# **ScintiAI: Automating SAS to PySpark Migration with a Multi-Agent AI System**

## **Abstract**

ScintiAI is a **next-generation AI-driven platform** designed to automate the conversion of legacy SAS code into scalable, modern PySpark pipelines. Traditional SAS systems are expensive, cloud-incompatible, and rapidly becoming outdated. ScintiAI addresses these challenges through a structured, multi-agent pipeline. Each agent—starting with a Parse Agent—is responsible for a critical stage of transformation, ensuring the entire process is modular, explainable, and production-ready. This white paper presents a comprehensive overview of ScintiAI’s architecture, agent responsibilities, technologies used, and its practical applications in real-world data modernization efforts, demonstrating how it delivers significant cost savings, accelerates cloud adoption, and enhances data agility for enterprises.

## **1. Introduction**

In today's rapidly evolving data landscape, organizations are increasingly seeking to modernize their data infrastructure to leverage cloud computing, enhance scalability, and reduce operational costs. A significant hurdle in this modernization journey is the prevalence of **legacy SAS codebases**. While SAS has historically been a powerful tool for data analysis and reporting, its proprietary nature, high licensing costs, and limited compatibility with modern cloud-native platforms present substantial challenges. Manually migrating these extensive codebases to open-source, cloud-friendly alternatives like PySpark is a labor-intensive, error-prone, and time-consuming endeavor, often spanning months or even years for large enterprises.

The strategic imperative to move away from monolithic, on-premise systems towards agile, scalable cloud-based solutions is undeniable. However, the sheer volume and complexity of existing SAS scripts, which often encapsulate decades of accumulated business logic, represent a formidable technical debt. This debt not only hinders innovation but also locks organizations into outdated technological paradigms. The critical need for an automated, intelligent, and reliable solution to bridge this gap between legacy SAS and modern PySpark environments has become paramount.

ScintiAI emerges as a **groundbreaking solution** to this critical problem. By harnessing the power of multi-agent AI systems and large language models (LLMs), ScintiAI automates the complex process of transforming legacy SAS code into production-ready PySpark pipelines. This automation not only accelerates the migration process but also significantly reduces the risk of human error and ensures the preservation of critical business logic. ScintiAI's innovative approach provides a structured, explainable, and auditable migration path, enabling organizations to unlock the full potential of their data in a cloud-native ecosystem without the traditional pain points associated with re-platforming.

## **2. Problem Statement: The Challenges of Legacy SAS Codebases**

The continued reliance on legacy SAS codebases poses several significant obstacles to enterprise-wide data modernization initiatives, creating a bottleneck for innovation and increasing operational overheads. Understanding these challenges is crucial to appreciating the value proposition of an automated solution like ScintiAI.

* **High Licensing Cost:** SAS is a proprietary software with substantial licensing fees, contributing to high operational expenditures for organizations. These costs are often tiered based on processing capacity, data volume, and the number of users, meaning that as an organization's data footprint and analytical needs grow, these costs can escalate dramatically and unpredictably. This creates a recurring financial burden that detracts from investments in other strategic areas like cloud infrastructure or advanced analytics tools. Furthermore, licensing models can be rigid, making it difficult for organizations to scale up or down dynamically based on demand, which is a key advantage of cloud computing.
* **Cloud Lockout:** SAS lacks native and seamless integration with leading cloud platforms such as Amazon Web Services (AWS), Microsoft Azure, or Databricks. While some workarounds exist, they are often cumbersome, inefficient, and fail to leverage the true elasticity and cost-effectiveness of cloud-native data processing environments.
* **Talent Shortage:** The demographic trend indicates a diminishing pool of skilled SAS developers. As older generations of highly experienced SAS programmers retire, finding and retaining talent proficient in legacy SAS becomes increasingly difficult. Universities and training programs are shifting their focus towards open-source technologies like Python, R, and Spark, leading to a decreasing supply of new SAS talent. This scarcity drives up recruitment costs, prolongs project timelines, and creates a significant knowledge transfer challenge for organizations relying heavily on SAS. The long-term sustainability of SAS-based operations is therefore at risk.
* **Manual Migration Risk:** Rewriting extensive SAS codebases to PySpark manually is inherently labor-intensive, highly susceptible to human error, and exceptionally challenging to validate comprehensively. A single complex SAS program can span thousands of lines, incorporating intricate data transformations, complex business rules, and statistical procedures. Manual conversion of such code introduces numerous opportunities for logical inconsistencies, syntax errors, and subtle deviations from the original business logic, which can lead to erroneous analytical results or operational failures. This approach often results in lengthy project timelines, significant budget overruns, and the potential introduction of bugs or logical inconsistencies that are costly and time-consuming to identify and rectify.

