Paper DV12

From Black Box to Glass Box: A Context-Aware Approach to SDTM Mapping Automation with Interactive Visual Validation

Rostislav Markov, Amazon Web Services, Inc., New York, USA
Jacques Lanoue, Merck & Co., Inc., North Wales, USA
Ulf Nielsen, MSD Merck Sharp & Dohme AG, Zurich, Switzerland

ABSTRACT

We present a specification-first approach to Study Data Tabulation Model (SDTM) automation in which artificial intelligence (AI) assists in generating machine-readable mapping specifications rather than performing "black-box" data transformations without lineage or reproducibility.

Our architecture separates data modeling, transformation logic, and specification generation into clearly defined layers. Dataset operations are expressed as transparent SQL-like modeling prior to SDTM mapping and derivation execution. Transformation functions are implemented as versioned Python modules deployed as cloud compute resources and managed through a controlled registry with explicit metadata. Machine-understandable JSON specifications ("MUS") reference only these approved functions and fully describe source datasets, dataset operations, mappings, derivations, and preprocessing steps.

The system operates on a reuse-first model. For standardized internal studies, 60–80% of mappings can be reused directly from prior specifications via a similarity-based "spec shopping" capability before AI augmentation is applied. For the residual mapping space, we leverage a large language model through a prompting pipeline that supplies study metadata, source variables, transformation functions, and strict response-format rules; responses are stored in the mapping store and always reviewed by programmers. An embedding-based retrieval pipeline is being added to further ground AI suggestions in study-specific artifacts (e.g., EDC exports) via a retrieval-augmented generation (RAG) pattern.

Specifications can be exported, augmented externally by other AI agents, and re-imported under strong validation and governance controls. An integrated engine observability dashboard overlays runtime execution state onto a static integration topology, providing statistical programmers with a UI-ready view of pipeline runs, per-domain metrics, and log correlation, without leaving the specification authoring tool.

We benchmark our design against a target of approximately 4,000 SDTM updates per year. The architecture is intended to scale to automation rates on the order of ~90%, with an estimated potential savings of ~32,000 hours annually and near day-one SDTM availability once fully rolled out and adopted. The result is a transparent, compliant, glass-box architecture that preserves human oversight while significantly improving efficiency.

INTRODUCTION

Clinical transformation to SDTM remains a labor-intensive and error-prone process. Traditional approaches rely on:

- rigid transformation libraries,
- centralized standards governance,
- superset relational schemas,
- commercial off-the-shelf tools updated periodically, and
- limited flexibility once execution begins.

Every clinical study introduces protocol-specific nuances, metadata variations, and bespoke derivations. As a result, rigid programmatic approaches are often perceived as overly restrictive or insufficiently adaptable by study teams. At the opposite extreme, AI-driven "black-box" transformation engines promise automation but introduce unacceptable risks in regulated environments:

- hidden transformation logic,
- limited traceability,
- reproducibility concerns, and
- regulatory uncertainty.

We propose an alternative. Rather than automating data transformation directly, we automate the creation and reuse of **transparent, auditable transformation specifications** and execute them with deterministic, observable engines. AI is applied as a constrained assistive tool at the specification layer, never as a "black box" execution engine.

GUIDING PRINCIPLES

The architecture is grounded in practical enterprise and regulatory realities. The following principles shape both design and operation.

1. **Specification-first**

Transformation intent is declared before it is executed. Specifications describe what should happen. Deterministic engines (here: cloud compute resources) perform the execution. AI operates at the specification layer. Execution remains governed and reproducible.

2. Reuse before prediction

Automation begins with reuse:

- mapping specifications are imported, cloned (or "shopped") and adapted,
- functions and derivations are reused across studies,
- historical patterns and promoted specs are leveraged first.

AI addresses residual ambiguity, proposing mappings where reuse is not available or obvious.

