PharmaSUG China 2026 - Paper AI-211

CodeGenius 2026: Connecting Enterprise Metadata, Standards, Interactive Execution, and AI Coding Agents for Statistical Programming

Jing Wang, Yaohui Zhu, Xuejie Zhou, Jinling Li, You Li, BeOne Medicines

ABSTRACT

As clinical trial programming becomes increasingly driven by study metadata, company standards, reusable templates, and rapid validation feedback, statistical programmers need an integrated workspace that extends beyond traditional code editing. CodeGenius has evolved from an AI-powered SAS editor into an enterprise statistical programming workspace that brings together metadata, standards, code development, execution, and AI assistance within a single analysis-scoped environment. Built on Monaco Editor and integrated with the ISPP (Integrated SAS Programming Platform) analysis workspace, CodeGenius uses backend APIs and workspace-level file management to connect SAS programs, logs, listings, datasets, study trackers, standard macros, templates, and SDTM/ADaM/TLF specifications in a unified workflow. Company metadata and programming standards are no longer static reference materials; instead, they directly enable macro documentation, template recommendation, standard header generation, and context-aware programming guidance. CodeGenius also supports interactive SAS execution directly within the editor. Users can submit the current program or selected code to a live SAS session, monitor execution status in real time, and review logs, listings, generated datasets, and output files without leaving the workspace, creating a closed feedback loop from coding to execution and review. A key advancement in CodeGenius 2026 is its AI Coding Agent for statistical programming. Unlike a general-purpose chatbot, the Coding Agent operates with full awareness of the current editor context, selected code, analysis workspace, company standards, and domain specifications. It supports SAS code generation, explanation, debugging, log analysis, and program modification. By combining enterprise metadata, interactive execution, and a domain-specific Coding Agent, CodeGenius 2026 offers a practical and compliant path toward agentic statistical programming in the pharmaceutical industry.

INTRODUCTION

In 2025, we introduced CodeGenius — an AI-powered SAS editor built on Monaco Editor and designed specifically for statistical programming (SP) in clinical trials. That first generation demonstrated that a modern, browser-based editor combining AI-driven code generation, domain-specific completion, and macro hover documentation could meaningfully improve programmer productivity while keeping data and compliance firmly under enterprise control. It validated a core thesis: that an editor tailored to statistical programming, rather than a generic AI coding tool, is the right foundation for the pharmaceutical industry, where CDISC conventions, validated macros, and audit requirements are non-negotiable.

Over the past year, the demands on statistical programming have continued to intensify. Studies are increasingly metadata-driven. Specifications for SDTM, ADaM, and TLF, study trackers, standard macros, and template libraries have become the true source of truth, and programs are expected to conform to them precisely and provably. At the same time, programmers need ever-faster validation feedback. The loop from writing code, running it, reading the log, and inspecting the resulting datasets and listings must be as tight as possible, because each iteration that requires leaving the editor, switching to a separate execution environment, and returning costs minutes of context-switching and breaks concentration. A standalone editor, however intelligent, cannot deliver this when the metadata lives in one system, the macros in another, and execution happens in a third.

CodeGenius 2026 responds to these pressures by evolving from an editor into an enterprise statistical programming workspace. Rather than treating code editing, metadata reference,

execution, and AI assistance as four disconnected tools, CodeGenius unifies them in a single analysis-scoped environment where each reinforces the others. Three advances anchor this evolution. First, enterprise metadata and standards integration turns company macros, templates, study trackers, and SDTM/ADaM/TLF specifications into active inputs that drive documentation, recommendation, and standard enforcement, rather than static reference material a programmer must consult separately. Second, interactive SAS execution lets programmers submit a whole program or a selected fragment to a live SAS session and review logs, listings, datasets, and outputs directly in the editor, closing the code-to-review loop. Third, an AI Coding Agent — a domain-aware assistant that, unlike a general chatbot, sees the current editor context, the active selection, the analysis workspace, company standards, and specifications — assists with generation, explanation, debugging, log analysis, and direct program modification.

This paper describes the architecture and technical implementation behind these three advances, the innovative features they enable for statistical programmers, and the path they open toward compliant, agentic statistical programming in clinical trials.

