PharmaSUG China 2026 - Paper AP169-2026

One Pipeline, Zero First Drafts: A Metadata-and-LLM Toolkit for Automated TFL Program Generation

Hefan Jing, Peilin Liu

Jiangsu Hengrui Pharmaceuticals Co., Ltd.

ABSTRACT

In clinical trial programming, producing Tables, Figures, and Listings (TFLs) typically begins with a manual, labor-intensive step: reading a Table of Contents (TOC), selecting the appropriate system macro for each output, and filling in the correct macro parameters. This paper presents generate_pdt, a Python-based pipeline that automates this "first-draft" process. The core idea is to decompose each TOC entry into three structured components — analysis type, filter conditions, and input dataset — then match these components to the appropriate system macro using a three-level strategy: exact matching, synonym matching, and semantic matching via a large language model (LLM). The tool generates a complete Program Deliverables Tracker (PDT) and ready-to-run SAS programs without any manual intervention. We describe the decomposition logic, the knowledge base structure, the three matching methods and their respective strengths and limitations, and discuss the tool's current gaps and future directions.

INTRODUCTION

The TFL production workflow in pharmaceutical clinical trials typically follows a well-established pattern: a Statistical Analysis Plan (SAP) defines what outputs to produce, a TOC lists them in a spreadsheet, and programmers select the appropriate system macro for each output and fill in the required parameters — whether in a tracking spreadsheet, a configuration file, or directly in the program call. The final step is writing the SAS programs themselves.

Most pharmaceutical companies maintain mature system macro libraries that handle the actual computation and formatting. The macros themselves are not the problem. The real problem is that **using system macros relies on manual decision-making**: for every output in the TOC, the programmer must select the right macro, identify which parameters are required, determine the allowed values for each parameter, and assemble the full parameter set correctly based on the output's title, section, and context. With 150+ outputs per study, this manual decision model leads to several concrete problems:

- **High learning cost** — for new hires, mastering dozens of system macros with hundreds of parameter combinations requires a long ramp-up period; for experienced programmers, each newly introduced macro still demands significant effort to learn its parameter interface and valid value combinations;
- **Error-prone** — manual parameter entry invites mismatched quotes, wrong flags, and incomplete conditions that only surface at batch-run time;
- **Inconsistency** — the same table implemented by different programmers on different projects often results in different macro configurations and parameter choices.

The cumulative effect is that programmers are less inclined to adopt the system macros and tend to fall back to copying programs from previous projects. This makes it difficult for the department to achieve the standardization and consistency that **a unified macro system was originally designed to deliver**.

The complete toolkit covers the full TFL production lifecycle: TOC generation, TFL metadata setup (PDT filling), initial program creation, batch execution, and output combination. This paper focuses specifically on the most intellectually demanding step — the **automated derivation of program names and macro parameters** from TOC metadata. Batch execution and output combination are handled by companion SAS macros and are outside the scope of this paper.

The core component presented here, generate_pdt, automates the PDT filling process by decomposing TOC entries and matching them against a structured knowledge base using a three-level matching

strategy. The current implementation operates in a **title-driven** mode: all matching decisions are based solely on the output title text. Footnotes, programming notes, and other contextual metadata embedded in the TOC are not yet consumed — this is the subject of the next planned iteration (see Future Directions).

THE CORE IDEA: TOC DECOMPOSITION

Our fundamental insight is that every TFL output in a clinical trial TOC can be decomposed into three independent components that together fully determine the system macro and its parameters:

COMPONENT A: ANALYSIS TYPE

The analysis type specifies *what kind* of analysis is being performed. Within a single ADaM domain, there are typically multiple analysis types. For example, within the AE domain (Section 14.3.1), common analysis types include:

- Overall summary (total count by treatment group)
- By SOC and PT (hierarchical breakdown)
- By PT only
- By severity, SOC and PT
- … (and more)

For the Laboratory domain (Section 14.3.4), analysis types include:

- Descriptive summary
- Shift table
- Shift by visit
- Abnormality incidence
- Abnormal values / listing
- … (and more)

Each analysis type maps to a specific program template and a base parameter set (sysparm3).

