PharmaSUG China 2026 - Paper CC179-2026

ClaudeAgentSDK and shinyAssistantUI: Extending Claude Code's AI Agent Capabilities to the R Ecosystem

Kaiping Yang and Jie Liu, BeOne Medicines

ABSTRACT

Background: Claude Code, Anthropic's AI coding agent, goes beyond traditional large language model (LLM) APIs by maintaining persistent sessions, executing tools, managing files, and completing multi-step tasks autonomously with built-in permission controls and hook-based extensibility. Although an official Python SDK (claude-agent-sdk-python) exposes these capabilities programmatically, no R equivalent existed — a significant gap for pharmaceutical R and Shiny developers.

Methods: We developed two open-source R packages. ClaudeAgentSDK ports the Python SDK, spawning Claude Code as a managed subprocess communicating via a bidirectional newline-delimited JSON stream protocol. It supports multi-turn conversations, streaming output, session management, interrupt handling, tool permission approval, and hook-based customization, achieving over 97% API parity across 699 automated tests. A coro-based async architecture ensures Shiny integration without blocking the R event loop. shinyAssistantUI provides the frontend as an htmlwidget wrapping @assistant-ui/react, delivering streaming token display, tool call visualization, slash command menus, and file attachments via a lightweight Shiny module. Backend-agnostic, it connects to ClaudeAgentSDK in a few lines of code. Both packages were developed using an AI-agent-assisted programming workflow with third-party LLM backends.

Results: The combined stack enables complete AI agent applications in R and Shiny. Pharmaceutical use cases include clinical data query assistants, SAS-to-R translation tools, regulatory drafting aids, and human-in-the-loop analysis pipelines. Both packages are open source on GitHub.

Conclusions: ClaudeAgentSDK and shinyAssistantUI bring Claude Code's full agent capabilities to R, giving pharma developers a production-ready stack for AI-powered Shiny applications. Attendees will leave with working example code for immediate adaptation.

INTRODUCTION

The pharmaceutical industry's adoption of R for statistical computing, clinical trial analysis, and regulatory submission has accelerated substantially over the past decade. R is now a primary language for TFL generation, CDISC dataset processing, and submission-ready analysis at many sponsor companies and CROs. At the same time, large language models (LLMs) have matured from conversational novelties into programmable agents capable of executing multi-step tasks, operating file systems, and calling external tools — under human supervision.

However, for pharma R developers, integrating these AI agent capabilities has required either (1) switching to Python ecosystems, (2) using stateless LLM chat APIs that lack persistent context, or (3) building brittle custom wrappers around command-line tools. None of these solutions integrates naturally with Shiny, the dominant R framework for interactive pharmaceutical applications. Furthermore, while Anthropic provides an official Claude Code extension for Visual Studio Code, no equivalent exists for RStudio — leaving the majority of pharma R users without native agent IDE support.

This paper introduces two complementary open-source R packages that close this gap: ClaudeAgentSDK and shinyAssistantUI. Together, they give R developers access to the full Claude Code agent stack — persistent sessions, tool execution, human-in-the-loop approval, streaming output — from within standard R and Shiny code. Attendees will leave with working example code, draw.io architecture diagrams, and concrete pharma use case patterns ready for immediate adaptation.

BACKGROUND: CLAUDE CODE AS AN AI AGENT PLATFORM

BEYOND CHAT APIS: PERSISTENT SESSIONS AND TOOLS

Most pharma teams' current LLM experience centers on stateless chat APIs: each API call is independent, the model has no memory between calls, and all context must be resent on every request. Claude Code (Anthropic, 2024) is architecturally different. It maintains a persistent session process that retains conversation history, has access to file system tools (read, write, edit), can execute shell commands, and manages ongoing multi-step tasks. These properties map directly onto pharmaceutical workflows where analysis often involves many sequential, interdependent steps.

Claude Code exposes a bidirectional newline-delimited JSON (NDJSON) stream protocol that allows external programs to drive it as a managed subprocess. An official Python SDK (claude-agent-sdk-python) wraps this protocol into a clean async API. However, no equivalent existed for R — until now.

THE R ECOSYSTEM GAP

Pharmaceutical statisticians and programmers working in R faced a binary choice: accept the limitations of stateless LLM APIs, or adopt Python. ClaudeAgentSDK eliminates this choice by implementing the full Python SDK protocol in idiomatic R, using R6 classes for the stateful client and transport layers, S3 types for message objects, and the coro package for non-blocking async execution compatible with Shiny's event loop.

PACKAGE 1: CLAUDEAGENTSDK

ARCHITECTURE OVERVIEW

ClaudeAgentSDK spawns Claude Code as a managed subprocess, communicating via its bidirectional NDJSON stream protocol. Figure 1 illustrates the full system architecture.