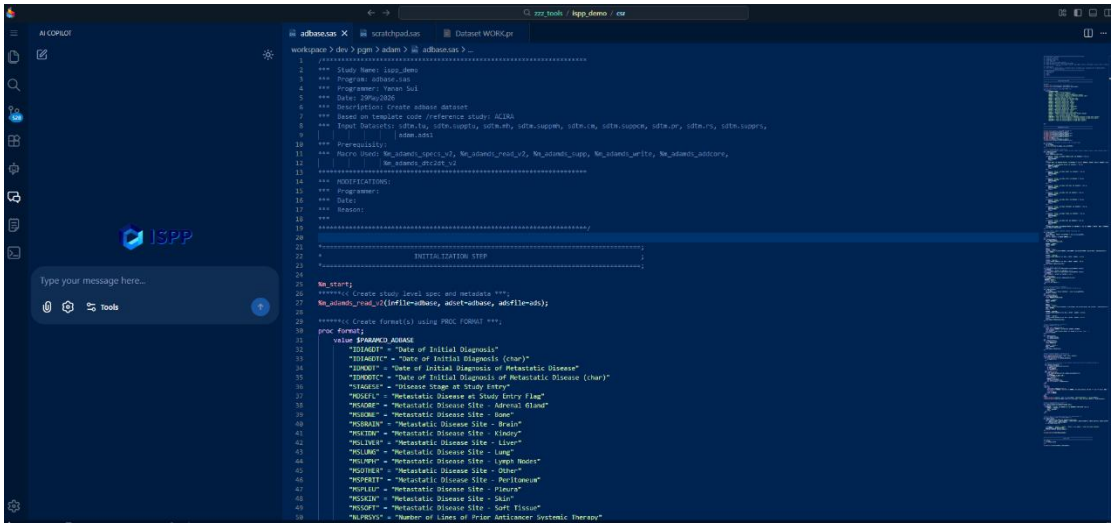
FROM EDITOR TO WORKSPACE: WHAT CHANGED SINCE 2025

The 2025 release concentrated on the editing experience and a streaming chatbot for code generation. Its unit of work was the file: a programmer opened a SAS program, edited it, asked the chatbot for help, and copied results back. The 2026 release shifts that unit from a file to an analysis workspace. Every CodeGenius session is now scoped to an analysis_id, and all files, execution requests, metadata lookups, and AI context flow through that scope. This single change is what allows the editor, the metadata, the live SAS session, and the AI to behave as one coordinated system rather than four cooperating tools, and it is the reason permissions and audit logging apply uniformly to every action a programmer takes.

The table below summarizes how each capability matured between the two generations.

Capability	CodeGenius 2025	CodeGenius 2026
Editing	Monaco-based SAS editor	Same, plus diff-preview apply and multi-output tabs
AI	Streaming chatbot (code generation)	Dual-mode: Coreflow chat **and** context-aware Coding Agent
Resources	Macro hover + completion	Live metadata, templates, standard headers, specs
Execution	External (SAS EG / batch)	Interactive in-editor SAS session + batch runs
Scope	Per-file	Analysis-scoped workspace (files, logs, listings, datasets)
Feedback loop	Open (run elsewhere)	Closed (code → run → log/dataset/listing → revise)

Table 1. CodeGenius 2025 vs. CodeGenius 2026



Display 1 — Editor with embedded chat

SYSTEM ARCHITECTURE AND TECHNICAL IMPLEMENTATION

SYSTEM ARCHITECTURE

CodeGenius 2026 retains the purpose-built layered architecture that balances cutting-edge AI capability with enterprise-grade security and domain-specific optimization, and extends it with two new layers — execution and workspace — that turn the editor into a complete programming environment.

