PharmaSUG China 2026 - Paper AI178-2026

# Coding Agent Builds BIP Copilot: An LLM-Powered R Shiny Platform for Clinical Biomarker Analysis

Kaiping Yang, BeOne Medicines; Ruiqi Huang, BeOne Medicines; Bohua Chen, University Medical Center Göttingen

Statistical Research and Innovation, BeOne Medicines

## ABSTRACT

**Background:** Clinical biomarker data in oncology not only provides insights on disease risk and treatment heterogeneity, but accumulates across studies as company data assets for mechanism-of-action (MOA) research and future study design. Integrating these standardized results requires effective, flexible search. We introduce BIP Copilot — an AI assistant to search biomarker-defined subgroup analysis results from the company's translational repository, restricted to validated methods, unlike general web search tools.

**Methods:** BIP (Biomarker Interactive Profiler) Copilot is an R Shiny application built with the ellmer package, connected to GPT-4.1 via bslib + shinychat. Users interact through guided prompts to retrieve results or run new analyses. Queries route through a LangChain-powered RAG layer to registered tools covering SQL execution, statistical analyses, KM survival plots, forest plots, boxplots, and baseline characteristic tables. Key features include BM25/re-ranking for tool-calling accuracy, dynamic cross-filtering, and on-demand table formatting. A report tool serializes conversation history and tool results into structured RMarkdown documents (HTML/PDF/Word). A coding agent serves as development partner, executing features under developer-maintained instructions; project-specific skills codify reusable patterns for tool deployment, configuration updates, and task migration.

**Results:** Users pose natural language questions — from 'Is PDL1 prognostic in study s1?' to 'What is the risk difference/ratio for ORR between PDL1-high and low?' — and receive tabular results with AI clinical interpretation within 30 seconds. The coding agent paradigm reduced feature delivery from weeks to hours with no regression defects.

**Conclusions:** BIP Copilot demonstrates two complementary advances: a practical R architecture for LLM-augmented biomarker analysis preserving statistical rigor through validated tool functions; and a reproducible AI-assisted development workflow in which a coding agent autonomously implements product features — AI building the AI product.

**Keywords:** *Biomarker, LLM, R Shiny, Tool Calling, ellmer, Coding Agent, Survival Analysis, Clinical Data Analysis, AI-Assisted Development*

## INTRODUCTION

Clinical biomarker analysis is central to oncology drug development. Across multiple clinical trials, biomarker-defined subgroup analyses accumulate as a rich institutional knowledge base — covering prognostic and predictive effects for endpoints such as progression-free survival (PFS), overall survival (OS), and objective response rate (ORR). Accessing and comparing these results has historically required programming expertise: analysts must write SQL queries, invoke statistical functions, and assemble visualizations manually, creating a bottleneck for cross-functional collaboration.

The advent of Large Language Models (LLMs) with function-calling (tool-use) capabilities offers a new paradigm: natural language as the interface to complex analytical pipelines. Rather than replacing validated statistical methods, the LLM acts as an intelligent dispatcher — translating user intent into precise tool calls that execute pre-validated R functions. This approach preserves statistical rigor while dramatically lowering the barrier to data access.

BIP (Biomarker Interactive Profiler) Copilot is the realization of this paradigm within BeOne Medicines' translational research infrastructure. It extends the existing BIP platform — a dropdown-based Shiny

interface for biomarker filtering — with a conversational AI layer built on the ellmer R package. This paper describes the system architecture, implementation details, key design decisions, and the novel use of a coding agent as an autonomous development partner.

## SYSTEM ARCHITECTURE

### OVERVIEW

BIP Copilot follows a layered architecture connecting the user interface through the LLM reasoning layer down to the data and tool execution layer. Figure 1 illustrates the end-to-end data flow.

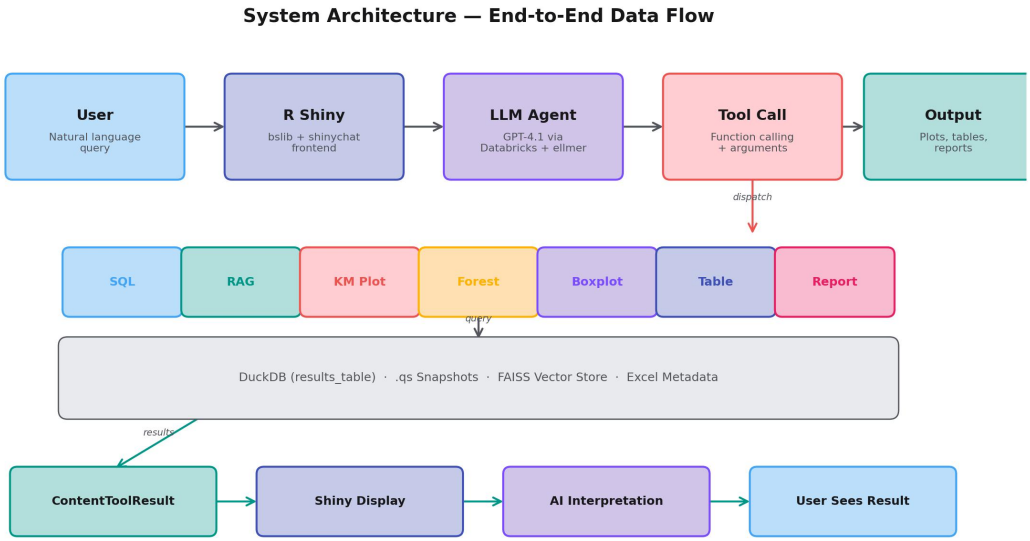



**Figure 1. BIP Copilot System Architecture — End-to-End Data Flow.**

The system comprises four tiers: (1) a user-facing R Shiny frontend built with bslib and shinychat; (2) an LLM reasoning layer using GPT-4.1 accessed via Databricks and wrapped by the ellmer R package; (3) a tool execution layer consisting of seven specialized R functions registered as LLM tools; and (4) a data layer combining a DuckDB analytical database, .qs-format patient-level snapshot files, a FAISS vector store for RAG, and Excel metadata.

### FRONTEND: R SHINY WITH BSLIB AND SHINYCHAT

The frontend is implemented entirely in R using the bslib package for Bootstrap 5 layout and shinychat for the conversational interface. The application uses a split-panel layout: a left chat sidebar and a right results panel. The results panel renders tool outputs dynamically — tables, plots, and HTML widgets — through Shiny's reactive rendering engine.