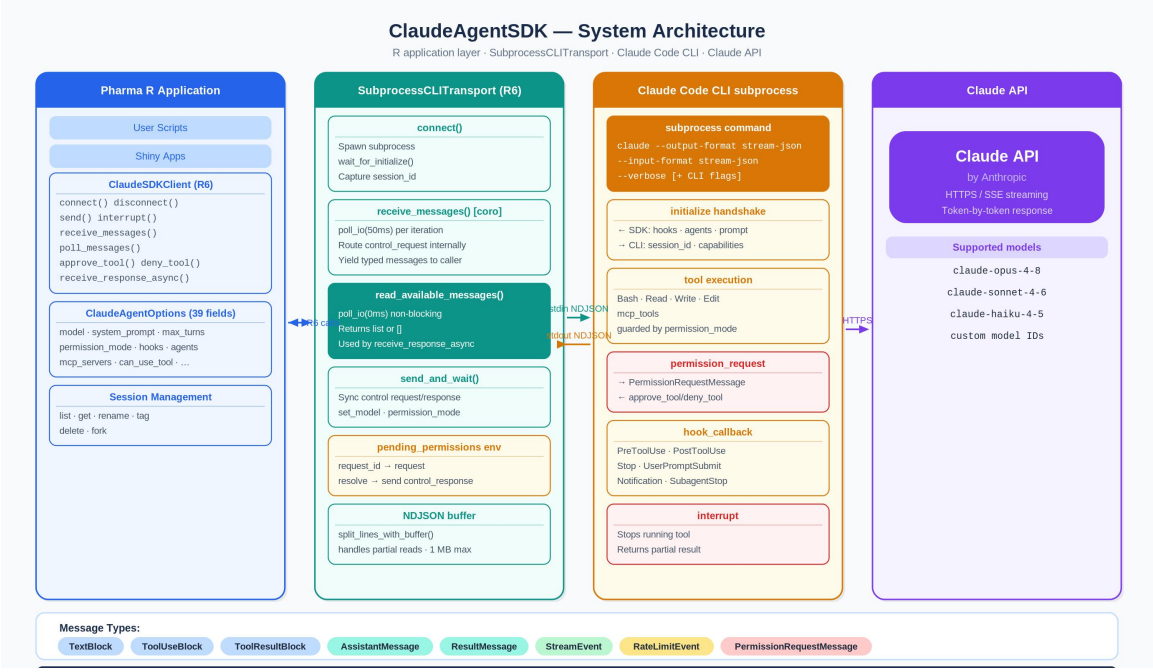

Figure 1. ClaudeAgentSDK System Architecture — R application layer, SubprocessCLITransport, Claude Code CLI subprocess, and Claude API

The package is organized into two main R6 classes. SubprocessCLITransport manages the low-level subprocess lifecycle — spawning the CLI with --output-format stream-json --input-format stream-json --verbose, buffering stdout, and routing control protocol messages (initialize, tool approval, interrupt). ClaudeSDKClient provides the user-facing API: connect(), send(), interrupt(), and receive_response_async() for Shiny integration.

All message types are lightweight S3 classes mirroring the Python SDK's types.py: TextBlock, ToolUseBlock, ToolResultBlock, AssistantMessage, ResultMessage, StreamEvent, RateLimitEvent, and PermissionRequestMessage. This type system ensures pattern-matching code (inherits(msg, 'StreamEvent')) works naturally in R.

KEY CAPABILITIES

ClaudeAgentSDK supports the following capabilities, listed in Table 1.

Capability	Python SDK	ClaudeAgentSDK (R)
Multi-turn sessions	Yes	Yes
Streaming output	Yes	Yes — StreamEvent type
Session management	Yes	Yes — list/get/rename/tag/delete/fork
Tool permission approval	Async callback	Message-driven + Shiny modal
Hook-based extensibility	Yes	Yes — 10 hook input types
Interrupt handling	Yes	Yes — client$interrupt()
Custom agents	Yes	Yes — AgentDefinition (13 fields)
Thinking config	Yes	Yes — Adaptive/Enabled/Disabled
API parity	Reference	97% (699 tests)

Table 1. ClaudeAgentSDK Capability Comparison with Python SDK

CODE EXAMPLE: CLINICAL DATA QUERY

The following example demonstrates a minimal multi-turn clinical data query session. The agent receives system prompt context describing the available CDISC datasets and responds with R analysis code.

```
library(ClaudeAgentSDK)

opts <- ClaudeAgentOptions(
  system_prompt = 'You are a clinical data analyst. Available datasets:
                   adsl, adae, adlb in the /data/ directory.',
  permission_mode = 'acceptEdits'
)
client <- ClaudeSDKClient$new(opts)
client$connect()

# Send natural language query
client$send('Show adverse event rates by treatment arm, serious AEs only')

# Collect streaming response
for (msg in client$receive_messages()) {
  if (inherits(msg, 'StreamEvent')) {
    cat(msg$delta)
  }
  if (inherits(msg, 'ResultMessage')) break
}
client$disconnect()
```

TESTING AND QUALITY

ClaudeAgentSDK ships with 699 automated tests across 12 test files. The majority require no live Claude Code CLI, enabling fast CI runs. Integration tests cover the full round-trip: connect, send, stream, interrupt, and session management. Parity with the Python SDK stands at 97% across options fields, session

management, error types, type definitions, client methods, and transport protocol. The three percent gap consists of intentional omissions: Python-only deprecated parameters, in-process MCP server types (R uses the mcptools subprocess approach instead), and mypy union type aliases with no R runtime value.

