LTIMindtree

WHITEPAPER

Rethinking SAS Modernization

An AI-Driven Blueprint for Modernizing Legacy SAS Workloads

A structured, multi-agent framework that automates SAS-to-PySpark transformation, preserving business logic, reducing migration risks, and accelerating cloud adoption while cutting operational costs.

Author

Ramesh Vanteru

Principal & Head of SAS COE, LTIMindtree

Table of contents

Executive Summary

This whitepaper is designed for CIOs, CTOs, data modernization leaders, and analytics heads grappling with the challenges of moving from legacy SAS systems to modern, cloud-native architectures. It offers a clear, actionable blueprint for organizations seeking to accelerate modernization without losing critical business logic or high operational risks.

Modern enterprises face mounting pressure to modernize data ecosystems and adopt cloud-native technologies. Yet, legacy SAS systems, long the backbone of data analytics, often stand in the way. These systems are costly, cloud-incompatible, and operationally rigid, locking critical business logic into proprietary frameworks. If you're still relying on SAS, you must recognize the operational risks, rising costs, and competitive disadvantages this creates. Manual migration to open-source alternatives like PySpark is possible, but it's slow, error-prone, and unsustainable at enterprise scale.

This whitepaper explores an AI-driven, multi-agent approach's architectural and methodological foundations to automate SAS-to-PySpark transformation. You'll see why traditional approaches fall short, how a modular migration framework addresses these gaps, and the practical benefits: cost reduction, risk mitigation, and faster time-to-cloud.

Real-world use cases illustrate how this approach aligns with enterprise modernization strategies while preserving decades of embedded business knowledge. Finally, we introduce Scintilla.AI, a next-generation platform built on these principles, and explore its role in helping organizations bridge the gap between legacy SAS systems and modern data architectures.

Why Does It Matter to Modernize Legacy SAS Workloads

Modernizing legacy data systems is more urgent than ever in an era of real-time analytics, AI-driven insights, and scalable cloud architectures.

Over decades, organizations across sectors, from financial services to healthcare, have invested heavily in SAS for data processing, reporting, and advanced analytics.

While SAS has historically offered reliability and statistical depth, it now represents an increasingly costly and inflexible foundation for modern data strategies.

Several structural challenges underline the urgency:

1. **Cloud-native architectures** thrive on elasticity, distributed processing, and integration with open-source ecosystems. Traditional SAS workloads are primarily built for on-premises environments and are only compatible with cloud-native pipelines

2. **Cost pressures** are intensifying, as proprietary licensing models scale poorly with growing data volumes and user demands

3. **Talent pipelines** are shifting away from proprietary languages toward Python, Spark, and related tools, creating operational risk

4. **Competitive pressure** requires data teams to iterate quickly, experiment, and deploy machine learning models, tasks better suited to open, modular platforms

Yet the transition from SAS to PySpark or other modern frameworks is not trivial. Many enterprises have accumulated thousands or millions of lines of SAS code, deeply intertwined with business processes and domain-specific rules. Rewriting this code manually can take years, drain budgets, and introduce substantial risk.

The question is not whether to modernize but how. This paper also argues that the answer lies in a structured, AI-driven approach that treats migration as a modular, explainable process that combines semantic parsing, LLM-based translation, automated validation, and iterative improvement.

The Challenges of Legacy SAS Codebases

Before modernizing, you must understand why legacy SAS environments are so complex to replace. These challenges create a bottleneck for innovation and increase operational overheads.