To overcome these multifaceted challenges, a robust, intelligent, and automated framework is urgently needed to handle this transformation at scale while preserving intricate business logic and ensuring data integrity. ScintiAI provides precisely such a framework, transforming a daunting task into a streamlined, efficient, and reliable process.

## **3. Solution Overview: ScintiAI Architecture**

ScintiAI utilizes a sophisticated **multi-agent architecture** that intelligently automates each phase of SAS to PySpark migration. This modular approach enhances explainability, debuggability, and scalability, providing a transparent and efficient conversion pipeline. The design philosophy behind ScintiAI prioritizes breaking down the complex migration problem into manageable sub-problems, each handled by a specialized agent, thereby mirroring the principles of a well-structured software development lifecycle.

### **3.1. Core Agent Pipeline**

The ScintiAI core comprises a structured pipeline of specialized AI agents, each responsible for a distinct stage of the transformation process. This sequential yet iterative design allows for a clear division of labor and facilitates robust error handling and self-correction.

#### **3.1.1. Parse Agent (Entry Point)**

The Parse Agent serves as the foundational component and the entry point for raw SAS code. Its primary responsibilities are analogous to the lexical analysis and parsing phases of a traditional compiler, preparing the raw, unstructured SAS code into a structured, machine-readable format.

* **Tokenizing and Structuring SAS Code:** The agent meticulously analyzes the input SAS code, breaking it down into discrete **tokens** (e.g., keywords, identifiers, operators, literals) and identifying logical blocks such as DATA steps, PROC statements (e.g., PROC SQL, PROC SORT, PROC MEANS), LIBNAME assignments, MACRO definitions, and embedded SQL segments. This process is analogous to the "compiler front-end" that prepares raw code for intelligent processing. It understands the grammatical structure of SAS, identifying the start and end of blocks, statements, and expressions. For example, it differentiates between a DATA step defining a new dataset and a PROC SQL block querying existing ones, treating them as distinct, manageable units for subsequent processing. This initial structuring is crucial because LLMs perform significantly better when provided with well-defined, contextualized chunks of code rather than an undifferentiated stream of characters.
* **Comment Handling:** It intelligently separates comments from executable code. SAS code often contains extensive comments that explain business logic, data definitions, or historical context. These comments are not merely discarded; they are either preserved for re-injection into the generated PySpark code (as documentation, ensuring the lineage of business understanding) or stored separately for auditability and context. This feature is vital for maintaining the "why" behind the code, which is often as important as the "what."
* **Creating Metadata:** The Parse Agent generates a structured, **intermediate representation (IR)** of the SAS code. This representation is not just a collection of tokens but a rich data structure that includes markers and metadata describing the relationships between code segments, variable definitions, data flow, and control flow. For instance, it might identify input datasets, output datasets, variables created or modified, joins performed, and conditional logic.

#### **3.1.2. LLM Rule Agent**

Following parsing, the LLM Rule Agent takes over the crucial role of converting the structured SAS blocks into equivalent PySpark code. This agent leverages the formidable power of Large Language Models (LLMs) such as OpenAI GPT (via Azure) or Google Gemini (via LangChain), operating as highly sophisticated translation engines.