PACKAGE 2: SHINYASSISTANTUI

ARCHITECTURE OVERVIEW

shinyAssistantUI is an htmlwidget that wraps @assistant-ui/react, a production-grade React chat component library. It provides Shiny developers with a full-featured AI chat UI — streaming token display, tool call visualization, slash command menus, and file attachments — without requiring React or JavaScript expertise. Figure 2 shows the data flow from user input to rendered streaming response.

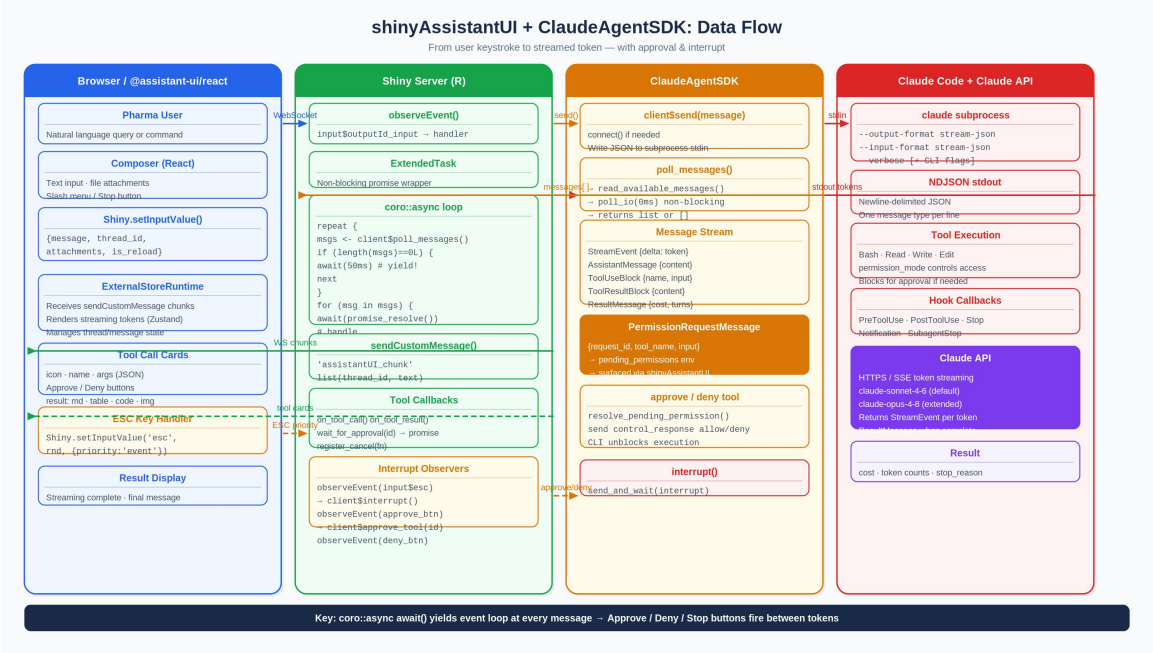

Figure 2. shinyAssistantUI + ClaudeAgentSDK data flow — React Composer, WebSocket, Shiny server, coro async loop, ClaudeAgentSDK, and Claude CLI

BACKEND-AGNOSTIC DESIGN

shinyAssistantUI is backend-agnostic: any R function that can stream text chunks can serve as the handler. The package ships with first-class integration for two backends. The make_claude_handler() function wires ClaudeAgentSDK for full agentic capabilities — persistent sessions, tool use, and human-in-the-loop approval. The make_ellmer_handler() function integrates the ellmer package (tidyverse, 2024), supporting any OpenAI-compatible LLM provider with automatic session persistence and tool-call approval cards. Any custom R streaming backend can also be connected via the documented handler contract. This design allows teams to switch AI backends without rewriting any UI code.

