Divya Multi-Agent Avatar System

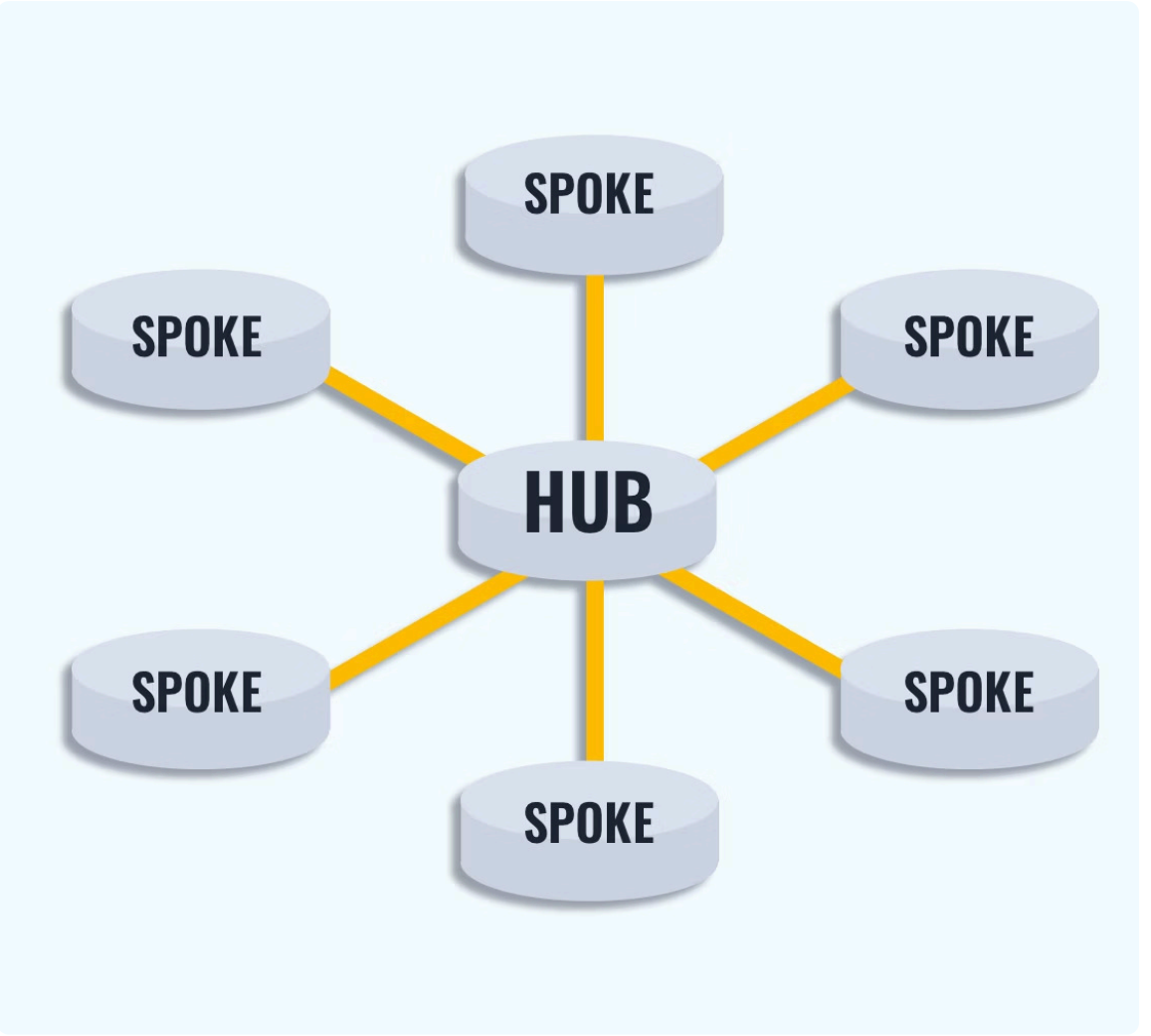
A sophisticated hub-and-spoke multi-agent architecture built on Google ADK, designed to orchestrate intelligent conversations and automated workflows through specialised agent collaboration.