COMPONENT B: FILTER CONDITIONS

The filter conditions specify *which subset* of data to include. These are extracted from the title and population specification. Examples:

Domain	Filter Variable	Subset Condition
AE	AEACN	Dose Increased / Drug Interrupted / Dose Reduced / Drug Withdrawn
AE	AESEV	MILD / MODERATE / SEVERE
AE	AESER	AESER="Y" (Serious AE)
AE	AEDIS	AEDIS="Y" (AE Leading to Discontinuation)
AE	AESDTH	AESDTH="Y" (AE Leading to Death)
Lab	PARCAT1	Hematology / Chemistry / Coagulation / Urinalysis…
Population	SAFFL	SAFFL="Y" (Safety Analysis Set)
Population	FASFL	FASFL="Y" (Full Analysis Set)

COMPONENT C: INPUT DATASET

The input dataset is determined by the section number in the TOC:

Section	Domain	Program Prefix
14.1	ADSL (Demographics)	tadsl_*
14.2	ADSL (Efficacy populations)	tadsl_*
14.3.1	ADAE (Adverse Events)	tadae_*
14.3.4	ADLB (Laboratory)	tadlb_*
14.3.5	ADVS (Vital Signs)	tadvs_*

Once all three components are determined, the system macro and full parameter string can be assembled mechanically. The challenge, then, is how to reliably identify these components from the natural-language title. This is where our three-level matching strategy comes in.

MATCHING METHOD 1: EXACT MATCHING

MECHANISM

Exact matching performs literal substring comparison between the TOC title and entries in the knowledge base. In SAS terms, this is equivalent to:

```
/* AE: keyword → filter condition */
if index(upcase(outtitle), "TREATMENT-EMERGENT") > 0 then sysparm =
'trtemfl="Y"';

/* AE: regex → analysis type */
if prxmatch("/SOC.*PT/i", outtitle) then analysis_type = "by_soc_pt";

/* Lab: keyword → parcat1 filter */
if index(upcase(outtitle), "URINALYSIS") > 0 then parcat1 = "Urinalysis";
```

The knowledge base (knowledge_base.xlsx) contains one row per known analysis pattern. Each row includes a title_shell column with the keyword to search for, and corresponding program name and parameter template columns.

COVERAGE

In our production experience across multiple clinical studies, exact matching covers approximately **80%** of all TOC entries. These are the well-established, standardized outputs that use consistent wording across projects.

LIMITATION: WHY EXACT MATCHING ALONE IS INSUFFICIENT

Exact matching fails when TOC authors use slightly different wording for the same analysis. Even trivial differences — a hyphen versus a space, "and" replaced by a comma — cause a miss:

Knowledge Base Keyword	Actual TOC Wording	Difference
"Treatment-Emergent"	"Treatment Emergent"	Hyphen vs space
"Vital Signs"	"Vital Sign Measurements"	Singular + extra word
"Shift Table of Laboratory"	"Laboratory Shift Analysis"	Word reordering

These are cases where a language model can easily identify semantic equivalence despite surface-level textual differences. This motivates the introduction of semantic matching as the next layer.

MATCHING METHOD 2: SEMANTIC MATCHING (EMBEDDING MODEL)

MODEL SELECTION: BGE-LARGE-ZH-V1.5

We selected BGE-large-zh-v1.5 (Beijing Academy of Artificial Intelligence, 326M parameters, 1024-dimensional embeddings) as our semantic backbone. This choice was driven by three factors:

- **Bilingual optimization** — BGE-large-zh is specifically trained for Chinese-English cross-lingual retrieval, matching our mixed-language TOC environment;
- **Deployment simplicity** — runs on CPU (no GPU required), with model weights available offline for air-gapped servers;
- **Proven retrieval quality** — ranked #1 on the C-MTEB leaderboard (a Chinese text retrieval benchmark) at time of selection for Chinese text retrieval tasks.