* **Prompted Conversion:** The LLM Rule Agent employs **finely tuned system prompts** to guide the LLM in understanding the semantic intent of the SAS code and translating it into idiomatic PySpark syntax while preserving the original business logic. These prompts are meticulously crafted to provide context, specify desired output format, and instruct the LLM on best practices for PySpark code generation. For example, a prompt for a SAS DATA step might include instructions to translate SET, MERGE, BY, IF-THEN-ELSE logic into PySpark operations like read, join, filter, withColumn, or when/otherwise clauses. This ensures accurate transformation of data manipulations, conditional statements, loops, and statistical procedures, leveraging the LLM's vast knowledge base of programming paradigms and data manipulation techniques. The LLM acts as an expert programmer capable of cross-language translation with semantic understanding.
* **Fallback Strategy:** For highly complex, ambiguous, or rare SAS blocks that the LLM might struggle to convert perfectly or confidently, ScintiAI implements a robust **fallback strategy**. In such cases, rather than failing the entire migration or generating incorrect PySpark, the original SAS code block is wrapped within a spark.sql('''original\_sas\_block''') construct. This approach is a critical safeguard. It allows the Spark environment to potentially execute the original SAS logic as a pass-through SQL query (if compatible) or, more importantly, flags this specific section for potential human review or further refinement by a domain expert.

#### **3.1.3. Validation Agent**

Once the PySpark code is generated by the LLM Rule Agent, the Validation Agent steps in to ensure its quality and correctness. This agent performs automated, programmatic checks to guarantee that the generated code meets critical criteria, significantly reducing the need for time-consuming and error-prone manual code reviews.

* **Completeness:** It verifies that the PySpark code includes all necessary Spark operations (e.g., .select(), .filter(), .groupBy(), .join(), .withColumn(), .orderBy()) corresponding to the transformations defined in the original SAS code. This involves comparing the logical operations identified in the Parse Agent's IR with the operations present in the generated PySpark. For example, if the SAS code performed a LEFT JOIN, the Validation Agent checks for a corresponding .join(..., how='left') in the PySpark.
* **Structural Integrity:** The agent checks for common syntax errors, missing parentheses, unclosed statements, incorrect function calls, or other structural flaws that would prevent the PySpark code from executing successfully within a Spark environment. This is akin to a static code analysis tool that ensures the generated code adheres to PySpark's grammatical rules. This check can involve using PySpark's own parsing capabilities or static analysis libraries.
* **Semantic Checks:** Basic logical validations are performed to ensure that key operations, such as data filtering conditions, aggregation logic (e.g., SUM, AVG), and variable transformations, are semantically equivalent to their SAS counterparts. While not a full data validation (which would involve running both codes and comparing outputs, a separate and often manual step), it confirms the presence of expected transformations. For instance, if SAS code WHERE AGE > 18 was present, the Validation Agent would confirm a filter(col('AGE') > 18) or similar expression exists in the PySpark. This ensures that the intent of the SAS code has been carried over in a logically sound manner.

#### **3.1.4. Feedback Agent**

The Feedback Agent introduces crucial retry logic and error-aware enhancement mechanisms, enabling ScintiAI to self-correct and continuously improve its output. This agent is a cornerstone of ScintiAI's robustness, transforming potential failures into learning opportunities.

* **Logs Failures:** It meticulously logs any errors or issues detected by the Validation Agent, providing a detailed record of areas requiring refinement. These logs capture the specific SAS code block, the problematic PySpark output, the type of validation error, and any relevant traceback information. This comprehensive logging is essential for both system self-correction and for providing transparent insights to human operators if manual intervention becomes necessary.
* **Retriggers LLM Agent:** Based on the logged failures, the Feedback Agent intelligently re-triggers the LLM Rule Agent. Crucially, it provides **updated context or modifies the original prompts**, guiding the LLM to generate a more accurate and compliant PySpark code segment. For example, if a validation error indicated a missing schema definition for a DataFrame, the Feedback Agent might re-prompt the LLM with an explicit instruction to infer or define the schema. If a join condition was incorrect, the prompt might include the specific column names and join type that were expected. This iterative refinement process allows the system to learn from its mistakes and improve the quality of its output without human intervention for common errors.