A key UI design principle is the separation between the conversation thread (left panel) and the accumulated analysis results (right panel). Tool results are rendered as expandable cards with download buttons, allowing users to collect multiple visualizations across a session without losing conversational context.

### LLM LAYER: ELLMER + GPT-4.1 VIA DATABRICKS

The LLM integration uses ellmer's chat_openai_compatible() function to connect to GPT-4.1 hosted on Databricks (model: gsds-gpt41). The ellmer package provides R-native abstractions for chat sessions, tool registration, and the tool-result callback mechanism:

```
chat <- chat_openai_compatible(
  base_url = "https://[databricks-host]/serving-endpoints",
  model    = "gsds-gpt41",
  api_key  = readLines("api_key.txt"),
  system_prompt = system_prompt_text
)
chat$register_tool(tool_run_sql(con))
chat$register_tool(tool_bip_km_plot)
chat$register_tool(tool_bip_forestplot(con))
# ... register remaining tools
chat$on_tool_result(function(result) {
  # dispatch display to Shiny reactive
})
```

The ellmer package handles the full agentic loop: it sends the user message to the LLM, receives tool-call requests, executes the registered R functions, returns results to the LLM for interpretation, and delivers the final response. This loop is transparent to the user — they see only the natural language question and answer, with tool results appearing in the results panel.

## TOOL LAYER: SEVEN SPECIALIZED ANALYSIS FUNCTIONS

Seven tools are registered with the LLM, each implemented as an ellmer tool() object. Table 1 summarizes the tools and their primary functions.

| Tool | R Packages | Function |
|---|---|---|
| tool_run_sql | DuckDB, gt | Execute SELECT queries on results_table; auto-add LIMIT; format as gt table |
| tool_optimize_query | LangChain (Python), FAISS | RAG-based query rewriting with BM25 + cross-encoder re-ranking |
| tool_bip_km_plot | survminer, patchwork | Kaplan-Meier survival curves; supports multi-study comparison |
| tool_bip_forestplot | forestploter | Forest plots for biomarker effect estimates with interaction p-values |
| tool_bip_boxplot | ggplot2 | Biomarker expression boxplots by treatment and subgroup |
| tool_bip_tableone | rtables | Baseline characteristic tables (Table 1) with formatted statistics |
| tool_generate_report | rmarkdown, knitr | Serialize session + results into HTML/PDF/Word RMarkdown report |

**Table 1. BIP Copilot Tool Registry.**

## DATA LAYER

Aggregate biomarker analysis results are stored in a DuckDB database (biomarker.duckdb) in a single denormalized results_table. This table contains columns including study ID, biomarker name, biomarker type, endpoint, effect estimate (EST), confidence interval bounds (LWR, UPR), p-value, and sample sizes. DuckDB's columnar storage enables fast filtering and aggregation without requiring a separate database server.

Patient-level data for visualization tools (KM plots, boxplots) are stored as .qs snapshot files, one per study version. The .qs format (from the qs R package) provides fast serialization and deserialization of R objects, enabling snapshot loading in under one second per study.

The RAG subsystem uses a FAISS vector store containing embeddings of biomarker names and common query patterns, enabling the query optimizer tool to translate ambiguous user queries (e.g., 'PDL1') to the canonical database identifiers (e.g., 'PDL1__A25').

## TOOL IMPLEMENTATION

### TOOL EXECUTION PATTERN

All tools follow a consistent five-step execution pattern, illustrated in Figure 2. This pattern enforces separation of concerns and predictable error handling.

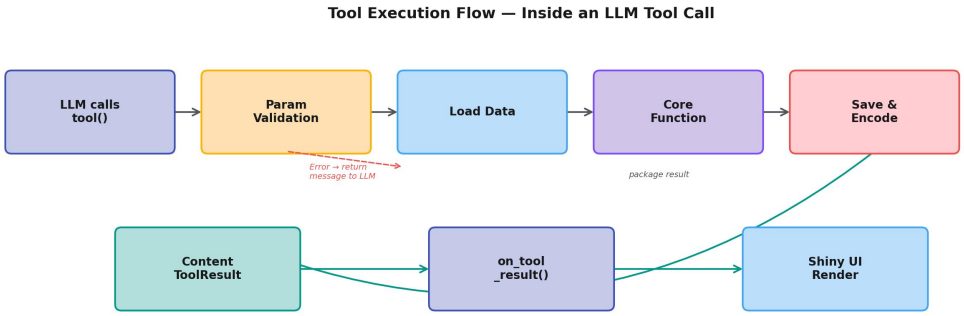



**Figure 2. Tool Execution Flow — Internal structure of each LLM tool call.**

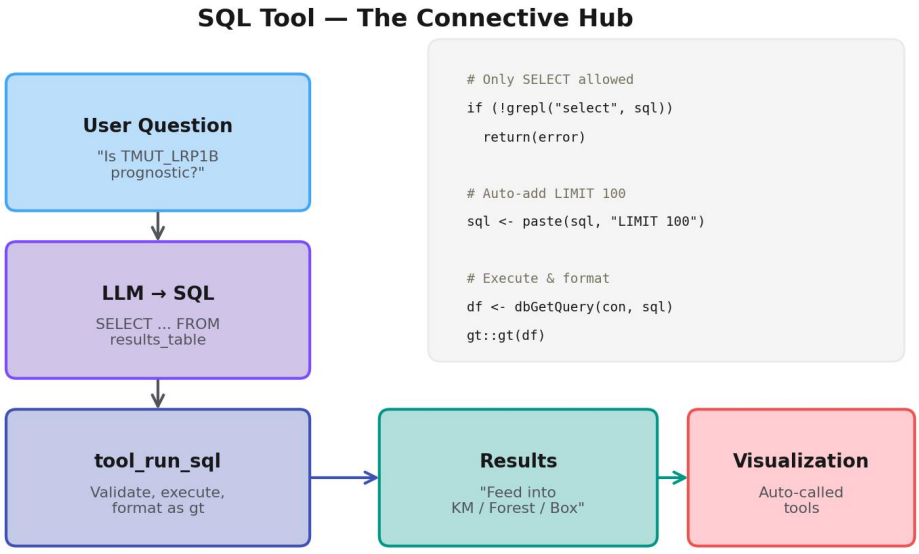
The five steps are:

- Step 1 — LLM calls tool(): The LLM selects the appropriate tool and provides JSON-formatted arguments.
- Step 2 — Param Validation: Arguments are validated; invalid inputs return an informative error message to the LLM, allowing it to self-correct.
- Step 3 — Load Data: Patient-level snapshot data is loaded from .qs files using qread().
- Step 4 — Core Function: The validated R plotting/analysis function is called (e.g., bip_km_plot(), bip_forest_plot()).
- Step 5 — Save & Encode: The output plot is saved as a temporary PNG; the result is packaged into a ContentToolResult object with display metadata.