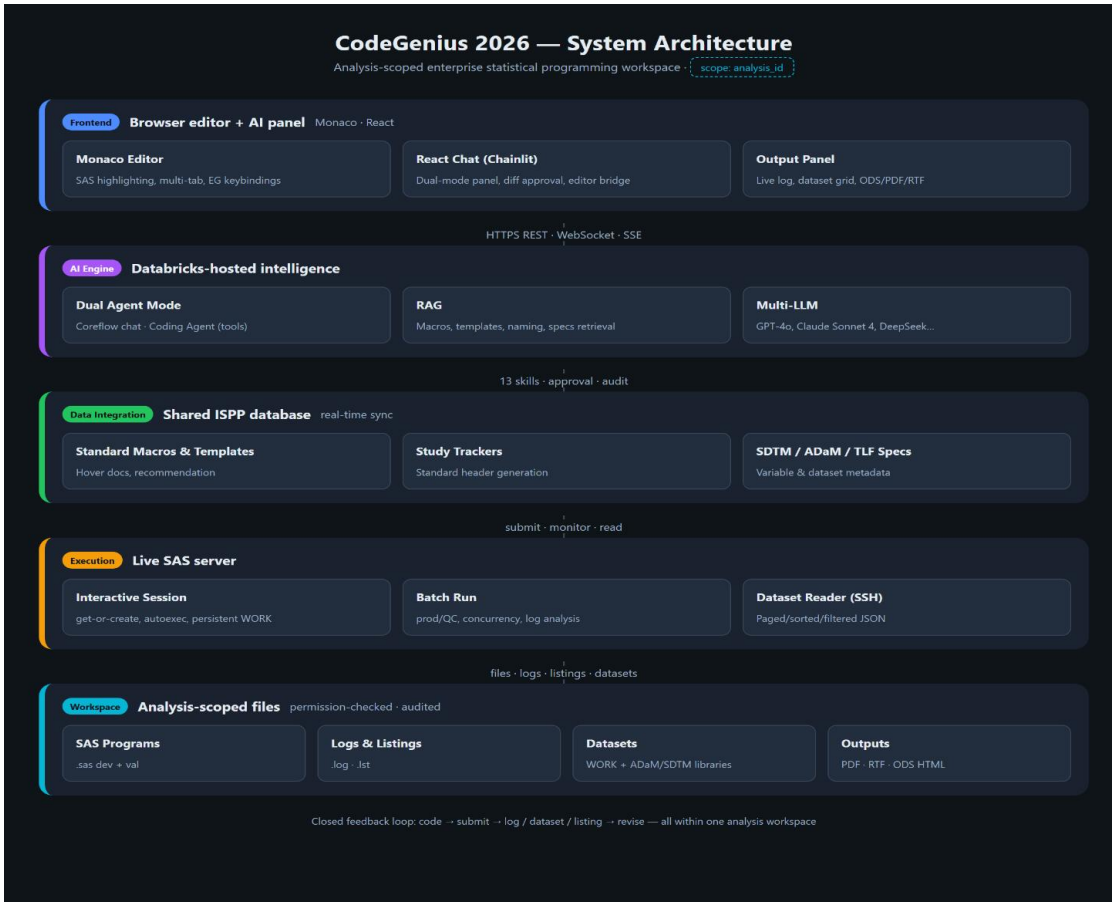


Figure 1. System Architecture in CodeGenius 2026

FRONTEND LAYER

Monaco Editor, the open-source engine behind Visual Studio Code, remains the core. It delivers context-aware SAS syntax highlighting that distinguishes data steps from PROC SQL, a multi-tab workspace for switching between programs, and SAS Enterprise Guide-style keybindings that ease the transition for programmers migrating from EG. The chat assistant is implemented as a React application embedded inside a Monaco sidebar panel, so AI lives beside the code rather than in a separate window. The panel and the editor communicate through a bidirectional event bridge, allowing AI features to read the active selection or file and write changes back precisely where the programmer is working.

AI ENGINE LAYER

Hosted on a dedicated Databricks platform for security and stability, this layer provides domain-optimized prompt engineering, multi-turn dialogue, and Retrieval-Augmented Generation (RAG) over corporate metadata, with access to multiple large language models. In 2026 it serves two distinct interaction modes from the same panel — a Coreflow conversational agent for open-ended questions and a statistical-programming Coding Agent for in-context program work — selected per message so the right model and prompt strategy apply to each request.

DATA INTEGRATION LAYER

Sharing the ISPP database, this layer keeps macros, templates, specifications, and study trackers synchronized in real time, so template recommendations and standard-header generation always reflect the current study's metadata rather than a stale snapshot.

EXECUTION LAYER (NEW)

A backend SAS session service runs programs interactively, streaming log entries and emitting job-status and step-completion events over WebSocket. A companion batch run service executes full programs and tracks status (running, success, failed, canceled). Together they let CodeGenius submit code and surface results without a separate execution tool.