### **3.2. System Architecture**

The robust architecture of ScintiAI is built upon a foundation of modern technologies, ensuring scalability, maintainability, and user-friendliness. The layered approach separates concerns and facilitates independent development and deployment of different components.

| Layer | Technology |
| --- | --- |
| **Frontend** | React.js, Material UI, Toastify, Framer Motion |
| **Backend** | FastAPI, Python, LangChain |
| **Agents** | Modular Python Classes with Defined Triggers |
| **LLMs** | OpenAI GPT (via Azure), Google Gemini (via LangChain) |
| **Execution** | Uvicorn, pip, npm, venv |

**Key Features and Benefits of the ScintiAI Architecture:**

* **Parse + LLM Combo:** This combination is fundamental. It enables **structured migration**, moving beyond raw text conversion to semantic understanding. The Parse Agent ensures that the LLM is not just guessing at translations but working with a semantically rich representation of the SAS code, leading to more accurate, reliable, and contextually appropriate PySpark output. This avoids the pitfalls of simple rule-based or regex-based conversions which lack true understanding.
* **Modular Agents:** The clear separation of responsibilities among Parse, LLM Rule, Validation, and Feedback Agents facilitates **easy debugging, extension, and scaling** of the migration process. If a new SAS PROC statement needs to be supported, the Parse Agent can be updated. If the LLM's translation quality needs improvement for a specific pattern, the LLM Rule Agent's prompts can be refined. Debugging an issue becomes a process of isolating the problematic agent, rather than sifting through a monolithic codebase. New agents can also be introduced for future enhancements, such as optimization or performance tuning.
* **Automated Validation:** The Validation Agent significantly reduces the need for **time-consuming and error-prone manual code review**. By programmatically checking for completeness, structural integrity, and basic semantic correctness, it catches common mistakes early in the process, freeing up human experts to focus on more complex business logic validation and edge cases. This dramatically accelerates the overall migration timeline.
* **Retry and Feedback Loops:** This intelligent mechanism allows the system to **auto-correct errors and continuously improve output** using validation logs. It creates a self-healing system that learns from its failures. This iterative refinement minimizes human intervention for common translation issues and enhances the overall robustness and accuracy of the generated PySpark code over time. It's a continuous improvement cycle embedded directly into the migration process.

## **4. Benefits of ScintiAI**

ScintiAI offers a compelling set of benefits that significantly enhance the efficiency and effectiveness of data modernization efforts, providing a clear return on investment for organizations grappling with legacy SAS systems.

* **Cost Reduction:** By automating the conversion of SAS to PySpark, ScintiAI dramatically reduces reliance on expensive SAS licenses. Organizations can gradually decommission their SAS environments, leading to significant savings in recurring software fees. Furthermore, the reduction in manual effort translates directly into lower labor costs, as fewer specialized SAS developers and data engineers are required for the migration process. The shift to open-source PySpark running on cloud infrastructure also offers more flexible and often lower compute costs compared to proprietary SAS hardware and software.
* **Accelerated Cloud Adoption:** ScintiAI is a direct enabler for cloud migration initiatives. By transforming legacy SAS ETL and analytical processes into cloud-native PySpark, it removes a major impediment to moving data workloads to platforms like AWS, Azure, or Databricks. This acceleration allows organizations to quickly leverage the scalability, elasticity, and cost-efficiency of cloud computing, realizing the benefits of cloud infrastructure much faster than with manual migration efforts.
* **Enhanced Scalability and Performance:** PySpark, being built on Apache Spark, offers unparalleled scalability for big data processing, far surpassing the limitations of traditional SAS. ScintiAI enables organizations to process larger volumes of data with greater speed and efficiency, supporting modern data initiatives like real-time analytics, large-scale data lakes, and complex machine learning pipelines. This performance boost can unlock new analytical capabilities and business insights.
* **Improved Agility and Innovation:** By moving to an open-source, cloud-compatible platform, organizations gain greater flexibility and agility. PySpark integrates seamlessly with a vast ecosystem of modern data tools, machine learning frameworks, and visualization platforms. This allows data teams to innovate faster, experiment with new technologies, and deploy data products more rapidly, without being constrained by proprietary vendor lock-in or outdated tools.
* **Reduced Risk and Error:** Automated migration significantly minimizes the risk of human error inherent in manual code conversion. The structured multi-agent approach, coupled with automated validation and feedback loops, ensures higher accuracy and consistency in the generated PySpark code. This leads to more reliable data pipelines and reduces the costly process of debugging and rectifying errors that can arise from manual rewrites.
* **Preservation of Business Logic:** ScintiAI's intelligent parsing and LLM-driven translation are designed to preserve the intricate business logic embedded within legacy SAS code. The focus is on semantic understanding and equivalent transformation, not just syntactic replacement. This ensures that years of accumulated business rules and analytical methodologies are accurately carried over to the new PySpark environment, maintaining continuity and avoiding costly re-engineering of core business processes.