FLOATING MODAL ASSISTANT MODE

shinyAssistantUI supports a floating modal assistant mode via the modal = TRUE parameter. When enabled, the chat widget renders as a small bubble overlay on top of any existing Shiny application — users click the bubble to expand a full chat panel, then collapse it to return to their analysis. This pattern is particularly valuable in pharmaceutical applications where the AI assistant augments an existing TFL viewer, data explorer, or submission portal without replacing it. Deployment requires no layout changes to the host application:

```
# Add to any existing Shiny app
```

```
ui <- fluidPage(
  # ... existing app UI ...
  assistantUIOutput('assistant', modal = TRUE)  # floating bubble
)

server <- function(input, output, session) {
  # ... existing server logic ...
  assistantUIServer('assistant', handler = make_claude_handler(options))
}
```

CODE EXAMPLE: MINIMAL INTEGRATION

The following example shows a complete Shiny application connecting shinyAssistantUI to ClaudeAgentSDK in twelve lines of server code.

```
library(shiny)
library(shinyAssistantUI)
library(ClaudeAgentSDK)

ui <- fluidPage(assistantUIOutput('chat', height = '80vh'))

server <- function(input, output, session) {
  client <- ClaudeSDKClient$new(ClaudeAgentOptions())

  assistantUIServer('chat', handler = function(message, on_chunk, on_done, on_error) {
    client$connect()
    client$send(message)
    client$receive_response_async(on_message = function(msg) {
      if (inherits(msg, 'AssistantMessage')) {
        for (blk in msg$content)
          if (inherits(blk, 'TextBlock')) on_chunk(blk$text)
      }
      if (inherits(msg, 'ResultMessage')) on_done()
    })
  })
}
shinyApp(ui, server)
```

Figure 3 shows the shinyAssistantUI running in a browser with streaming output and tool call cards visible.

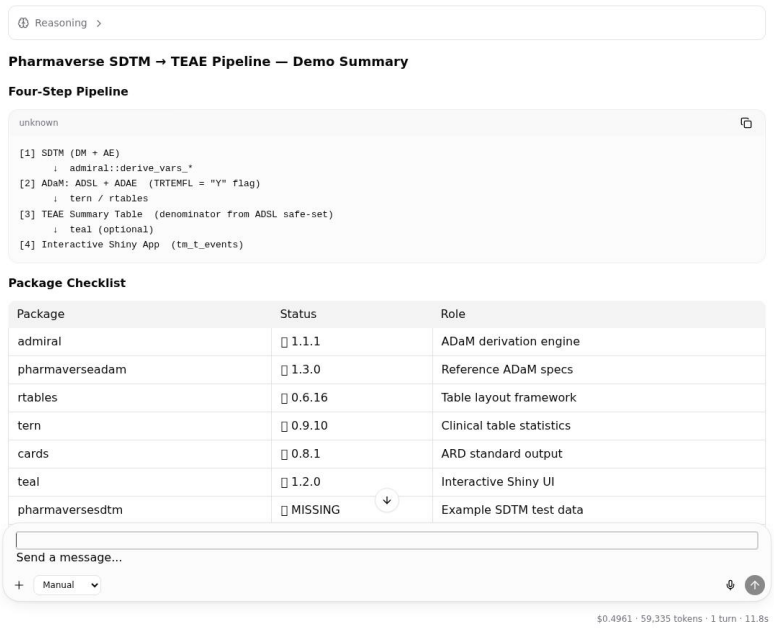

Figure 3. shinyAssistantUI running in Shiny — Pharmaverse SDTM→TEAE Pipeline demo with streaming chat

RSTUDIO ADDIN: BRIDGING THE VS CODE GAP

While Anthropic provides an official Claude Code extension for Visual Studio Code, no equivalent exists for RStudio — the IDE used by the majority of pharmaceutical statisticians and programmers. shinyAssistantUI closes this gap by shipping a registered RStudio addin, claude_addin(), that launches a full Claude Code chat session directly inside the RStudio Viewer pane or a floating dialog, rooted at the current R project.

The addin is context-aware: for each new prompt, it samples the active editor file path and any selected code via rstudioapi, appending them to Claude Code's system prompt as live context. This means the user can highlight a block of R code, open the addin, and ask 'explain this' or 'refactor for tidyverse' without any manual copy-paste. The workspace @-mention system provides gitignore-aware fuzzy file completion, allowing the user to type @data/adsl to reference project files directly in the chat composer.

Four permission modes control how aggressively Claude Code operates on the project: default (approval card for every tool call), plan (read-only analysis), acceptEdits (file edits auto-approved, shell commands still prompt), and bypassPermissions (no prompts — fastest, suitable for isolated development environments). Conversation history is stored in ~/.claude_addin_session_map.rds, persisting across project switches and RStudio restarts.