MECHANISM

Our initial design after exact matching was to use semantic embedding as the sole fallback. For entries not matched by exact keywords, we compute cosine similarity between the TOC title and candidate knowledge base entries:

```
from transformers import AutoModel, AutoTokenizer
import torch

model = AutoModel.from_pretrained('BAAI/bge-large-zh-v1.5')
tokenizer = AutoTokenizer.from_pretrained('BAAI/bge-large-zh-v1.5')

# Encode and compute cosine similarity
title_vec = encode(toc_title)       # query embedding
kb_vecs = encode(candidate_entries) # corpus embeddings

similarities = cosine_similarity(title_vec, kb_vecs)

best_idx = argmax(similarities)
if similarities[best_idx] >= 0.55:
    return AutoMatch(candidate_entries[best_idx])
else:
    return FlagForReview()  # NEED_REVIEW = "Y"
```

THRESHOLD SELECTION

The acceptance threshold of 0.55 is an **empirical estimate** derived from internal testing across several clinical projects. It was chosen to balance recall (accepting valid matches) against precision (avoiding false positives). No rigorous cross-validated optimization has been performed on this value — it represents a conservative working point that may be refined as more production data accumulates.

PERFORMANCE OPTIMIZATION: SHORTLIST PRE-FILTER

A naïve implementation would compute BGE embeddings for all candidate knowledge base entries for each unmatched TOC title. With a knowledge base containing hundreds of entries and potentially dozens of unmatched titles, this would result in thousands of model inference calls, each requiring ~100ms on CPU.

Our optimization is to first rank all candidates using Python's lightweight SequenceMatcher (character-level similarity) and select only the **top-8** candidates before invoking the embedding model. This reduces model inference calls by approximately 92%, bringing the entire semantic stage to 1–2 seconds even on CPU-only hardware.

```
# Shortlist: lightweight char-level pre-filter
from difflib import SequenceMatcher

ratios = [(i, SequenceMatcher(None, toc_lower, kb_lower).ratio())
```

```
            for i, kb_lower in enumerate(all_kb_entries)]
top_8 = sorted(ratios, key=lambda x: x[1], reverse=True)[:8]

# Only these 8 candidates go through BGE inference
for idx, _ in top_8:
    sim = cosine(model.encode(toc_title), model.encode(kb_entries[idx]))
    ...
```

LIMITATION 1: SEMANTIC SIMILARITY ≠ FUNCTIONAL EQUIVALENCE

The fundamental problem with semantic matching is that textually similar titles may refer to completely different analyses:

Title A	Title B	Cosine Sim	Issue
"Serious Adverse Event" (SAE)	"Severe Adverse Event"	0.79 ✔	Accepted, but filter: AESER="Y" vs AESEV="SEVERE"
"Treatment-Emergent AE"	"Treatment-Related AE"	0.82 ✔	Accepted, but filter: TRTEMFL="Y" vs TRAEFL="Y"

LIMITATION 2: DOMAIN-SPECIFIC ABBREVIATIONS ARE UNKNOWN

A more critical limitation emerged during production deployment: BGE-large-zh-v1.5 is a *general-purpose* language model trained on broad web corpora. It has **no knowledge of clinical analysis abbreviations**. When a TOC title uses "TEAE" and the knowledge base entry says "Treatment-Emergent Adverse Event", the embedding model sees two unrelated short text strings and returns similarity well below 0.55:

Abbreviation	Full Term	Cosine Similarity	Result
TEAE	Treatment-Emergent Adverse Event	< 0.55	Miss ✗
SAE	Serious Adverse Event	< 0.55	Miss ✗
TRAE	Treatment-Related Adverse Event	< 0.55	Miss ✗

This is a structural limitation that cannot be resolved by adjusting the similarity threshold — it requires explicit domain knowledge injection.