3. Program reproducibility

All production transformations are programmatic:

- Dataset operations are explicit (via SQL),
- functions are version-controlled and deployed as cloud compute resources,
- outputs are reproducible from mapping specifications and function code, and
- mapping predictions occur in a preview environment; only validated specifications are promoted to production.

4. Code governance

Governance is enforced with:

- controlled function registry,
- semantic versioning and approvals,
- test-driven documentation,
- unit tests and metadata descriptors, and
- parameter schema validation.

Users cannot bypass architectural controls by injecting arbitrary code into specifications. AI suggestions must adhere to the same constraints.

5. Function granularity

Transformation functions are:

- modular and parametrized,
- independently testable, and
- reused globally or study-specifically.

Granularity maximizes reuse while keeping behavior understandable.

6. Transparency by design

Every mapping is:

- linked to a versioned function,
- traceable to its origin (reuse, AI, manual, import), and
- observable at runtime via an integrated execution dashboard.

There are no hidden transformations.

7. **Human oversight**

Human validation remains a regulatory safeguard:

- AI suggestions are visible, editable and attributable,
- approval is explicit and versioned, and
- mapping coverage and review status are visible in the web graphical user interface (GUI).

The human programmer remains decisively "in the loop."

8. Interoperability

Mapping specifications are portable artifacts:

- Exportable from the SDTM authoring tool,
- augmentable by external AI agents or scripts,
- re-importable under validation and governance.

The system does not depend on a single AI model.

9. Observability

Execution telemetry includes:

- function invocation traces,
- error localization,
- runtime metrics and record counts per domain, and
- version lineage from run back to specification and function code.

Troubleshooting is interactive and transparent within the SDTM tool.

10. **Simplicity**

Architectural simplicity is enforced through separation of duties:

- JSON for mapping specifications,
- Git for code governance,
- SQL/dataframe operations for data modeling,
- programmatic execution orchestrated by state machines, and
- AI is applied where ambiguity exists in mapping design.

ARCHITECTURE OVERVIEW

The system architecture is modular and separating data modeling, mapping and derivations, governance, and execution. We focus on the aspects that matter to statistical programmers: how functions are managed, how specifications are authored, and how the engine behaves.

FUNCTIONS REGISTRY

A GitHub-based registry contains reusable global functions and derivations:

- implemented in Python/Pandas,
- deployed as cloud compute resources,
- released via JFrog,
- validated with Pytest and behavior-driven documentation,
- described by metadata (inputs, outputs, SDTM domains, parameter schemas, parameterization).

Static and dynamic metadata are indexed in a search index (OpenSearch), making functions discoverable by both the authoring web GUI and the transformation engine orchestrator. Only approved functions for a given study/Implementation Guide (IG) version are surfaced in the web GUI.

SPECIFICATION AUTHORING TOOL

The authoring tool is a reactive web GUI. It enables users to:

- inspect dataset inventory,
- configure dataset operations,
- design source-to-target mappings and derivations,
- track mapping coverage and origin (manual/reused/AI/import),
- reuse mappings from other studies,
- invoke AI-based mapping predictions,
- manage study-specific configuration parameters,
- manage specification versions,
- export/import specifications,
- publish to a preview environment and promote to production.

Mapping specifications are portable JSON artifacts. Users can:

1. export (download) a specification,
2. submit to an external AI agent or script,
3. augment or refactor mappings,
4. re-import the specification, and
5. review highlighted differences.

Validation upon import ensures:

- function existence in the registry,
- version approval,
- parameter schema compliance, and
- domain compatibility.

External AI cannot bypass governance controls. While curated context improves AI output quality, responses remain non-deterministic. Human review of AI-generated code is non-negotiable. Our workflow enforces this through preview environment testing and controlled promotion to production.

TRANSFORMATION ENGINE

The engine enables programmatic dataset transformations. It:

- parses the specification,
- constructs state machines executing the specified transformation steps/using the prebuilt cloud compute resources corresponding to each function,
- manages error handling (for example, missing source forms referenced in the specification),
- writes intermediate dataframes and traces for each transformation,
- logs execution details and audit trail.