The addin is installed automatically with the shinyAssistantUI package and appears in the RStudio Addins menu as 'Claude Code Chat'. It requires no configuration beyond having ClaudeAgentSDK and a valid Anthropic API key. The following code launches the addin programmatically with auto-approved file edits:

```
# From RStudio Addins menu → 'Claude Code Chat'
# Or call directly:
library(shinyAssistantUI)

# Default: approval card for every tool call
claude_addin()

# Auto-approve file edits, prompt for shell commands
claude_addin(permission_mode = 'acceptEdits')
```

```
# Read-only analysis and planning
claude_addin(permission_mode = 'plan')

# Floating dialog instead of Viewer pane
claude_addin(viewer = 'dialog', permission_mode = 'acceptEdits')
```

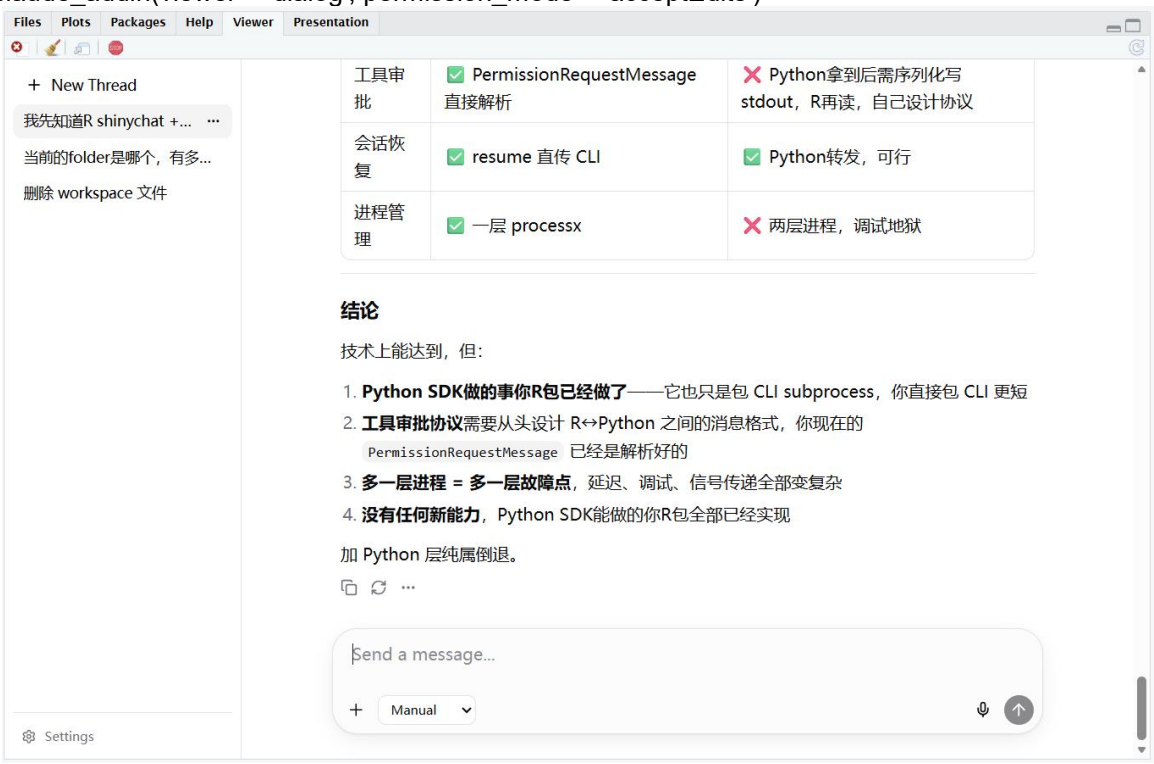

Figure 4. shinyAssistantUI RStudio Addin — Claude Code Chat in the RStudio Viewer pane

PHARMACEUTICAL USE CASES

Figure 5 provides an overview of the four primary pharmaceutical use cases enabled by the combined stack.