## **5. Use Cases**

ScintiAI's versatility makes it applicable across various critical data modernization scenarios, providing tailored solutions for diverse organizational needs.

* **Enterprise SAS Decommissioning:** For large organizations that have accumulated decades of SAS code, a complete **decommissioning of legacy SAS infrastructure** can be a daunting, multi-year project. ScintiAI enables a systematic and safe approach to this transition. Instead of a "rip and replace" strategy, ScintiAI facilitates a phased migration, converting critical SAS applications and data pipelines piece by piece. This allows enterprises to gradually reduce their reliance on expensive SAS licenses, minimize operational disruptions, and ultimately achieve a complete transition to more cost-effective and agile PySpark environments. It handles the sheer volume of code by providing a scalable automation engine.
* **Cloud Migration Initiatives:** ScintiAI significantly accelerates the migration of **on-premise data processing workloads to cloud platforms** like AWS, Azure, or Databricks. Many organizations are held back from full cloud adoption by their on-premise SAS footprint. ScintiAI directly addresses this by automating the conversion of complex SAS ETL processes, data transformations, and reporting logic into cloud-native PySpark code. This allows data to be processed natively within cloud environments, leveraging services like AWS S3/Glue/EMR, Azure Data Lake/Synapse/Databricks, leading to improved performance, scalability, and reduced infrastructure management overhead compared to trying to run SAS in the cloud.
* **Accelerated AI/ML Readiness:** The journey to advanced Artificial Intelligence and Machine Learning often requires data in a format compatible with modern ML frameworks. Many enterprises have their historical data preparation and feature engineering pipelines locked within SAS. ScintiAI transforms these **SAS-based data preparation and feature engineering pipelines into PySpark**, making data readily available and compatible with modern AI/ML frameworks and tools like TensorFlow, PyTorch, or scikit-learn. This allows data scientists to work directly with cleaned, transformed data in a scalable PySpark environment, greatly accelerating model development, training, and deployment. It bridges the gap between traditional data analytics and cutting-edge machine learning.
* **COE Automation Tools for Consultants:** For consulting firms and Centers of Excellence (COEs) specializing in data modernization, ScintiAI provides a powerful **automation tool to rapidly assess and migrate client SAS codebases**. Instead of relying on manual code review and rewriting, consultants can leverage ScintiAI to quickly analyze a client's SAS footprint, generate initial PySpark conversions, and focus their efforts on validating complex business logic and optimizing the migrated pipelines. This enhances their service offerings, improves project turnaround times, reduces delivery costs, and allows them to take on more migration projects efficiently. It serves as a force multiplier for data modernization consultants.

## **6. Setup Instructions**

Setting up ScintiAI involves configuring both the backend and frontend components. These instructions provide a clear, step-by-step guide to get the system up and running in a development environment.

### **6.1. Backend Setup**

The backend handles the core logic of the multi-agent system, including parsing, LLM interaction, validation, and feedback.

