PharmaSUG China 2026 - Paper AI201

Stop Building, Start Prompting: Leveraging LLM-Native Capabilities for TFL Automation with Minimal Scaffolding

Zhongqian Xu, Evopoint Biosciences Co., Ltd.

ABSTRACT

Clinical TFL generation traditionally requires substantial study-specific engineering, including Word-shell parsing, variable mapping, statistical programming, and output formatting. This paper presents ADaM2TFL, an exploratory design and working prototype in which a compact Markdown "skill file" provides an LLM coding agent with project context: directory layout, shell conventions, statistical rules, formatting constraints, and known failure patterns. Reusable deterministic utilities support shell normalization, metadata caching, and structural validation, while the agent interprets these artifacts and generates candidate R programs and RTF outputs. The objective is not to eliminate software, but to minimize per-study custom scaffolding. A five-phase workflow combines configuration, shell confirmation, metadata caching, code generation, and verification, with human review retained at key decision points. The design also includes an experimental knowledge-accumulation loop and an exploratory shell-free mode; both are target capabilities rather than validated production features. Experience with the prototype suggests that many failures can be addressed by improving the governing context instead of adding study-specific code. The central proposition is therefore that, for LLM-native clinical reporting, part of the engineering bottleneck shifts from building automation infrastructure to authoring precise, reviewable context.

INTRODUCTION

Tables, Figures, and Listings (TFLs) are core deliverables of clinical trial reporting. Despite decades of automation through SAS macros, open-source R packages and commercial metadata and validation platforms, study-specific TFL production remains labor-intensive. Teams must still interpret Word shells, map requirements to ADaM variables, implement statistical logic, format outputs, and validate the result. Much of this work recurs across studies even when the underlying reporting patterns are similar.

Large language models (LLMs) with long-context windows, code-generation capabilities, and tool use create a different design opportunity. Rather than encoding every project convention in a bespoke framework, an LLM agent can be given reviewed context and allowed to navigate project artifacts, inspect data summaries, interpret normalized shell metadata, and compose candidate statistical programs. The practical question is therefore not whether an LLM can produce R code, but what combination of context, deterministic utilities, human review, and validation is sufficient to make that process useful and inspectable.

This paper presents ADaM2TFL, a skill-file approach for a long-context LLM coding agent. The skill file supplies workflow instructions, directory conventions, statistical rules, formatting constraints, and known failure patterns, supported by a small collection of reusable utilities and reference documents. We describe the architecture, workflow, and design patterns that emerged from applying the prototype to clinical-reporting tasks. The contribution is an exploratory design case study, not a validated production system or a comparative benchmark. Specific timing values are included only to illustrate the role of metadata caching; formal evaluation of accuracy, reproducibility, and operating performance remains future work.

THE SKILL FILE AS A MINIMAL HARNESS

The central insight is that an LLM agent needs a strong contextual contract and a small set of deterministic tools, rather than a large study-specific automation platform. The skill file describes what varies by project: file locations, naming conventions, domain rules, review gates, and output constraints. Reusable capabilities handle common operations such as reading sas7bdat files, normalizing Word-shell content, checking cache freshness, and writing R code with the reporter package. This design does not remove all software; it reduces the amount of custom software that must be created for each study.