MATCHING METHOD 3: ABBREVIATION DICTIONARY

MOTIVATION

To address the embedding model's inability to resolve domain-specific abbreviations, we introduced a deterministic abbreviation dictionary layer. Unlike fine-tuning the model (which is expensive, non-reproducible, and loses auditability), this approach maintains full transparency while precisely targeting the LLM's blind spot.

MECHANISM

In the knowledge base, the title_shell column registers abbreviations and their full-form equivalents, separated by #:

```
title_shell: "Treatment-Emergent#TEAE#treatment-emergent adverse event"
title_shell: "Serious Adverse Event#SAE#serious AE"
title_shell: "Treatment-Related#TRAE#drug-related adverse event"
title_shell: "Demographics#Demographic and Baseline
Characteristics#Baseline Demographics"
```

During matching, the system splits title_shell by # and performs literal substring comparison for each variant. If any registered variant appears in the TOC title, the entry is matched deterministically — no model inference required.

WHY NOT FINE-TUNE THE EMBEDDING MODEL?

- **Cost** — fine-tuning requires labeled training data and GPU resources for each update;
- **Reproducibility** — model weights change with each training run, making audit trails impossible;
- **Scope** — clinical abbreviations are project-level naming conventions, not general language knowledge;
- **Auditability** — a deterministic lookup table can be inspected and verified by non-technical reviewers.

EXECUTION ORDER

In the production implementation, the three methods execute in this order:

1. **Exact keyword match** — the primary literal keyword from title_shell (before the first #);
2. **Abbreviation dictionary match** — all registered #-separated variants;
3. **Semantic embedding match** — only invoked when both deterministic methods fail.

This ensures deterministic methods always take priority, and the probabilistic LLM serves only as a final safety net.

CANDIDATE SELECTION: PRIORITY-BASED TIE-BREAKING

When multiple matching methods produce candidates for the same TOC entry, the system must select the single best match. We use a **priority-based ranking** with four tie-breaking criteria applied in order:

```
sort_key = (priority↑, confidence↓, specificity↓, length_gap↑)
```

Dimension	Meaning	Effect
priority	1=exact, 2=synonym, 3=semantic	Deterministic methods always win
confidence	Cosine similarity (or 1.0 for exact)	Higher confidence preferred
specificity	Number of tokens matched	More specific patterns preferred
length_gap	|len(title) − len(pattern)|	Closer length match preferred

The tuple is compared lexicographically with the lowest value winning. This guarantees that an exact keyword match always takes priority over a semantic match, regardless of how high the cosine similarity may be — because deterministic keyword matching is provably correct, while semantic similarity can produce false positives (e.g., “Serious” vs “Severe”).

FALLBACK WHEN PYTORCH IS UNAVAILABLE

If the runtime environment lacks GPU hardware or PyTorch is not installed, the tool automatically disables semantic matching and operates in exact + abbreviation dictionary mode only. This ensures the tool is always functional, with or without ML infrastructure.

THE THREE-LEVEL MATCHING STRATEGY

The three matching methods form a cascading pipeline, applied in order:

Level	Method	Coverage	Nature	Failure Mode
1	Exact Match	~80%	Deterministic	Wording variations
2	Abbreviation Dictionary	~15%	Deterministic	Unseen abbreviations
3	Semantic Match (BGE)	~5%	Probabilistic	False positives (similar ≠ same)

Design principle: **prioritize deterministic methods**. The LLM is invoked only as a last resort, minimizing non-reproducible behavior while still handling novel inputs gracefully.