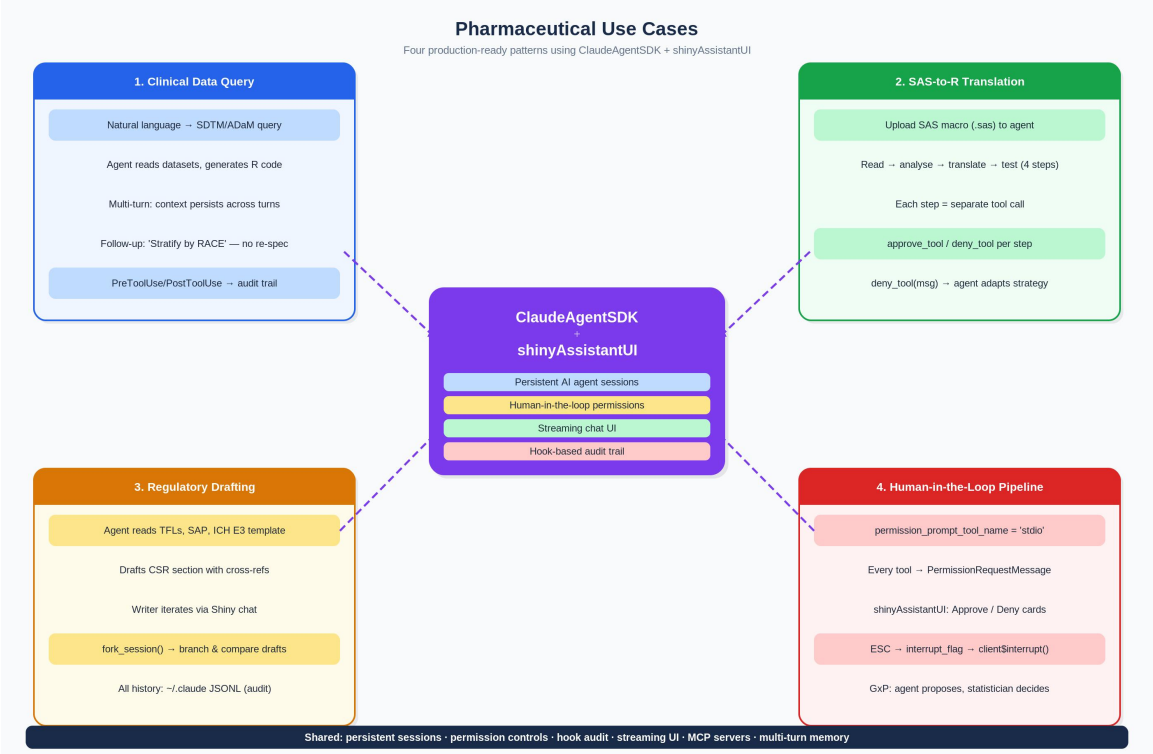

Figure 5. Pharmaceutical use cases — four patterns enabled by ClaudeAgentSDK + shinyAssistantUI

USE CASE 1: CLINICAL DATA QUERY ASSISTANT

Biostatisticians and clinical programmers frequently need to query SDTM and ADaM datasets for ad-hoc analyses outside the formal TFL pipeline. Traditionally this requires either writing R or SAS code from scratch or waiting for a programmer. With ClaudeAgentSDK, a Shiny application can accept natural language queries ('What is the median time to first AE by treatment group for patients with prior cardiac history?') and translate them into executable R code against the project's CDISC datasets.

The agent maintains session context across turns, so a follow-up query ('Now stratify by RACE') requires no re-specification of the original question. The permission_mode setting controls whether the agent can read files autonomously or must request approval for each file access — a critical control for GxP environments.

USE CASE 2: SAS-TO-R TRANSLATION TOOL

Many pharma companies are mid-migration from SAS to R. Legacy SAS macros (%ADTTE, %MMRM, internal utility macros) represent thousands of person-hours of validated logic that must be faithfully reproduced in R. Manual translation is slow and error-prone.

ClaudeAgentSDK enables a multi-step translation workflow: (1) the agent reads and analyzes the SAS macro, (2) proposes an equivalent R function using tidyverse or base R idioms, (3) generates testthat unit tests against known SAS outputs, and (4) runs the tests. Each step is a separate tool call that the statistician can approve, modify, or reject via the shinyAssistantUI interface. The deny_tool() method allows rejecting specific tool executions while keeping the session alive — the agent adapts based on the rejection message.

USE CASE 3: REGULATORY DOCUMENT DRAFTING AID

Clinical Study Report (CSR) writing is document-intensive, requiring consistent cross-referencing between TFLs, the protocol, the Statistical Analysis Plan, and ICH E3 guidelines. A ClaudeAgentSDK-powered

Shiny application can assist medical writers by: reading TFL outputs and prior-section drafts, generating ICH-aligned draft text, and allowing iterative revision via conversational follow-up.

Session management features — rename_session(), tag_session(), fork_session() — enable document version branching. A medical writer can fork a session at any checkpoint, explore an alternative phrasing branch, and discard it if the original was superior. All session history persists in ~/.claude/projects/ JSONL files, providing an audit trail without any additional infrastructure.

USE CASE 4: HUMAN-IN-THE-LOOP ANALYSIS PIPELINE

Regulatory submissions require that statistical analyses are conducted under human oversight. ClaudeAgentSDK's permission model maps directly onto this requirement. When permission_prompt_tool_name = 'stdio' is set and no synchronous can_use_tool callback is provided, tool use requests flow through the message stream as PermissionRequestMessage objects. The Shiny application can display these as modal dialogs, and the statistician approves or denies each tool call individually via client$approve_tool(request_id) or client$deny_tool(request_id).

The coro::async architecture ensures that the approval modal remains responsive even while the agent is running: the 50-millisecond yield in the polling loop allows Shiny input events (button clicks, ESC key) to fire between token deliveries. This pattern is demonstrated in the package's example 15 (examples/15_shinychat_tool_approval_msgdriven.R) and is the recommended production pattern for GxP-adjacent applications.

