the coherence of the reasoning chain may break down when confronted with the non-local effects and state-dependent behaviors.

To assess the impact of Chain-of-Thought (CoT) reasoning on syntax generation, we integrated CoT within our existing framework, utilizing `Llama-3-7b`, `gpt-3.5-Turbo-0125`, and `gpt-4-1106-preview` as to maintain consistency across studies.

The steps involved in the evaluation process were as follows:

- Implemented initial step-by-step meta-prompting to apply *Chain-of-Thought* (CoT) reasoning in smart contract code generation. However, it was observed that often degraded the output format, turning it un-compilable.
- Developed a custom LLM function to equip the agent with memory, facilitating self-dialogue through prompts for iterative planning and testing while retaining all information centralized.
- Designed and tested multiple prompt configurations that explicitly guided the model to break down the problem and articulate intermediate reasoning steps.
- Performance was measured using the same *SH-Eval* and SuAV-*Eval* (*FFQ-3-hint*) metrics, providing a robust basis for comparison with previous results.

Backbone	Strategy	Method	SH-Eval	SuAV-Eval
Llama-3-7b	Prompting	CoT	18.81	0.40
	Baseline		**19.11**	**0.53**
gpt-3.5-turbo-0125	Prompting	CoT	17.05	0.45
	Baseline		**17.12**	**0.55**
gpt-4-1106-preview	Prompting	CoT	13.65	0.55
	Baseline		**13.39**	**0.75**

Table 7.6: Performance of CoT prompting on SH-Eval and SuAV (TD-3-hint) benchmarks. The results indicate minimal improvements over baseline, suggesting limited efficacy of CoT for code generation.

The application of CoT prompting in smart contract code generation did not yield significant improvements in performance metrics. The marginal changes observed across SH-Eval and SuAV-Eval, summarized in Table 7.6, suggest that while CoT can enhance reasoning, it may not intrinsically benefit the syntactic and logical structure required for code generation.

Chain of Thought (CoT) prompting demonstrated minimal improvement in code generation performance across all models (see Figure 7.14), with most metrics either marginally improving or declining compared to baselines. Llama-3-7b showed slight improvements in SH-Eval (19.11 vs 18.81 baseline) but decreased in SuAV-Eval (0.40 vs 0.53 baseline), while maintaining relatively stable TD and FFQ metrics. GPT-3.5-turbo-0125 experienced minor decreases in both SH-Eval (17.05 vs 17.12) and SuAV-Eval (0.45 vs 0.55), coupled with fluctuations in TD and FFQ. GPT-4-1106-preview displayed the most pronounced variations, with improved SH-Eval (13.65 vs 13.39) but a dramatic decrease in SuAV-Eval (0.55 vs 0.75), exhibiting a sharp decline in TD and FFQ after the first iteration, followed by partial recovery.

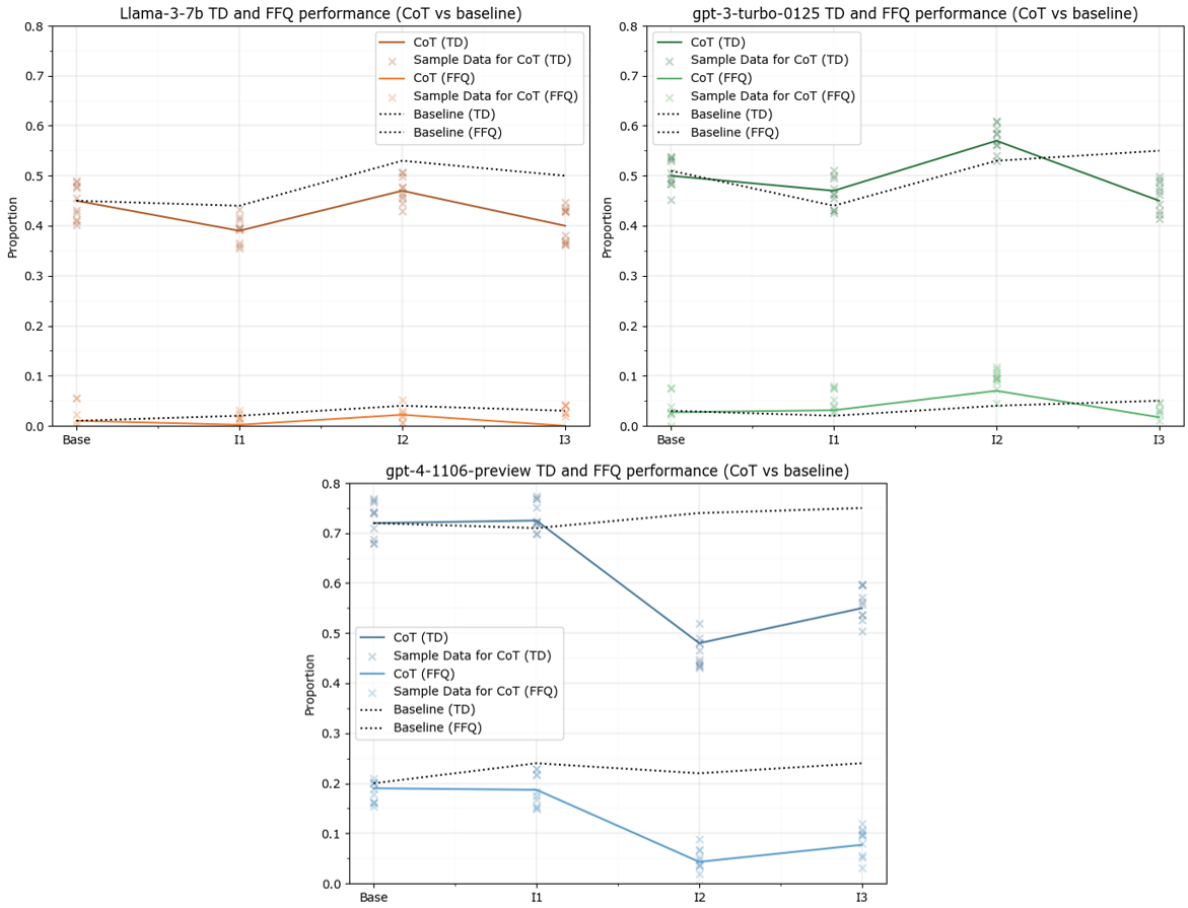


Figure 7.14: Illustration of the performance of Chain of Thought (CoT) reasoning applied to smart contract code alignment tasks, comparing True Density (TD) and False-Free Queries (FFQ) metrics across three hinting iterations (X1, X2, X3) using the SuAV module. The results are shown for multiple models, including *Llama-3-7b*, *gpt-3.5-turbo-0125*, and *gpt-4-1106-preview*, with their respective baselines.

These findings suggest an *Overthinking paradox* [HDS24] in AI models, where more capable models may be more prone to performance inconsistencies when subjected to CoT prompting. The additional reasoning steps introduced by CoT appear to lead advanced models like GPT-4 to explore more complex or tangential paths, potentially moving away from optimal solutions for code generation tasks. This phenomenon is evidenced by the sharp fluctuations in GPT-4's performance metrics, particularly in TD and FFQ, which generally moved in tandem across all models, indicating a strong correlation between information density and code quality in generated outputs. The complex interplay between model capability and prompting techniques underlines the need for carefully tailored approaches in leveraging advanced AI models for specialized tasks like code generation.

In conclusion, the application of Chain of Thought (CoT) prompting to smart contract code generation tasks yielded minimal improvements and, in some cases, even led to performance declines across various models. These results suggest that while CoT can enhance reasoning in certain domains, it may not be inherently beneficial for tasks requiring precise syntactic and logical structures, such as code generation. The sequential nature of CoT reasoning appears to be misaligned with the non-linear, multi-faceted nature of code creation, where developers often need to consider multiple aspects simultaneously.

- CoT prompting demonstrates limited efficacy in improving code generation performance, with minimal gains and occasional declines across different models and benchmarks (SH-Eval and SuAV-Eval).
- The sequential nature of CoT reasoning appears misaligned with the multi-faceted requirements of code generation, suggesting a need for more specialized prompting techniques that consider both syntactic correctness and logical coherence.
- Performance variations across models and iterations indicate an *Overthinking paradox*, where more capable models (e.g., GPT-4) may be more prone to inconsistencies when subjected to CoT prompting for code generation tasks.
- Future research should focus on developing prompting strategies that better align with the unique challenges of code generation, particularly addressing the complex interplay between model capability and task-specific requirements.

Future work should focus on developing methods that can better leverage the unique capabilities of different language models while addressing the particular challenges of generating syntactically correct and logically sound code.

7.4.4 Hybrid Model with Conversational Agents

This method has been drafted as a drop-in replacement for Chain of Thought (CoT) to simulate agents in a very simplistic way, solely for ablation testing purposes. The implementation and features used are limited in scope and do not represent a complete or production-ready system. This method is still far from being suitable for production use, as it was created with the sole purpose of ablation testing and is not yet included in any formal pipeline or workflow. The results and performance of this method should be interpreted with caution, considering its experimental nature and limited scope. This implementation is not intended for real-world applications and should only be used in controlled research environments.

Building upon the insights from the Chain of Thought (CoT) experiments, which yielded marginal improvements in smart contract generation, we devised a hybrid model utilizing a multi-agent conversational approach. This model integrates multiple specialized agents that collaborate through an iterative dialogue to enhance the quality and alignment of generated smart contracts.

The hybrid model involves three primary agents: the *alignment agent*, the *health assessment agent*, and the *security assessment agent*. Each agent focuses on a distinct aspect of the smart contract, ensuring comprehensive evaluation and refinement.

- **Alignment agent:** checks for alignment with the questions hinted back from the SuAV module. It incorporates these questions into the prompt for writing the smart contract, ensuring the generated code aligns with the specified requirements.
- **Health assessment agent:** evaluates the gas consumption of the smart contract, providing insights and suggestions to optimize gas usage.

- **Security assessment agent:** examines the smart contract for potential security vulnerabilities and bad practices, ensuring the code adheres to best practices in smart contract security.

The proposed solution for automated turn-taking between multiple LLM agents features a modular and extensible architecture. Each agent, instantiated from the `LLM_Conversation` class, is configured with a distinct `system_prompt` to define its behavior and initialized with the language model and a temperature parameter to control creativity.

The conversation flow is managed all along by the `ConversationController`, which cycles through the agents, ensuring each one responds in turn. This controller maintains a turn index, enabling seamless transitions between agents. The conversation loop iterates, with each agent generating contextually relevant responses based on the history, updating the prompt for the next turn.

To conclude the conversation, a predefined condition checks for specific markers indicating the end. The final response is then summarized, and the generated code is cleaned of any specific markers using regex, ensuring the output is structured and ready for use.

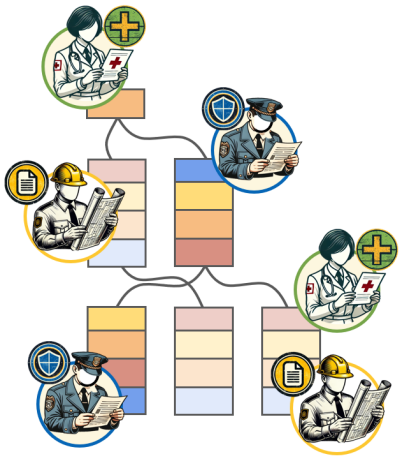


Figure 7.15: CoT Conceptual Illustration

```python
class LLM_Conversation(LLM):

    def __init__(
        self,
        system_prompt: Optional[str] = None,
        time_enabled: bool = False,
        autosave: bool = False,
        save_filepath: str = None,
        language_model: str = "gpt-3.5-turbo",
        temperature: float = 0.5,
        context_length: int = 8192,
        rules: str = None,
        *args,
        **kwargs,
    ):
        super().__init__(system_prompt, language_model, temperature)

        self.time_enabled = time_enabled
        self.autosave = autosave
        self.save_filepath = save_filepath
        self.context_length = context_length
        self.rules = rules

        self.conversation_history = []
        self._start_memory()

        if self.system_prompt: self.add("System", self.system_prompt)
        if self.rules: self.add("User", self.rules)

    def add(self, role: str, content: str) -> None:
    def delete(self, index: int) -> None:
    def update(self, index: int, role: str, content: str) -> None:
    def query(self, index: int) -> None:
    def search(self, keyword: str) -> List:
    def export_conversation(self, filename: str) -> None:
    def import_conversation(self, filename: str) -> None:
    def clear(self) -> None:
```

Listing 7.7: Implementation of the `Conversation` class

To facilitate communication among these agents, we developed a new class called `LLM_Conversation`. This class utilizes the capabilities of large language models (LLMs) to enable dynamic and context-aware dialogues between the agents. The `LLM_Conversation` class ensures that each agent can share its findings and receive feedback from the others, creating a synergistic environment.

The hybrid model was evaluated using the same metrics as previous experiments: SH-Eval and SuAV-Eval (FFQ-3-hint). The results demonstrated significant improvements over the baseline and previous prompting techniques.

