Inflect-GTM: An Autonomous Multi-Agent System for Go-to-Market Automation

Mintae Kim

July 2025

Abstract

We present *Inflect-GTM*, an end-to-end autonomous agent framework designed to automate Go-to-Market (GTM) tasks such as customer onboarding, data analysis, document generation, and follow-up communications. The system employs a **multi-agent architecture** where specialized agents (RootAgent, AnalystAgent, DocumentWriterAgent, PostDemoAgent) collaborate via shared memory to accomplish complex workflows. Each agent is built around a Large Language Model (LLM) configured with a specific role prompt and augmented with tool-use capabilities. The framework integrates with external services including Google Workspace (Sheets, Docs, Gmail, Calendar) and Slack through dedicated tool APIs, enabling agents to perceive and act on real-world data. We also implement a **dynamic agent builder** using a FastAPI server, allowing on-demand creation of custom agents with specified tasks and tools. We detail the system pipeline, agent implementations at the code level, memory management strategies, and tool integration methods. Our analysis situates Inflect-GTM in the context of recent advancements in autonomous agents, such as the ReAct paradigm for LLM-based reasoning and action and multi-agent collaboration frameworks. We include a comprehensive pipeline diagram and pseudocode to illustrate the agent orchestration. The Inflect-GTM project demonstrates how a modular, multi-agent approach can automate complex business processes, leveraging the strengths of LLMs and pragmatic tool use while maintaining global state and inter-agent coordination.

1 Introduction

Large Language Models (LLMs) have recently demonstrated strong capabilities in reasoning, generation, and sequential decision-making. Beyond text completion, recent frameworks explore *agent-based paradigms*, where an LLM is guided by system prompts and equipped with tools to observe and act in external environments. Approaches such as ReAct [2] show that combining chain-of-thought reasoning with tool use improves grounding and task completion in complex environments.

Expanding this idea, **multi-agent systems**—where each LLM-based agent is assigned a specific sub-role—offer a path to modularity, robustness, and interpretability. Examples like AutoGen [**?**] and MetaGPT [1] demonstrate how agents acting under specialized roles (analyst, planner, coder) can jointly complete large-scale tasks via shared memory or messaging protocols.

We introduce **Inflect-GTM**, an open-source multi-agent framework for automating the Go-to-Market (GTM) process—a structured workflow encompassing customer segmentation, onboarding document generation, and post-demo communication. Inflect-GTM includes four core agents:

- **RootAgent** – loads customer records from Google Sheets and orchestrates the agent pipeline.
- **AnalystAgent** – generates segmentation logic and onboarding strategies using LLM reasoning and code synthesis.
- **DocumentWriterAgent** – produces onboarding documents tailored to each segment and strategy.
- **PostDemoFollowupAgent** – parses meeting logs and dispatches contextual follow-up via Gmail, Slack, and Calendar.

Each agent runs on a local LLM backend (LLaMA 3.1 via Ollama) and is equipped with a modular toolset (Google Sheets, Docs, Gmail, Calendar, Slack). Agents communicate via a **GlobalMemory** singleton and store turn-level context in **LocalMemory**. The architecture prioritizes transparency, agent autonomy, and external integration.

In addition to the fixed onboarding pipeline, Inflect-GTM includes a **FastAPI-based Agent Builder API**, allowing users to instantiate custom task-specific agents at runtime. Users can specify prompts and tools, and the system leverages LangChain's `create_react_agent` and LangGraph under the hood to construct deterministic reasoning agents.

Our contributions are threefold:

1. A modular LLM agent framework for structured GTM automation.
2. A unified memory-sharing mechanism for coordinated agent interaction.
3. A runtime agent instantiation API for general-purpose task delegation.

Inflect-GTM demonstrates how LLMs can transition from passive assistants to autonomous agents acting across business-critical pipelines. We hope this work can guide further efforts in building controllable, tool-augmented agentic systems.

2 System Architecture and Workflow

2.1 Overview

Inflect-GTM is designed as a modular, extensible multi-agent system where autonomous agents perform specialized roles in automating the Go-to-Market (GTM) process. The architecture supports two primary operating modes:

1. **Pipeline Mode**: A predefined sequence of four cooperating agents autonomously completes the GTM onboarding workflow, from ingesting raw customer data to crafting and dispatching follow-up communications.
2. **Dynamic Agent Mode**: A one-off agent is instantiated at runtime via REST API based on user-defined instructions and selected tools. This mode supports ad-hoc queries and flexible deployments.