Architecture Overview

Hub-and-Spoke Design

At the core of this system lies the **multi_content_hub_agent**, which serves as the central intelligence coordinator. This hub receives all incoming user queries and programmatic calls, then intelligently routes each request to the most appropriate specialist agent based on the nature of the enquiry.

This architectural pattern provides exceptional modularity and extensibility. Each specialist agent operates independently with its own tools and capabilities, whilst the hub maintains overall system coherence. The design allows for seamless addition of new agents without disrupting existing functionality, making the system highly adaptable to evolving requirements.



The hub-and-spoke model fundamentally differs from monolithic architectures by distributing responsibilities across purpose-built components. Each agent encapsulates specific domain knowledge and tooling, reducing complexity whilst improving maintainability. This separation of concerns ensures that debugging, testing, and enhancement of individual agents can proceed independently, accelerating development cycles and reducing system-wide risk.

Furthermore, the architecture supports parallel processing capabilities, as multiple agents can handle different requests simultaneously. This concurrency improves system throughput and responsiveness, particularly critical when handling time-sensitive operations such as scheduled polls or real-time information retrieval.

Content Specialist Agents

RSS Agent

Monitors external news feeds via the fetch_rss() tool, aggregating updates from sources like TechCrunch and delivering concise summaries of current technology developments.

Search Agent

Executes web lookups using google_search tool for real-time external information that cannot be sourced from internal systems, ensuring up-to-date responses.

Blog Agent

Provides answers from internal blog content through the search_blog() tool, which searches and aggregates relevant entries from the knowledge base.

These content-focused agents form the informational backbone of the system. The RSS Agent continuously monitors designated feeds, applying natural language processing to extract key points and summarise lengthy articles into digestible updates. This capability proves invaluable for staying current with rapidly evolving technology landscapes without overwhelming users with information.

The Search Agent bridges the gap between internal knowledge and the broader web, employing sophisticated query formulation to retrieve relevant external information. It applies filters and relevance scoring to ensure that only high-quality, pertinent results reach users. Meanwhile, the Blog Agent serves as an institutional memory, making historical insights and documented best practices immediately accessible through semantic search capabilities that go beyond simple keyword matching.

Interactive Engagement Agents

1

Poll Agent

Manages daily quiz and polling workflows by reading JSON files from poll_1/ and kpolls_2/ directories.

- get_today_poll1_question()
- get_today_poll1_answer()
- get_today_poll2_question()
- get_today_poll2_answer()

2

Fun Agent

Delivers lightweight entertainment content including programming jokes through the get_joke() tool, adding personality to interactions.

3

Birthday Agent

Generates personalised birthday greetings as an LLM-only agent with no additional tools, invoked by the hub for celebration messages.

These engagement-focused agents transform the system from a mere information provider into an interactive companion. The Poll Agent orchestrates structured knowledge-testing exercises, maintaining separate question and answer repositories that enable sophisticated daily quiz mechanics. By separating questions from answers and managing two distinct poll streams, the system can offer varied difficulty levels and thematic variety to sustain long-term user engagement.

The Fun Agent addresses a crucial aspect often overlooked in technical systems: the human need for levity and connection. By providing contextually appropriate humour, it helps maintain user engagement during extended interaction sessions. The Birthday Agent, whilst simple in implementation, exemplifies the power of LLM-only agents—it requires no tools because the language model's generative capabilities suffice for creating personalised, heartfelt messages that feel authentic rather than templated.