Backbone	Strategy	Method	SH-Eval	SuAV-Eval
Llama-3-7b	Agents	LLM Conversation	21.05	0.66
	Baseline		**19.11**	**0.50**
gpt-3.5-turbo-0125	Agents	LLM Conversation	15.30	0.63
	Baseline		**17.12**	**0.55**
gpt-4-1106-preview	Agents	LLM Conversation	14.90	0.79
	Baseline		**13.39**	**0.75**

Table 7.7: Performance of Conversational Agents on SH-Eval and SuAV (TD-3-hint) benchmarks. The results indicate substantial improvements over baseline, demonstrating the efficacy of the hybrid model.

The results, presented in Table 7.7, indicate substantial improvements over the baseline and previous prompting techniques:

- **Llama-3-7b:** This model exhibited the most substantial progress among all tested systems. It achieved a remarkable 24.53% increase in its SuAV-Eval score, elevating it from 0.53 to 0.66. This dramatic improvement emphasizes the potential of hybrid models to significantly enhance the performance of smaller or less advanced language models, potentially bridging the gap with more sophisticated systems.
- **GPT-3.5-turbo-0125:** The performance of this model also saw considerable advancement with the hybrid approach. It demonstrated a 14.55% improvement, raising its SuAV-Eval score from 0.55 to 0.63. This notable enhancement suggests that even well-established models can benefit significantly from the integration of conversational agents, potentially expanding their capabilities and applications.
- **GPT-4-1106-preview:** While this advanced model also benefited from the hybrid approach, its improvement was more modest compared to the others. It experienced a 5.33% increase in its SuAV-Eval score, rising from 0.75 to 0.79. This smaller but still significant improvement indicates that even state-of-the-art models can be further refined through hybrid techniques, albeit with diminishing returns as baseline performance increases.

The results reveal an inverse relationship between a model's initial capabilities and the magnitude of improvement achieved through the hybrid approach. Llama-3-7b, starting from a relatively modest SuAV-Eval score of 0.53, experienced a dramatic 24.53% leap to 0.66, while gpt-3.5-turbo-0125, occupying a middle ground, saw a notable but less extreme boost of 14.55%, rising from 0.55 to 0.63 (as seen both in Figure 7.16 and Table 7.7). This suggests that the hybrid model effectively compensates for the baseline limitations of less advanced language models in the domain. Despite its already impressive baseline score of 0.75, gpt-4-1106-preview still managed a 5.33% increase to 0.79. While this improvement may seem modest in percentage terms, it represents pushing the boundaries of state-of-the-art performance. The fact that even this advanced model showed measurable gains highlights the potency of the hybrid approach.

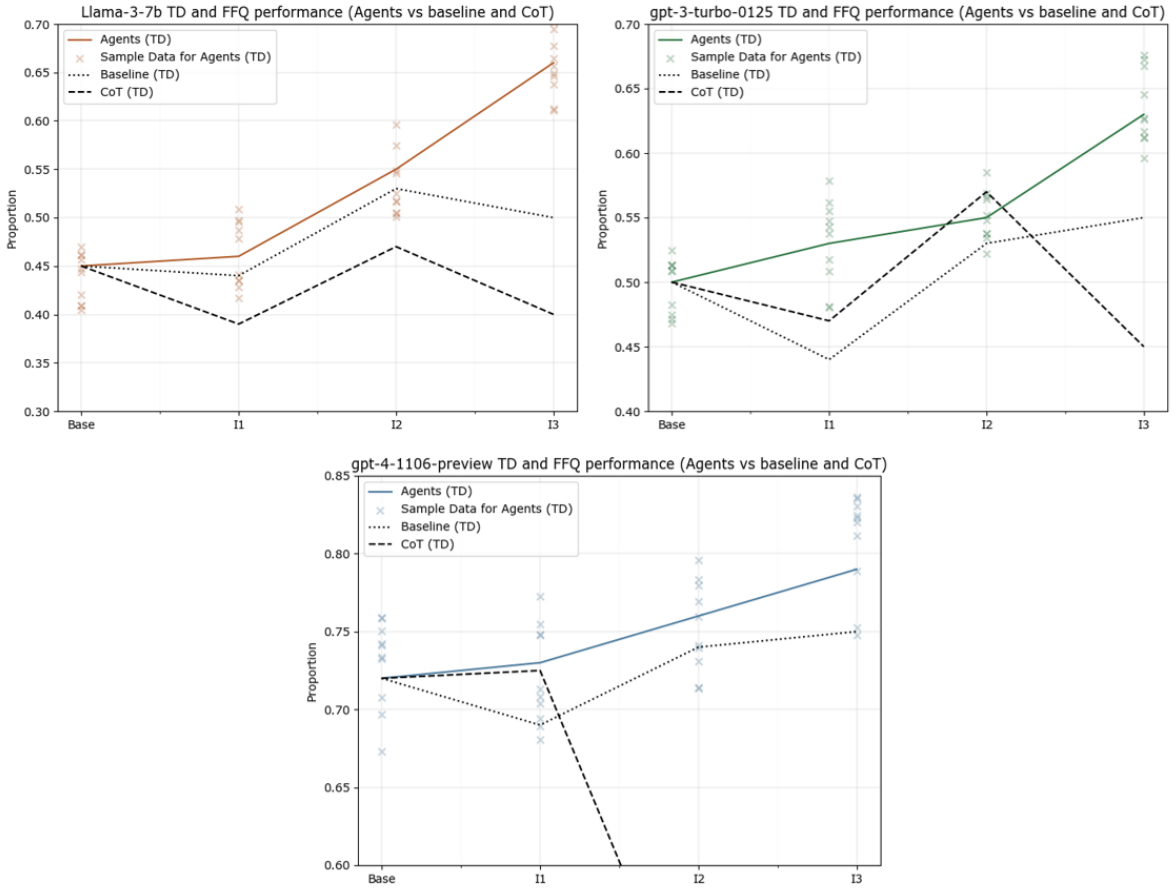


Figure 7.16: Performance comparison of Agents vs Baseline and Chain of Thought (CoT) approaches across different language models, focusing solely on True Density (TD) for clarity. The graph illustrates the TD values over three hinting iterations (Base, I1, I2, I3) for Llama-3-7b, gpt-3-turbo-0125, and gpt-4-1106-preview models. False-Free Queries (FFQ) data is omitted in this visualization to enhance readability and focus on TD trends.

- SH-Eval scores showed improvements across all tested models, with Llama-3-7b demonstrating the most significant increase from 19.11 to 21.05.
- SuAV-Eval scores consistently improved for all models, with Llama-3-7b showing the largest gain of 24.53% (from 0.50 to 0.66).
- The hybrid model using conversational agents yielded substantial improvements over the baseline, particularly for less advanced models, suggesting its potential as a performance equalizer.
- Even the most advanced model (gpt-4-1106-preview) showed a modest but notable improvement of 5.33% in SuAV-Eval score, indicating the approach's effectiveness across different model capabilities.

However, it's crucial to note that these improvements in SuAV-Eval scores, while promising, should be interpreted cautiously. Further research is needed to determine how these gains translate to real-world performance, potential trade-offs in other areas, and the specific mechanisms by which the hybrid approach enhances each model's capabilities.

Conclusions

The development and evaluation of artificial intelligence, especially non-deterministic models, face significant challenges vital for responsible progress. This work has made progress in addressing controllability for code generation, focusing on syntax and performance metrics. The success of the SuAV (Survey Augmented Generation for Validation) assessment, integrated into the MaLB-SC (Multi-Agent LLM-Based Smart Contract Generation Module) system, shows promising results in terms of TD (True Density) and FFQ (False-Free Queries) outcomes.

Building on these achievements, the SuAV method represents an approach to alignment challenges, incorporating self-reflection concepts. It uses a dual-agent framework with an inquisitor and a panel of scrutineers to assess and validate smart contracts. However, its current scope is limited to reasoning layers and does not yet include advanced parameter adjustments through machine learning algorithms, highlighting the need for further research in sophisticated alignment methods.

- MaLB-SC as a DSS (Decision Support System) addresses a critical gap in Solidity smart contract development, where standardized programming practices are often lacking. By implementing a forward-looking approach that bridges this gap, the system provides ground for more consistent and reliable software development methodologies.
- The culmination of this research effort is embodied in the Survey Augmented Generation for Validation (SuAV) technique. By employing a dual-agent framework with an inquisitor and a panel of scrutineers, SuAV provides a systematic method for validating smart contracts against original requirements. This innovative technique offers valuable insights for future research in AI-assisted software development and general alignment endeavours, at the core of state-of-the-art research.
- As LLMs evolve rapidly, their deployment as agents interacting with diverse external environments, including the internet, software systems, robotics, and human interfaces, is expanding at an exponential rate, as well as their intellectual capabilities. This underlines the relevance of research into controllability and alignment.
- In the near future, smart contract generation is likely to become a standard LLM feature, along with various forms of code generation and other complex tasks. This progression emphasizes the need for robust methodologies to evaluate and control these systems, ensuring their reliability, safety, and alignment with human values.

The study's exploration of different reasoning methodologies, including the use of DSPy for declarative syntax prompting and implementing various agent types, provides a solid foundation for future research in AI-assisted software development. Hence, the principles and methodologies developed in this research, such as the modular architecture of MaLB-SC and the SuAV technique, are expected to remain relevant throughout the era of LLMs. Insights gained regarding reasoning capabilities for information compute models may prove valuable in shaping future paradigms of generative and planning-oriented AI.

Thus, the research presented here contributes to the current state of the art and lays the groundwork for continued exploration in alignment, controllability, and the ethical implications of increasingly capable AI systems. As we move forward, it will be critical to expand our methodologies and benchmarks on increasingly powerful models, ensuring that our ability to systematize safety evolves with their growing capabilities.

Epilogue: The Flywheel Effect

"You see, I told you it couldn't be done without turning the whole country into a factory. You have done just that."

— Niels Bohr

To Edward Teller, upon learning of the scale of the Manhattan Project in 1944

This essay is an extension of the previous chapter on conclusions, presenting my personal views and analysis on the current state and future trajectory of artificial intelligence based on information gathered over several years from various sources. My perspective has been shaped by recent rapid developments in the field, as well as in-depth examination of economic models like Tom Davidson's work, Leopold Aschenbrenner's extensive "Situational Awareness" report, and insights from interviews with experts like Yann LeCun and Guillaume Verdon, hosted by figures such as Dwarkesh Patel and Lex Fridman. While acknowledging more pessimistic perspectives, this essay advocates for a future where AI development is managed effectively and remains under human control. This humble contribution attempts to organize my learnings, understand connections, and share modest insights, hoping to contribute to the ongoing dialogue about this transformative technology. Although the scope of this epilogue may seem wide-ranging compared to the thesis's specific focus on controlling a LLM model for a highly-focused use case, it aims to contextualize our modest project within the broader AI landscape, recognizing that, in the long run, technological awareness has proven to be of equal importance to responsible advancement.

As we stand on the brink of a new technological era, an extraordinary funding escalation has been set in motion, transforming our world in ways we are only beginning to comprehend. In recent years, we've witnessed an exponential growth in AI capabilities, fueled by massive investments from tech giants and ambitious startups alike. The race to develop ever-more powerful AI systems has led to breakthroughs that seemed unimaginable just a decade ago. From language models that can engage in human-like conversations to AI-powered tools that generate art, music, and even code, the boundaries of what's possible are constantly expanding. As AI revenue grows rapidly, trillions of dollars are being invested in hardware, IT facilities, and network operation centers. The industrial ramp-up, including growing US electricity production by tens of percent, is poised to be intense. This is partly because AI hardware has been improving much more quickly than Moore's Law[3]. We have specialized chips for AI workloads, moving from CPUs to GPUs, adapting chips for Transformers, and reducing precision number formats

[3]Coined by Gordon Moore in 1965, Moore's Law is the observation that the number of transistors on a microchip doubles approximately every two years, though the cost of computers is halved. This trend has significantly influenced the exponential growth in computing power.

from fp64/fp32 for traditional supercomputing to fp8 on H100s[4]. Spending a million dollars on a model was once considered outrageous. Now, the figures in boardroom strategies grow by another zero every six month, seeing $100 billion and even trillion-dollar clusters, with financial limits and GDP constraints making further increases challenging. This current landscape is already dominated by an intense rivalry among leading technology firms, who are committing extensive computing power to achieve the most powerful AI capabilities. For instance, the supercomputer being build for OpenAI boasts more than 285,000 CPU cores, 10,000 GPUs, and 400 gigabits per second of network connectivity for each GPU server. Meanwhile, other companies are pushing their limits, enlisting the best ML engineers and firing on all cylinders to build colossal data centers in a race to dominate the frontier. Such infrastructure and pursuit represent the pinnacle of current technology, but the true measure of AI's success should not be determined solely by its raw computational power or capabilities. Instead, it should be judged by how effectively we can align these powerful systems with human values, needs, and ethical considerations. This is not just a rhetorical flourish; it is a real and urgent issue that has captured the concern of some of the most prominent figures in the field of AI. Luminaries such as Geoffrey Hinton, often referred to as the "Godfather of AI," the controversial tech magnate Elon Musk, and the renowned researcher Ilya Sutskever, who just started an SSI company to contribute to a solution, have all voiced their apprehensions about the direction and potential risks of AI development. Ensuring that safety grows in tandem with capabilities requires vast efforts, beyond those committed to generative research.