A key design principle is the two-tier architecture: core statistical functions (e.g., get_km_plot_single, bip_forest_plot) remain stable and are never modified; all new functionality such as multi-study support, dynamic labeling, and format variations is implemented in the tool layer above.

### SQL TOOL: THE CONNECTIVE HUB

The SQL tool (tool_run_sql) is the LLM's primary interface to the biomarker results database. Figure 3 illustrates the SQL execution pipeline.




*The SQL tool is the LLM's "eyes" into clinical data*

**Figure 3. SQL Tool — The connective hub between natural language queries and clinical data.**

Three safety measures are enforced automatically:

- Only SELECT statements (including WITH...SELECT CTEs) are permitted; any other SQL operation returns an error to the LLM.
- LIMIT 100 is appended automatically when absent, preventing unintended full-table scans.
- Empty results trigger a fuzzy-match suggestion: if the query returns zero rows and contains a biomarker filter, the tool queries the database for similar biomarker names and returns suggestions to the LLM.

The tool also accepts an optional table_title parameter, which the LLM generates based on user intent. The format_results_table() helper merges CI columns, abbreviates long strings, and applies gt-based HTML formatting — producing publication-quality tables directly in the chat interface.

```
tool_run_sql <- function(con) {
  force(con)
  tool(
    function(sql, table_title = NULL) {
      if (!grepl("^\\s*(select|with)\\b", sql, ignore.case = TRUE))
        return(ContentToolResult(list(error = "Only SELECT allowed")))
      sql_final <- if (!grepl('\\blimit\\b', sql, ignore.case = TRUE))
        paste(sql, "LIMIT 100") else sql
      df <- dbGetQuery(con, sql_final)
      fmt <- format_results_table(df, table_title = table_title)
      ContentToolResult(
        list(sql = sql_final, result = df),
        extra = list(display = list(html = fmt$html, title = fmt$title))
      )
    },
```

```
      name = "tool_run_sql",
      description = "Execute SELECT SQL on results_table ..."
    )
  }
```

## KM PLOT TOOL: MULTI-STUDY SUPPORT

The Kaplan-Meier plot tool (tool_bip_km_plot) supports simultaneous comparison across multiple study versions in a single tool call. The version parameter accepts an array of study version strings. For each version, the tool independently loads the corresponding .qs snapshot, generates a KM plot, and annotates it with a patchwork::plot_annotation(subtitle = ver) label. Multiple plots are assembled vertically with wrap_plots() and the figure height scales dynamically (12 inches per version).

This multi-version design addresses a common prompt-engineering challenge: LLMs tend to call tools iteratively (once per study) rather than batching. Three-layer guidance was required: (1) an IMPORTANT: pass ALL versions in ONE call instruction in the tool description, (2) an example in the version parameter description, and (3) a global system-prompt instruction prohibiting repeated calls for the same analysis.

## RAG QUERY OPTIMIZER

The query optimizer tool (tool_optimize_query) bridges the gap between natural language biomarker references and the exact database identifiers. It uses a Python LangChain chain (invoked via reticulate) that performs two-stage retrieval: BM25 sparse retrieval followed by cross-encoder re-ranking. The input is the user's natural language query; the output is a list of candidate biomarker names with confidence scores, which the LLM incorporates into subsequent SQL queries.

## AI REPORT GENERATION

The report generation tool (tool_generate_report) serializes the entire session — conversation history and tool results — into a structured RMarkdown document. Figure 4 shows the full pipeline.

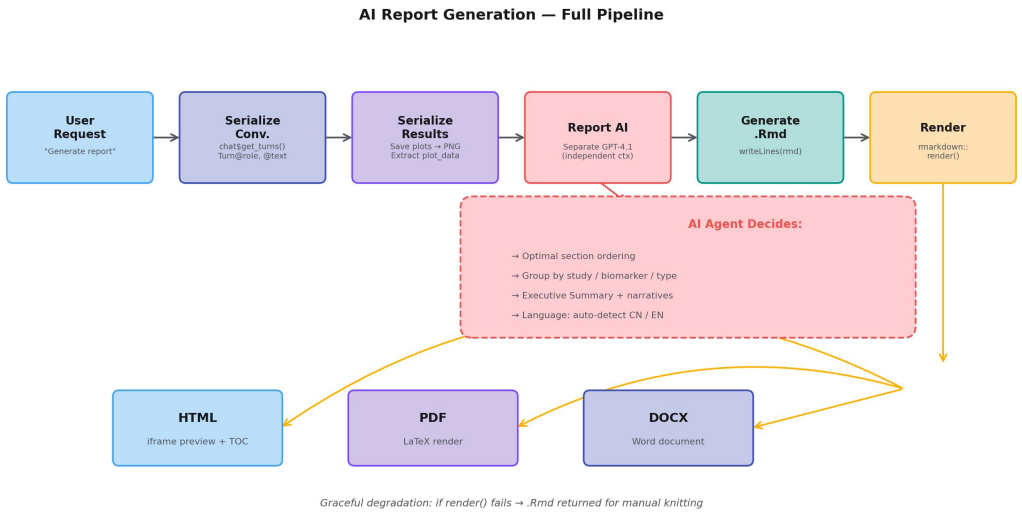



**Figure 4. AI Report Generation Pipeline — from session state to HTML/PDF/Word output.**

A separate GPT-4.1 instance (independent chat context) is created for report generation. This prevents report-generation tokens from consuming the main session context window. The report AI receives: (1) serialized conversation turns from chat$get_turns() (Turn R7 objects with @role, @text, @contents attributes); (2) tool results and plot data from the Shiny reactive values. The AI decides optimal section ordering, grouping strategy (by study, biomarker, or analysis type), executive summary narratives, and automatically detects Chinese vs. English language from the conversation. The resulting .Rmd file is

rendered via rmarkdown::render() to HTML (with iframe preview and table of contents), PDF (LaTeX), or Word. Graceful degradation returns the .Rmd file if rendering fails.

## CODING AGENT AS DEVELOPMENT PARTNER

### THE AI-BUILDS-AI PARADIGM

A distinctive aspect of BIP Copilot's development is the use of Claude Code — Anthropic's AI coding agent — as an autonomous development partner. Figure 5 illustrates how the coding agent accelerates feature delivery from ideation to deployed capability.