REASSEMBLY: FROM COMPONENTS TO RUNNABLE CODE

Once the three-step decomposition is complete, the system **reassembles** the identified components into a ready-to-run SAS program. Consider the following TOC entry:

```
OUTREF:   14.3.1-1.1          (output reference number)
OUTTITLE: Summary of TEAEs by SOC and PT (Safety Analysis Set)  (output
title)
OUTPOP:   Safety Analysis Set  (analysis population)
```

The pipeline processes this as follows:

Step 1: Input Dataset (from section number)

```
14.3.1 → section sheet "s14_3_1" → domain: ADAE → prefix: tadae
```

Step 2: Analysis Type (synonym match)

```
title contains "TEAEs"
→ hits synonym: "Treatment-Emergent#TEAE"
→ pgm = "tadae_teae_bysocpt"
```

Step 3: Filter Conditions (auto-assigned)

```
TEAE analysis → analysis filter: trtemfl="Y"
"Safety Analysis Set" → population filter: SAFFL="Y"
```

TEMPLATE-BASED CODE GENERATION

With the program name and parameters determined, the system looks up the corresponding SAS template from a **program_template_list**. This template list maps each analysis type (matched by keyword/regex against the program name) to a pre-written SAS template file:

OUTTYPE	KEYWORD (regex)	Template File
Table	tadae_teae	tadae_template.sas
Table	tadae_sae	tadae_template.sas
Table	tadlb_desc	tadlb_template.sas
Table	tadlb_shift	tadlb_shift_template.sas
Table	tadsl_pop	tadsl_pop_template.sas
Table	tadsl_demo	tadsl_demo_template.sas

The %generate_program macro then assembles three parts into the final .sas file:

1. **Standard header** — auto-populated with file name, developer, date, project/protocol, input/output datasets;
2. **Template body** — read from the matched template file, containing the analysis-specific macro call and logic scaffold;
3. **Macro parameters** — the filter conditions and analysis parameters determined during decomposition.

Generated Output (tadae_teae_bysocpt.sas)

```
/* --- Standard Header --- */
File Name : tadae_teae_bysocpt.sas
Developer : H. Jing
Date      : 2026-06-23
```

```
Output    : 14.3.1-1.1

/* --- Template Body --- */
%macro tadae_teae_bysocpt;
  %load_parameters
  /* ... analysis template logic ... */
%mend;
%tadae_teae_bysocpt;

/* --- Macro Parameters (passed via SAS SYSPARM option) --- */
SYSPARM: trtemfl="Y" | SAFFL="Y"
```

In our implementation, macro parameters are delivered to the SAS program through the SYSPARM system option — a pipe-delimited string that the system macro parses at runtime. Other organizations may pass parameters through conventional macro arguments or configuration files; the matching logic is independent of the delivery mechanism.

The result is a complete, runnable SAS program file. The programmer's starting point is no longer a blank file — it is a pre-filled program that only requires study-specific adjustments.

BENEFITS

After matching is complete, the system macro and full macro parameters for every TOC entry are determined. The tool generates:

- A complete PDT with program names and parameters filled in;
- Ready-to-run SAS programs that invoke the correct system macros with the correct parameters.

No manual first-draft work is required. The programmer's role shifts from *authoring* to *reviewing*. Each benefit directly addresses one of the three pain points identified in the Introduction:

Key advantages:

- **From TOC to Runnable Programs** — PDT filling time reduced from hours to minutes; new programmers can be productive immediately without memorizing macro libraries;
- **Correct Fill + Safety Net** — zero manual first drafts; programmers start from review, not from scratch; uncertain matches are automatically flagged for human verification;
- **Standardizing Derivation** — deterministic output ensures same TOC always produces same result; every decision is traceable to a knowledge base entry; the knowledge base is project-independent and grows over time.

STANDARDIZING DERIVATION

An important secondary benefit is that the tool drives departmental consensus on both ADaM dataset derivation **and** the PDT fill-in approach. Because filter variables and program-template mappings are pre-defined in the knowledge base and automatically assigned to each TFL, all programmers within the department use the same flag variables for the same analysis types and the same program structures for the same output patterns.