Both modes are built atop a shared infrastructure consisting of LLM-powered reasoning, structured memory sharing, and tool-based execution. Each agent executes a perception-reasoning-action loop mediated by an LLM (served locally via Ollama), and can invoke external tools to observe or affect real-world state.

2.2 System Components

Figure 1 depicts the system's key components:

- **Agents**: Agents (e.g., `RootAgent`, `AnalystAgent`) are Python classes inheriting from a common base and initialized with a system prompt, LLM instance, local memory, shared global memory, and tool registry. Each agent's 'run()' method wraps a single round of reasoning and optional tool use.
- **GlobalMemory and LocalMemory**: `GlobalMemory` is a singleton key-value store shared across agents, used to pass structured data such as parsed spreadsheets, segmentation code, and LLM outputs. `LocalMemory` maintains each agent's prompt-response history for context window construction and traceability.

- **External Tools**: Tool interfaces include `GoogleSheetsTool`, `GmailTool`, `GoogleCalendarTool`, and `SlackTool`. These are lightweight wrappers with standardized 'run()' methods, enabling deterministic input-output interaction with real-world services.
- **LLM Backend**: All agent reasoning is conducted using a local instance of LLaMA 3.1 served via Ollama, interfaced using `langchain_ollama`. The system uses low-temperature decoding for deterministic code and text generation.
- **FastAPI Server**: For dynamic agent creation, a FastAPI server is exposed. It provides REST endpoints to create agents, list tools, and interact with instantiated agents via chat-like calls. Internally, this leverages LangChain's `create_react_agent()` and LangGraph to create structured reasoning agents on demand.

2.3 Pipeline Execution Flow

In **pipeline mode**, agents execute sequentially with memory-based coordination:

1. **RootAgent** loads a customer dataset from a Google Sheet and stores the parsed data structure into `GlobalMemory`.
2. **AnalystAgent** uses the LLM to generate Python code, which it executes to partition the customers into segments. It also produces a dictionary mapping each segment to an onboarding strategy.
3. **DocumentWriterAgent** uses the segments and strategies to generate onboarding documents for each group, with content adapted from the strategy.
4. **PostDemoFollowupAgent** loads a transcript or meeting summary, generates contextual follow-up messages, and dispatches them via integrated APIs such as Gmail and Slack.

At each stage, agents write to `GlobalMemory`, creating a modular state that is read by the next agent. This loose coupling through shared memory improves interpretability, testability, and fault isolation.

2.4 Dynamic Agent Workflow

In **dynamic mode**, a user sends a POST request to `/api/create_agent` with:

- A natural-language task description.
- A list of desired tools (e.g., Gmail, Calendar).

The server then:

1. Embeds the task into a structured system prompt.
2. Loads tool implementations from a centralized `tool_registry`.
3. Constructs a ReAct-style agent via `create_react_agent()`.
4. Opens an endpoint at `/api/chat` for ongoing interaction.

This mode supports on-demand tasks like summarizing documents, analyzing calendars, or generating one-off emails—without modifying the pipeline logic.

2.5 Pipeline Architecture and Agent Coordination

Figure 1 illustrates the architecture of Inflect-GTM as a sequential, memory-driven multi-agent pipeline. The system begins with the **RootAgent**, which retrieves and parses raw customer data—typically from a shared Google Sheet—and stores it in a centralized **GlobalMemory** shared across all agents.

The **AnalystAgent** accesses this memory to synthesize Python segmentation logic using an LLM, executes the generated code to cluster customers, and stores both the segmentation function and its output. It also generates a strategy dictionary mapping each segment to a tailored onboarding plan.

The **DocumentWriterAgent** consumes the segments and strategies to generate personalized onboarding documents via LLM prompts, writing the results back to shared memory.

The final agent, **PostDemoFollowupAgent**, parses meeting logs, composes follow-up emails, and executes actions such as sending emails or scheduling meetings through integrated tools like Gmail, Calendar, and Slack.