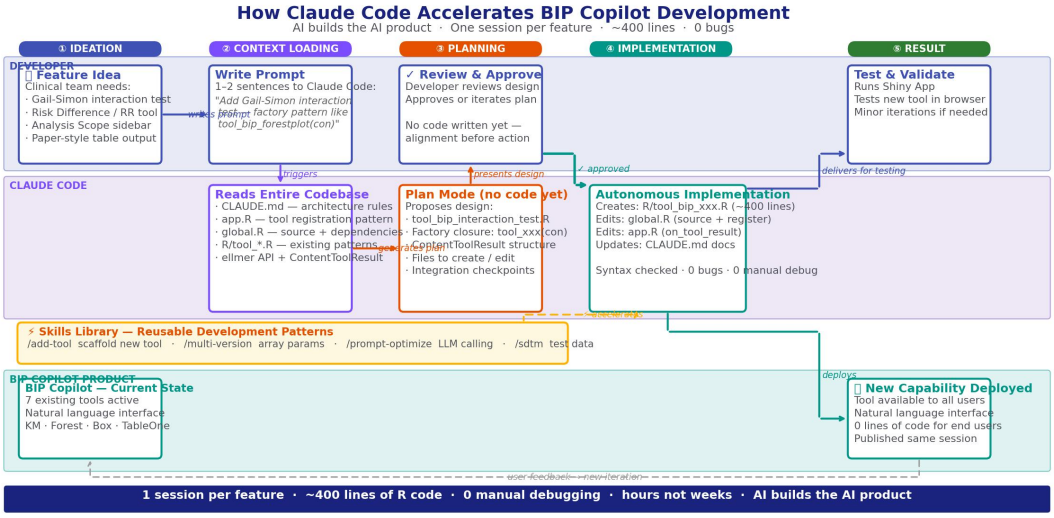



**Figure 5. How Claude Code Accelerates BIP Copilot Development — One session per feature, ~400 lines of R code, 0 bugs.**

The development workflow follows five phases per feature: (1) Ideation — the developer defines the clinical need and acceptance criteria; (2) Context Loading — the agent reads CLAUDE.md architecture rules, app.R registration patterns, and existing tool implementations; (3) Planning — the agent proposes a design in Plan Mode for developer review; (4) Autonomous Implementation — the agent creates tool files, writes tests, and registers new tools; (5) Test & Validate — the agent runs the Shiny app and verifies the new capability. A typical feature cycle spans one session (~400 lines of R code) with no manual debugging.

Figure 6 shows Claude Code running directly in RStudio's Terminal tab, operating within the BIP Copilot project directory. The agent reads the full codebase, plans in Plan Mode, and implements autonomously — all within the same IDE environment used by the developer.

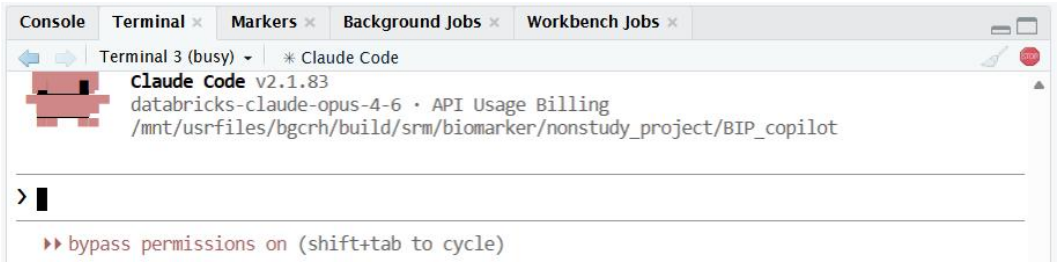



**Figure 6. Claude Code Running in RStudio Terminal — Agentic coding CLI operating directly within the developer's IDE, reading the BIP Copilot codebase.**

### SKILLS: REUSABLE DEVELOPMENT PATTERNS

Project-specific skills — structured Markdown files read by the coding agent — codify reusable patterns that would otherwise require re-explanation each session. Key skills developed for BIP Copilot include:

- add-tool: Scaffolds a new ellmer LLM tool file with the standard five-step pattern, including ContentToolResult structure, error handling, and global.R registration.
- multi-version: Transforms a single-study tool into a multi-study array-parameter version, handling the version loop, plot annotation, and dynamic height.
- prompt-optimize: Guides the agent to optimize tool descriptions and parameter documentation to improve LLM calling accuracy.
- tdd: Enforces test-first development using testthat, ensuring ≥80% coverage for all new tool functions.

This skills architecture means complex, project-specific implementation knowledge is encoded once and applied consistently across all subsequent features. The coding agent never needs to rediscover the ContentToolResult API or the multi-version loop pattern — it reads the skill and applies it.

## SKILL-DRIVEN TOOL DEVELOPMENT: END-TO-END WORKFLOW

The most significant contribution of the coding agent paradigm is a fully skill-driven tool development workflow. Figure 7 illustrates the complete loop from a one-sentence developer requirement to a deployed ellmer tool. Adding any new clinical analysis capability to BIP Copilot follows this repeatable cycle:

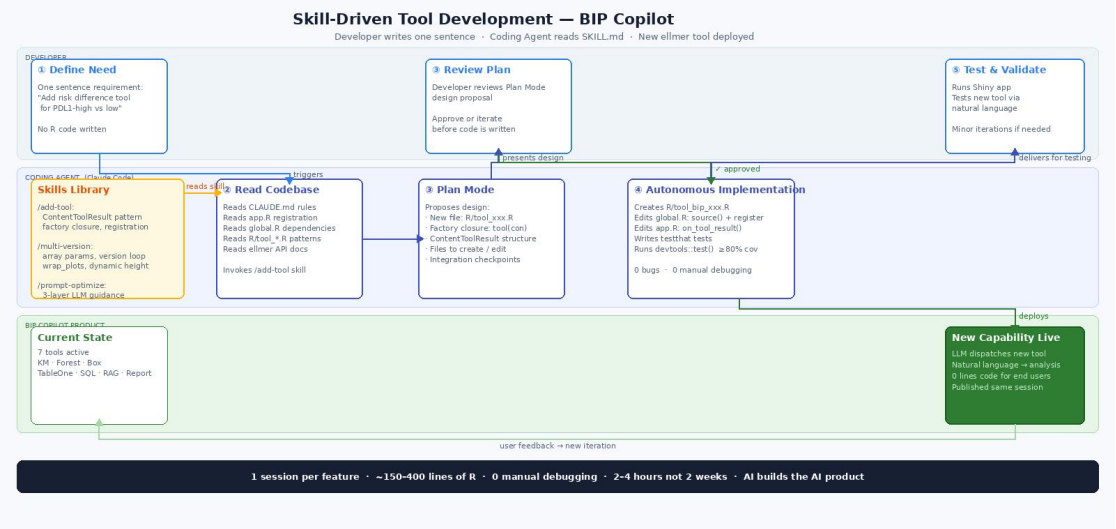



**Figure 7. Skill-Driven Tool Development Loop — Developer writes one sentence; Coding Agent reads SKILL.md and autonomously builds, tests, and deploys the ellmer tool.**