Tool Architecture and Integration

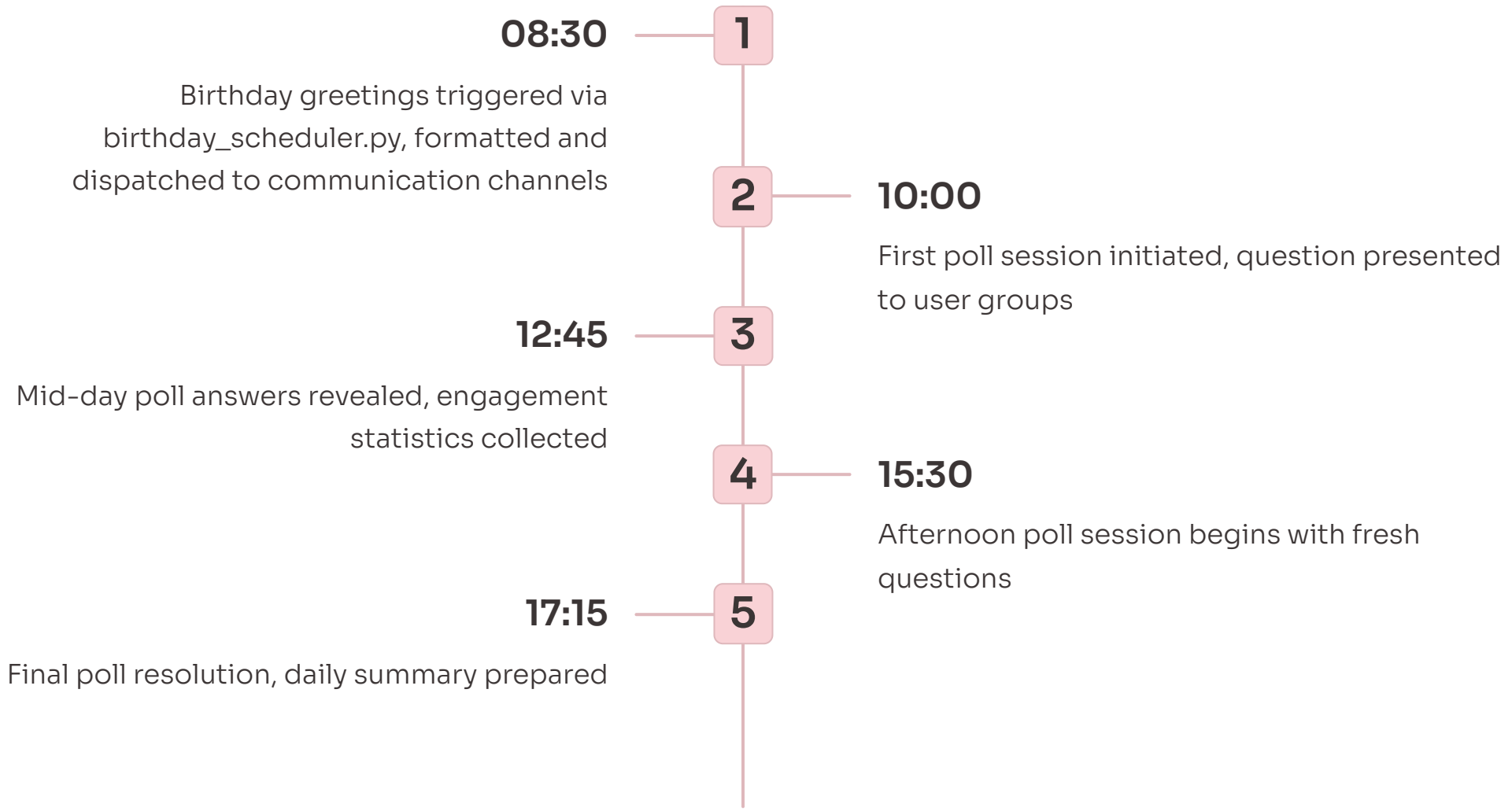
Each specialist agent leverages carefully designed tools that encapsulate specific functionality. These tools represent the boundary between the agent's reasoning capabilities and external system interactions. The tool architecture follows a consistent pattern: each tool accepts well-defined parameters, performs a discrete operation, and returns structured results that the agent can incorporate into its response.

The **fetch_rss()** tool, for instance, handles all the complexities of RSS feed parsing, connection management, and error handling, presenting the agent with clean, normalised article data. Similarly, **search_blog()** abstracts the intricacies of full-text search, relevance ranking, and result aggregation, allowing the Blog Agent to focus on synthesising information rather than managing search mechanics.