WORKSPACE LAYER (NEW)

An analysis-scoped file manager exposes SAS programs, logs, listings, datasets, and output files through backend APIs, so the workspace that hosts editing also hosts execution results and supplies context to the AI.

ANALYSIS-SCOPED WORKSPACE

The defining structural change in 2026 is that everything is scoped to a study analysis through an analysis_id carried in the URL and attached to every backend and AI request. This makes the workspace the consistent unit for file access, execution, output review, and agent context. A programmer always works within one analysis, sees only its files and outputs, and any AI request automatically inherits that scope. Because every operation routes through the analysis, permissions and audit logging apply uniformly — a programmer cannot accidentally read or run against another study's data, and every action is recorded. This is what makes a powerful, AI-assisted, execution-capable workspace compatible with clinical data governance.

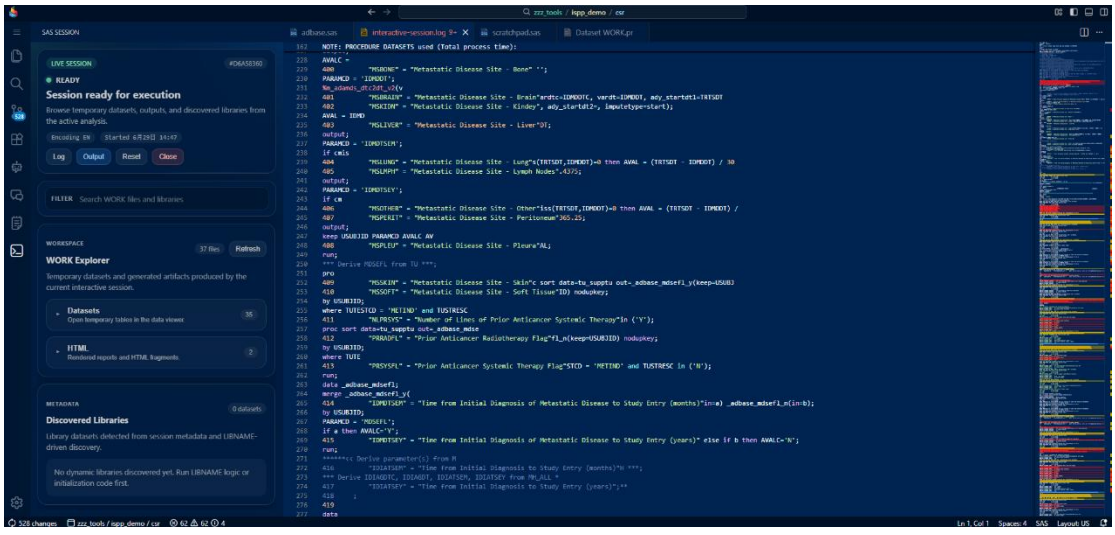
EDITOR - AI BRIDGE

Context awareness depends on the AI being able to see and change exactly what the programmer is looking at. CodeGenius implements this through a bidirectional bridge between the React chat panel and Monaco, built on custom DOM events. The bridge lets AI features read the current selection — or the full file when nothing is selected — capture the precise selection range, and write code back to either the selection or the whole file. Because the range is captured at the moment the request is sent, an edit applies correctly even after the selection visually clears, and the user can choose to insert, replace a selection, or rewrite the file.

```
// getEditorContent returns selection (with its range) or full file
const getEditorContent = (type = 'auto') => new Promise(resolve => {
  window.addEventListener(CHAINLIT_EDITOR_CONTENT_RESULT, onResult)
  window.dispatchEvent(new CustomEvent(CHAINLIT_GET_EDITOR_CONTENT, { detail:
{ type } }))
})
// replaceEditorContent applies AI output to the captured range or the file
const replaceEditorContent = (content, selectionRange) =>
  window.dispatchEvent(new CustomEvent(CHAINLIT_REPLACE_EDITOR_CONTENT,
    { detail: { content, selectionRange } }))
```

INTERACTIVE SAS EXECUTION

Interactive execution closes the gap that previously forced programmers to leave the editor for a separate environment. CodeGenius exposes a complete REST lifecycle for live SAS sessions, scoped to the analysis: create (get-or-create), submit, status, cancel, reset, and destroy, plus dedicated endpoints to browse the session WORK directory and read its datasets and files. The result is a closed loop entirely inside CodeGenius — write, submit, watch the log stream, inspect the datasets and listings, and revise — with no context-switching.