The engine is data-driven by the MUS specification. It does not contain hard-coded study-specific logic.

SPECIFICATION AUTHORING WORKFLOW

DATASET OPERATIONS

Before any SDTM mapping occurs, programmers can define **dataset operations** in the GUI to perform:

- query definition (select columns, set distinct flags),
- pre-mapping joins across source tables,
- intermediate dataset preview,

- basic column validation.

This layer is implemented as structured “SQL-like” operations (TableAndColumns, Join, Filter, GroupBy) that are translated to SQL or dataframe operations at runtime. All modeling precedes mapping functions, which keeps SDTM mappings simpler and easier to review. Below figure illustrates a simple example.

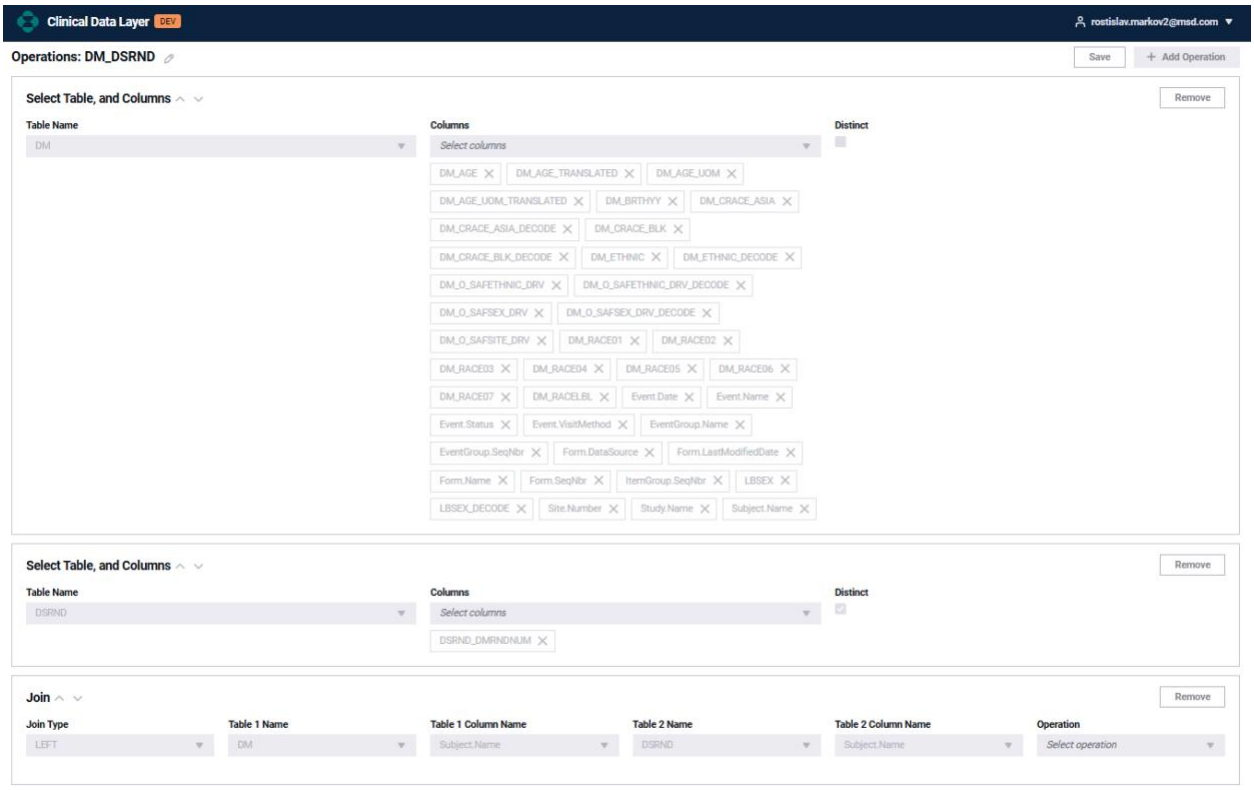


Figure 1. Dataset operations (simple configurator screen)

An optional **advanced SQL** mode allows power users to enter custom SQL within governed constraints and validate it against the source data schema before running the engine. This helps catch syntax-level errors early and prevents failures during the execution.

