Solus: An end-to-end AI software developer.

Adam Blumenfeld

President, CSX Labs

This paper presents Solus, a proposed end-to-end AI software development solution. It examines the rapid advancement of the generative AI space and identifies a gap in the current ecosystem: while existing tools such as GitHub's Copilot, Replit's Ghostwriter, and Amazon's Codewhisperer offer inline code completions to aid developers, there is no comprehensive autonomous software development solution. Furthermore, these systems often require manual intervention and revision. The Solus system aims to solve these issues by utilizing self-critical, multi-role AI agents to undertake all aspects of software development, including managing dependencies, refactoring, debugging, and generating business logic. It also proposes a robust, secure, and scalable approach that is capable of handling enterprise-level projects, with transparency and traceability of operations. The paper delves into the system architecture, discussing key components such as a control plane, agents, and resources. It makes a case for the unique design of Solus, which allows for scalability, customization, and optimization of the generative process by compartmentalizing agents and abstracting resources.

1 Introduction

1.1 Market Background

The seamless transfer of knowledge and data across nations powered the information age. Global access to electronic devices created high-growth platforms and services, springing millions of jobs in content creation, software development, and more. An estimated 26.9 million professional developers practiced worldwide as of 2022, growing 14.5% from 2018[1]. The inception of large language models gave rise to the generative AI space, with platforms growing at a record pace, such as OpenAI's ChatGPT, which rose to 100 million users in two months[2]. One implication of generative AI is that it automates large portions of the jobs created in the information revolution in the tech, legal, financial, media, and support fields, to name a few[3]. S&P Global Market Intelligence predicts generative AI market revenue to hit $36 billion by 2028, forecasting code generators to lead the expansion at a compound annual growth rate of 72.9%[4]. The ability of language models to generate code has sparked new products such as GitHub's Copilot[5], Replit's Ghostwriter[6], and Amazon's Codewhisperer[7], which provide inline code completions saving developers time from repetitive tasks, routine lookups, and forgotten implementations.

Inline completions are helpful but far from autonomous, requiring manual developer intervention and revision.

Projects like Smol[8], AutoGPT[9], and BabyAGI[10] aim to perform tasks autonomously by making language models reason and evaluate themselves. However, projects like AutoGPT and BabyAGI are general in project scope, leading to a jack-of-all-trades issue.

In contrast, Smol is simple though it lacks capability and requires much human intervention.

1.2 Problems

1.2.1 Scope Limits Capability

Tools like AutoGPT and BabyAGI aim to achieve artificial general intelligence (AGI), a system that can learn to carry out any task a human or animal can perform[11]. Although a novel goal, refraining from tailoring the generative process to a specific task limits the value, the systems can bring to such a complex task as software development. A capable AI software developer project

Adam Blumenfeld: adamb@csxlabs.org

must tailor agents to the software development lifecycle.

1.2.2 Multi-Agent Collaboration

There are no effective frameworks for multi-agent communication and collaboration on a resource in an organized manner. Software development is an inherently complex process, with many scoped concerns spanning a project with many dependencies and business-related tradeoffs to weigh in. As agents act on a project's resources, a standard protocol must exist for compartmentalizing working areas, communicating issues, and resolving conflicts.

1.2.3 Manual Feedback Loop

Most tools require an amount of human feedback and approval when operating to stay on track and approve resource usage. Language model costs incurred during critical reasoning and self-regulation limit more capable projects, such as AutoGPT. These costs are a barrier to making multi-agent systems possible, as the quality of these tools' performance as software developers render the cost unjustified. Therefore, an AI software developer system must have transparent resource monitoring and self-guidance, ensuring the quality of iteration.

1.3 Solution

Solus *will be*[1] an end-to-end AI software development solution utilizing self-critical, multi-role AI agents.

1.3.1 System Requirements

Solus must be robust, secure, and scalable to be a suitable system for enterprises to trust with their projects.

We achieve this by compartmentalizing agents and adding layers of abstraction to inter-agent communication, allowing us to optimize, shard, and secure resources under the hood.

[1]NOTE: We are raising funding to build Solus due to the steep upstart costs. Contact the corresponding author if you want to contribute financially.

1.3.2 Product Requirements

The system must be capable of doing most software development tasks, such as managing dependencies, refactoring, debugging, and generating business logic and documentation related to the project and business goals. It must execute these tasks self-regulating while ensuring all operations are transparent, traceable, and intervenable. We accomplish this by orchestrating the agents in a service mesh where sidecars intercept communications between agents and resources and beam them to a centralized control plane that controls downstream processes and provides visibility over the operation.