Each agent interfaces with the external world through appropriate tools—represented as dashed arrows in Figure˜**??**fig:pipeline—while internal data and coordination flow through solid lines via `GlobalMemory`. This architecture enables modular, tool-augmented autonomy while maintaining synchronized, state-aware coordination between agents.

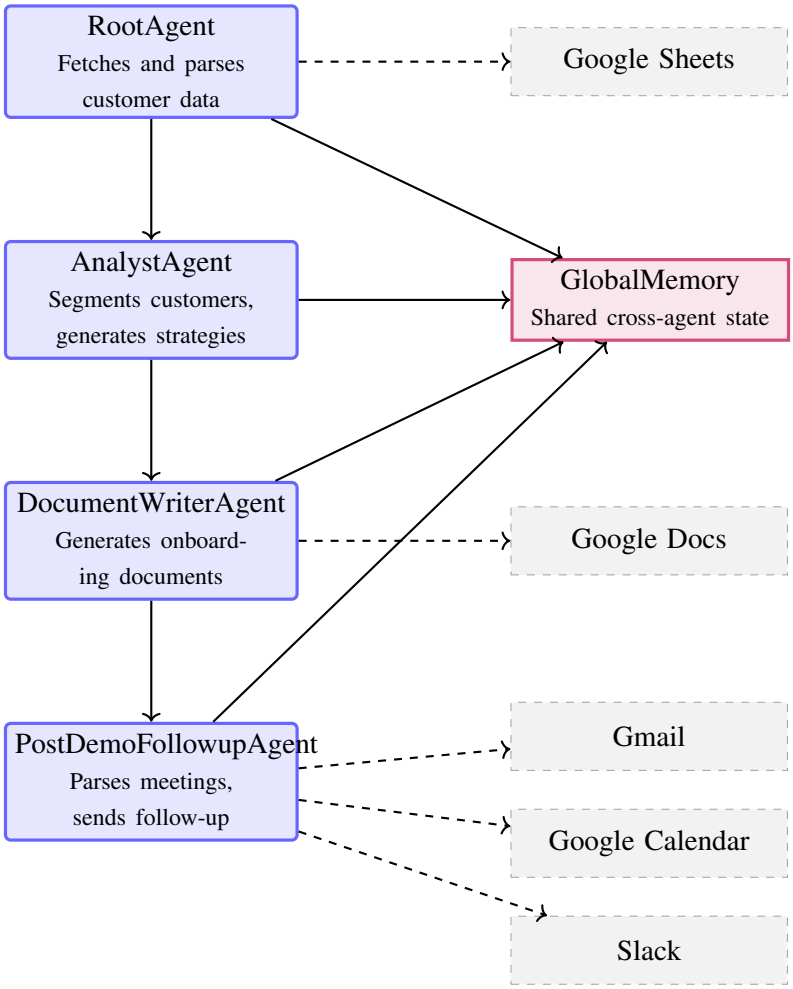

Figure 1: Inflect-GTM Pipeline: Agents, Shared Memory, and External Tool Interactions

3 System Architecture and Workflow

3.1 Overview

Inflect-GTM is designed as a modular, extensible multi-agent system where autonomous agents perform specialized roles in automating the Go-to-Market (GTM) process. The architecture supports two primary operating modes:

1. **Pipeline Mode**: A predefined sequence of four cooperating agents autonomously completes the GTM onboarding workflow, from ingesting raw customer data to crafting and dispatching follow-up communications.
2. **Dynamic Agent Mode**: A one-off agent is instantiated at runtime via REST API based on user-defined instructions and selected tools. This mode supports ad-hoc queries and flexible deployments.

Both modes are built atop a shared infrastructure consisting of LLM-powered reasoning, structured memory sharing, and tool-based execution. Each agent executes a perception-reasoning-action loop mediated by an LLM (served locally via Ollama), and can invoke external tools to observe or affect real-world state.