FUNCTIONS

The web GUI exposes the functions registry directly to the programmer:

- A searchable registry of both global and study-specific functions, searchable by name, purpose, domain, or study context.
- Hover tooltips and details panes that show concise descriptions, expected inputs/outputs, and prerequisites.
- Version selection and, where needed, the ability to disable specific functions or versions for a study.

Programmers can quickly find the right derivation or transformation without leaving the mapping screen, while governance ensures that only authorized functions are available for a given study.

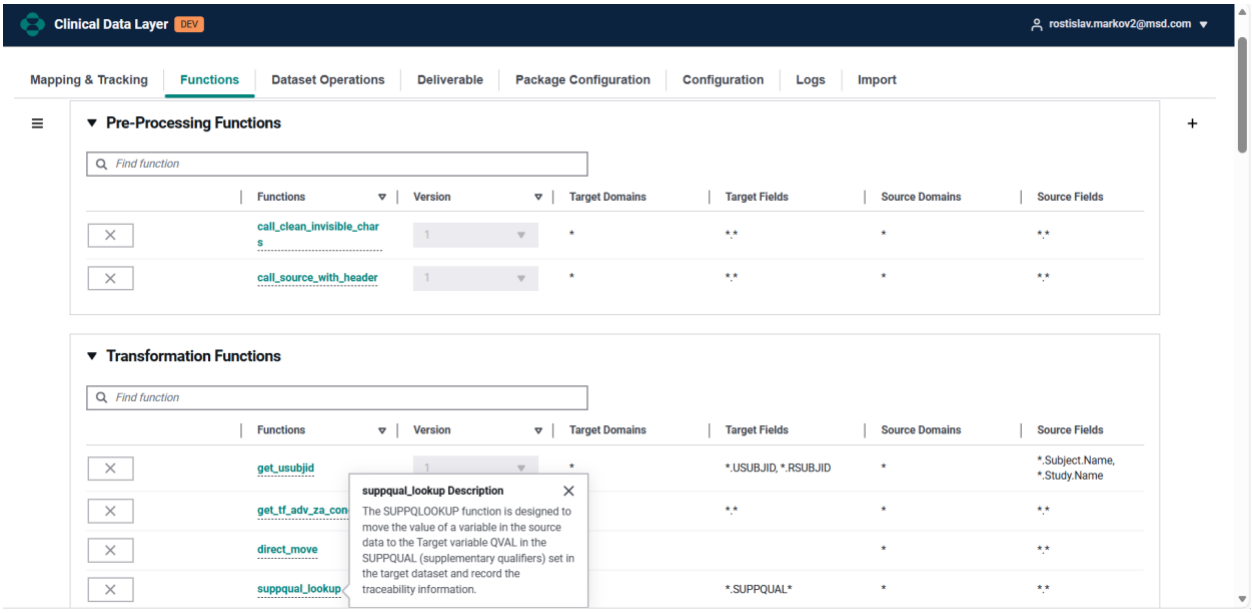


Figure 2. Functions in the GUI

SOURCE-TARGET MAPPINGS

The core of the authoring experience is the **source-to-target mapping grid**, which includes:

- variable-level mapping rows,
- mapping completeness tracking,
- function selection (including parameterized functions),
- conditional parameters (such as constant values for assign-type functions),
- tooling (such as reuse and import).