1.3.3 Business Requirements

Solus must be versatile, profitable, and sustainable. By making agents customizable components with a clear separation of concerns, we can give Solus a clear path of expansion to other applications in the generative AI space. By operating on a cloud model, we can stay profitable on a low operating margin, minimizing costs for us and our customers. To make Solus sustainable, we plan on creating a plugin ecosystem for optimized agent types for specific tasks and technologies. These developer ecosystems give Solus an economic moat over other initiatives. We also plan on building Solus to be open-source to ensure the most significant impact on the developer community. Open source allows us to create an ecosystem around our technology while maintaining profit and advantage due to enterprise trust in our platform.

2 System Architecture

2.1 Components

As shown in Figure 1, the Solus system comprises three primary components:

1. **Control Plane**: The control plane orchestrates agents and manages resources. It also provides an entry point for users to interact with the system and monitor operations. Agents and Resources beam all actions to the control plane, which analyzes them and decides how to proceed.

2. **Agents**: Agents act on resources and interact with each other to complete the task given

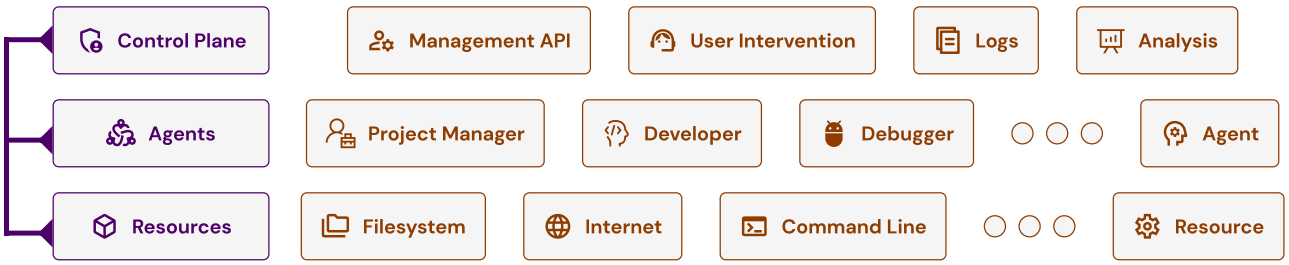

Figure 1: The system comprises a control plane, a service mesh, and a resource layer. The **control plane** orchestrates agents and manages resources. The **agents** are responsible for doing the generation, acting on different resources. **Resources** provide an interface for agents to interact with their environment, such as the filesystem and the internet, to get context and complete tasks.

by the user. Agents are customizable components with roles and a set of allowed operations. They communicate with each other to regulate each other. For example, a Project Management agent may help a Software Engineer agent to stay working on items relevant to the business concerns of the system. Agents are also responsible for self-regulation and monitoring their resource usage.

3. **Resources**: Agents interact with resources such as the filesystem, internet, and databases to complete tasks. Resources are abstracted away from agents, allowing us to optimize and shard them under the hood. Resources are also responsible for monitoring and reporting their usage to the control plane.

The abstraction of agents and resources decouples the generative process from the underlying infrastructure, allowing us to optimize and scale the system without affecting the generative process. It also provides room for expansion and customization, allowing us to create a plugin ecosystem for agent types and resource types.

2.2 Inter-Service Communication

As shown in Figure 2, the services interact through a service mesh. Sidecars intercept and standardize services' communications and beam them to the control plane. The control plane keeps logs of all system activity, intervening when needed or requested by the user. The other components of the system regulate their communications and complete actions independently. A service mesh allows agents and resources to handle their interactions while keeping a centralized log of all activity in the control plane. It will enable

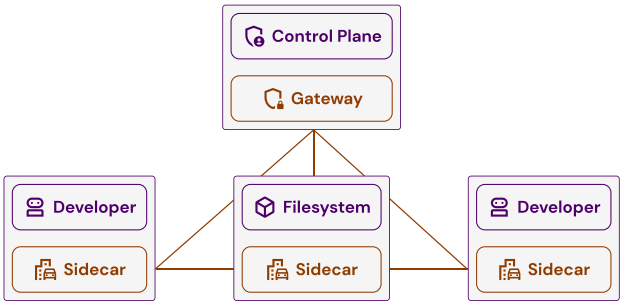

Figure 2: The services interact through a service mesh. Sidecars intercept and standardize services' communications and beam them to the control plane.

us to monitor and regulate the system while abstracting away the underlying infrastructure from the agents.