Display 2 — Interactive run + live log

Get-or-create sessions with study initialization. When a programmer first runs code, the backend either reuses a healthy session or provisions a new one tied to the user and analysis. New sessions are bootstrapped with the study's initialization program (autoexec) resolved

from the analysis metadata, so libnames, formats, and standard macros are loaded exactly as in production. Because the session is persistent, WORK datasets and macro state survive across submissions: a programmer can run a DATA step, then a PROC, and the second sees the first's tables — mirroring SAS Enterprise Guide's interactive feel.

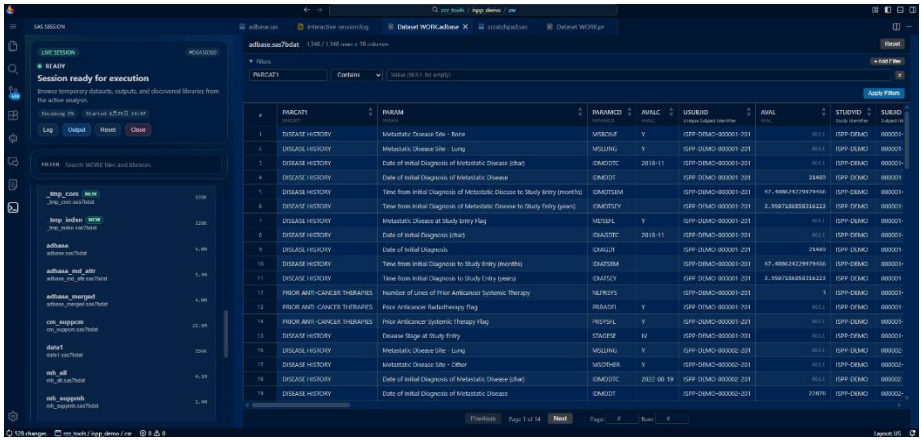
Submission and live log streaming. Code (whole program or selection) is submitted with a monotonically increasing seq_id; the call is accepted asynchronously (HTTP 202) and execution proceeds on the SAS server. Log lines are buffered per sequence and streamed to a per-session live log tab that is reused across submissions, so repeated runs never accumulate stale tabs. A polling refresh keeps the view current and a sas_session_step_complete WebSocket event appends a clear "Log complete" marker. A cancel endpoint sends an interrupt to the running step, and a reset endpoint clears WORK and macro state and invalidates dataset caches.

```
// Live log tab streams session output and marks completion
socketService.subscribe('sas_session_step_complete',
  ({ session_id, seq_id }) => { if (match) { completed = true; refresh() } })
```

WORK introspection and dataset reading. A WORK-files endpoint lists generated artifacts grouped by type — datasets, HTML/ODS output, images, PDF, RTF, and text — hiding SAS internal files. To read a SAS7BDAT dataset, the backend executes a reader script over a pooled per-user SSH connection on the SAS server; the script paginates, sorts, and filters locally and returns JSON that ISPP passes straight through, so no large files are transferred and no parsing happens on the ISPP host. The same path reads non-WORK libraries (e.g. ADAM) by libname, and ODS HTML output for each submission is fetched per seq_id. Every endpoint verifies session ownership, rejects path traversal with a strict filename pattern, and caps reads at 5 MB.

Outputs in the Workspace

Execution is only useful if its results are immediately reviewable. Datasets open in a paged, filterable, and sortable viewer (server-side validated page, page_size ≤ 1000, column allow-list, asc/desc, and a JSON filter array), so a programmer confirms derivations without exporting to another tool. Logs, listings, and ODS HTML/PDF/RTF outputs render in the same workspace, tracked separately for development and validation/QC. By placing dev and val artifacts beside the editor, CodeGenius becomes the place where validation feedback is read and acted on, not just where code is written.