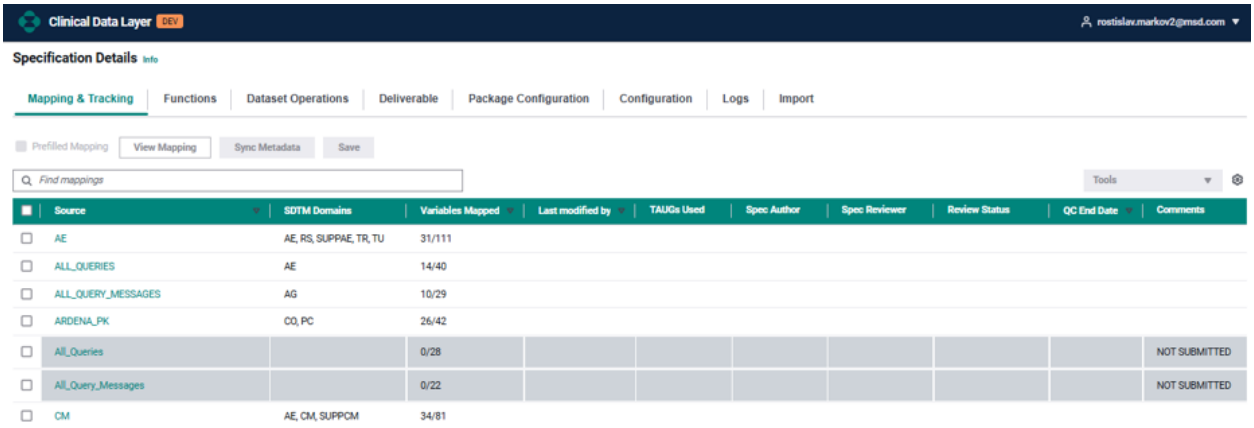


Figure 3. Domain mapping overview (summary view, redacted)

For each domain, programmers can track

- source table and variable,
- SDTM domain and variable,
- transformation function and version, and
- the programmer who last changed/accepted the mapping.

This makes mapping ownership and history visible at a glance.

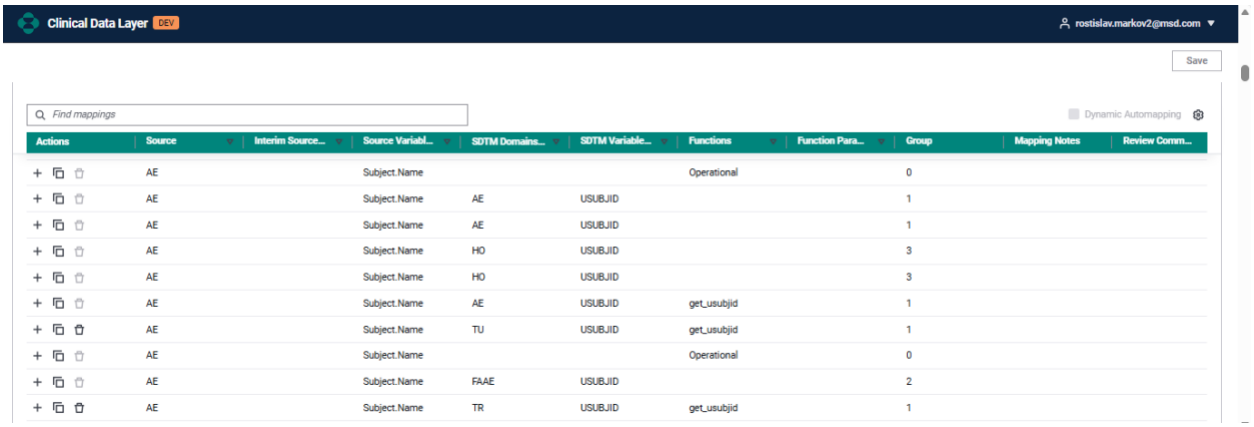


Figure 4. Source mapping screen

OBSERVABILITY DASHBOARD

The observability dashboard gives programmers a visual view of each engine run directly inside the authoring tool. For each specification, the dashboard shows a **run history** with:

- execution status (success / failure / in-progress),
- start/end times and duration,
- high-level error counts and record counts.