The service mesh design pattern was initially used to manage microservices in a cloud-native environment due to its de-coupling of the network layer from the application layer[12]. Due to the scale and compute requirements of competent language models[13], we must run agents on separate machines. A service mesh allows us to scale the system horizontally without affecting the generative process.

2.3 Agents

As shown in Figure 3, agents represent the people in a team, interacting with resources and each other to complete the project tasks. We utilize two models, a deliberator and an executor, to collaborate on tasks. Using a model to reason throughout a task before performing actions improves the quality of generations[14], similar to how humans reason through ideas before acting on them (most of the time). The deliberator gets high-level context, such as business values and principles. The executor gets lower-level

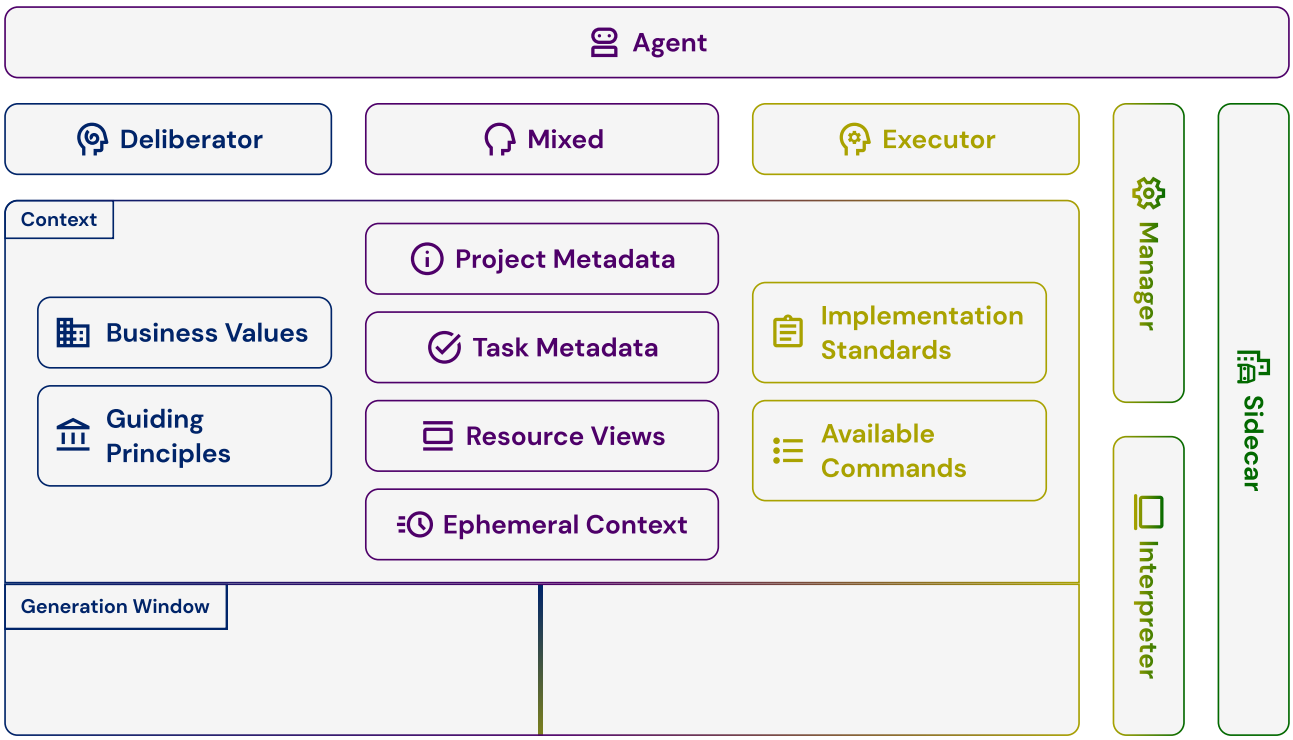

Figure 3: Agents represent the people in a software team. They act on resources and interact with each other to complete the task given by the user. They use two language models, a deliberator and an executor. The **deliberator** conducts unstructured reasoning about the current task, forming a critical chain of thinking and action. The **executor** takes the reasoning from the deliberator and converts it to commands according to a structured specification provided by the Agent's context. The **interpreter** serializes and parses the output from the executor, forming a set of requests to send to the manager. The **manager** manages the lifecycle of the agent's generations and updates the local context. It takes commands from the interpreter to send through the sidecar. As described in Figure 2, the **sidecar** manages all inbound and outbound requests between agents, beaming them to the control plane for processing and delegating them to the manager.

context like implementation standards-for example, a codebase style guide-and its available commands. The models also have shared context about the project, current task, resource views, and ephemeral context (short-term memory).