For example, in AE analyses, the tool assigns TRTEMFL="Y" (a company-defined analysis flag for treatment-emergent events) for treatment-emergent AEs and TRAEFL="Y" for drug-related AEs. This standardizes how programmers derive the ADAE dataset — at the department level, we add a TRAEFL variable to ADAE and populate it with "Y" for drug-related events. Similarly, for laboratory analyses, descriptive summaries uniformly use ANL01FL="Y" as the analysis flag, while shift tables use ANL02FL="Y".

While the CDISC ADaM Implementation Guide does not mandate specific flag variable names or derivation rules, the automated parameter assignment effectively creates departmental standards for these derivations, reducing variability and facilitating cross-project consistency.

Beyond matching quality, the practical automation gain is substantial: repetitive setup tasks are automated end-to-end, including TOC intake normalization, PDT filling, and starter SAS program generation. This shifts programmer effort from boilerplate drafting to clinically meaningful validation and refinement.

NEED_REVIEW SAFETY NET

All uncertain matches (cosine < 0.55) are automatically flagged with NEED_REVIEW=Y and highlighted in red in the output Excel workbook. This visual cue ensures programmers immediately identify rows requiring human verification before the generated parameters are used in production.

NEXT STEP

FROM TITLE-DRIVEN TO TOC-DRIVEN

The current implementation is **title-driven**: all matching decisions are based solely on the output title. This is sufficient for macro selection and basic parameter assignment (~95% coverage), but generated programs use "safe default" logic and may lack project-specific conditions.

The most impactful next step is to evolve toward a **TOC-driven** mode that additionally consumes footnotes and programming notes embedded in the TOC spreadsheet. For example, a footnote stating "Display only subjects with baseline and post-baseline values" would be parsed and injected as a conditional filter into the generated program. This evolution does not change the matching architecture — it extends the post-match parameter refinement stage to produce more robust, project-specific initial programs.

KB MAINTENANCE AUTOMATION

Generate update suggestions from review logs when new macros or structures are introduced, keeping the knowledge base current without requiring manual periodic audits.

BETTER EMBEDDING MODEL

Evaluate domain-specific or newer bilingual embedding models to improve semantic matching accuracy beyond the current BGE-large-zh-v1.5. Candidates include models fine-tuned on clinical programming terminology or newer architectures with better cross-lingual alignment.

PRECISION TUNING

Add stricter post-check rules to reduce semantic near-miss false positives — for example, token-overlap guards that verify shared key tokens between the TOC phrase and the matched KB entry, length-ratio filters that reject matches with large character-count disparities, and synonym pre-screening that ensures core clinical terms align before accepting a semantic score.

CONCLUSION

We have presented generate_pdt, a metadata-driven pipeline that eliminates manual first-draft coding in TFL production. The core approach is to decompose each TOC entry into three components (analysis type, filter conditions, input dataset) and match them using a three-level strategy (exact → synonym → semantic). The design prioritizes deterministic methods, invokes LLM only as a fallback, and always preserves human oversight through review flags.

The tool has been deployed across multiple clinical studies, demonstrating significant time savings while maintaining auditability and programmer control. Its knowledge base grows with each project, continuously improving matching coverage without requiring model retraining.

The pattern — *Classify with AI, Derive with Rules, Verify with Humans* — is applicable beyond TFL production to other repetitive tasks in clinical programming that require semantic understanding but demand deterministic, auditable outputs.

ACKNOWLEDGMENTS

The author thanks colleagues at Jiangsu Hengrui Pharmaceuticals for their feedback during development and production deployment of this tool.

CONTACT INFORMATION

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

Hefan Jing
Jiangsu Hengrui Pharmaceuticals Co., Ltd.
hefan.jing@hengrui.com

Peilin Liu
Jiangsu Hengrui Pharmaceuticals Co., Ltd.
peilin.liu.pl71@hengrui.com