However, we are not even on track. The USA's leading AI labs treat security as an afterthought, partly due to their benefits orientation and the limited understanding of latent interpretability[5] and alignment. These models remain black boxes, and as they grow in complexity, they become increasingly challenging to evaluate and understand. Here lies the paradox of Reinforcement Learning from Human Feedback (RLHF): it fails to scale effectively to superhuman models because human feedback is inherently limited and cannot fully capture the complexity of these advanced systems. As AI capabilities continue to surge, this limitation becomes a critical bottleneck. Thus, if AI capabilities keep progressing, we urgently need methodologies adapted to our less rapidly evolving and mostly stable human minds, helping us preserve our ability to oversee and manage these systems effectively.

Even so, there are reasons to be hopeful when reflecting on the history of AI, the field having always been marked by waves of optimism and skepticism. Back in 2016, Reinforcement Learning (RL) was the buzzword in AI, and most believed it was going to be the key to achieving Artificial General Intelligence[6] (AGI). Mastering RL required a deep grasp of Markov Decision Processes, Bellman equations, dynamic programming, and other complex concepts. However, at one Europe's top AI conference, Yann LeCun offered a different perspective: "RL is just a small part of the overall intelligence framework. The bulk of the intelligence is in unsupervised learning, with supervised learning enhancing it, and RL adding further improvements." LeCun's statement, unconventional at the time, has proven prescient. In today's models like ChatGPT, most computational effort goes into unsupervised or self-supervised learning to predict the next token. Supervised fine-tuning focuses on instruction following, while RL enhances conversational abilities. Currently, he's making another controversial statement: that autoregressive models might be a dead end. He does not imply they will disappear entirely, but suggests that their potential may be limited. Instead of inferential models, he proposes a higher layer where reasoning transcends the real input space and operates within a neuro-symbolic latent space. This space compresses all modalities—images, text, audio—integrating them seamlessly. Here, continuous

[4]Refers to NVIDIA's H100 Tensor Core GPUs, which are designed specifically for AI and high-performance computing workloads. These GPUs support advanced features like mixed-precision training, significantly accelerating deep learning tasks.

[5]The ability to understand and explain the internal representations and processes of a machine learning model, particularly those that are not directly observable. This is crucial for ensuring that AI systems are transparent and their decisions are understandable.

[6]Artificial General Intelligence (AGI) refers to a type of artificial intelligence that possesses the ability to understand, learn, and apply knowledge across a wide range of tasks at a human-like level of proficiency. The concept of AGI has its roots in early AI research and philosophy, with pioneers like Alan Turing and John McCarthy exploring the potential for machines to exhibit general cognitive capabilities. In contemporary context, AGI remains a theoretical goal, distinct from narrow AI, which is designed to excel in specific tasks, and its development poses significant technical and ethical challenges.

gradient-based reasoning is applied, allowing for sophisticated analysis. From this abstract layer, one can decode the information into any desired format using methods such as autoregressive models or diffusion techniques. This focus on multimodality highlights the path to creating a model capable of learning from any input in a connected way, with high fidelity and low latency, compared to biological intelligence.

The truth is, none of that might be necessary to revolutionize the world, and reasoning systems may not be required to achieve something extraordinary. Even if new capabilities cease to emerge and we never create AI systems capable of true reasoning, what we have today is already sufficient to fundamentally transform our world, especially given the ongoing incremental advances. With the optimization capabilities of Mamba[7] and the compression of current Bitnet[8] models, we are seeing an acceleration in the decentralization of these models with more densified software, while hardware is gradually catching up. The contributions of the open-source community are particularly notable. For instance, just one week after Meta released the Llama series 3, they extended the context from 8K to 128K, trained multiple high-performing fine-tunes, and achieved inference speeds of over 800 tokens per second. And most notably, tiny LLMs start to demonstrate comparable proficiency in mathematics to frontier models by utilizing the same techniques Google employed to master Go, namely Monte Carlo Tree Search (MCTS) and backpropagation. The Llama-3-8b model achieves a remarkable 96.7% on the GSM8K math benchmark, surpassing GPT-4, Claude, and Gemini, despite having 200 times fewer parameters, an achievement that has significant implications for future planning and the development of emerging capabilities. In the near future, we might use a single model trained with stochastic depth[9], allowing for flexible layer usage during prediction. This could enable arbitrary layer repeats during generation, allowing users to choose how much compute to invest, and potentially relegating retrieval-augmented generation[10] (RAG) to tasks where explainability is critical, such as in the medical field. If we achieve a significant breakthrough in infinite context, the need for RAG might diminish, as gradient descent could learn to fetch the correct context automatically, eliminating the need for manual RAG heuristics.

LeCun makes a valid point, as the excitement surrounding LLMs far surpasses their current level of development, both within AI as a whole and for LLMs specifically. These models have been trained exclusively on internet text, leaving out many other types of information. Despite access to an enormous amount of reading material—equivalent to over 200,000 years—and requiring far more training trials than humans, they remain lossy compressors, offering only a small glimpse into the future potential of these technologies. The bottleneck for LLMs, if it exists, would appear when these algorithms, even at their highest refinement, reach a ceiling—a ceiling defined by a lack of true understanding, context, and consciousness. This absence of genuine comprehension, beyond mere patterns and data correlations, distinguishes them from humans.

And this would suggest that we are still at the beginning, as LLMs represent the early seeds of what the transformer paradigm can evolve into, giving us ample, or at least some, room to address safety and alignment concerns. However, despite this optimism, the relentless scaling efforts by major corporations continue to dominate the landscape. Even if most top AI researchers haven't signed open

[7] A cutting-edge language model that utilizes a state space model approach instead of traditional attention mechanisms, enabling more efficient processing of long sequences. This innovative design allows Mamba to scale linearly with sequence length, offering potential superior performance over Transformers in tasks that demand extensive contextual understanding. Mamba maintains comparable or even better results across various language modeling benchmarks, showcasing its advanced capabilities in handling complex language processing tasks.

[8] A pioneering approach in Large Language Model (LLM) architecture that utilizes binary (1-bit) or ternary (-1, 0, 1) representations for model parameters. This innovative technique dramatically reduces model size and computational requirements while maintaining performance comparable to full-precision models. BitNet enables more efficient deployment of LLMs on resource-constrained devices, offering significant improvements in latency, memory usage, throughput, and energy consumption.

[9] A technique used in training deep neural networks where certain layers are randomly dropped during training, which helps in improving the model's generalization and reducing overfitting.

[10] A method in natural language processing where a model retrieves relevant documents or information to augment the generated output, improving the accuracy and relevance of the responses. It is particularly useful for tasks requiring access to external knowledge bases.

letters advocating to slow down capabilities research in favor of focusing on safety, the uncertainty over who is genuinely enhancing safety versus who is merely buying time to catch up on benefits remains a cause for concern. The reality is that we don't know how close LLMs can get to human understanding solely through increased computational power. The pace of deep learning progress in the last decade has been extraordinary. Just ten years ago, getting a deep learning system to recognize basic images was groundbreaking. Now, we constantly create new, tougher challenges, only to see them rapidly surpassed. Major benchmarks that used to take decades to overcome now feel like they are surpassed in mere months. As of June 2024, consider how far back in time one would need to go for this technology to appear as unachievable science fiction—three years seems accurate. While there is always room for uncertainty, it is confidently predicted that advancements in training, data, parameters, and computational power will significantly enhance AI capabilities. From GPT-2 to GPT-4, AI advanced from the level of a preschooler to that of a smart high schooler within four years. Analyzing trendlines in computational power (approximately 0.5 orders of magnitude per year), algorithmic efficiencies (approximately 0.5 orders of magnitude per year), and the transition from chatbots to more autonomous agents, we can anticipate a similar qualitative leap by 2027.

Some might argue that LLMs entirely replacing human work is still far-fetched. Even if we halted research, and assuming they're not yet a significant risk, we should remember that simpler AI, like the one behind Facebook's feed, was implicated in failing to prevent its misuse during the Myanmar genocide. Tools, including AI, simplify tasks. Unlike statistical techniques used in network algorithms, these models present significant challenges as black boxes, with advances in interpretability and alignment lagging behind their capabilities. The timeline for NLP AI remains uncertain, but we already recognize that the benchmarks we use to evaluate LLMs—many of which exceed what an average human could solve—are becoming insufficient.

As we continue to scale up the clusters, these models, despite lacking rational behavior and consciousness, still uncover semantic connections between concepts in unexpected and impressive ways. An in the horizon is a far more interesting leap, since althogh being convinced that current models are indeed convserational, they are still like echo chambers. In essence, we just need to teach models a sort of System II outer loop that lets it reason through difficult, long-horizon projects. This approach would represent a significant academic breakthrough, allowing models to generate synthetic data from their own inferences about hidden knowledge, much like the flywheel effect[11], where the system continually improves by learning from its own outputs. If we momentarily imagine that progress continues at its current pace, we might foresee a future where numerous intelligences assist in refining new paradigms and models semi-autonomously, inherently relying on digital-native rails for executing payments[12]. This looped development is already evident in organizations like Anthropic and OpenAI, highlighted by the recent release of Sonnet-3.5, a state-of-the-art model. As Sam Altman has pointed out, larger-scale systems are essential for training more powerful models. This suggests that we will soon be compressing a decade of algorithmic progress into shorter timeframes, leading to a rapid increase in intelligence. However, this acceleration poses challenges for our alignment methods, which may struggle to keep pace. The concern lies not with incremental advancements in techniques and conceptual adjustments of the models but with the rapid pace of investment in software and hardware RD. This suggests rapid GDP growth, potentially multiplying tenfold in two decades, driven by positive feedback loops and reaching near-total levels of automation quickly, according to Tom Davidson's economic model.

While machine learning advancements have surprised us before, predicting future breakthroughs

[11] This term is borrowed from mechanical engineering, where a flywheel stores rotational energy. In the context of AI, it refers to the phenomenon where improvements in one part of the system (like data quality or model architecture) lead to benefits in other areas, which then feed back into further improvements. For example, better models generate better synthetic data, which in turn helps to train even better models.

[12] AI agents, lacking physical presence and legal identity, are unable to establish traditional bank accounts, which poses a significant limitation within the fiat currency system. This makes the reliance on cryptocurrency and digital-native assets not just convenient but necessary. As AI models become more integrated into economic activities, using decentralized and digital-native payment methods is a natural progression. Cryptocurrencies offer a universal medium of exchange that aligns perfectly with the autonomous and borderless nature of AI agents, ensuring seamless and efficient transactions in a digital economy. Yet, this is a matter for another thesis

in LLMs remains challenging. There is no guaranteed timeline for improvement, whether in generation or alignment, but it is certain that reliably controlling AI systems far smarter than us remains an unresolved technical challenge. There is no reason to expect that a small group of intelligences will consistently follow human commands in the long term unless we address alignment and establish effective constraints, especially in federated systems. Although there is currently a bias towards the over-centralization of AI due to the density of compute and centralized data training, over time, as we exhaust the available data on the internet, it makes sense to consider a more ubiquitous AI. This shift is supported by the increasing trend in compute density seen in hardware advancements, allowing AI to gather information and test hypotheses in a distributed manner. Centralized cybernetic control—having one massive intelligence fusing many sensors to accurately perceive the world, predict variables, and act—has never been optimal, further complicating the task of alignment. For these reasons, I believe that our efforts should focus more on preparing for the next generations of models—emphasizing security, monitoring, preparedness, safety, adversarial robustness, (super)alignment, confidentiality, societal impact, and related topics. By doing so, we could scale this technology responsibly in the future, training it on all available data. We would reach a much greater level of maturity than we have today, thereby avoiding legal conflicts. Ideally, we would implement some form of licensing solution that permits the training of models on copyrighted material. Ultimately, our goal should be to harmonize the immense potential of AI with the values and safeguards necessary for a just and equitable society, ensuring that these advancements benefit humanity as a whole.