3.2 System Components

Figure 1 depicts the system's key components:

- **Agents**: Agents (e.g., `RootAgent`, `AnalystAgent`) are Python classes inheriting from a common base and initialized with a system prompt, LLM instance, local memory, shared global memory, and tool registry. Each agent's 'run()' method wraps a single round of reasoning and optional tool use. Notably, the agents are stateful: intermediate results (e.g., segmentation functions, follow-up drafts) are stored and reused across calls.
- **GlobalMemory and LocalMemory**: `GlobalMemory` is a singleton key-value store shared across agents, used to pass structured data such as parsed spreadsheets, segmentation code, and LLM outputs. `LocalMemory` maintains each agent's prompt-response history for context window construction and traceability.
- **External Tools**: Tool interfaces include `GoogleSheetsTool`, `GmailTool`, `GoogleCalendarTool`, and `SlackTool`. These are lightweight wrappers with standardized 'run()' methods, enabling deterministic input-output interaction with real-world services.
- **LLM Backend**: All agent reasoning is conducted using a local instance of LLaMA 3.1 served via Ollama, interfaced using `langchain_ollama`. The system uses low-temperature decoding for deterministic code and text generation.
- **FastAPI Server**: For dynamic agent creation, a FastAPI server is exposed. It provides REST endpoints to create agents, list tools, and interact with instantiated agents via chat-like calls. Internally, this leverages LangChain's `create_react_agent()` and LangGraph to create structured reasoning agents on demand.

3.3 Pipeline Execution Flow

In **pipeline mode**, agents execute sequentially with memory-based coordination:

1. **RootAgent** loads a customer dataset from a Google Sheet and stores the parsed data structure into `GlobalMemory`. It uses `GoogleSheetsTool` with a specific cell range (e.g., `range A1 F100`) to ingest the tabular records and converts them into structured dictionaries.

2. **AnalystAgent** uses the LLM to generate Python code (e.g., a 'segment$_{c}ustomers$'$function), whichitexecutestopa$

3. **PostDemoFollowupAgent** loads a transcript or meeting summary, generates contextual follow-up messages, and dispatches them via integrated APIs such as Gmail and Slack. The agent queries `GoogleCalendarTool` to resolve relevant event context and sends mail through `GmailTool`.

At each stage, agents write to `GlobalMemory`, creating a modular state that is read by the next agent. This loose coupling through shared memory improves interpretability, testability, and fault isolation.

3.4 Dynamic Agent Workflow

In **dynamic mode**, a user sends a POST request to `/api/create_agent` with:

- A natural-language task description.
- A list of desired tools (e.g., Gmail, Calendar).

The server then:

1. Embeds the task into a structured system prompt.
2. Loads tool implementations from a centralized `tool_registry`.
3. Constructs a ReAct-style agent via `create_react_agent()`.
4. Opens an endpoint at `/api/chat` for ongoing interaction.

This mode supports on-demand tasks like summarizing documents, analyzing calendars, or generating one-off emails—without modifying the pipeline logic.

3.5 Pipeline Architecture and Agent Coordination

Figure~**??**fig pipeline illustrates the architecture of Inflect-GTM as a sequential, memory-driven multi-agent pipeline. The system begins with the **RootAgent**, which retrieves and parses raw customer data—typically from a shared Google Sheet—and stores it in a centralized **GlobalMemory** shared across all agents.

The **AnalystAgent** accesses this memory to synthesize Python segmentation logic using an LLM, executes the generated code to cluster customers, and stores both the segmentation function and its output. It also generates a strategy dictionary mapping each segment to a tailored onboarding plan.

The **DocumentWriterAgent** consumes the segments and strategies to generate personalized onboarding documents via LLM prompts, writing the results back to shared memory. If dynamic document creation is required, the agent may invoke `GoogleDocsTool`.

The final agent, **PostDemoFollowupAgent**, parses meeting logs, composes follow-up emails, and executes actions such as sending emails or scheduling meetings through integrated tools like Gmail, Calendar, and Slack.

Each agent interfaces with the external world through appropriate tools—represented as dashed arrows in Figure 1—while internal data and coordination flow through solid lines via `GlobalMemory`. This architecture enables modular, tool-augmented autonomy while maintaining synchronized, state-aware coordination between agents.

4 Discussion and Conclusion

Inflect-GTM demonstrates the feasibility and utility of using modular LLM-based agents to automate complex business processes. By integrating modern orchestration frameworks, tool abstraction layers, and a shared memory design, the system aligns well with emerging paradigms in multi-agent AI and real-world task automation.