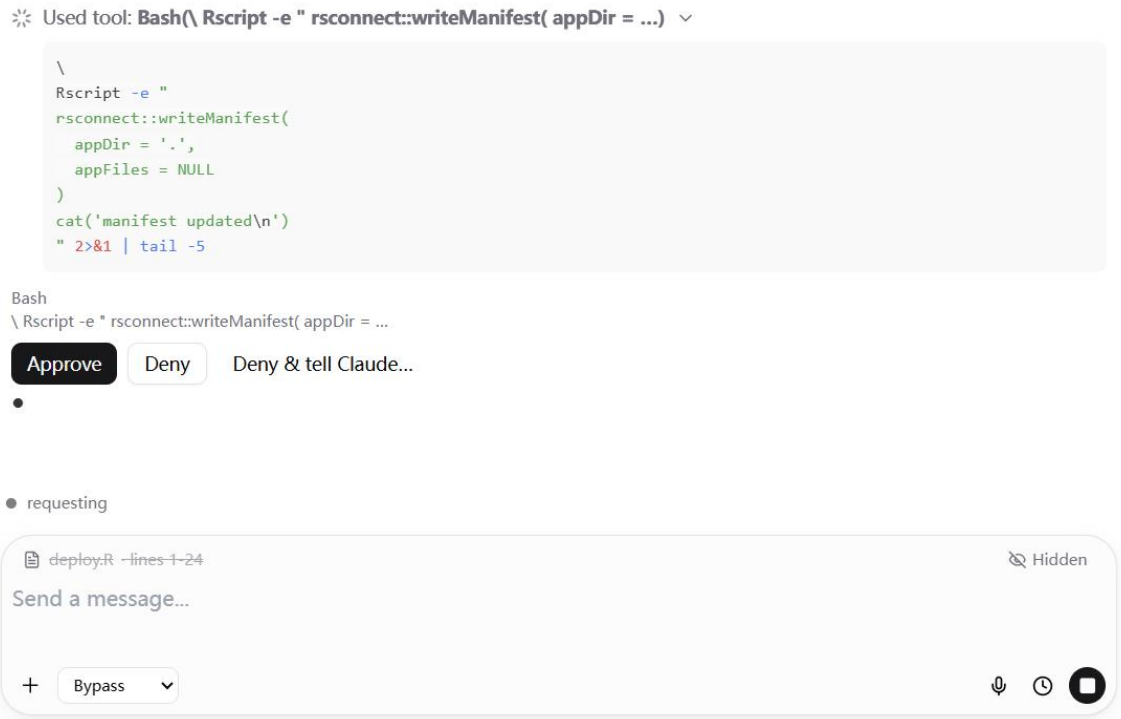

Figure 6. Tool approval card in shinyAssistantUI — Approve / Deny / Deny & tell Claude buttons during active agent session

AI-AGENT-ASSISTED DEVELOPMENT WORKFLOW

Both ClaudeAgentSDK and shinyAssistantUI were themselves developed using an AI-agent-assisted programming workflow, with Claude Code as the development agent and third-party LLM backends for code generation. This is not a coincidence: the packages were designed by eating their own cooking.

The development workflow consisted of four phases. First, the Python SDK source was analyzed by an agent to produce a structured parity matrix. Second, the agent generated R implementations function by

function, writing tests before implementations (TDD). Third, integration tests were run against a live Claude Code CLI to validate round-trip behavior. Fourth, bugs found during integration testing were reported back to the agent as structured bug reports, which it used to propose and apply fixes.

Over the course of development, eight systematic bugs were identified and fixed via a formal parity audit comparing R and Python behavior across all 39 options fields, 15 client methods, and all message type constructors. The audit was itself conducted by a multi-agent workflow: one agent read the Python SDK, one read the R implementation, and a third synthesized the comparison.

For pharma development teams, this workflow has direct applicability. Code migrations, validation documentation, and test generation are all tasks where AI agents excel when given persistent context, file system access, and human approval at each critical step — exactly what ClaudeAgentSDK enables.

DISCUSSION

LIMITATIONS AND CONSIDERATIONS

Several limitations should be considered when evaluating ClaudeAgentSDK for pharmaceutical use. First, the package requires a local Claude Code CLI installation, which in turn requires an Anthropic account and API key. Organizations with strict data egress controls should ensure that any patient data passed to the agent in system prompts or tool results is appropriately de-identified or governed by a suitable DPA.

Second, Claude Code's tool execution occurs within the permissions of the R process running it. In Shiny Server or Posit Connect environments, this is typically a restricted service account, which limits blast radius. However, the permission_mode setting should be set conservatively ('default' rather than 'bypassPermissions') in any environment where the agent has access to production data or systems.

Third, while 699 tests provide strong coverage, ClaudeAgentSDK is new software. Teams deploying it in validation-critical workflows should treat it as COTS software requiring qualification testing against their specific use cases, not as a validated system.

COMPARISON WITH ALTERNATIVES

The ellmer package (tidyverse, 2024) provides a clean R interface to multiple LLM providers but operates on stateless request-response semantics. It is well-suited for single-turn generation tasks. ClaudeAgentSDK complements rather than competes with ellmer: for multi-step agentic workflows with file system access, ClaudeAgentSDK is the appropriate choice; for single-turn text generation or summarization, ellmer's simpler interface may be preferable. shinyAssistantUI explicitly supports both backends via make_claude_handler() and make_ellmer_handler() respectively.