Resource views are the agent's view of the resources it interacts with. For example, a software engineer agent may have a view of the codebase, the filesystem, and the internet. A human software engineer opens files in their editor and websites in their browser and uses the terminal to run commands. The agent's resource views are similar to the human's but are abstracted away from the agent. The agent does not know how to open a file or run a command. It only knows how to request the manager to open a file or run a command. The manager then delegates the request to the sidecar, updating the resource view when the action is complete.

Their ephemeral context includes a shared truncated history of the deliberator and executor's generations. This context simulates the short-term memory human developers use when executing tasks.

2.4 Resources

As shown in Figure 4, resources interact with all entities involved with the task to complete it. For example, a software engineer agent may need to interact with the filesystem, the internet, and a database to complete a task. The agent will send requests to the manager, such as to get the contents of a file or run a command. The manager will then delegate the request to the sidecar, returning the result to the agent once the action is complete.

The resources are abstracted from the agents, allowing us to implement effective access control and rate limiting. For example, a software engineer agent may access the filesystem and the internet, not the database. If the software en-

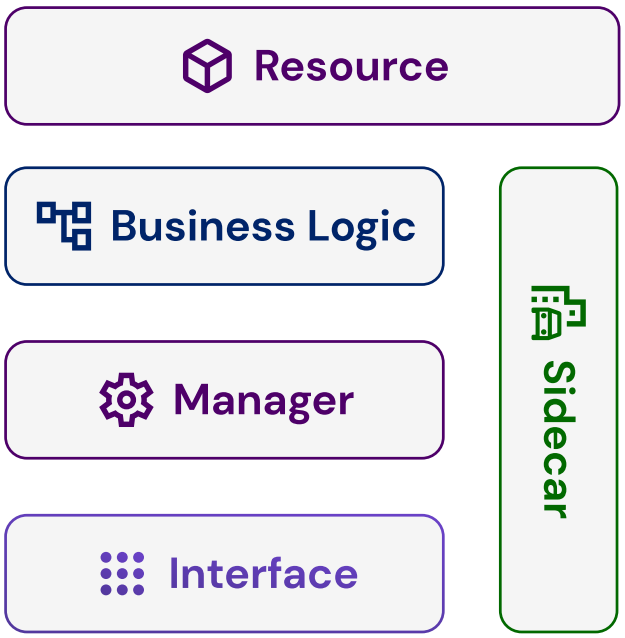

Figure 4: Resources manage all the underlying infrastructure and outside entities in the generation process. They consist of **business logic** that accesses the resource itself, a **manager** that schedules actions and monitors the resource, an **interface** that contains and validates the specification for interacting with the given resource, and a **sidecar** that manages all inbound and outbound requests made to the resource, beaming to the control plane for processing and delegating them to the manager.

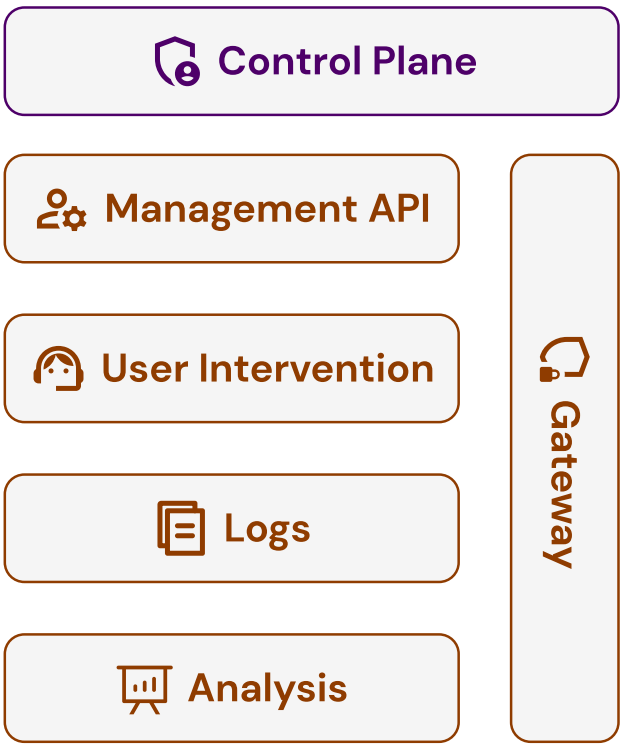

Figure 5: The control plane orchestrates agents and resources while providing total visibility and analysis into the operation. It contains a **management API** that provisions resources and agents, populating their contexts, managing service discovery, and regulating service usage and costs. The control plane also contains a **user intervention** interface for users to monitor and intervene with intermittent tasks. The **logs** are a centralized store of all information flowing through the systems, as resources and agents beam every interaction to the control plane. An **analysis** component analyzes the logs to interpret the operation's state. This state can contain resource usage, task tracking, and generative metrics. Finally, the **gateway** is the entry point for all requests to the control plane. It manages network layer authentication and authorization, ensuring only authorized users and services can access the control plane.

gineer agent is compromised, the attacker cannot access the database due to the access control implemented in the database resource. The resource also allows us to implement rate limiting, preventing the agent from overloading expensive resources. For example, the agent may only be able to run one command at a time, preventing it from overloading the CPU.