ARCHITECTURE

Figure 1. Architecture of the Skill File Harness

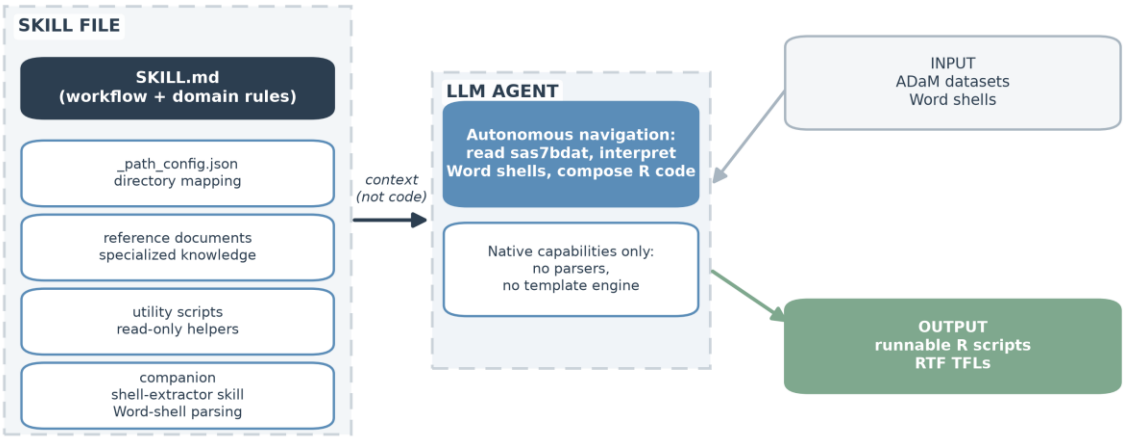


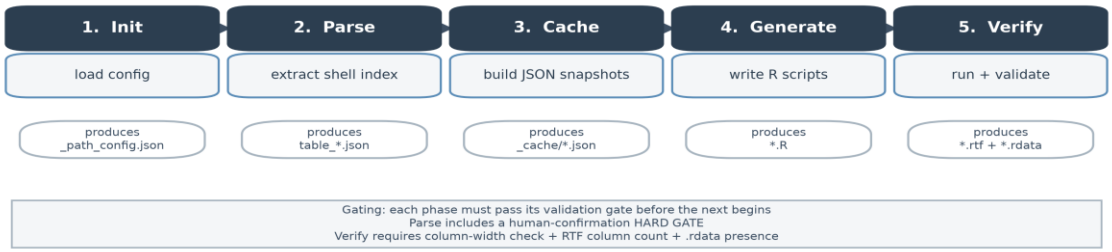
Figure 1 summarizes the architecture. The SKILL FILE zone contains the core SKILL.md, which defines workflow orchestration and domain rules, together with four supporting layers: _path_config.json for directory mapping, reference documents for specialized knowledge, deterministic utility scripts for derived artifacts and validation, and a companion shell-extractor skill that converts Word shells into normalized metadata. These resources are supplied to the LLM AGENT as context and tools. The agent then explores ADaM metadata, interprets the confirmed shell structure, generates candidate R scripts, and runs validation steps. The architecture therefore shifts most study-specific logic into reviewable context while retaining reusable code where deterministic behavior is valuable.

Two design choices are particularly important. First, progressive disclosure keeps the main skill concise: it provides enough information to start, while detailed references are opened only when required, such as layout guidance for a multi-column listing or domain guidance for variables that contain delimiter-separated composite values. Second, configuration over convention makes the workflow portable: _path_config.json defines project-specific directories and is reviewed before downstream work begins. Together, these choices limit prompt size without hiding project decisions.

WORKFLOW ORCHESTRATION

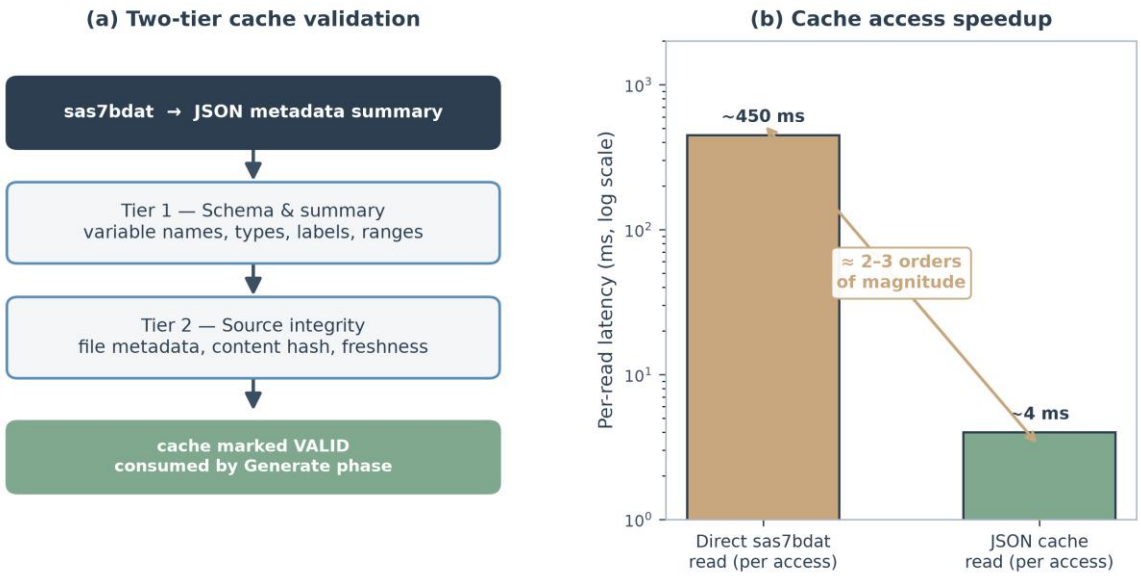
The skill defines a five-phase pipeline (Figure 2). Init loads and confirms the project configuration. Parse extracts shell metadata and requires human confirmation. Cache creates variable-level JSON metadata summaries and checks whether they remain consistent with the source files. Generate selects single-table or batch mode and writes candidate R scripts using the reporter package. Verify executes the scripts and performs structural output checks. Each phase defines input preconditions and validation criteria, and a failed criterion returns the workflow to the relevant phase. The Parse confirmation is intentionally retained as a hard human gate.