This tool-based design provides several architectural advantages. Firstly, tools can be tested, debugged, and optimised independently of agent logic. Secondly, multiple agents can potentially share tools where appropriate, reducing code duplication. Thirdly, tools create clear integration points for external systems—adding a new data source often requires only implementing a new tool rather than modifying agent behaviour.

The polling tools demonstrate another important pattern: reading structured data from the filesystem. By maintaining question and answer files in JSON format within organised directories (**polls_1/** and **polls_2/**), the system gains flexibility in content management. Non-technical team members can update poll content by modifying JSON files without touching agent code, exemplifying the principle of separation between content and logic.

Automated Scheduling Framework



The scheduling framework represents a critical component that transforms reactive agents into proactive system participants. Separate Python scheduler scripts (**poll_scheduler.py** and **birthday_scheduler.py**) operate as independent processes, each running continuous loops using the schedule library to trigger actions at precise times.

These schedulers don't merely invoke agent tools—they orchestrate complete workflows. For instance, poll_scheduler.py retrieves questions, waits for scheduled answer times, then formats responses appropriately for the target communication channel. This workflow management ensures consistent user experiences, as polls arrive predictably and answers follow at expected intervals, building habitual engagement patterns.

The birthday scheduler exemplifies event-driven automation. By checking for upcoming birthdays at a designated morning time (08:30), formatting personalised messages through the Birthday Agent, and dispatching them to appropriate channels, the system creates meaningful touchpoints that strengthen community bonds. This proactive outreach capability distinguishes sophisticated agent systems from simple request-response chatbots.