Display 3 — Dataset viewer

AI CODING AGENT

The assistant offers two modes, toggled in the chat header and persisted per user: a **Coreflow** conversational agent for general questions and a **Coding Agent** for in-context program work. Coding mode is not a wrapped chatbot — it is a tool-using agent with full workspace context, implemented as a lightweight custom orchestrator over an OpenAI-compatible endpoint (no heavy framework). Each request runs a bounded reason-act loop: the model is offered a catalogue of typed skills, may call one or more, receives their results, and iterates up to a configurable limit (20 iterations, 300-second timeout, three automatic retries per skill) until it produces a final answer. Editor context — the active selection or full file — is injected into the first turn, so the model reasons about the programmer's real code, not a detached snippet.

Thirteen domain skills in four groups. The agent's capabilities are concrete, statistical-programming skills rather than generic chat: file skills (read, write, edit, list) confined to the analysis pgm directory; knowledge skills that pull ADaM/SDTM variable specs, dataset structures, and dataset lists from company metadata via a cached spec service; code skills for spec-driven SAS generation, explanation, and fixing; and execution skills that run programs through the existing batch system, parse logs for errors/warnings/notes with grouped issues and fix suggestions, and compare production vs QC outputs for validation. The agent therefore spans the full task — look up a spec, generate code, run it, read the log, fix, re-run.

Human-in-the-loop approval. Skills that modify files or execute code are flagged requires_approval. Before a write or edit lands, the agent computes the proposed content, diffs it, and pushes a Monaco diff preview to the chat panel (show_diff_preview); the coroutine blocks until the programmer clicks Keep or Discard. Execution skills ask for explicit approval too. This preserves the review discipline statistical programming demands — nothing reaches a file or the SAS server without a human decision.

Compliance by construction. File access is constrained by a regex to the study's pgm tree, path traversal and symlinks are rejected, and every session, skill call, approval, success, and error is recorded by an audit logger. Combined with analysis scoping, this keeps an autonomous-leaning agent inside clinical data-governance boundaries.

```
// Coding-mode messages carry editor + analysis context
getMessageMetadata() // { agent_mode: 'coding', analysis_id, editor_context }
// requires_approval skill → show_diff_preview → Keep applies, Discard rejects
```

INNOVATIVE FEATURES AND IMPLEMENTATION

Metadata-driven assistance. Macro documentation, template recommendation, and standard-header generation draw on live ISPP metadata and study trackers. Typing adsl surfaces the company ADSL template; standard headers are generated from study-tracker fields; hovering a macro shows its purpose, parameters, and manual link — all from current data, not static files.

Context-aware Coding Agent. Full awareness of the selection, file, workspace, standards, and specifications lets the agent generate, explain, debug, and analyze logs against the actual program rather than a detached snippet.

Interactive and batch execution. Submit a selection or a full program to a live session, or dispatch batch runs, and monitor status in real time without leaving the editor.

Integrated outputs. Logs, listings, datasets (filter, sort, page), and PDF/RTF outputs sit in one panel, organized per dev/val track.

Safe apply. A diff preview with Keep/Discard stands between any AI suggestion and the file, preserving review discipline.

One workspace. Editing, execution, review, and AI are scoped to a single analysis, uniformly permission-checked and auditable.

FUTURE DEVELOPMENT

CodeGenius 2026 establishes the workspace, the execution loop, and the Coding Agent; the next phase deepens their autonomy and reach. We plan to extend the agent toward multi-step plans that span generation, execution, log analysis, and fixes with explicit checkpoints, so it can carry a task further while keeping the programmer in control. We will broaden RAG over the historical study-program library to recommend and customize reference programs for new studies, strengthen validation support with automated comparison against QC outputs and standard-compliance checks, and tighten integration so dataflow and collaboration span more internal and external platforms. The aim throughout is to keep advancing efficiency and quality while preserving the compliance posture that makes the tool usable in clinical trials.

CONCLUSION

CodeGenius 2026 grows from an AI-powered editor into an enterprise statistical programming workspace, uniting metadata, standards, code, execution, and AI within one analysis-scoped environment. By making company knowledge active rather than static, closing the code-to-review loop with interactive execution, and adding a Coding Agent that understands the programmer's real context, it offers a practical and compliant path toward agentic statistical programming in the pharmaceutical industry. CodeGenius remains committed to applying advances in AI to real business scenarios, keeping programmers at the center while removing the friction between writing, running, and validating clinical trial code.