2.5 Control Plane

As shown in Figure 5, the control plane contains a centralized control point and information store of the system. As every action beams to the control plane, it can monitor and regulate the system without navigating and routing requests between agents and resources. Using a centralized information store also allows us to analyze the operation's state, providing aggregate metrics and insights into the system's operation and granular metrics and insights into individual agents and resources. The idea of using a centralized log to analyze the operation's state is similar to Apache Kafka[15], which uses a centralized log as an event-driven data store for real-time data processing.

The control plane also contains a management API for provisioning resources and agents, managing service discovery, and regulating service usage and costs.

3 Conclusion

The Solus system utilizes the wide-reaching potential of generative AI to revolutionize the field of software development. Creating an autonomous end-to-end AI software develop-

ment solution expands the functions of current AI developers from inline completion methods to fully-fledged, autonomous programming processes. Employing a multi-agent system ensures that software development's traditional compartmentalization is maintained, providing targeted solutions for each aspect of the development cycle. Furthermore, its service mesh, compartmentalized agents, and abstracted resources allow optimization, scalability, and security to be maintained, letting Solus easily handle enterprise-scale projects. The business model balances profitability with creating a thriving open-source developer ecosystem. This development represents a significant leap forward in the evolution of software development, wholeheartedly embracing the era of AI.

References

[1] Qubit Labs. How Many Programmers are there in the World and in the US? [2023]. *Qubit Labs*, nov 29 2022. [Online; accessed 2023-07-16].

[2] David Carr. Chatgpt Topped 1 Billion Visits in February. *Similarweb*, mar 7 2023. [Online; accessed 2023-07-16].

[3] Aaron Mok and Jacob Zinkula. Chatgpt may be coming for our jobs. Here are the 10 roles that AI is most likely to replace. *Insider*, jun 4 2023. [Online; accessed 2023-07-16].

[4] SungHa Park. Generative AI Software Market Forecast to Expand Near 10 Times by 2028 to $36 Billion, S&P Global Market Intelligence Says. *S&P Global*, jun 8 2023. [Online; accessed 2023-07-16].

[5] Thomas Dohmke. Github Copilot is generally available to all developers. *The GitHub Blog*, jun 21 2022. [Online; accessed 2023-07-16].

[6] Amjad Masad, Samip Dahal, Giuseppe Burtini, and Alexandre Cai. Ghostwriter AI & Complete Code Beta. *Replit Blog*, sep 8 2022. [Online; accessed 2023-07-16].

[7] Sylvia Engdahl. Amazon CodeWhisperer, Free for Individual Use, is Now Generally Available. *Amazon Web Services*, apr 13 2023. [Online; accessed 2023-07-16].

[8] Anton Osika. Current state of virtual developers. *Future tools*, jun 4 2023. [Online; accessed 2023-07-16].

[9] Sabrina Ortiz. What is Auto-GPT? Everything to know about the next powerful AI tool. *Zdnet*, apr 14 2023. [Online; accessed 2023-07-16].

[10] Sriram Parthasarathy. Meet BabyAGI — The Autonomous AI Agent to Streamline Your Tasks. *Towards AI*, may 10 2023. [Online; accessed 2023-07-16].

[11] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of Artificial General Intelligence: Early experiments with GPT-4. https://arxiv.org/abs/2303.12712, mar 22 2023.

[12] Wubin Li, Yves Lemieux, Jing Gao, Zhuofeng Zhao, and Yanbo Han. Service Mesh: Challenges, State of the Art, and Future Research Opportunities. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 122–1225.

[13] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and Efficient Foundation Language Models. https://arxiv.org/abs/2302.13971, feb 27 2023.

[14] Yao Fu, Hao Peng, Tushar Khot, and Mirella Lapata. Improving Language Model Negotiation with Self-Play and In-Context Learning from AI Feedback. https://arxiv.org/abs/2305.10142, may 17 2023.

[15] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, volume 11, pages 1–7. Athens, Greece, 2011.