Appendix A

Appendix: Coverage Sets

First Coverage Set: Some agents needed previous training to better align the output with expected results, as some tasks are marked by a diverse array of possible outcomes and a limited tolerance for deviation from expected results, especially in classification. Thus, JSON coverage sets were produced with Sony's approval to train and validate the behavior of certain agents (as explained in subsection Compilable Computing Entities). In this initial appendix, the first set of data coverage, as delineated in the main body of the work, is presented in detail, directly from ∼`/data/off_topic_descriptions`.

```
[
    "",
    "Smart contracts offer transformative potential for fan engagement platforms by
        enabling automated, transparent interactions directly between fans and creators
        or sports teams. These self-executing contracts built on blockchain technology
        can manage voting mechanisms, merchandise sales, or exclusive content access
        without intermediary oversight. This not only reduces operational costs but also
        enhances trust, as fans can see exactly how their contributions are being used
        and benefit from an immutable ledger that records all transactions.",
    "Smart contract programming involves writing code that is deployed on a blockchain.
        The most popular language for this is Solidity, used primarily for Ethereum.
        Smart contracts are compiled by a special compiler designed to convert the high-
        level language into a lower-level language understood by the blockchain. For
        instance, the Solidity compiler (solc) transforms Solidity scripts into Ethereum
        Virtual Machine (EVM) bytecode, which can be executed on the Ethereum network,
        ensuring the contract performs exactly as written once deployed",
    "Launching a successful fan engagement platform necessitates a robust business
        strategy. Initial funding might be sourced from venture capital, emphasizing the
        innovative use of blockchain to attract tech-savvy investors. Partnerships with
        prominent influencers or sports teams could drive initial user adoption, while a
        multi-tiered subscription model can sustain long-term revenue. Marketing
        strategies should focus on digital channels to tap into the tech-oriented
        audience, leveraging social media for brand visibility and engagement.",
    "From a software development perspective, multiple stakeholders are involved in fan
        engagement platforms. Developers and project managers are crucial for overseeing
        the platform's architecture and ensuring that milestones are met. Designers focus
         on user experience and interface to make the platform appealing and easy to use.
         QA engineers ensure the software's reliability and security. Finally, content
        managers and marketing professionals play essential roles in curating engaging
        content and driving user growth",
    "Loyalty 3.0 represents the next generation of loyalty programs, integrating
        gamification, big data analytics, and blockchain technology. This approach not
        only rewards transactions but also interaction and engagement, creating a more
        holistic view of loyalty that values a customer's entire relationship with a
        brand. By leveraging blockchain, Loyalty 3.0 can ensure that rewards are
        distributed fairly and transparently, enhancing trust and participation in the
        program.",
    "Smart Contract Smart Contract Smart Contract Smart Contract Smart Contract Smart
        Contract Smart Contract Smart Contract.",
    "Single smart contracts have lots of functionalities and can be used in various
        industries, since they offer very technical resources.",
    "Code binds agreement, Blocks chain truth in the ledger, Trust is automatic.
        Transactions secure, Ledger logs every action, Immutable trace. Contracts self-
```

```
        execute, Mistakes are past, trust in code, Blockchain does ensure.",
    "The Digital Canvas initiative leverages the concept of programmable interactions,
        much like a smart contract, but for artists and galleries. The platform allows
        digital creators to set up automated licensing agreements, ensuring their art is
        shared or sold under specific conditions. Just as blockchain technology records
        transactions, Digital Canvas maintains a digital ledger of all art transactions,
        providing transparency and security without the need for third-party oversight.",
    "In the realm of professional networking, a Virtual Handshake Protocol could mirror
        the functionalities of a smart contract by automating connections and
        endorsements based on pre-set criteria. This protocol, powered by machine
        learning algorithms, can analyze user profiles and match individuals with
        potential mentors, partners, or employers, streamlining the networking process
        while ensuring a high level of data integrity and user consent.",
    "Educational platforms are exploring avenues similar to smart contracts for
        automating the issuance of certificates and managing course enrollments. By using
         predefined criteria, these platforms can autonomously verify a student's
        achievements and issue certificates without manual intervention. This system
        enhances the reliability of educational credentials and simplifies administrative
         processes.",
    "Strategic implementation of digital contracts in the healthcare sector could
        revolutionize patient engagement and data management. By setting predefined terms
         for data access and patient care protocols, hospitals can automate many of their
         processes, similar to how smart contracts operate on blockchain platforms. This
        ensures compliance with regulations and consistency in patient care across
        multiple facilities.",
    "Imagine a digital marketing campaign that functions like a smart contract. Campaign
        elements are executed automatically based on audience engagement metrics. This
        could include the release of exclusive content when certain interaction
        thresholds are met, mimicking the conditional execution of smart contracts but
        applied to marketing strategies",
    "In the design world, the concept of a 'Design Compiler' could function similarly to
        how smart contracts are compiled. This tool would automatically transform high-
        level design ideas into detailed schematics and prototypes, streamlining the
        process from conception to creation, ensuring designers' original specifications
        are met accurately.",
    "The Smart Scheduler for conference management could draw from the principles of
        smart contracts to automate the planning and execution of events. Based on
        predefined rules and participant feedback, sessions could be dynamically
        scheduled and resources allocated efficiently, ensuring optimal engagement and
        operational transparency",
    "An automated feature rollout system in software development could employ mechanisms
        akin to those in smart contracts to ensure seamless and error-free deployment of
        new features across platforms. By establishing clear conditions for deployment
        and rollback strategies, this system could significantly reduce downtime and
        improve user experience.",
    "In urban development, a 'City Contract' system could automate zoning and land use
        decisions based on real-time data and community input, much like a smart contract
         executes terms based on the blockchain. This would facilitate more dynamic urban
         planning and ensure compliance with environmental and societal needs",
    "A futuristic 'Consent Ledger' in online platforms could function like a smart
        contract for privacy settings, automatically adjusting users' privacy based on
        their preferences and activities. This would enhance user trust and control,
        providing a transparent record of consent and changes over time, similar to the
        immutable nature of blockchain records.",
    "The Culinary Code system operates on a platform that automates recipe adjustments
        based on ingredient availability and dietary preferences, mirroring the
        adaptability of smart contracts. Imagine a digital chef that dynamically modifies
         your dinner plans ensuring optimal nutrition and taste with the precision of
        blockchain-like technology.",
    "This is a highly technical description of a Smart Contract.",
    "The smart contract must be able to produce tokens."
]
```