4.1 Modularity and Role Assignment

A key design principle in Inflect-GTM is the decomposition of the Go-to-Market (GTM) workflow into independent, role-specific agents. This mirrors patterns observed in prior work such as MetaGPT [1], where agents operate as members of a virtual team, each contributing to a different phase of the pipeline. Such modularity enhances maintainability, testability, and extensibility. For example, new agents such as a `SalesAgent` or `SupportAgent` can be introduced without affecting existing logic, provided they conform to the shared memory interface.

4.2 LLM-Centric Reasoning with Grounded Action

Each agent relies on a local LLM for reasoning, decision-making, and generation. Crucially, agents are not limited to passive text completion. Instead, they can invoke tools to fetch real-world data or perform actions, following the ReAct [2] strategy of interleaving "thought" and "action" steps. This grounding of language in environment interactions improves both reliability and utility. Moreover, the use of a local LLM backend (e.g., LLaMA 3.1 via Ollama) ensures data privacy and cost control, while deterministic generation (temperature = 0) increases reproducibility—an important factor in automation pipelines.

4.3 Memory and State Management

Inflect-GTM employs a hybrid memory architecture:

- **GlobalMemory** enables agents to persist and access shared intermediate results, acting as a scratchpad that supports blackboard-style coordination.
- **LocalMemory** provides agent-specific conversational history, used to construct prompts and retain task context.

While simple, this design is effective in a fixed pipeline setting. In more complex workflows, advanced memory solutions (e.g., vector stores, learned memory modules) could be integrated to support long-term reasoning or retrieval-based augmentation.

4.4 Deployment and Integration Considerations

By exposing all functionality via a FastAPI server, Inflect-GTM supports integration with external interfaces (e.g., web frontends, Slack bots, cron jobs). The dynamic agent creation endpoint (`/api/create_agent`) allows clients to instantiate task-specific agents at runtime, making the system usable as an "agent-as-a-service" backend. This aligns with modern deployment practices in LLM agent systems, where cloud-based orchestration, session tracking, and external monitoring are common.

4.5 Limitations and Future Work

While Inflect-GTM demonstrates the viability of multi-agent automation, it has several limitations. First, it relies heavily on deterministic and correctly structured LLM output such as valid Python functions and well-formed JSON. Although prompt engineering helps mitigate this, parsing errors and execution failures still occasionally occur. Second, the current memory model is static and global, which constrains dynamic workflows and long-horizon planning. More expressive memory structures or learned planners may be required for advanced autonomy. Third, the framework lacks built-in safeguards. In production, moderation layers and error-handling routines are essential to prevent undesirable actions, such as sending incorrect or unauthorized communications.

Looking forward, we aim to extend Inflect-GTM with support for asynchronous and event-driven workflows by incorporating job queues or reactive scheduling. Communication among agents could be made more explicit and decoupled from shared memory via structured messaging protocols. Finally, we envision a user-facing interface—potentially block-based or graphical—for composing custom agent workflows and toolchains, lowering the barrier for domain experts to create bespoke automations.

4.6 Conclusion

Inflect-GTM provides a fully functional and extensible framework for automating Go-to-Market workflows through the use of LLM-powered agents. Its modular architecture integrates both fixed pipelines and dynamic agent instantiation, illustrating how structured orchestration and autonomous behavior can co-exist in a real-world deployment. By bridging high-level language capabilities with pragmatic tool execution and shared memory synchronization, the system exemplifies how generative AI can be harnessed beyond chat interfaces to drive coordinated task automation.

We believe Inflect-GTM is a step toward more capable enterprise-grade agent systems. As LLMs become more controllable and tool ecosystems mature, such frameworks may enable highly customizable, interpretable, and safe AI workflows in domains like sales, support, research, and operations. The codebase and methodology presented here aim to serve as a foundation for continued exploration and refinement in this direction.

References

[1] Sirui Hong et al. Metagpt: Meta programming for a multi-agent collaborative framework. arXiv preprint arXiv:2308.00352, 2023. https://arxiv.org/abs/2308.00352.

[2] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629, 2022. https://arxiv.org/abs/2210.03629.