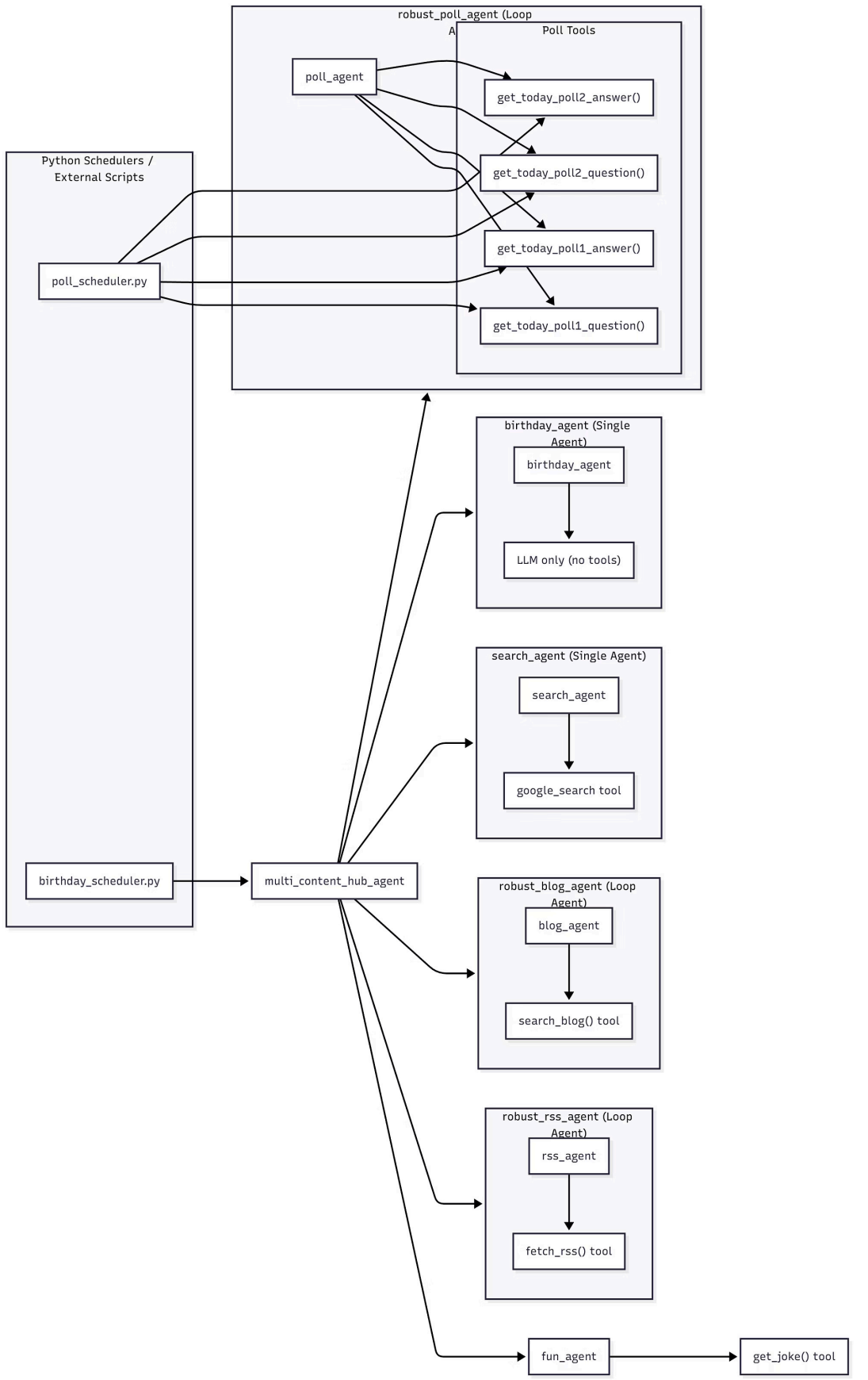
Multi-Channel Communication Integration

Channel Flexibility

The system's schedulers format messages and dispatch them to multiple external channels, including:

- WhatsApp groups
- Email distribution lists
- Chat platforms
- Custom integrations

This channel-agnostic design ensures that content reaches users wherever they prefer to engage, maximising visibility and participation rates.

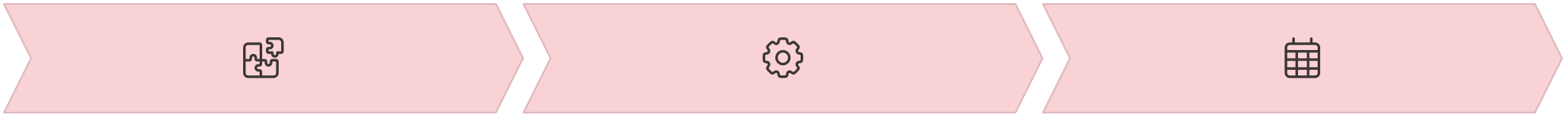


Multi-channel integration addresses a fundamental challenge in modern systems: users fragment their attention across numerous platforms, and any single-channel solution inevitably misses significant portions of the intended audience. By abstracting the message formatting and dispatch logic from the core agent capabilities, the architecture allows each scheduler to adapt content for different platforms' requirements—character limits, formatting conventions, media support—without duplicating agent intelligence.

The schedulers implement a strategic buffering layer between agent-generated content and channel-specific APIs. When a poll question emerges from the Poll Agent, the scheduler can simultaneously format it as a WhatsApp message with emoji enhancements, a clean email with HTML formatting, and a Slack message with interactive buttons. This parallel dispatch ensures synchronised delivery across channels, maintaining conversation coherence for users who participate across multiple platforms.

Furthermore, the multi-channel approach provides resilience. Should one communication platform experience downtime or API issues, the system continues operating through alternative channels. This redundancy proves particularly valuable for time-sensitive content like birthday greetings or daily polls, where delivery timing significantly impacts effectiveness.

System Extensibility and Modularity



Add New Agents

Introduce specialist agents for new domains without modifying existing components—simply register with the hub

Expand Tool Library

Create new tools and assign them to appropriate agents, extending capabilities incrementally

Deploy Schedulers

Implement additional scheduler scripts for new automated workflows, operating independently

The system's modular architecture fundamentally enables sustainable growth. Adding a new specialist agent—perhaps a **Calendar Agent** for meeting management or a **Translation Agent** for multilingual support—requires only implementing the agent's specific logic and tools, then registering it with the hub's routing logic. Existing agents continue operating without modification, and the hub's decision-making process naturally extends to accommodate the new capability.