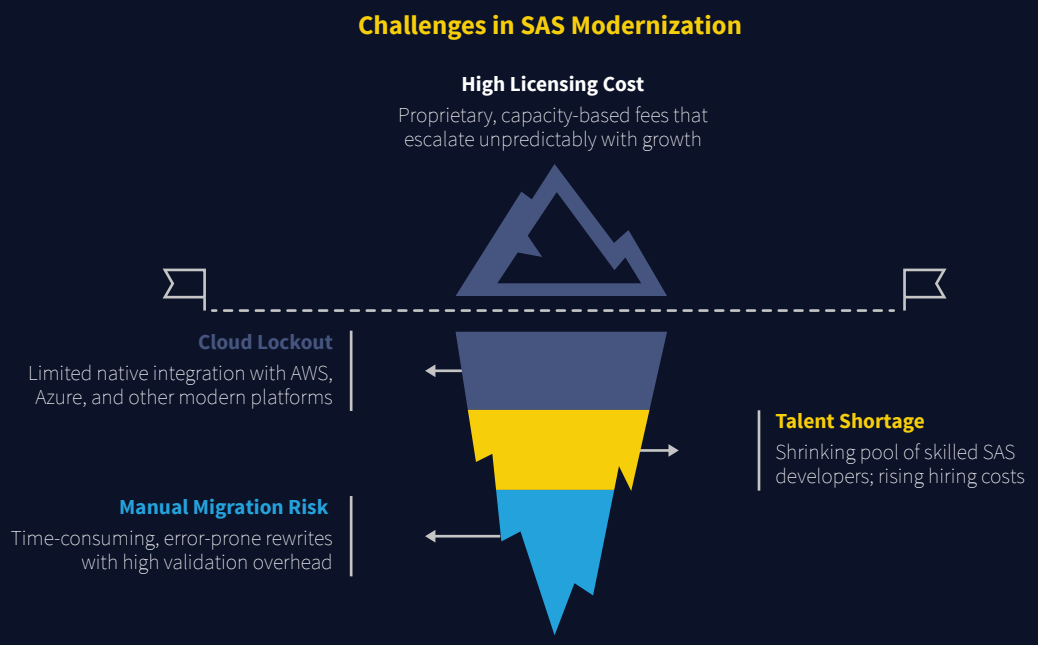

Figure 1: Key challenges of SAS modernization

1. 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.

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, a key advantage of cloud computing.

2. 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 and inefficient and fail to leverage the true elasticity and cost-effectiveness of cloud-native data processing environments.

3. 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 tricky.

Universities and training programs are shifting their focus towards open-source technologies like Python, R, and Spark, decreasing the 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.

4. Manual Migration Risk

Rewriting extensive SAS codebases to PySpark manually is inherently labour-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, an intelligent and automated framework is urgently needed to handle this transformation at scale while preserving intricate business logic and ensuring data integrity.

Beyond Manual Rewrites:
An AI-Driven, Multi-Agent Approach

Modernizing legacy SAS code does not have to mean starting from scratch. An emerging alternative reframes migration as a structured, AI-enabled process built around four key architectural layers:

01 **Parsing and structuring** the legacy code

02 **Semantic translation** into modern frameworks

03 **Automated validation** to ensure correctness

04 **Feedback-driven refinement** to improve over time

05 **Optimizing** code to increase efficiency

Each layer corresponds to specialized "agents" within the framework. Let's examine these in detail.

The Parse Agent: creating semantic structure

Before code can be translated, it must be understood. The Parse Agent acts like a compiler's front-end:

1. Tokenization

Breaking raw SAS code into discrete elements: keywords, variables, literals, and operators

2. Logical segmentation

Recognizing structural constructs (DATA steps, PROC statements, MACRO definitions)

3. Metadata extraction

Identifying data lineage: input datasets, output targets, join keys, filter conditions

4. Comment preservation

Retaining human explanations for downstream documentation

By converting free-form SAS code into a structured intermediate representation (IR), the Parse Agent provides downstream agents with the semantic context needed for accurate translation.

The LLM Rule Agent: context-aware translation

Large Language Models (LLMs) excel at pattern recognition, cross-language translation, and understanding semantic intent. In this layer:

1. The LLM Rule Agent uses carefully engineered prompts to instruct the model on converting SAS constructs into idiomatic PySpark equivalents

2. Prompts embed domain knowledge (e.g., mapping PROC SQL joins to PySpark DataFrame APIs)

3. The LLM operates as an "expert programmer" capable of translating data steps, statistical procedures, and transformation logic while retaining business intent

The agent employs fallback strategies for highly complex or ambiguous code blocks: wrapping code in spark.sql() constructs for human review, ensuring no logic is lost.

The Validation Agent: programmatic quality control

Even expert-generated code requires review. The Validation Agent automates this:

1. Completeness checks

Verifying all transformations from the Parse Agent are represented in the PySpark code

2. Structural integrity

Catching syntax errors, missing parentheses, or invalid DataFrame operations

3. Semantic checks

Confirming that filter conditions, joins, and aggregations preserve original intent

This agent drastically reduces manual testing effort and ensures a consistent migration baseline.

The Feedback Agent: learning from mistakes

No AI system is perfect on the first pass. The Feedback Agent introduces retry logic and iterative refinement:

- Logs errors from the Validation Agent
- Adjusts prompts or adds context before re-invoking the LLM Rule Agent
- Continues until code passes validation or flags it for human review

Over time, this feedback loop helps the system adapt to organization-specific coding styles and business logic nuances.

The Optimize Agent: serving output code

Finally, when the chunks of code are validated completely, the Optimize Agent merges the chunks in order, enhancing code performance and efficiency, reducing redundancies and improving execution.