Figure 2. Five-Phase TFL Generation Pipeline



Three mechanisms merit emphasis. First, cache-first metadata access reduces repeated exploratory reads. The cache records variable names, types, labels, cardinality, representative values, numeric ranges, and missing-value counts, allowing the agent to inspect a compact JSON summary rather than repeatedly opening the source sas7bdat file. The target validation design in Figure 3 combines schema and summary checks with source integrity and freshness checks; the current prototype implements freshness checks using file metadata and content hashes. In a representative development measurement, a cached metadata lookup took approximately 4 ms compared with approximately 450 ms to reopen the SAS file. These values illustrate the expected order of magnitude and are not presented as a general benchmark. Second, single-pass data exploration scans the variables needed for a target TFL together, including character values, numeric ranges, labels, and missingness, rather than querying variables one at a time. Third, batch mode parameterizes filter conditions when several TFLs share the same structure, while confirmed shell metadata remains the source for titles and footnotes.

Figure 3. Two-Tier Cache Validation and Performance Impact



DESIGN PATTERNS FOR LLM-NATIVE AUTOMATION

Several reusable patterns emerged while developing the ADaM2TFL prototype (Table 1). The most important is to express domain and layout constraints in reviewed skill text and to pair them with deterministic checks where feasible. This makes the governing rules visible to reviewers while avoiding the assumption that instruction text alone guarantees compliance. A second pattern is to document recurring failures by symptom, likely cause, and resolution so that the agent can use prior experience during regeneration. Together, these patterns treat context as a maintained engineering artifact rather than an informal prompt.

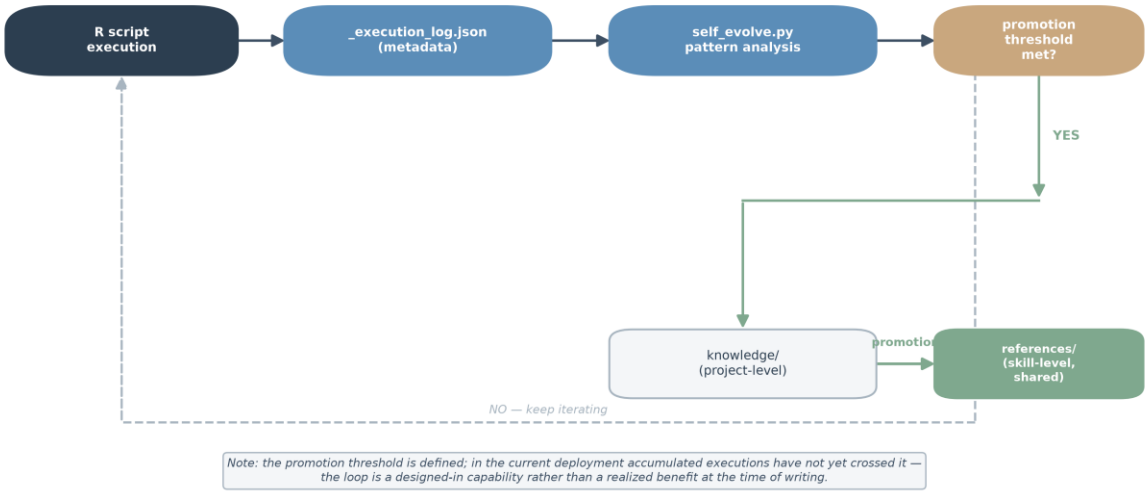
Table 1. Design Patterns for LLM-Native TFL Automation

Pattern	Description	ADaM2TFL Example
Safety Rules as Context	Express domain and layout constraints in reviewed skill text, and pair them with deterministic checks where feasible.	Composite-value matching guidance; automated column-width checks.
Failure Documentation	Document recurring symptoms, likely causes, and resolutions so the agent can use prior experience during regeneration.	Catalog of percent-escaping, footnote, file-lock, encoding, and column-overflow failures.
Knowledge Accumulation	Aggregate execution metadata and surface recurring patterns as promotion candidates for human review.	Experimental analysis of composite variables, recurring filters, and error-fix pairs.
Progressive Disclosure	Keep essential workflow context in the main skill and open detailed references only when required.	A concise SKILL.md supported by focused cache, layout, API, error, and batch references.
Configuration over Convention	Use explicit, reviewed configuration so the same skill can be applied across projects without editing its core instructions.	_path_config.json defines and confirms project-specific directories.
Companion Skill Integration	Delegate specialized document normalization to a reusable companion skill and consume validated artifacts downstream.	The shell-extractor skill converts Word shells into a confirmed, normalized index.