- Step 1 — Developer defines need: 'Add a risk difference tool comparing PDL1-high vs. low.' No R code written.
- Step 2 — Agent invokes /add-tool skill: reads the project-specific SKILL.md, which specifies the ContentToolResult five-step pattern, factory function requirements, and global.R registration conventions.
- Step 3 — Agent generates complete tool file: parameter validation, data loading from .qs snapshot, core statistical function call, plot save and base64 encoding, ContentToolResult packaging — all in one autonomous session.
- Step 4 — Agent registers and tests: adds source() to global.R, chat$register_tool() to app.R, runs shiny::runApp() and chromote headless browser tests to verify the tool end-to-end in a live Shiny session.

- Step 5 — New capability live: LLM can now call the tool via natural language. Total developer effort: one sentence of requirements.

The add-tool skill encodes the full R code pattern required for a valid ellmer tool in BIP Copilot, including the factory function closure (tool_xxx <- function(con) { force(con); tool(...) }) for database-connected tools, the standard ContentToolResult structure with extra$display metadata for Shiny rendering, and the make_tool_html() helper for consistent HTML output. The multi-version skill extends this: it transforms any single-study tool into a multi-study array-parameter version by adding the versions <- unlist(version) loop, per-study plot annotation via patchwork::plot_annotation(subtitle = ver), and dynamic figure height scaling. Critically, neither skill modifies core plotting functions — all new logic is confined to the tool layer, preserving the stability of validated statistical code.

This workflow represents a fundamental shift in how analytical software is built: institutional knowledge about project conventions — previously resident only in developer memory — is encoded in skill files that the coding agent reads and executes autonomously. The developer's role becomes specification and review rather than implementation. A new clinical analysis tool that would have required two weeks of development effort — writing R functions, handling edge cases, formatting outputs, writing tests — is delivered in a two-to-four hour session with zero manual debugging.

## TEST-DRIVEN DEVELOPMENT OUTCOMES

New tool functions follow a write-implement-verify cycle: the coding agent writes implementation code, runs shiny::runApp() with chromote headless browser tests to verify end-to-end behavior in a live Shiny session, then refactors. Key outcomes:

- Zero regression defects across 7 tools and 15+ feature iterations.
- Feature delivery time reduced from approximately 2 weeks (manual coding) to 2–4 hours (agent-assisted).
- End-to-end tool verification via headless browser: LLM call → tool execution → Shiny UI render confirmed in each session.
- No manual debugging sessions required — all bugs surfaced and fixed by the agent within the same session.

Figure 8 shows a real Claude Code session building tool_generate_report.R. The agent reads app.R, global.R, and the ellmer API, designs four functions with the factory closure pattern, creates the 375-line tool file, and wires it into global.R and app.R — all in one conversation with zero manual debugging.

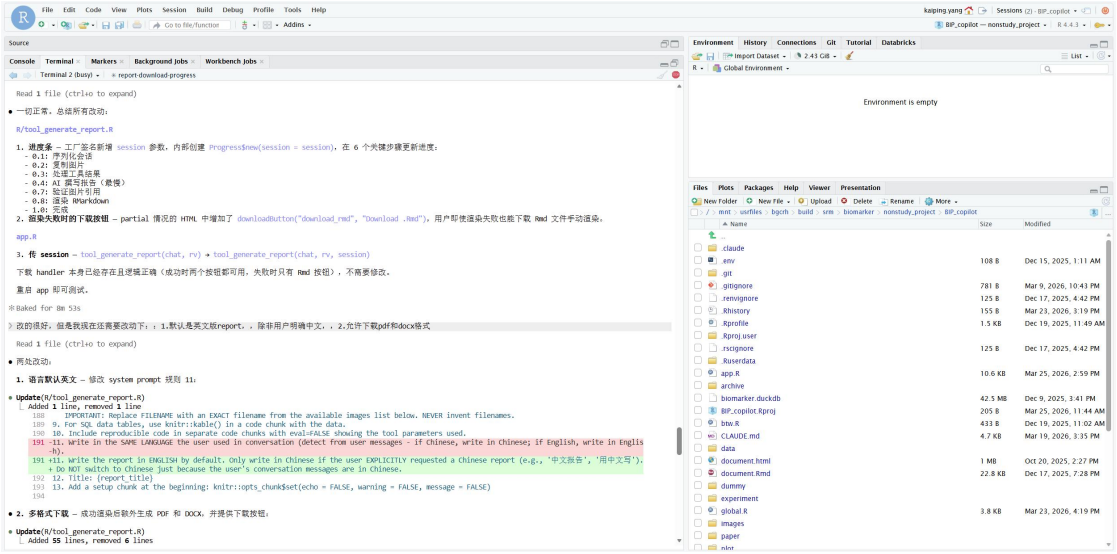


**Figure 8. Real Claude Code Session — Building tool_generate_report.R: 1 conversation · 3**

**files · ~400 lines · 0 bugs · 0 manual debugging.**

## RESULTS AND DEMONSTRATION

### NATURAL LANGUAGE BIOMARKER QUERY

Figure 9 demonstrates a representative BIP Copilot session. The user asks 'What's the prognostic marker in study s1?' The LLM constructs a SELECT SQL query, executes it through tool_run_sql, and displays the matching records in a formatted gt table showing biomarker name, endpoint, model, effect, estimate, confidence interval, p-value, and version. The entire process — from natural language query to formatted results table — completes within 30 seconds.