Second Coverage Set: This set contains valid descriptions, representative of the expected input distribution, and structured breakdowns of their features, each characterized with the following attributes: id, name, scope, input, constraints, output, primary scenario, alternative scenario. In this second appendix, the second set of data coverage, as delineated in the main body of the work, is presented in detail, directly from ∼/`data/smart_contract_descriptions`.

```
[
    {
        "title": "event",
        "descriptions": [
            "It has to manage 50,000 tokens available for a concert, with each token representing one ticket. Users are limited to purchasing one ticket each, but those with Golden status can buy up to three tickets to transfer to other users. The ticket sales are divided into two phases. The first phase lasts for 5 minutes, and the second phase is triggered one week after the first one ends. If the event is cancelled, compensation includes an extra 25% for Golden ticket holders, 5% for Platinum, and no extra compensation for Bronze ticket holders.",
            ">",
            ">",
            ">"
        ],
        "requirements": [
            {
                "id": "REQ01",
                "name": "Token Allocation",
                "scope": "Managing the distribution and tracking of 50,000 concert tickets represented as tokens.",
                "input": "Total number of tokens (50,000), each ticket request.",
                "constraints": "Fixed total supply of tokens.",
                "output": "Each successful transaction decreases the total number of available tokens.",
                "primary_scenario": "User requests a ticket and if tokens are available, one token is allocated to the user.",
                "alternative_scenario": "All tokens are sold, no more tokens can be allocated."
            },
            {
                "id": "REQ02",
                "name": "User Purchase Limit",
                "scope": "Restricting standard users to purchasing only one ticket.",
                "input": "User status (standard), ticket purchase request.",
                "constraints": "User status must be standard; ticket count per standard user cannot exceed one.",
                "output": "Validation result of purchase request (approved or denied).",
                "primary_scenario": "Standard user purchases their first ticket successfully.",
                "alternative_scenario": "Standard user attempts to purchase more than one ticket; the system denies the request."
            },
            {
                "id": "REQ03",
                "name": "Golden Status Exception",
                "scope": "Allowing Golden status holders enhanced purchasing capabilities.",
                "input": "User status (Golden), ticket purchase requests.",
                "constraints": "User status must be verified as Golden; ticket count for Golden users cannot exceed three.",
                "output": "Validation result of purchase request for up to three tickets.",
                "primary_scenario": "Golden status user purchases up to three tickets successfully.",
                "alternative_scenario": "Golden status user attempts to purchase more than three tickets; the system denies the request."
            },
            {
                "id": "REQ04",
                "name": "Sales Phases",
                "scope": "Managing the ticket sales timeline in two phases.",
                "input": "Current time/date, phase status.",
                "constraints": "Sales must occur strictly within the defined phase durations.",
                "output": "Phase status (active or inactive).",
                "primary_scenario": "A user makes a purchase during an active sales phase.",
                "alternative_scenario": "A user attempts to purchase outside the active sales phase; the system denies the request."
            },
            {
                "id": "REQ05",
                "name": "Phase Timing",
                "scope": "Scheduling and triggering the two sales phases.",
```

```
                "input": "Start times for each phase, duration of the first phase.",
                "constraints": "Exact timing must be adhered to; the second phase begins exactly one week after the first ends.",
                "output": "Current phase status based on the timeline.",
                "primary_scenario": "System transitions from the first phase to the second phase exactly one week after the first ends.",
                "alternative_scenario": "Time checks occur frequently to ensure phase timings are respected."
            },
            {
                "id": "REQ06",
                "name": "Event Cancellation Compensation",
                "scope": "Event Cancellation Compensation",
                "input": "Calculating compensation for ticket holders if the event is cancelled.",
                "constraints": "Ticket holder's status (Golden, Platinum, Bronze), event cancellation status. Only applies if they bought in the first phase.",
                "output": "Compensation rates differ by status: 25% for Golden, 5% for Platinum, none for Bronze.",
                "primary_scenario": "Event is cancelled, and compensation is distributed according to user status.",
                "alternative_scenario": "Event proceeds as planned; no compensation is distributed."
            }
        ]
    },
    {
        "title": "auctioning",
        "descriptions": [
            "The goal is to auction limited VIP tickets to fans for a concert. Each ticket is represented as a unique token. Limit each address to bid a maximum of 3 times. Bids must be at least 5% higher than the current highest bid. Emit new highest bids and conclude when a time limit expires.",
            ">",
            ">",
            ">"
        ],
        "requirements": [
            {
                "id": "REQ01",
                "name": "Unique Token Representation",
                "scope": "Representing each VIP concert ticket as a unique token using the ERC-721 standard.",
                "input": "Concert details (artist, venue, date), token generation request.",
                "constraints": "Each token must be uniquely identifiable and comply with the ERC-721 standard.",
                "output": "Generation of a new unique token for each VIP ticket.",
                "primary_scenario": "A request is made to generate a token, and a unique ERC-721 token is minted for a VIP ticket.",
                "alternative_scenario": "Token generation fails due to system errors or non-compliance with ERC-721 standards."
            },
            {
                "id": "REQ02",
                "name": "Bid Limitation Per Address",
                "scope": "Limiting the number of bids an address can place during the auction to three.",
                "input": "Bidder's address, bid attempt.",
                "constraints": "Each address can only make three bids during the entire auction period.",
                "output": "Acceptance or rejection of a bid based on the bid count from the same address.",
                "primary_scenario": "Bidder places a bid and has not reached the three-bid limit, so the bid is accepted.",
                "alternative_scenario": "Bidder attempts to place a fourth bid, and the system rejects this bid."
            },
            {
                "id": "REQ03",
                "name": "Bid Increment Requirement",
                "scope": "Enforcing a rule that each new bid must be at least 5% higher than the current highest bid.",
                "input": "Amount of the new bid, current highest bid.",
                "constraints": "New bids must be quantifiably 5% higher than the current highest bid.",
```

```
                "output": "Validation result of the bid increment (approved or denied).",
                "primary_scenario": "A new bid is successfully placed that is 5% higher than the previous highest bid.",
                "alternative_scenario": "A bid is placed that does not meet the 5% higher requirement and is therefore rejected."
            },
            {
                "id": "REQ04",
                "name": "Emitting Bid Updates",
                "scope": "Notifying participants of a new highest bid through event emissions in the smart contract.",
                "input": "New highest bid details.",
                "constraints": "Each new highest bid must trigger an event emission.",
                "output": "Event emission that notifies all participants of the new highest bid.",
                "primary_scenario": "A new highest bid is placed, and an event is emitted to update participants.",
                "alternative_scenario": "A highest bid is placed, but due to a system fault, the event is not emitted."
            },
            {
                "id": "REQ05",
                "name": "Auction Time Limit",
                "scope": "Managing the auction duration with a set time limit, after which no further bids can be accepted.",
                "input": "Current time/date, auction end time.",
                "constraints": "Bids are only accepted within the predefined auction time frame.",
                "output": "Status of auction (active or concluded).",
                "primary_scenario": "A bid is placed within the active auction time and is accepted.",
                "alternative_scenario": "A bid attempt is made after the auction time has expired, and it is rejected."
            }
        ]
    },
    {
        "title": "upgrading",
        "descriptions": [
            "It is intented to manage tiered VIP memberships, allowing fans to upgrade or downgrade their membership based on the tokens they hold. Define membership levels as Bronze, Silver, and Gold, each with associated perks. Upgrading requires the owner to have a sufficient number of additional tokens (100 tokens for Silver, 200 for Gold). Include functionality for members to voluntarily lower their level, refunding tokens proportionally based on their tenure on the platform (70% if >2 years, 50% if >1 year, 30% if >6 months). Once downgraded, a member must wait 12 days before upgrading again.",
            ">",
            ">",
            ">"
        ],
        "requirements": [
            {
                "id": "REQ01",
                "name": "Tiered VIP Membership Management",
                "scope": "Managing tiered VIP memberships using an ERC-721-based smart contract where memberships are categorized into Bronze, Silver, and Gold levels.",
                "input": "Membership upgrade or downgrade request, member's current token count and membership level.",
                "constraints": "Memberships are strictly categorized into three levels with specific token requirements for each.",
                "output": "Member's new membership level following a successful upgrade or downgrade.",
                "primary_scenario": "Member requests an upgrade and has the necessary tokens for the next level, which is then processed successfully.",
                "alternative_scenario": "Member requests an upgrade without sufficient tokens, and the request is denied."
            },
            {
                "id": "REQ02",
                "name": "Membership Upgrade Requirements",
                "scope": "Defining and enforcing token requirements for upgrading membership levels among Bronze, Silver, and Gold.",
                "input": "Member's current membership level, number of tokens held, upgrade request.",
                "constraints": "100 additional tokens needed for Silver, 200 for Gold.
```

```
                    Member must not have downgraded within the last 12 days.",
                "output": "Approval or denial of the membership upgrade based on token count and upgrade cooldown.",
                "primary_scenario": "Member meets the token requirement and cooldown period, successfully upgrades membership.",
                "alternative_scenario": "Member does not meet token requirement or is within cooldown period, and the upgrade is denied."
            },
            {
                "id": "REQ03",
                "name": "Voluntary Membership Downgrading",
                "scope": "Allowing members to voluntarily downgrade their membership level and receive a token refund based on their tenure.",
                "input": "Member's current level, tenure on the platform, downgrade request.",
                "constraints": "Token refund rates: 70% if tenure >2 years, 50% if >1 year, 30% if >6 months. Downgrade is followed by a 12-day upgrade cooldown.",
                "output": "Number of tokens refunded and new membership level.",
                "primary_scenario": "Member requests a downgrade and is eligible for a refund based on tenure; tokens are refunded accordingly.",
                "alternative_scenario": "Member requests a downgrade but does not meet the minimum tenure for a refund; downgrade proceeds without refund."
            },
            {
                "id": "REQ04",
                "name": "Upgrade Cooldown Management",
                "scope": "Managing the cooldown period preventing members from upgrading their membership level again too soon after downgrading.",
                "input": "Date of last downgrade, current date, upgrade request.",
                "constraints": "12-day cooldown period after any downgrade.",
                "output": "Approval or denial of the upgrade request based on cooldown compliance.",
                "primary_scenario": "Member attempts to upgrade more than 12 days after a downgrade and is successful.",
                "alternative_scenario": "Member attempts to upgrade within 12 days of downgrading; the request is denied."
            }
        ]
    },
    {
        "title": "rewards",
        "descriptions": [
            "Design a smart contract that tracks and rates fan interactions for each content piece. Fans earn points that influence their ranking on the platform, with interactions categorized as likes, comments, shares, and views. Each interaction type carries different weightings (1 point for likes, 10 points for shares, 20 points for comments). Fans can earn a maximum of 100 points per day from each activity. Platinum and Gold tier members can earn up to 150 and 200 points, respectively, but the extra points expire after a week.",
            ">",
            ">",
            ">"
        ],
        "requirements": [
            {
                "id": "REQ01",
                "name": "Interaction Tracking and Rating",
                "scope": "Tracking and rating fan interactions with content pieces using a smart contract. Interactions include likes, comments, shares, and views, each contributing differently to a fan's overall points and ranking on the platform.",
                "input": "Type of interaction (like, comment, share, view), fan's interaction details.",
                "constraints": "Interactions are quantified with specific points: 1 point for likes, 10 for shares, 20 for comments.",
                "output": "Updated points and ranking for the fan based on their interactions.",
                "primary_scenario": "Fan interacts with a content piece (e.g., likes it), and points are added to their total score.",
                "alternative_scenario": "Fan reaches the daily point limit from a specific interaction type; further points from that interaction type are not added until the next day."
            },
            {
                "id": "REQ02",
                "name": "Daily Point Limits",
```

```
                "scope": "Enforcing a maximum point limit that a fan can earn per day from each type of interaction to prevent point inflation and ensure equitable ranking.",
                "input": "Fan's daily interaction count, type of interaction.",
                "constraints": "Standard limit is 100 points per day per activity. Adjusted limits for Platinum (150 points) and Gold (200 points) tier members, with expiration rules for extra points.",
                "output": "Total points earned by the fan for the day, considering the daily limits.",
                "primary_scenario": "Fan earns points up to the daily limit for an interaction type.",
                "alternative_scenario": "Fan exceeds the daily point limit for an interaction type; excess points are not counted."
            },
            {
                "id": "REQ03",
                "name": "Tier-Based Point Adjustment",
                "scope": "Adjusting daily point limits based on membership tier (Platinum and Gold), with special rules for the expiration of extra points earned beyond the standard limit.",
                "input": "Fan's membership tier, points earned from interactions, date of points earning.",
                "constraints": "Platinum members can earn up to 150 points, and Gold members up to 200 points per activity, but extra points expire after one week.",
                "output": "Adjusted and valid points based on tier and expiration of extra points.",
                "primary_scenario": "Platinum or Gold tier fan earns points exceeding the standard limit, and the system tracks the expiration of these points.",
                "alternative_scenario": "Extra points expire after a week, reducing the fan's total points if they were not utilized within that timeframe."
            }
        ]
    },
    {
        "title": "subscription",
        "descriptions": [
            "A smart contract is needed to manage multiple subscription levels where fans pay a monthly fee in tokens to access exclusive content. Subscription levels include Basic (10 tokens/month), Premium (50 tokens/month), and Elite (100 tokens/month). Automatically deduct fees from a user's token balance every 30 days. If there are insufficient funds, the subscription is paused. Implement auto-renewal features, and if a subscription expires, access to exclusive content is immediately revoked until renewal.",
            ">",
            ">",
            ">"
        ],
        "requirements": [
            {
                "id": "REQ01",
                "name": "Subscription Level Management",
                "scope": "Managing multiple subscription levels (Basic, Premium, Elite) within a smart contract, allowing fans to access exclusive content based on their subscription tier.",
                "input": "Fan's subscription level selection, token payment.",
                "constraints": "Subscription fees: Basic at 10 tokens/month, Premium at 50 tokens/month, Elite at 100 tokens/month.",
                "output": "Activation or update of the fan's subscription status based on the paid tier.",
                "primary_scenario": "Fan selects a subscription level and successfully pays the required tokens, activating the subscription.",
                "alternative_scenario": "Fan selects a subscription level but lacks sufficient tokens to pay the fee, preventing activation."
            },
            {
                "id": "REQ02",
                "name": "Automatic Fee Deduction",
                "scope": "Automatically deducting subscription fees from a user's token balance every 30 days to maintain active subscriptions.",
                "input": "Current date, last payment date, user's token balance, subscription level.",
                "constraints": "Fees are deducted every 30 days, corresponding to the subscription level's cost.",
                "output": "Updated token balance and subscription status (active or paused).",
                "primary_scenario": "It is time for the fee deduction, the fan has
```

```
                    sufficient balance, and the fee is successfully deducted, keeping the
                     subscription active.",
253                 "alternative_scenario": "Insufficient balance at the time of fee
                    deduction leads to the subscription being paused."
254             },
255             {
256                 "id": "REQ03",
257                 "name": "Subscription Auto-Renewal",
258                 "scope": "Implementing an auto-renewal feature for subscriptions,
                    ensuring continuous access to exclusive content unless manually
                    cancelled or paused due to insufficient funds.",
259                 "input": "Subscription expiry date, user's token balance, user's renewal
                    settings.",
260                 "constraints": "Subscriptions attempt to auto-renew unless funds are
                    insufficient or the user has cancelled.",
261                 "output": "Renewal of subscription and deduction of corresponding fee, or
                     failure to renew due to cancellation or insufficient funds.",
262                 "primary_scenario": "Subscription is due for renewal, auto-renewal is
                    enabled, and the user has sufficient tokens; subscription is renewed.
                    ",
263                 "alternative_scenario": "Subscription is due for renewal but cannot be
                    processed due to insufficient funds or user cancellation; access is
                    revoked until issues are resolved."
264             },
265             {
266                 "id": "REQ04",
267                 "name": "Access Revocation on Expiry",
268                 "scope": "Revoking access to exclusive content immediately if a
                    subscription expires without renewal, maintaining the integrity and
                    exclusivity of content.",
269                 "input": "Subscription status, access request to exclusive content.",
270                 "constraints": "Access to content is directly tied to the active status
                    of a subscription.",
271                 "output": "Access granted if subscription is active; access denied if
                    subscription is expired.",
272                 "primary_scenario": "Fan tries to access content with an active
                    subscription, and access is granted.",
273                 "alternative_scenario": "Subscription has expired due to non-renewal or
                    insufficient funds, and access to content is denied."
274             }
275         ]
276     },
277     {
278         "title": "voting",
279         "descriptions": [
280             "Fans can vote on various categories using tokens. Votes are weighted by the
                 number of tokens held. Categories include Best Video, Best Fan Art, and
                 Best Comment. Each category has its own voting function, where 1 token
                 equals 1 vote. Fans can allocate tokens to different categories but must
                 hold at least 10 tokens to participate. Voting is open for 30 days.",
281             ">",
282             ">",
283             ">"
284         ],
285         "requirements": [
286             {
287                 "id": "REQ01",
288                 "name": "Token-Based Voting System",
289                 "scope": "Implementing an ERC721 smart contract for a Fan Engagement
                    Platform that allows fans to vote on different categories (Best Video
                    , Best Fan Art, Best Comment) using tokens, where the number of
                    tokens held by a fan determines their voting power.",
290                 "input": "Fan's token count, category selection, number of tokens
                    allocated for voting.",
291                 "constraints": "Each token equates to one vote. Fans must hold at least
                     10 tokens to participate in voting.",
292                 "output": "Recorded votes for each category based on tokens allocated by
                     fans.",
293                 "primary_scenario": "Fan allocates tokens to a category, and votes are
                    added to the total count of that category based on the number of
                    tokens used.",
294                 "alternative_scenario": "Fan attempts to vote with fewer than 10 tokens,
                    and the voting attempt is rejected."
295             },
296             {
297                 "id": "REQ02",
298                 "name": "Category-Specific Voting Functions",
299                 "scope": "Creating separate voting functions within the smart contract
```

```
                    for each category, allowing fans to allocate tokens specifically for
                    Best Video, Best Fan Art, and Best Comment.",
300                 "input": "Category-specific vote request, number of tokens allocated.",
301                 "constraints": "Separate functions must exist for each category to
                    prevent vote misallocation.",
302                 "output": "Successful recording of votes in the correct category based on
                     fan's token allocation.",
303                 "primary_scenario": "Fan selects a category, allocates a specific number
                    of tokens, and the tokens are correctly applied as votes in that
                    category.",
304                 "alternative_scenario": "Fan mistakenly tries to allocate tokens to a non
                    -existent category, and the transaction is nullified."
305             },
306             {
307                 "id": "REQ03",
308                 "name": "Voting Eligibility and Minimum Token Requirement",
309                 "scope": "Ensuring that only fans with at least 10 tokens can participate
                     in the voting process, maintaining fairness and engagement in voting
                    .",
310                 "input": "Fan's token holdings, voting request.",
311                 "constraints": "Fans need to hold a minimum of 10 tokens to be eligible
                    for voting.",
312                 "output": "Validation of fan's eligibility to vote based on token
                    holdings.",
313                 "primary_scenario": "Fan meets the minimum token requirement and is
                    allowed to vote.",
314                 "alternative_scenario": "Fan does not meet the minimum token requirement
                    and is denied the ability to vote."
315             },
316             {
317                 "id": "REQ04",
318                 "name": "Voting Period Management",
319                 "scope": "Managing the open period for voting across all categories,
                    ensuring the voting process is open for a designated timeframe (30
                    days).",
320                 "input": "Current date, start and end dates of the voting period.",
321                 "constraints": "Voting must only occur within the 30-day period after
                    initiation.",
322                 "output": "Status of voting period (open or closed).",
323                 "primary_scenario": "Fan votes within the designated 30-day period, and
                    the vote is accepted.",
324                 "alternative_scenario": "Fan attempts to vote outside of the 30-day
                    period, and the vote is not accepted."
325             }
326         ]
327     },
328     {
329         "title": "distribution",
330         "descriptions": [
331             "Program a highly efficient ERC721 Smart Contract for the Fan Engagement
                    Platform to distribute rewards based on activity and token holdings at
                    the end of each month. Fans with more than 200 tokens held and at least
                    50 activity points accumulated in a month qualify for rewards. Rewards
                    consist of awarding 10% of held tokens as additional tokens. Distribution
                     occurs automatically on the last day of each month, emitting an event.",
332             ">",
333             ">",
334             ">"
335         ],
336         "requirements": [
337             {
338                 "id": "REQ01",
339                 "name": "Reward Qualification Criteria",
340                 "scope": "Defining and validating the criteria for fans to qualify for
                    monthly rewards on the Fan Engagement Platform, based on token
                    holdings and activity points.",
341                 "input": "Fan's total tokens, monthly activity points.",
342                 "constraints": "Fans must hold more than 200 tokens and accumulate at
                    least 50 activity points in a month to qualify for rewards.",
343                 "output": "Determination of whether a fan qualifies for the monthly
                    reward.",
344                 "primary_scenario": "Fan exceeds both the token holding and activity
                    point thresholds, qualifying for rewards.",
345                 "alternative_scenario": "Fan does not meet one or both of the thresholds,
                     and does not qualify for rewards."
346             },
347             {
348                 "id": "REQ02",
```

```
                "name": "Reward Calculation and Distribution",
                "scope": "Automatically calculating and distributing rewards to qualified fans at the end of each month, where the reward is 10% of the tokens they hold.",
                "input": "Qualified fan's token holdings, current date.",
                "constraints": "Reward calculation occurs only if the fan qualifies and on the last day of the month.",
                "output": "Additional tokens credited to the fan's account, proportional to their held tokens.",
                "primary_scenario": "On the last day of the month, tokens are automatically credited to the accounts of fans who qualify.",
                "alternative_scenario": "It is not the last day of the month, or the fan does not qualify; no tokens are distributed."
            },
            {
                "id": "REQ03",
                "name": "Event Emission upon Reward Distribution",
                "scope": "Emitting an event each time rewards are distributed to notify the system and potentially other linked services or interfaces.",
                "input": "Details of the reward distribution (fan identifier, amount of tokens distributed).",
                "constraints": "An event must be emitted for each reward distribution, capturing all relevant details.",
                "output": "Event log that provides transparency and traceability of the reward distribution process.",
                "primary_scenario": "Rewards are distributed and an event is emitted detailing the distribution.",
                "alternative_scenario": "No rewards are distributed (due to non-qualification or other reasons), hence no event is emitted."
            }
        ]
    },
    {
        "title": "referral",
        "descriptions": [
            "Smart Contract that rewards fans for referring friends, with bonuses based on the number of successful referrals. Successful referrals are counted when the referred friend creates an account and buys a ticket for an event. Bonuses include 10 tokens for 5 referrals, 30 tokens for 10, and 70 tokens for 20. Additionally, if a referred friend further refers others, the original referrer receives 150 tokens.",
            ">",
            ">",
            ">"
        ],
        "requirements": [
            {
                "id": "REQ01",
                "name": "Referral Tracking System",
                "scope": "Implementing a tracking system within an ERC721 smart contract to monitor successful referrals, where a referral is deemed successful once the referred friend creates an account and purchases an event ticket.",
                "input": "Referrer's ID, referred friend's account creation and ticket purchase status.",
                "constraints": "Referrals must be confirmed by both account creation and an event ticket purchase by the referred friend.",
                "output": "Incremented referral count for the referrer upon successful referral confirmation.",
                "primary_scenario": "A referred friend creates an account and purchases a ticket, successfully confirming a referral and increasing the referrer's count.",
                "alternative_scenario": "A referred friend creates an account but does not purchase a ticket, resulting in an unsuccessful referral that does not increase the count."
            },
            {
                "id": "REQ02",
                "name": "Referral Bonus Distribution",
                "scope": "Calculating and distributing bonuses to fans based on the number of successful referrals made. Bonuses are tiered based on referral milestones.",
                "input": "Referrer's ID, number of successful referrals.",
                "constraints": "Bonuses are set at 10 tokens for 5 referrals, 30 tokens for 10 referrals, and 70 tokens for 20 referrals. Distribution occurs as each milestone is reached.",
                "output": "Tokens awarded to referrers upon reaching specific referral milestones.",
```

```
                "primary_scenario": "Referrer reaches a referral milestone and is automatically awarded the corresponding token bonus.",
                "alternative_scenario": "Referrer's total referrals do not meet any of the bonus thresholds, resulting in no bonus distribution."
            },
            {
                "id": "REQ03",
                "name": "Second-Level Referral Rewards",
                "scope": "Rewarding the original referrer with additional tokens if their referred friends refer others to the platform.",
                "input": "Original referrer's ID, successful second-level referral activity.",
                "constraints": "The original referrer earns a bonus only when a referred friend successfully refers others.",
                "output": "150 tokens awarded to the original referrer for each successful second-level referral.",
                "primary_scenario": "A referred friend successfully refers another person who creates an account and purchases a ticket, triggering a second-level reward for the original referrer.",
                "alternative_scenario": "A referred friend attempts to refer another person, but the referral does not complete the required actions (account creation and ticket purchase), so no second-level reward is issued."
            },
            {
                "id": "REQ04",
                "name": "Event Emission for Reward Activities",
                "scope": "Emitting blockchain events for each reward activity related to referrals to ensure transparency and traceability within the smart contract operations.",
                "input": "Details of the referral reward distribution (referrer's ID, number of tokens awarded).",
                "constraints": "An event must be emitted whenever tokens are distributed as a referral reward.",
                "output": "Blockchain event logs that detail each instance of reward distribution for auditing and tracking purposes.",
                "primary_scenario": "Tokens are awarded for reaching a referral milestone or for second-level referrals, and an event is emitted detailing these transactions.",
                "alternative_scenario": "No tokens are awarded due to failure to meet referral criteria, hence no event is emitted."
            }
        ]
    }
]
```