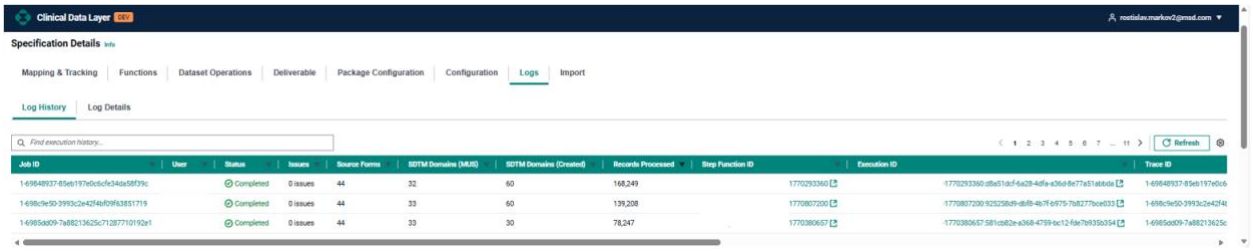


Figure 5. Log history of individual runs per specification version (overview, redacted)

Clicking into a run opens a **pipeline view** that represents the actual processing steps, including:

- preprocessing,
- dataset operations,
- SDTM Minus mappings,
- per-domain outputs, and
- final write-out to the data platform.

Each step is color-coded by status and annotated with metrics such as number of records processed and number of issues. From this view, programmers can:

- quickly see which domain or step failed or is slow,
- drill into domain-specific metrics, and
- follow deep links to the underlying logs (e.g., in OpenSearch) if needed.

This significantly reduces the time and effort required to understand “what happened” in an SDTM run, especially when troubleshooting partial failures or unexpected outputs.

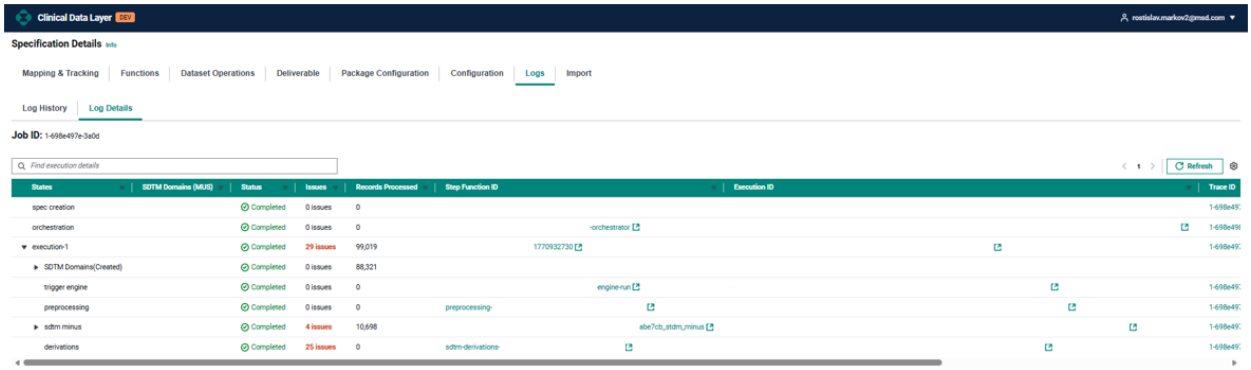


Figure 6. Log details of individual runs (overview, redacted)

SPECIFICATION SYNTAX

CONCEPTUAL OVERVIEW

The transformation logic is captured in a single, machine-readable JSON document. This file is the contract between the authoring tool (where programmers design mappings and dataset operations), and the engine (which executes those mappings). For a given study, the specification describes:

- the study and SDTM context (header),
- variable-level source-to-target mappings (SDTM Minus),
- dataset operations (joins, filters, aggregations),
- derivation functions and their execution order,
- pre-processing steps applied to the source data, and
- the set of transformation functions referenced by the spec.