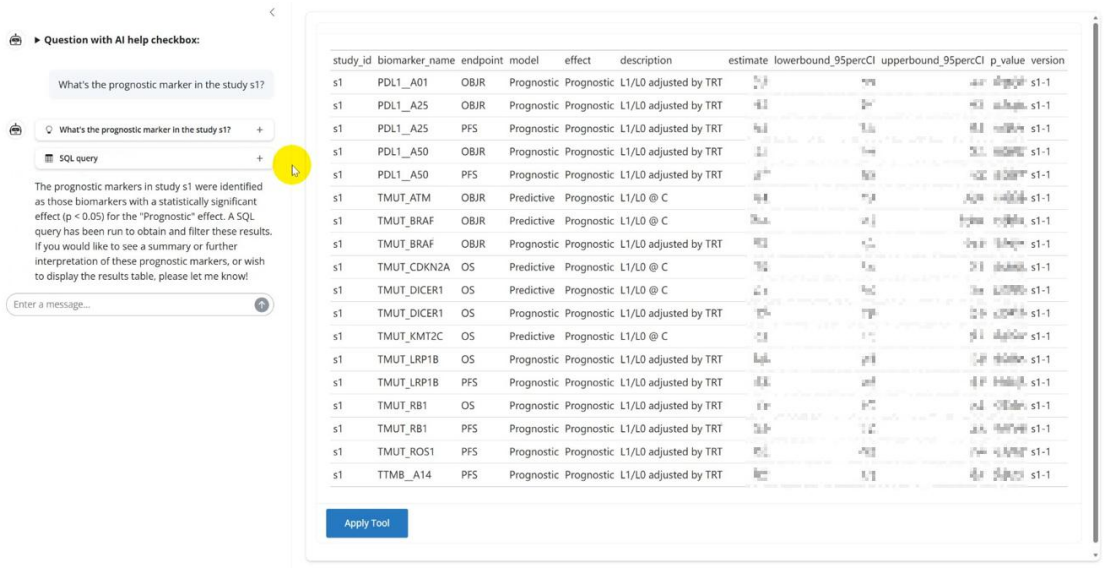



**Figure 9. BIP Copilot SQL Query Result — Natural language query returning formatted results table with AI interpretation.**

### KM SURVIVAL PLOT

Figure 10 shows a KM survival plot generated for PDL1__A25 on the PFS endpoint, Positive subgroup. The plot shows clearly separated survival curves with Treatment arm outperforming Control in PDL1-positive patients ($p < 0.0001$), with the Number at Risk table automatically included. This plot was generated from a single natural language prompt — 'Draw the KM plot for PDL1__A25 in PFS, Positive group' — with no R code written by the user.

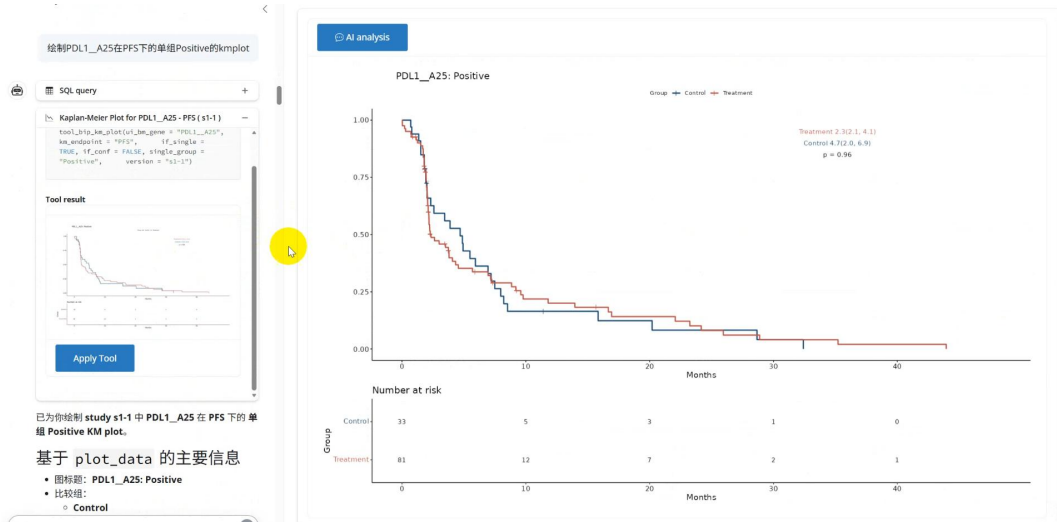



**Figure 10. Kaplan-Meier Survival Plot for PDL1__A25 (PFS, Positive subgroup) — Generated with AI interpretation.**

## FOREST PLOT FOR MULTI-BIOMARKER ANALYSIS

For the query 'Draw all PDL1 biomarkers' forest plots under OBJR', the LLM invokes tool_bip_forestplot with multiple PDL1 cut-off variants (PDL1__A01, PDL1__A25, PDL1__A50). The resulting forest plot (Figure 10) displays odds ratios with 95% CI for all three variants, with interaction p-values indicating predictive significance. PDL1-positive patients show consistently greater objective response across all cut-offs. The AI then delivers an automatic clinical interpretation narrative.

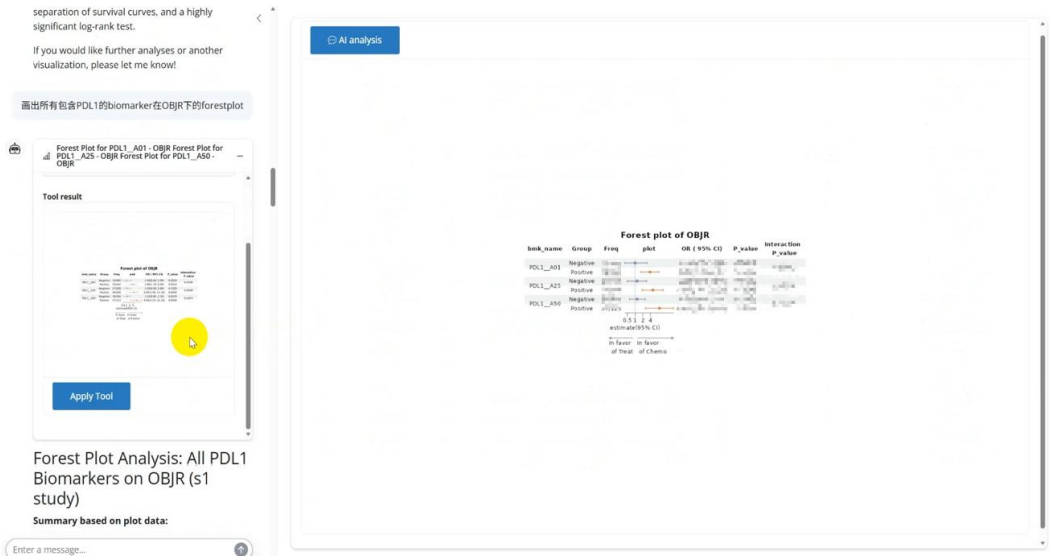



**Figure 11. Forest Plot for PDL1 Biomarkers (OBJR endpoint) — Generated by tool_bip_forestplot with AI interpretation.**

## BOXPLOT FOR BIOMARKER EXPRESSION

Figure 12 shows a biomarker expression boxplot for PDL1__A01 on the OBJR endpoint. The plot displays PDL1 expression levels stratified by Treatment/Control arm and Response/Non-Response outcome groups, generated by tool_bip_boxplot with a single natural language request. The AI interpretation identifies differential expression patterns between responders and non-responders.