KNOWLEDGE ACCUMULATION AND SELF-EVOLUTION

The target architecture includes an experimental knowledge-accumulation loop (Figure 4). Executions can record structured metadata such as filters, warnings, failures, and cache behavior. An analysis utility aggregates these records across four areas: composite-value variables, recurring filter conditions, error-fix pairs, and domain-to-TFL mappings. When a pattern meets a defined recurrence threshold, the tool can surface it as a promotion candidate. A human reviewer then decides whether the pattern is sufficiently general and well supported to be incorporated into shared skill references.

Figure 4. Self-Evolution Loop



The maturity of this loop is limited. Pattern analysis and promotion recommendations are implemented, but automatic execution-log capture and reference updates are not yet fully integrated into the main workflow. In addition, accumulated executions in the current deployment have not crossed the promotion threshold. Figure 4 should therefore be read as the target operating model for a human-reviewed learning loop, not as evidence of an autonomous system that has already improved itself in production.

DISCUSSION

ADaM2TFL rebalances traditional automation. Reusable parsers, cache builders, templates, and validators remain, but the study-specific layer is expressed primarily through a structured context document rather than a new custom framework. This separation makes project rules easier to review and allows the same core tools to be reused across studies. The central claim is thus not that prompting replaces software, but that well-designed context can replace a meaningful portion of repetitive study-level engineering.

An exploratory extension is shell-free generation. Once the agent has enough domain context, it can draft a TFL program from a natural-language request. The governed five-phase workflow, however, still treats confirmed shell metadata as the default specification and review gate. Shell-free generation should therefore be understood as a promising authoring mode for prototypes or clearly specified requests, not as a general replacement for approved shells in production reporting.

Self-review mechanisms improve traceability but do not establish statistical validity by themselves. The design uses three layers. First, shell-derived metadata is not consumed downstream until a reviewer confirms the title, population, structure, and footnotes. Second, development logs and execution records provide an audit trail of the variables, filters, and decisions used to create an output. Third, the Verify phase checks script execution, output presence, and structural characteristics such as column geometry; planned extensions include explicit denominator, missing-value, and content checks. These safeguards support review and debugging, but they do not replace independent statistical QC or study-specific validation.

Several limitations should temper expectations. Results depend on the capabilities and version of the underlying LLM, and instruction text reduces but does not eliminate model sensitivity or nondeterminism. Complex Word layouts can still challenge shell extraction and interpretation. Current validation is primarily structural rather than a formal proof of statistical correctness, and the approach has not yet been evaluated through a systematic multi-study benchmark. The knowledge-accumulation loop remains experimental. Operational topics such as model governance, privacy, validated environments, and independent QC must also be addressed before production adoption.

Future work will expand coverage to complex figures and derived endpoints, integrate execution logging with the knowledge loop, strengthen deterministic QC checks, and define a governed path for shell-free requests. A systematic evaluation should measure code correctness, output fidelity, reviewer effort, reproducibility across model versions, and failure modes across multiple studies. These steps are necessary to determine where the approach is reliable enough for validated clinical-reporting workflows.

CONCLUSION

The ADaM2TFL prototype shows how a concise Markdown skill file, supported by reference documents and reusable deterministic utilities, can guide an LLM agent through clinical TFL programming. The approach reduces the need for per-study custom parsers, bespoke template engines, and orchestration code without claiming to eliminate software or human review. Its principal design patterns - reviewed rules as context, progressive disclosure, configuration over convention, companion-skill integration, staged validation, and human-reviewed knowledge accumulation - are applicable beyond a single TFL workflow. The remaining challenge is to describe requirements precisely, connect them to reliable checks, and validate the resulting system at the standard required for clinical reporting.

REFERENCES

[1] Bosak, D. “reporter: Creates Statistical Reports.” R package version 1.4.4, 2024. Available at https://cran.r-project.org/package=reporter.

[2] Wickham, H. et al. “Welcome to the Tidyverse.” Journal of Open Source Software, 4(43):1686, 2019.

[3] Brown, T. et al. “Language Models are Few-Shot Learners.” Advances in Neural Information Processing Systems, 33:1877–1901, 2020.

[4] Anthropic. “Introducing Claude’s Extended Thinking.” Anthropic Blog, 2024. Available at https://www.anthropic.com/news/extended-thinking.

[5] Anthropic. “Tool Use (Function Calling) with Claude.” Anthropic Documentation, 2024. Available at https://docs.anthropic.com/en/docs/build-with-claude/tool-use.

[6] OpenAI. “Codex: A Collaborative AI Coding Agent.” OpenAI Documentation, 2025. Available at https://openai.com/codex.

CONTACT INFORMATION

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

Zhongqian Xu

Evopoint Biosciences Co., Ltd.

zhongqian.xu@evopointbio.com