Third Coverage Set:
The DSPy library is utilized once again for training the inquisitor figure from the SuAV architecture to generate pertinent questions based on the contract. This ensures the questions remain within the scope of Y/N queries, which can be easily answered through boolean scrutineer output structures. Consequently, the module includes a coverage set, similar to the interaction coalitions, for training the inquisitor on effective query formulation. This data has been retrieved directly from ~`/data/insights`.

```
[
    {
        "smart_contract_description": "Program a highly efficient ERC721 Smart Contract for the Fan Engagement Platform to distribute rewards based on activity and token holdings at the end of each month. Fans with more than 200 tokens held and at least 50 activity points accumulated in a month qualify for rewards. Rewards consist of awarding 10% of held tokens as additional tokens. Distribution occurs automatically on the last day of each month, emitting an event.",
        "question": "How are activity points tracked and calculated?"
    },
    {
        "smart_contract_description": "Program a highly efficient ERC721 Smart Contract for the Fan Engagement Platform to distribute rewards based on activity and token holdings at the end of each month. Fans with more than 200 tokens held and at least 50 activity points accumulated in a month qualify for rewards. Rewards consist of awarding 10% of held tokens as additional tokens. Distribution occurs automatically on the last day of each month, emitting an
```

```
            event.",
        "question": "What types of activities contribute to the activity points?"
    },
    {
        "smart_contract_description": "Program a highly efficient ERC721 Smart Contract
            for the Fan Engagement Platform to distribute rewards based on activity and
            token holdings at the end of each month. Fans with more than 200 tokens held
            and at least 50 activity points accumulated in a month qualify for rewards.
            Rewards consist of awarding 10% of held tokens as additional tokens.
            Distribution occurs automatically on the last day of each month, emitting an
            event.",
        "question": "Is there a cap on the number of tokens a user can hold or earn
            through rewards?"
    },
    {
        "smart_contract_description": "Program a highly efficient ERC721 Smart Contract
            for the Fan Engagement Platform to distribute rewards based on activity and
            token holdings at the end of each month. Fans with more than 200 tokens held
            and at least 50 activity points accumulated in a month qualify for rewards.
            Rewards consist of awarding 10% of held tokens as additional tokens.
            Distribution occurs automatically on the last day of each month, emitting an
            event.",
        "question": "How is the 10% reward calculated and distributed?"
    },
    {
        "smart_contract_description": "Program a highly efficient ERC721 Smart Contract
            for the Fan Engagement Platform to distribute rewards based on activity and
            token holdings at the end of each month. Fans with more than 200 tokens held
            and at least 50 activity points accumulated in a month qualify for rewards.
            Rewards consist of awarding 10% of held tokens as additional tokens.
            Distribution occurs automatically on the last day of each month, emitting an
            event.",
        "question": "What happens if a user falls below the 200 token threshold before
            the end of the month but has accumulated 50 activity points?"
    },
    {
        "smart_contract_description": "It is intended to manage tiered VIP memberships,
            allowing fans to upgrade or downgrade their membership based on the tokens
            they hold. Define membership levels as Bronze, Silver, and Gold, each with
            associated perks. Upgrading requires the owner to have a sufficient number of
             additional tokens (100 tokens for Silver, 200 for Gold). Include
            functionality for members to voluntarily lower their level, refunding tokens
            proportionally based on their tenure on the platform (70% if >2 years, 50% if
             >1 year, 30% if >6 months). Once downgraded, a member must wait 12 days
            before upgrading again.",
        "question": "How are the perks for each membership level defined and updated?"
    },
    {
        "smart_contract_description": "It is intended to manage tiered VIP memberships,
            allowing fans to upgrade or downgrade their membership based on the tokens
            they hold. Define membership levels as Bronze, Silver, and Gold, each with
            associated perks. Upgrading requires the owner to have a sufficient number of
             additional tokens (100 tokens for Silver, 200 for Gold). Include
            functionality for members to voluntarily lower their level, refunding tokens
            proportionally based on their tenure on the platform (70% if >2 years, 50% if
             >1 year, 30% if >6 months). Once downgraded, a member must wait 12 days
            before upgrading again.",
        "question": "Is there a mechanism to prevent users from upgrading immediately
            after downgrading to game the system?"
    },
    {
        "smart_contract_description": "It is intended to manage tiered VIP memberships,
            allowing fans to upgrade or downgrade their membership based on the tokens
            they hold. Define membership levels as Bronze, Silver, and Gold, each with
            associated perks. Upgrading requires the owner to have a sufficient number of
             additional tokens (100 tokens for Silver, 200 for Gold). Include
            functionality for members to voluntarily lower their level, refunding tokens
            proportionally based on their tenure on the platform (70% if >2 years, 50% if
             >1 year, 30% if >6 months). Once downgraded, a member must wait 12 days
            before upgrading again.",
        "question": "How is tenure on the platform calculated and verified for token
            refunds?"
    },
    {
        "smart_contract_description": "It is intended to manage tiered VIP memberships,
            allowing fans to upgrade or downgrade their membership based on the tokens
            they hold. Define membership levels as Bronze, Silver, and Gold, each with
            associated perks. Upgrading requires the owner to have a sufficient number of
```

```
            additional tokens (100 tokens for Silver, 200 for Gold). Include
            functionality for members to voluntarily lower their level, refunding tokens
            proportionally based on their tenure on the platform (70% if >2 years, 50% if
            >1 year, 30% if >6 months). Once downgraded, a member must wait 12 days
            before upgrading again.",
        "question": "Are there any fees associated with upgrading or downgrading
            membership?"
    },
    {
        "smart_contract_description": "It is intended to manage tiered VIP memberships,
            allowing fans to upgrade or downgrade their membership based on the tokens
            they hold. Define membership levels as Bronze, Silver, and Gold, each with
            associated perks. Upgrading requires the owner to have a sufficient number of
            additional tokens (100 tokens for Silver, 200 for Gold). Include
            functionality for members to voluntarily lower their level, refunding tokens
            proportionally based on their tenure on the platform (70% if >2 years, 50% if
            >1 year, 30% if >6 months). Once downgraded, a member must wait 12 days
            before upgrading again.",
        "question": "Can members upgrade or downgrade multiple levels at once (e.g., from
            Gold to Bronze)"
    },
    {
        "smart_contract_description": "A smart contract is needed to manage multiple
            subscription levels where fans pay a monthly fee in tokens to access
            exclusive content. Subscription levels include Basic (10 tokens/month),
            Premium (50 tokens/month), and Elite (100 tokens/month). Automatically deduct
            fees from a user's token balance every 30 days. If there are insufficient
            funds, the subscription is paused. Implement auto-renewal features, and if a
            subscription expires, access to exclusive content is immediately revoked
            until renewal.",
        "question": "How is the token balance checked and fees deducted automatically
            every 30 days?"
    },
    {
        "smart_contract_description": "A smart contract is needed to manage multiple
            subscription levels where fans pay a monthly fee in tokens to access
            exclusive content. Subscription levels include Basic (10 tokens/month),
            Premium (50 tokens/month), and Elite (100 tokens/month). Automatically deduct
            fees from a user's token balance every 30 days. If there are insufficient
            funds, the subscription is paused. Implement auto-renewal features, and if a
            subscription expires, access to exclusive content is immediately revoked
            until renewal.",
        "question": "Is there a grace period before pausing the subscription due to
            insufficient funds?"
    },
    {
        "smart_contract_description": "A smart contract is needed to manage multiple
            subscription levels where fans pay a monthly fee in tokens to access
            exclusive content. Subscription levels include Basic (10 tokens/month),
            Premium (50 tokens/month), and Elite (100 tokens/month). Automatically deduct
            fees from a user's token balance every 30 days. If there are insufficient
            funds, the subscription is paused. Implement auto-renewal features, and if a
            subscription expires, access to exclusive content is immediately revoked
            until renewal.",
        "question": "How is the auto-renewal feature managed and can users opt-out?"
    },
    {
        "smart_contract_description": "A smart contract is needed to manage multiple
            subscription levels where fans pay a monthly fee in tokens to access
            exclusive content. Subscription levels include Basic (10 tokens/month),
            Premium (50 tokens/month), and Elite (100 tokens/month). Automatically deduct
            fees from a user's token balance every 30 days. If there are insufficient
            funds, the subscription is paused. Implement auto-renewal features, and if a
            subscription expires, access to exclusive content is immediately revoked
            until renewal.",
        "question": "What happens to the unused portion of a subscription if a user
            manually cancels before the end of the 30-day period?"
    },
    {
        "smart_contract_description": "A smart contract is needed to manage multiple
            subscription levels where fans pay a monthly fee in tokens to access
            exclusive content. Subscription levels include Basic (10 tokens/month),
            Premium (50 tokens/month), and Elite (100 tokens/month). Automatically deduct
            fees from a user's token balance every 30 days. If there are insufficient
            funds, the subscription is paused. Implement auto-renewal features, and if a
            subscription expires, access to exclusive content is immediately revoked
            until renewal.",
        "question": "Can users upgrade or downgrade their subscription level at any time,
```

```
                and how is the fee adjustment handled?"
    },
    {
        "smart_contract_description": "Design a smart contract that tracks and rates fan
            interactions for each content piece. Fans earn points that influence their
            ranking on the platform, with interactions categorized as likes, comments,
            shares, and views. Each interaction type carries different weightings (1
            point for likes, 10 points for shares, 20 points for comments). Fans can earn
             a maximum of 100 points per day from each activity. Platinum and Gold tier
            members can earn up to 150 and 200 points, respectively, but the extra points
             expire after a week.",
        "question": "How are interactions tracked and points allocated in real-time?"
    },
    {
        "smart_contract_description": "Design a smart contract that tracks and rates fan
            interactions for each content piece. Fans earn points that influence their
            ranking on the platform, with interactions categorized as likes, comments,
            shares, and views. Each interaction type carries different weightings (1
            point for likes, 10 points for shares, 20 points for comments). Fans can earn
             a maximum of 100 points per day from each activity. Platinum and Gold tier
            members can earn up to 150 and 200 points, respectively, but the extra points
             expire after a week.",
        "question": "What mechanisms are in place to prevent fraudulent interactions or
            spamming?"
    },
    {
        "smart_contract_description": "Design a smart contract that tracks and rates fan
            interactions for each content piece. Fans earn points that influence their
            ranking on the platform, with interactions categorized as likes, comments,
            shares, and views. Each interaction type carries different weightings (1
            point for likes, 10 points for shares, 20 points for comments). Fans can earn
             a maximum of 100 points per day from each activity. Platinum and Gold tier
            members can earn up to 150 and 200 points, respectively, but the extra points
             expire after a week.",
        "question": "How are extra points for Platinum and Gold members calculated and
            tracked?"
    },
    {
        "smart_contract_description": "Design a smart contract that tracks and rates fan
            interactions for each content piece. Fans earn points that influence their
            ranking on the platform, with interactions categorized as likes, comments,
            shares, and views. Each interaction type carries different weightings (1
            point for likes, 10 points for shares, 20 points for comments). Fans can earn
             a maximum of 100 points per day from each activity. Platinum and Gold tier
            members can earn up to 150 and 200 points, respectively, but the extra points
             expire after a week.",
        "question": "What happens to the points once they expire after a week for higher-
            tier members?"
    },
    {
        "smart_contract_description": "Design a smart contract that tracks and rates fan
            interactions for each content piece. Fans earn points that influence their
            ranking on the platform, with interactions categorized as likes, comments,
            shares, and views. Each interaction type carries different weightings (1
            point for likes, 10 points for shares, 20 points for comments). Fans can earn
             a maximum of 100 points per day from each activity. Platinum and Gold tier
            members can earn up to 150 and 200 points, respectively, but the extra points
             expire after a week.",
        "question": "Is there a leaderboard or ranking system to display fan rankings
            based on points?"
    }
]
```