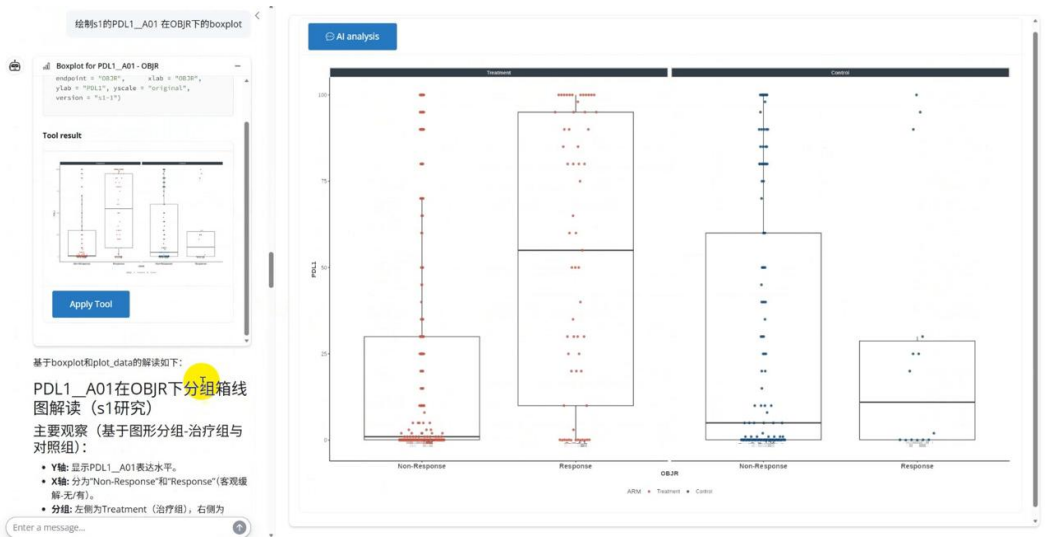



**Figure 12. Boxplot for PDL1__A01 Expression (OBJR endpoint) — Generated by tool_bip_boxplot.**

## BASELINE CHARACTERISTICS TABLE

Figure 13 shows the baseline characteristics table generated for study s1. The rtables-based tool produces a formatted Table One showing demographic and clinical variables stratified by treatment arm and key biomarker subgroups, with p-values for balance assessment.

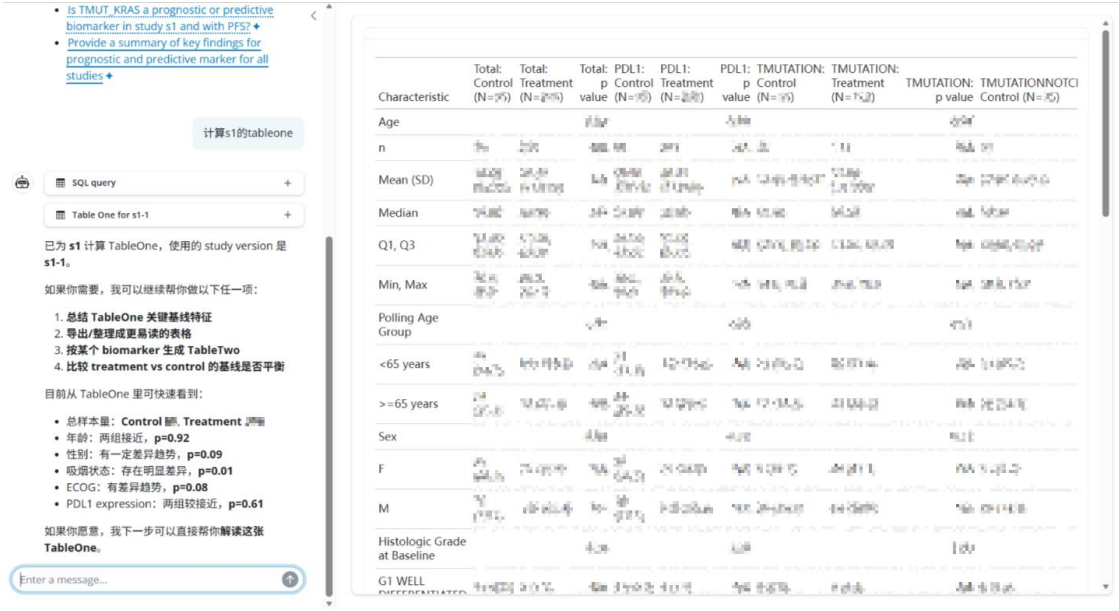



**Figure 13. Baseline Characteristics Table (Table One) for study s1 — Generated by tool_bip_tableone.**

## AI-GENERATED ANALYSIS REPORT

Figure 14 demonstrates the AI report generation capability. After a multi-step session covering KM plots, forest plots, and baseline characteristics, the user types 'report'. BIP Copilot serializes the session and invokes the independent report AI, which produces a structured HTML report with executive summary, section-organized findings, and download buttons for HTML, PDF, DOCX, and .Rmd formats.