- Merges chunks of code received from Validation and Feedback agents
- Executes the predefined prompt to optimize the merged code to enhance efficiency and improve execution

Design principles and architecture

A multi-agent system for SAS modernization must be designed to handle large-scale code migrations efficiently, adapt to evolving requirements, and provide clear, explainable outputs for compliance and audit readiness.

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 Architecture:

1. 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 proper understanding.

2. 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.

The Parse Agent can be updated to support a new SAS PROC statement. The LLM Rule Agent's prompts can be refined if the LLM's translation quality needs improvement for a specific pattern.

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.

3. 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 human experts to focus on more complex business logic validation and edge cases. This dramatically accelerates the overall migration timeline.

4. Retry and Feedback Loops

This intelligent mechanism allows the system to auto-correct errors and improve output using validation logs. It creates a self-healing system that learns from its failures.

This iterative refinement minimizes human intervention for standard 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.

Practical Benefits for Enterprises

An AI-driven, multi-agent migration framework offers more than speed. It changes modernization's economics and risk profile.

01

Cost savings

Lower SAS license fees, reduced manual engineering effort, and better use of cloud-native compute

02

Faster cloud adoption

Modern code integrates natively with platforms like Databricks, AWS Glue, or Azure Synapse

03

Improved agility

Teams can iterate on PySpark pipelines, integrate ML frameworks, and deploy new data products faster

04

Risk reduction

Automated validation minimizes silent data corruption and reduces dependence on scarce SAS specialists

05

Business logic preservation

Semantic parsing ensures decades of embedded rules are migrated, not discarded.

Use Cases

Real-world applications illustrate why this approach matters:

Enterprise SAS decommissioning

Large organizations often aim to retire SAS systems to reduce cost and complexity. AI-driven migration enables a phased approach, converting pipelines incrementally and validating outputs, rather than a risky “big bang” rewrite.

Cloud migration initiatives

Many data workloads remain on-premises because SAS code is complex to migrate. Automated translation into PySpark unlocks native execution in cloud platforms, improving scalability and cutting infrastructure management costs.

AI/ML enablement

SAS pipelines often include feature engineering steps vital for machine learning. Translating them into PySpark allows direct integration with TensorFlow, PyTorch, or scikit-learn, accelerating model development.

Consulting and COE automation

Data modernization consultancies can use AI-powered tools to scan client SAS codebases, propose migration plans, and automate first drafts, reducing delivery timelines and costs while focusing human expertise on complex edge cases.

Scintilla.AI: From Principle to Platform

LTIMindtree's Scintilla.AI embodies these architectural and methodological principles in a production-ready platform.

Built as a modular multi-agent system, Scintilla.AI automates SAS-to-PySpark migration by combining parsing, LLM-driven translation, validation, and feedback-driven refinement.

The platform preserves business logic, ensures code quality, and scales across thousands of legacy scripts.

Key capabilities include:

1. A semantic parsing engine that transforms raw code into structured representations
2. Context-aware translation using leading LLMs, guided by tailored prompts
3. Automated validation to catch syntax and semantic issues early
4. Feedback loops that iteratively refine output with minimal human intervention

Scintilla.AI helps enterprises reduce operational risk, cut costs, and accelerate cloud modernization strategies by turning complex, error-prone migration into a repeatable, explainable process.

Conclusion

Modernizing legacy SAS systems is not simply a technical upgrade; it's a strategic move toward agility, scalability, and competitive differentiation. Yet manual rewrites are rarely feasible at enterprise scale.

A structured, AI-driven, multi-agent approach offers a realistic path forward: it preserves historical business logic, improves accuracy, and delivers faster time-to-cloud.

Platforms like Scintilla.AI operationalize these principles, transforming modernization from a high-risk, multi-year effort into an incremental, automated process.

As organizations look to unlock new value from data, AI-powered automation stands out as an enabler of transformation in the cloud-native era.

References

AI Agents: Mastering Agentic RAG - Part 5, Shivam Goyal,Microsoft Educator Developer Blog, March 31, 2025

https://techcommunity.microsoft.com/blog/educatordeveloperblog/ai-agents-mastering-agentic-rag---part-5/4396171

Fully managed Retrieval Augmented Generation options, AWS:

https://docs.aws.amazon.com/prescriptive-guidance/latest/retrieval-augmented-generation-options/rag-fully-managed.html

Author's Profile

Ramesh Vanteru

Principal & Head of SAS COE, LTIMindtree

Ramesh is a SAS subject matter expert with nearly twenty years of experience in designing and developing data and SAS analytical applications. He is SAS Advanced Certified and a Snowflake SnowPro Certified Architect, offering technical advice on building SAS and Snowflake data platforms with extensive expertise in consulting, taxation, BFSI, healthcare, and cloud data platform implementation.