Articles

[Cho56] Noam Chomsky. "Three models for the description of language". In: *IRE Transactions on information theory* 2.3 (1956), pp. 113–124. URL: https://chomsky.info/wp-content/uploads/195609-.pdf.

[Ros58] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain". In: *Psychological review* 65.6 (1958), p. 386. URL: https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf.

[You67] Daniel H Younger. "Recognition and parsing of context-free languages in time n3". In: *Information and control* 10.2 (1967), pp. 189–208. URL: https://core.ac.uk/download/pdf/82305262.pdf.

[Roy70] Winston W. Royce. "Managing the Development of Large Software Systems". In: *Proceedings of IEEE WESCON* (1970), pp. 1–9. URL: https://www.praxisframework.org/files/royce1970.pdf.

[Hop82] John J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC346238/pdf/pnas00447-0135.pdf.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. URL: https://www.cs.utoronto.ca/~hinton/absps/naturebp.pdf.

[Rab89] Lawrence R Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286. URL: https://www.cs.ubc.ca/~murphyk/Bayes/rabiner.pdf.

[Sza96] Nick Szabo. "Smart contracts: building blocks for digital markets". In: *EXTROPY: The Journal of Transhumanist Thought* 16.18 (1996).

[Hoc97] Sepp et al. Hochreiter. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780. URL: https://www.bioinf.jku.at/publications/older/2604.pdf.

[Wol+02] Jonathan R Wolpaw et al. "Brain–computer interfaces for communication and control". In: *Clinical neurophysiology* 113.6 (2002), pp. 767–791. URL: https://www.cs.cmu.edu/~tanja/BCI/BCIreview.pdf.

[Mik+13] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv preprint arXiv:1301.3781* (2013). URL: https://arxiv.org/pdf/1301.3781.

[Lit+17] Geert Litjens et al. "A survey on deep learning in medical image analysis". In: *Medical image analysis* 42 (2017), pp. 60–88. URL: https://arxiv.org/pdf/1702.05747.

[Sha+17] Noam Shazeer et al. "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer". In: *arXiv preprint arXiv:1701.06538* (2017). URL: https://arxiv.org/abs/1701.06538.

[Vas+17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017). URL: https://arxiv.org/pdf/1706.03762.

[Bak18] Amir Bakarov. "A Survey of Word Embeddings Evaluation Methods". In: *arXiv preprint arXiv:1801.09536* (2018).

[Dev+19] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2019). URL: https://arxiv.org/abs/1810.04805.

[Liu+19] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *arXiv preprint arXiv:1907.11692* (2019). URL: https://arxiv.org/abs/1907.11692.

[Qi+19] Peng Qi et al. "Answering Complex Open-Domain Questions Through Iterative Query Generation". In: *arXiv preprint arXiv:1910.07000* (2019). URL: https://arxiv.org/abs/1910.07000.

[SKG19] Christian Stoll, Lena Klaaßen, and Ulrich Gallersdörfer. "The Carbon Footprint of Bitcoin". In: *Joule* 3.7 (2019), pp. 1647–1661. DOI: 10.1016/j.joule.2019.05.012.

[Bro+20] Tom B Brown et al. "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165* (2020). URL: https://arxiv.org/abs/2005.14165.

[Bom21] et al. Bommasani. "On the Opportunities and Risks of Foundation Models". In: *arXiv preprint arXiv:2108.07258* (2021). URL: https://arxiv.org/abs/2108.07258.

[Che+21] Mark Chen et al. "Evaluating Large Language Models Trained on Code". In: *arXiv preprint arXiv:2107.03374* (2021). URL: https://arxiv.org/abs/2107.03374.

[Dos+21] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *arXiv preprint arXiv:2010.11929* (2021). URL: https://arxiv.org/abs/2010.11929.

[Jum+21] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596 (2021), pp. 583–589. DOI: 10.1038/s41586-021-03819-2.

[Nak+21] Rei Nakano et al. "WebGPT: Browser-assisted Question-answering with Human Feedback". In: *arXiv preprint arXiv:2112.09332* (2021). URL: https://arxiv.org/abs/2112.09332.

[Ahn+22] Michael Ahn et al. "Do as I can, not as I say: Grounding Language in Robotic Affordances". In: *arXiv preprint arXiv:2204.01691* (2022). URL: https://arxiv.org/abs/2204.01691.

[Che22] et al. Chen. "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks". In: *arXiv preprint arXiv:2211.12588* (2022). URL: https://arxiv.org/abs/2211.12588.

[Doh+22] David Dohan et al. "Language Model Cascades". In: *arXiv preprint arXiv:2207.10342* (2022). URL: https://arxiv.org/abs/2207.10342.

[Hof+22] Jordan Hoffmann et al. "Training Compute-Optimal Large Language Models". In: *arXiv preprint arXiv:2203.15556* (2022). URL: https://arxiv.org/abs/2203.15556.

[Hua+22] Jiaxin Huang et al. "Large language models can self-improve". In: *arXiv preprint* (2022). eprint: arXiv:2210.11610. URL: https://arxiv.org/abs/2210.11610.

[Kad+22] Saurav Kadavath et al. "Language Models (Mostly) Know What They Know". In: *arXiv preprint arXiv:2207.05221* (2022). URL: https://arxiv.org/abs/2207.05221.

[Kha+22] Omar Khattab et al. "Demonstrate-Search-Predict: Composing retrieval and language models for knowledge-intensive NLP". In: *arXiv preprint arXiv:2212.14024* (2022). URL: https://arxiv.org/abs/2212.14024.

[Kho22] et al. Khot. "Decomposed Prompting: A Modular Approach for Solving Complex Tasks". In: *arXiv preprint arXiv:2210.02406* (2022). URL: https://arxiv.org/abs/2210.02406.

[Koj22] et al. Kojima. "Large Language Models are Zero-Shot Reasoners". In: *arXiv preprint arXiv:2205.11916* (2022). URL: https://arxiv.org/abs/2205.11916.

[Wan+22] Xuezhi Wang et al. "Self-Consistency Improves Chain of Thought Reasoning in Language Models". In: *arXiv preprint arXiv:2203.11171* (2022). URL: https://arxiv.org/abs/2203.11171.

[Wei+22] J. Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *arXiv preprint arXiv:2201.11903* (2022). URL: https://arxiv.org/abs/2201.11903.

[Yao+22] Shunyu Yao et al. "ReAct: Synergizing Reasoning and Acting in Language Models". In: *arXiv preprint arXiv:2210.03629* (2022). URL: https://doi.org/10.48550/arXiv.2210.03629.

[Che+23] Guangyao Chen et al. "AutoAgents: A Framework for Automatic Agent Generation". In: *arXiv preprint arXiv:2309.17288* (2023). URL: https://arxiv.org/abs/2309.17288.

[Den23] et al. Deng. "Rephrase and Respond (RaR)". In: *arXiv preprint arXiv:2311.04205* (2023). URL: https://arxiv.org/abs/2311.04205.

[Dhu+23] Dhruv Dhuliawala et al. "Knowing What LLMs DO NOT Know: A Simple Yet Effective Self-Detection Method". In: *arXiv preprint arXiv:2310.17918* (2023). URL: https://arxiv.org/abs/2310.17918.

[Eva23] Brian D. Evans. "How Web3 could Revolutionize Loyalty Programs". In: *theCustomer* (2023). URL: https://thecustomer.net/how-web3-could-revolutionize-loyalty-programs/.

[Fan23] Fanprime. "Web3 Loyalty Programs: 5 Benefits for Customer Loyalty". In: *Fanprime Blog* (2023). URL: https://fanprime.io/blog/web3-loyalty-programs/.

[Guo+23] Qingyan Guo et al. "Connecting large language models with evolutionary algorithms yields powerful prompt optimizers". In: *arXiv preprint* (2023). eprint: arXiv:2309.08532. URL: https://arxiv.org/abs/2309.08532.

[Kad23] et al. Kadavath. "Self-Consistency Boosts Calibration for Math Reasoning". In: *arXiv preprint arXiv:2403.09849* (2023). URL: https://arxiv.org/abs/2403.09849.