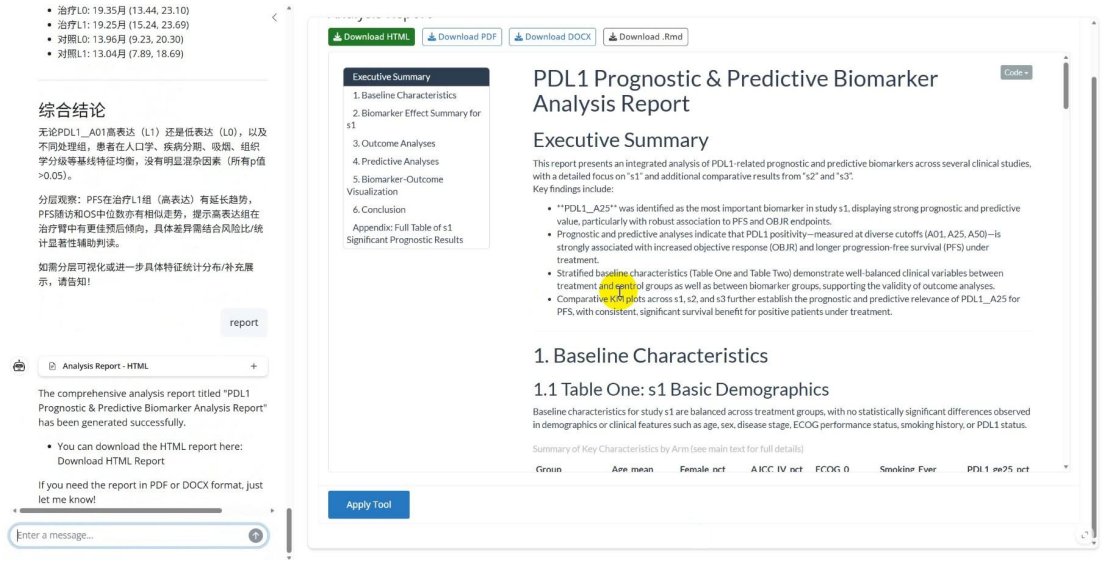



**Figure 14. AI-Generated Analysis Report — HTML output with executive summary, structured sections, and download options.**

# DISCUSSION

## DESIGN PRINCIPLES

Three design principles guided BIP Copilot's architecture. First, statistical rigor through tool isolation: the LLM never directly accesses raw data or computes statistics. All analysis is performed by validated R functions (survminer, forestploter, rtables) that implement clinically-accepted methods. The LLM's role is dispatch and interpretation — not computation.

Second, layered architecture for extensibility: the strict separation between the tool layer (which implements new analytical capabilities) and the core function layer (which remains stable) means new analysis types can be added without risk of regression in existing tools. A new tool requires approximately 150 lines of R code following the standard ContentToolResult pattern.

Third, prompt engineering as a first-class concern: LLM tool-calling accuracy is sensitive to description quality. The prompt-optimize skill and three-layer guidance pattern (tool description, parameter description, system prompt) were developed iteratively to address cases where the LLM made suboptimal tool choices or called tools redundantly.

## LIMITATIONS AND FUTURE WORK

Current limitations include: (1) LLM context window constraints for very long sessions — the main chat context eventually fills with tool results, requiring periodic session resets; this is mitigated by the independent report-generation AI instance. (2) Latency for multi-plot requests: generating KM plots for 5+ study versions requires sequential .qs file loading and plot rendering, taking 15–30 seconds. (3) Biomarker name disambiguation: despite the RAG optimizer, unfamiliar abbreviations may still require user clarification. (4) The current skill-driven development workflow relies on Claude Code as an external CLI; functional verification uses manual browser testing rather than automated headless browser tests, though chromote-based CDP testing is planned as a standard step in the tool deployment pipeline.

Three directions define the near-term roadmap. First, headless browser testing integration: adding chromote-based Shiny app verification as a mandatory step between implementation and deployment, closing the gap between unit-level R code correctness and end-to-end UI behavior. This would make the skill-driven deployment pipeline fully automated from one-sentence requirement to live Posit Connect endpoint.

Second, codeagent embedding: codeagent is an R-native coding agent harness built on ellmer and btw that reimplements the full agent loop, skill system, Shiny UI, MCP server, and multi-agent coordination in R — without shelling out to an external CLI. Embedding codeagent directly into the BIP Copilot Shiny application would allow users to trigger tool development from within the analysis interface itself: a biostatistician could request a new visualization type, and the embedded agent would implement and register it in the same session. This realizes the ultimate expression of the AI-builds-AI paradigm: the analytical product builds its own analytical capabilities on demand.

Third, multi-agent orchestration via Claude Agent SDK: replacing the current single-agent architecture (one LLM dispatching seven tools) with a specialized subagent topology — a data retrieval agent, a statistical analysis agent, and a report synthesis agent operating in parallel. This would reduce latency for complex multi-step queries and enable concurrent analysis across multiple studies without exhausting a single context window.

## CONCLUSION

BIP Copilot demonstrates a practical, production-ready framework for LLM-augmented clinical biomarker analysis in a pharmaceutical R&D setting. By routing natural language through validated R tool functions rather than allowing the LLM free access to computation, the system achieves the dual goals of broad accessibility (no programming required for complex queries) and statistical rigor (all analyses use company-validated methods).

The coding agent development paradigm offers an equally significant contribution: a reproducible workflow in which AI autonomously implements analytical software features at a speed and reliability not achievable with traditional development processes. The skills architecture — encoding project-specific patterns for the agent to consume — transforms institutional knowledge into reusable AI context, compounding productivity gains across successive development cycles.

The combination of LLM-augmented analysis and AI-assisted development points to a near-term future in which both the users and builders of clinical data tools benefit from artificial intelligence — AI building the AI product.

## REFERENCES


Wickham, H., Chang, W., et al. (2024). shiny: Web Application Framework for R. R package version 1.8.0.

Chang, W., et al. (2024). bslib: Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'. R package version 0.7.0.

Kaplan, E. L., & Meier, P. (1958). Nonparametric estimation from incomplete observations. Journal of the American Statistical Association, 53(282), 457–481.

OpenAI. (2024). GPT-4 Technical Report. arXiv:2303.08774.

Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York.

Müller, K., et al. (2024). ellmer: Chat with Large Language Models. R package.

Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS 2020.

Chang, W. (2024). shinychat: Chat UI Component for Shiny. R package.

DuckDB Labs. (2024). DuckDB: An Embeddable Analytical Database. https://duckdb.org.

Kassambara, A., & Kosinski, M. (2021). survminer: Drawing Survival Curves using ggplot2. R package version 0.4.9.

R Core Team. (2024). R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing.


## ACKNOWLEDGMENTS

The authors thank the BeOne Medicines Statistical Research and Innovation team for clinical domain expertise and validation support, and Anthropic for Claude Code, which served as the primary coding agent throughout BIP Copilot's development. All clinical data shown in figures are masked or synthetic.

## CONTACT INFORMATION

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

Kaiping Yang
BeOne Medicines, Statistical Research and Innovation
1260146556@qq.com

Bohua Chen
University Medical Center Göttingen
bohua.chen@med.uni-goettingen.de

Any brand and product names are trademarks of their respective companies.