This extensibility extends beyond agents to tools and schedulers as well. If the RSS Agent needs to support a new feed format, developers can add a new tool without touching the agent's reasoning logic. If business requirements demand a new automated workflow—perhaps weekly summary reports—a new scheduler script can be developed and deployed independently, following the established pattern of the poll and birthday schedulers.

The architecture also supports experimentation and gradual rollout. New agents can be deployed in a shadow mode where they receive requests and generate responses but don't yet influence production outputs. This allows thorough testing with real usage patterns before full activation. Similarly, new tools can be implemented alongside existing ones, enabling A/B testing of different approaches to the same functionality.

Perhaps most importantly, the modular design distributes cognitive load across development teams. Different engineers can own different agents, becoming domain experts in their respective areas whilst maintaining a consistent integration pattern. This specialisation improves code quality, accelerates feature development, and reduces the risk of cascading failures from changes in one area affecting unrelated functionality.

Technical Implementation Highlights

Google ADK Foundation

Built on Google's Agent Development Kit, providing robust agent orchestration, natural language understanding, and seamless integration with Google's AI ecosystem

Python Implementation

Leverages Python's extensive libraries, asynchronous capabilities, and clean syntax for maintainable agent and scheduler code

JSON Configuration

Stores poll questions, answers, and configuration in structured JSON files, enabling non-technical content updates

Schedule Library

Employs Python's schedule library for reliable, time-based task execution across all automated workflows

The choice of Google ADK as the foundational framework provides several strategic advantages. ADK offers pre-built components for common agent patterns, reducing boilerplate code and allowing developers to focus on business logic rather than infrastructure. Its integration with Google's broader AI ecosystem enables seamless access to advanced language models, cloud services, and future platform enhancements without requiring architectural changes.

Python serves as an ideal implementation language for this system. Its extensive library ecosystem provides ready-made solutions for RSS parsing, web scraping, scheduling, and API integration. The language's dynamic typing and expressive syntax accelerate development, whilst its widespread adoption in the AI community ensures access to cutting-edge tools and frameworks as they emerge.

The JSON-based content management approach reflects pragmatic system design. By storing poll questions and answers in JSON files within organised directories, the system achieves a clean separation between code and content. This enables content creators to update daily polls without developer intervention, reducing operational overhead and enabling rapid content iteration based on user feedback.

System Benefits and Future Potential

Current Capabilities

- Intelligent query routing through centralised hub
- Real-time news aggregation and summarisation
- Automated daily polls with scheduled reveals
- Personalised birthday greeting generation
- Internal knowledge base search and retrieval
- External web search integration
- Multi-channel message distribution
- Entertainment content delivery

Future Extensions

- Meeting scheduling and calendar agents
- Document generation and formatting agents
- Code review and analysis agents
- Multilingual translation agents
- Data visualisation and reporting agents
- Sentiment analysis and feedback agents
- Integration with additional communication platforms
- Advanced analytics and usage insights

This multi-agent system delivers immediate practical value whilst establishing a foundation for unlimited future expansion. The current implementation demonstrates the viability of the hub-and-spoke pattern across diverse use cases—from real-time information retrieval to scheduled automated workflows to personalised content generation. Users benefit from a single, consistent interface that intelligently routes their needs to the most appropriate specialist, creating a seamless experience that belies the system's underlying complexity.

Looking forward, the architecture's extensibility positions the system to evolve alongside organisational needs. As new challenges emerge or opportunities arise, appropriate agents can be developed and integrated without disrupting existing functionality. This evolutionary capacity transforms the system from a static tool into a living platform that grows more capable over time, continuously adapting to serve its users more effectively.

Key Takeaway: The Divya Multi-Agent Avatar exemplifies modern agent-based system design—modular, extensible, and purpose-built for both immediate utility and long-term evolution. By orchestrating specialist agents through a central hub and enabling proactive automation via independent schedulers, the system delivers sophisticated conversational AI capabilities whilst maintaining architectural clarity and operational simplicity.