[Mad+23] Aman Madaan et al. "Self-Refine: Iterative Refinement with LLMs". In: *arXiv preprint arXiv:2303.17651* (2023). URL: https://arxiv.org/abs/2303.17651.

[Ope23] OpenAI. "GPT-4 Technical Report". In: *arXiv preprint arXiv:2303.08774* (2023). URL: https://arxiv.org/pdf/2303.08774.

[Par+23] Joseph Seunghyun Park et al. "Generative Agents: Interactive Simulacra of Human Behavior". In: *arXiv preprint arXiv:2304.03442* (2023). URL: https://arxiv.org/abs/2304.03442.

[Pou23] et al. Pourreza. "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction". In: *arXiv preprint arXiv:2304.11015* (2023). URL: https://arxiv.org/abs/2304.11015.

[Pry+23] R. Pryzant et al. "Automatic prompt optimization with "gradient descent" and beam search". In: *arXiv preprint arXiv:2305.03495* (2023). URL: https://arxiv.org/abs/2305.03495.

[Qin+23] Hanmeng Qin et al. "Evaluating the Logical Reasoning Ability of ChatGPT and GPT-4". In: *arXiv preprint arXiv:2304.03439* (2023). URL: https://arxiv.org/pdf/2304.03439.

[Rig+23] Maria Rigaki et al. "Out of the Cage: How Stochastic Parrots Win in Cyber Security Environments". In: (Aug. 2023). URL: https://arxiv.org/abs/2308.12086.

[SMK23] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. "Are Emergent Abilities of Large Language Models a Mirage?" In: *arXiv preprint arXiv:2304.15004* (2023). URL: https://arxiv.org/abs/2304.15004.

[Sch+23] Timo Schick et al. "Toolformer: Language Models Can Teach Themselves to Use Tools". In: *arXiv preprint arXiv:2302.04761* (2023). URL: https://arxiv.org/abs/2302.04761.

[She+23] Yujia Shen et al. "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in HuggingFace". In: *arXiv preprint arXiv:2303.17580* (2023). URL: https://arxiv.org/abs/2303.17580.

[Shi23] et al. Shinn. "Reflexion: an autonomous agent with dynamic memory and self-reflection". In: *arXiv preprint arXiv:2303.11366* (2023). URL: https://arxiv.org/abs/2303.11366.

[The23] The Audiencers. "The loyalty flywheel: increasing customer loyalty and ARPU with a new sales perspective and not only for publishers". In: *The Audiencers Blog* (2023). URL: https://theaudiencers.com/the-loyalty-flywheel-increasing-customer-loyalty-and-arpu-with-a-new-sales-perspective-and-not-only-for-publishers/.

[Tou+23] Hugo Touvron et al. "LLaMA: Open and Efficient Foundation Language Models". In: *arXiv preprint arXiv:2302.13971* (2023). URL: https://arxiv.org/abs/2302.13971.

[Vac+23] Anna Vacca et al. "Functional suitability assessment of smart contracts: A survey and first proposal". In: *Journal of Software: Evolution and Process* e2636 (2023). DOI: 10.1002/smr.2636. URL: https://onlinelibrary.wiley.com/doi/10.1002/smr.2636.

[Wei+23] Jerry Wei et al. "Larger language models do in-context learning differently". In: *arXiv preprint arXiv:2303.03846* (2023). URL: https://arxiv.org/abs/2303.03846.

[Wes23] et al. Weston. "System 2 Attention (is something you might need too)". In: *arXiv preprint arXiv:2311.11829* (2023). URL: https://arxiv.org/abs/2311.11829.

[Wu+23] Qingyun Wu et al. "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework". In: *arXiv preprint arXiv:2308.08155* (2023). URL: https://arxiv.org/abs/2308.08155.

[Xu23] et al. Xu. "Re-reading (RE2)". In: *arXiv preprint arXiv:https://arxiv.org/abs/2309.06275v2* (2023). URL: https://arxiv.org/abs/2309.06275v2.

[Xue23] et al. Xue. "Reversible Chain-of-Thought for LLMs". In: *arXiv preprint arXiv:2305.11499* (2023). URL: https://arxiv.org/abs/2305.11499.

[Yan+23] Chengrun Yang et al. "Large language models as optimizers". In: *arXiv preprint* (2023). eprint: arXiv:2309.03409. URL: https://arxiv.org/abs/2309.03409.

[Yao+23] S. Yao et al. "Tree of thoughts: Deliberate problem solving with large language models". In: *arXiv preprint arXiv:2305.10601* (2023). URL: https://arxiv.org/abs/2305.10601.

[Zha+23] Y. Zhang et al. "Cumulative reasoning with large language models". In: *arXiv preprint arXiv:2308.04371* (2023). URL: https://arxiv.org/abs/2308.04371.

[AI24] Meta AI. "Introducing Meta Llama 3: The most capable openly available LLM". In: *Meta AI Blog* (2024). URL: https://ai.meta.com/blog/meta-llama-3/.

[Ant24] Anthropic. "Introducing the next generation of Claude". In: *Anthropic Blog* (2024). URL: https://www.anthropic.com/news/claude-3-family.

[Bha+24] Aman Bhargava et al. "What's the Magic Word? A Control Theory of LLM Prompting". In: *arXiv preprint arXiv:2310.04444v3* (2024). URL: https://arxiv.org/abs/2310.04444v3.

[MIT24] MIT Technology Review. "OpenAI's new GPT-4o model lets people interact using voice or video in the same model". In: *MIT Technology Review* (May 2024). URL: https://www.technologyreview.com/2024/05/13/1092358/openais-new-gpt-4o-model-lets-people-interact-using-voice-or-video-in-the-same-model/.

[NNB24] Khanh Nghiem, Anh Minh Nguyen, and Nghi D.Q. Bui. "Envisioning the Next-Generation AI Coding Assistants: Insights & Proposals". In: *arXiv preprint arXiv:2403.14592* (2024). URL: https://arxiv.org/abs/2403.14592.

[Ope24a] OpenAI. "Hello GPT-4o". In: *OpenAI Blog* (May 2024). URL: https://openai.com/index/hello-gpt-4o/.

[RG24] Matthew Renze and Erhan Guven. "Self-Reflection in LLM Agents: Effects on Problem-Solving Performance". In: *arXiv preprint arXiv:2405.06682* (2024). URL: https://doi.org/10.48550/arXiv.2405.06682.

[Tec24] TechTimes. "Microsoft's Strategic $1 Billion Investment in OpenAI to Counter Google's AI Dominance$". In: *TechTimes* (2024). URL: https://www.techtimes.com/articles/304187/20240501/microsofts-strategic-1-billion-investment-openai-counter-googles-ai-dominance.htm.

[The24] The Guardian. "New GPT-4o AI model is faster and free for all users, OpenAI announces". In: *The Guardian* (May 2024). URL: https://www.theguardian.com/technology/article/2024/may/13/openai-new-chatgpt-free.

[Yan+24] Jiuding Yang et al. "SIFiD: Reassess Summary Factual Inconsistency Detection with LLM". In: *arXiv preprint arXiv:2403.07557* (2024). URL: https://arxiv.org/abs/2403.07557.

Books

[TS06] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, 2006. ISBN: 9780132392273.

[Kah11] Daniel Kahneman. *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux, 2011.

[PM14] Roger S. Pressman and Bruce R. Maxim. *Software Engineering: A Practitioner's Approach*. 8th. McGraw-Hill Education, 2014. ISBN: 978-0078022128.

[Nar+16] Arvind Narayanan et al. *Bitcoin and Cryptocurrency Technologies*. Princeton University Press, 2016. ISBN: 9780691171692.

[Ant17] Andreas M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. 2nd. O'Reilly Media, 2017. ISBN: 9781491954386.

[Dan17] Chris Dannen. *Introducing Ethereum and Solidity*. Apress, 2017.

[AW18] Andreas M. Antonopoulos and Gavin Wood. *Mastering Ethereum*. O'Reilly Media, Inc., 2018. ISBN: 9781491971932.

[SKH19] K. Solorio, R. Kanna, and D. H. Hoover. *Hands-on Smart Contract Development with Solidity and Ethereum: From Fundamentals to Deployment*. O'Reilly Media, 2019.

Misc

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. N.p., Web. 30 Jan 2014. https://bitcoin.org/bitcoin.pdf. 2008.

[BC14] Anton Badev and Matthew Chen. *Bitcoin: Technical Background and Data Analysis*. 2014. URL: https://www.federalreserve.gov/econresdata/feds/2014/files/2014104pap.pdf.

[KPM19] KPMG. *The Truth About Customer Loyalty*. https://assets.kpmg.com/content/dam/kpmg/ar/pdf/the-truth-about-customer-loyalty.pdf. 2019.

[Git21] GitHub. *Introducing GitHub Copilot: Your AI Pair Programmer*. https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/. Accessed: 2024-06-03. 2021.

[NVI21] NVIDIA. *NVIDIA Ampere Architecture*. https://www.nvidia.com/en-us/data-center/nvidia-ampere-gpu-architecture/. Accessed: 2024-06-03. 2021.

[Rud21] Sebastian Ruder. *A review of the recent history of natural language processing*. 2021. URL: https://www.ruder.io/a-review-of-the-recent-history-of-nlp/.

[Sta21] Statista. *Global market size of loyalty management*. 2021. URL: https://statista.com/statistics/1295852/loyalty-management-market-size-world/.

[Ant22] Antavo. *Global Customer Loyalty Report 2022*. https://antavo.com/wp-content/uploads/2021/12/Global-Customer-Loyalty-Report-2022-by-Antavo.pdf?_hsmi=61681060&_hsenc=p2ANqtz--nwVcL-u61RHvlUkruaQg4i_TylQOh6sf21QVko-06sVpajDDiPZr5WDlN_b3CH6ln1jDcxqJy2fcdTNI4bf2hPywKEw. 2022.

[Con23a] LangChain Contributors. *LangChain: A Language Model Chain Framework*. Accessed: 2024-06-11. 2023. URL: https://www.langchain.com.

[Con23b] Llama Index Contributors. *Llama Index: An Indexing Library for Large Language Models*. Accessed: 2024-06-11. 2023. URL: https://www.llamaindex.com.

[Goo23a] Google. *Introducing Bard: An experimental conversational AI service*. Accessed on 2023-06-21. 2023. URL: https://blog.google/technology/ai/bard-google-ai-search-updates/.

[Goo23b] Google. *Introducing Gemini: our largest and most capable AI model*. Accessed on 2023-06-21. 2023. URL: https://blog.google/technology/ai/google-gemini-ai/.

[Kha+23] Omar Khattab et al. *DSPy: Programming—not Prompting—Large Language Models*. 2023. URL: https://arxiv.org/pdf/2310.03714.

[Lun23] Ingrid Lunden. *Google invests* $300M$ *in artificial intelligence startup Anthropic*. Accessed on 2023-06-21. 2023. URL: https://techcrunch.com/2023/02/03/google-invests-300m-in-artificial-intelligence-startup-anthropic/.

[Mic23] Microsoft. *Semantic Kernel: A Multimodal AI Kernel*. Accessed: 2024-06-11. 2023. URL: https://github.com/microsoft/semantic-kernel.

[Ros23] Mark Roszak. *Large language models in FinTech: A boon or bane for compliance officers?* FinTech Global. 2023. URL: https://fintech.global/2023/07/20/large-language-models-in-fintech-a-boon-or-bane-for-compliance-officers/.

[LM-23] LM-SYS. *Chatbot Arena: An Open Platform for Evaluating Large Language Models.* https://chat.lmsys.org. Accessed on 2023-06-21. 2023.

[Vin23] James Vincent. *Google's Gemini demo was faked.* Accessed on 2023-06-21. 2023. URL: https://www.theverge.com/2023/12/7/23991672/google-gemini-ai-demo-fake-video.

[WM23] Daisuke Wakabayashi and Cade Metz. *Google Stumbles in Chatbot Race, Rushing to Catch Up.* Accessed on 2023-06-21. 2023. URL: https://www.nytimes.com/2023/03/29/technology/google-chatbot-ai-race.html.

[Wan+23] Xinlei Wang et al. *Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution.* 2023. URL: https://arxiv.org/abs/2309.16797.

[Age24] AgentLayer Team. *AgentLayer: Decentralized AI governance and model interoperability framework.* Accessed: 2024-02-11. 2024. URL: https://www.agentlayer.xyz/whitepaper.

[Dev24] Devika. *Devika Repository.* https://github.com/stitionai/devika. Accessed: 2024-03-03. 2024.

[Ini24] SWE-Bench Initiative. *Benchmarking AI Performance.* https://www.swebench.com/. Accessed: 2024-05-03. 2024.

[Ope24b] OpenDevin. *OpenDevin Repository.* https://github.com/OpenDevin/OpenDevin. Accessed: 2024-03-03. 2024.