1. Create a Virtual Environment:  
   It is highly recommended to use a virtual environment to manage dependencies and avoid conflicts with other Python projects.  
   Bash  
   python -m venv venv
2. Activate the Virtual Environment:  
   Before installing dependencies or running the backend, activate the virtual environment.
   * **Windows:**  
     Bash  
     venv\Scripts\activate
   * **macOS/Linux:**  
     Bash  
     source venv/bin/activate
3. Install Dependencies:  
   Navigate into your backend directory (if not already there) and install all required Python packages using pip and the requirements.txt file. This file lists all the libraries necessary for the backend to function correctly (e.g., FastAPI, LangChain, OpenAI/Google API clients).  
   Bash  
   cd backend # If you are not already in the backend directory  
   pip install -r requirements.txt  
     
   *Note: Ensure your requirements.txt file is properly populated with all necessary dependencies.*
4. Configure LLM API Keys:  
   Before running the backend, you will need to set environment variables for your chosen LLM provider (OpenAI or Google Gemini). For example:
   * For OpenAI: export OPENAI\_API\_KEY="your\_openai\_api\_key"
   * For Google Gemini: export GOOGLE\_API\_KEY="your\_google\_api\_key"
   * If using Azure OpenAI, additional environment variables related to your Azure endpoint and deployment name will be required as per LangChain documentation.  
     It's best practice to use a .env file and a library like python-dotenv for managing these credentials securely, or to configure them directly in your deployment environment.
5. Run the Backend Server:  
   Once dependencies are installed and API keys are set, you can start the backend server using Uvicorn, an ASGI server implementation for Python. The --reload flag is useful for development, as it automatically restarts the server when code changes are detected.  
   Bash  
   uvicorn main:app --reload  
     
   The backend will typically run on http://127.0.0.1:8000 by default.

### **6.2. Frontend Setup**

The frontend provides the user interface for interacting with ScintiAI, allowing users to input SAS code and view the generated PySpark output.

1. Navigate to the Frontend Directory:  
   Change your current directory to the frontend folder where the React.js application resides.  
   Bash  
   cd frontend
2. Install Node Modules:  
   Install all necessary Node.js packages and dependencies for the React application using npm.  
   Bash  
   npm install
3. Install Material UI and Other Dependencies:  
   Ensure that specific UI libraries and other utilities are installed. These provide the visual components and interactive elements for the user interface.  
   Bash  
   npm install @mui/material @emotion/react @emotion/styled framer-motion react-toastify @mui/icons-material  
     
   *Note: It's good practice to have all frontend dependencies listed in the package.json file, so a single npm install typically suffices if the package.json is comprehensive.*
4. Start the Frontend Development Server:  
   Launch the React development server. This will compile the React application and serve it, usually with hot-reloading enabled for development convenience.  
   Bash  
   npm start
5. Access the Application:  
   Once both the backend and frontend servers are running, open your web browser and navigate to the local address where the frontend is hosted.  
   http://localhost:3000  
     
   You should now see the ScintiAI user interface, ready for interaction.

## **7. Conclusion**

ScintiAI represents a paradigm shift in legacy code migration. By leveraging a structured **multi-agent AI system**, it redefines the process of transforming legacy SAS code into modern, scalable PySpark pipelines. Its foundational Parse Agent ensures intelligent code understanding, meticulously breaking down complex SAS into a structured intermediate representation. This structured approach is critical, moving beyond superficial text translation to a deep, semantic comprehension of the original business logic.

The power of the LLM Rule Agent then comes to the forefront, providing accurate and context-aware conversion by leveraging advanced Large Language Models and finely tuned prompts. This is complemented by the integrated Validation and Feedback Agents, which form a robust self-correction mechanism. These agents collectively guarantee the quality and correctness of the generated PySpark code, minimizing manual intervention, maximizing efficiency, and continuously improving the system's performance.

Whether an organization aims for a complete SAS shutdown, a strategic piece-by-piece modernization, or rapid cloud adoption, ScintiAI is engineered to make the transformation seamless, safe, and highly scalable. It addresses the critical challenges of high licensing costs, cloud lockout, talent shortages, and manual migration risks, offering a clear path to a modernized data ecosystem. ScintiAI empowers enterprises to unlock the full potential of their data in the cloud-native era, driving innovation, reducing technical debt, and positioning them for sustainable growth in the ever-evolving data landscape. Its modular, explainable, and production-ready design makes it an indispensable tool for any organization embarking on its data modernization journey.