FUTURE WORK

shinyAssistantUI's backend-agnostic handler contract enables a growing ecosystem of R agent packages that share the same Shiny UI layer. Two packages are under active development, each targeting a distinct backend scenario:

- CodexAgentSDK — An R SDK for OpenAI Codex agents using the same subprocess protocol and handler interface as ClaudeAgentSDK. Pharma teams will be able to choose between Claude Code and Codex based on infrastructure and data governance requirements while reusing all existing shinyAssistantUI components and human-in-the-loop approval patterns.
- codeagent — A fully R-native coding agent built on ellmer and btw that requires no external CLI dependency. It implements the agent loop, tool execution, permission system, context compaction, skill system, and sub-agent coordination entirely within R. This approach eliminates the Claude Code CLI installation requirement and broadens the range of compatible LLM providers to any ellmer-supported backend.

Together these packages extend the R agent ecosystem: shinyAssistantUI provides the Shiny UI surface; ClaudeAgentSDK, CodexAgentSDK, and codeagent provide agent backends ranging from Claude Code to OpenAI Codex to fully R-native; and the shared handler contract ensures that advances in any layer benefit the full stack without UI rewrites.

CONCLUSION

ClaudeAgentSDK and shinyAssistantUI bring Claude Code's full AI agent capabilities to the R ecosystem. The combined stack enables pharmaceutical developers to build production-ready AI-powered Shiny applications without leaving R: persistent multi-turn sessions, streaming output, human-in-the-loop tool approval, hook-based audit logging, and a full-featured chat UI are all available via standard R package installation.

The four pharmaceutical use cases described — clinical data query assistants, SAS-to-R translation, regulatory document drafting, and human-in-the-loop analysis pipelines — represent high-value, high-feasibility targets for early adoption. The human-in-the-loop pattern in particular aligns directly with GxP principles: the agent proposes, the statistician decides, and the audit trail is built in.

shinyAssistantUI also ships as a registered RStudio addin ('Claude Code Chat'), bringing native Claude Code agent access to RStudio users without requiring VS Code. This addresses a key gap in the pharma R ecosystem, where RStudio remains the dominant IDE. Context-aware and permission-gated, the addin integrates directly into the RStudio Viewer pane and is rooted in the user's current R project.

Both packages are open source on GitHub. Attendees are encouraged to clone the repositories, review the 24 included examples, and adapt the patterns to their own pharmaceutical workflows.

Two companion packages under active development — CodexAgentSDK and codeagent — will extend this ecosystem to additional agent backends and fully R-native agent loops respectively. Details are provided in the Future Work section.

REFERENCES

Anthropic. 2024. "Claude Code: Agentic Coding." Anthropic Documentation. Available at https://docs.anthropic.com/en/docs/claude-code

Anthropic. 2025. "claude-agent-sdk-python." GitHub. Available at https://github.com/anthropics/claude-agent-sdk-python

Yang, K. 2025. "ClaudeAgentSDK: R SDK for Claude Code." GitHub. Available at https://github.com/kaipingyang/ClaudeAgentSDK

Yang, K. 2025. "shinyAssistantUI: Shiny htmlwidget wrapping @assistant-ui/react." GitHub. Available at https://github.com/kaipingyang/shinyAssistantUI

assistant-ui. 2024. "@assistant-ui/react: React components for AI chat interfaces." GitHub. Available at https://github.com/assistant-ui/assistant-ui

Chang, W., Cheng, J., Allaire, J.J., et al. 2024. shiny: Web Application Framework for R. R package version 1.9.0. Available at https://CRAN.R-project.org/package=shiny

Csárdi, G., Henry, L. 2024. coro: Coroutines for R. R package version 1.1.0. Available at https://CRAN.R-project.org/package=coro

Chang, W. 2023. R6: Encapsulated Classes with Reference Semantics. R package version 2.5.1. Available at https://CRAN.R-project.org/package=R6

Wickham, H., Bryan, J. 2023. ellmer: A Consistent Interface to Large Language Models. GitHub. Available at https://github.com/tidyverse/ellmer

ICH. 1995. "E3: Structure and Content of Clinical Study Reports." International Council for Harmonisation of Technical Requirements for Pharmaceuticals for Human Use.

Wickham, H., Averick, M., Bryan, J., et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4(43):1686.

ACKNOWLEDGMENTS

The author thanks the Anthropic team for open-sourcing Claude Code and its Python SDK, the assistant-ui team for the React chat component library, and colleagues for feedback on pharmaceutical use case requirements.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kaiping Yang
BeOne Medicines
Email: 1260146556@qq.com
GitHub: https://github.com/kaipingyang/ClaudeAgentSDK
GitHub: https://github.com/kaipingyang/shinyAssistantUI