Because the format is standardized, specifications can be exported, imported, validated, and reused across studies and environments.

MAIN SECTIONS

A typical specification includes the following top-level sections:

- header: study context, SDTMIG version, and audit metadata.
- sdtmMinus: source to target variable mappings, expressed per source table and variable.
- studyDatasetOperations: declarative dataset operations (joins, filters, column selections) that run before SDTM mappings.
- sdtmDerivations: study level derivation functions that operate across rows or domains (e.g., informed consent dates, first/last dose).
- preProcessingFunctions: functions applied before mapping/derivation (e.g., cleaning invisible characters, handling repeated headers).
- transformationFunctions: the list of all transformation functions referenced in sdtmMinus and sdtmDerivations with their IDs and versions.

The engine reads this structure and constructs an execution plan without additional configuration.

SAMPLE SYNTAX

Below is an illustrative example using the specification syntax.


```
{
  "header": {
    "Study ID": "STUDY-001",
    "Study Name": "STUDY-001_internal",
    "SDTM Version": "SDTMIG v3.4",
    "Dictionary Versions": {
      "Meddra": "",
      "WHODD": "",
      "LOINC": "",
      "CT": ""
    },
    "Specification Name": "STUDY-001_internal_sdtmig3.4_v01"
  },

  "sdtmMinus": [
    {
      "SourceTable": "AE",
      "SourceVariable": "AEACN_RAW",
      "InterimDataset": "",
      "TargetList": [
        {
          "SDTMDomain": "AE",
          "SDTMVariable": "AEACN",
          "SDTMTransformationFunction": "uuid-direct-move",
          "SDTMTransformationFunctionName": "direct_move (v1)",
          "FunctionParameterValue": "",
          "Group": "1",
          "SDTMCodelist": "",
          "SDTMType": "text"
        }
```

```
    ]
   },
   {
    "SourceTable": "AE",
    "SourceVariable": "AEENDTC_RAW",
    "InterimDataset": "",
    "TargetList": [
     {
      "SDTMDomain": "AE",
      "SDTMVariable": "AEENDTC",
      "SDTMTransformationFunction": "uuid-iso8601",
      "SDTMTransformationFunctionName": "ISO8601 (v1)",
      "FunctionParameterValue": "",
      "Group": "1",
      "SDTMCodelist": "",
      "SDTMType": "text"
     }
    ]
   }
  ],

  "studyDatasetOperations": {
   "datasetOperations": [
    {
     "DM_EXAMPLE_JOIN": [
      {
       "type": "TableAndColumns",
       "sequence": 1,
       "tableName": "DM",
       "tableColumns": "Study.Name,Subject.Name,DM_AGE,DM_BRTHYR",
       "distinct": false,
       "filterText": ""
      },
      {
       "type": "TableAndColumns",
       "sequence": 2,
       "tableName": "RAND",
       "tableColumns": "RANDNUM",
       "distinct": true,
       "filterText": ""
      },
      {
       "type": "Join",
       "sequence": 3,
       "joinType": "LEFT",
       "table1Name": "DM",
       "table1ColumnName": "Subject.Name",
       "table2Name": "RAND",
       "table2ColumnName": "Subject.Name",
       "operation": "",
       "additionalJoinConditions": []
      }
     ]
    }
   ]
  },

  "sdtmDerivations": [
   {
```

```
      "Function": "uuid-get-informed-consent",
      "Version": 1,
      "Name": "std_fn_get_informed_consent",
      "Sequence": 0
    },
    {
      "Function": "uuid-get-first-dose",
      "Version": 1,
      "Name": "std_fn_get_first_dose",
      "Sequence": 1
    }
  ],

  "preProcessingFunctions": [
    {
      "Function": "uuid-clean-invisible-chars",
      "Version": 1,
      "Name": "std_fn_clean_invisible_chars"
    }
  ],

  "transformationFunctions": [
    {
      "Function": "uuid-direct-move",
      "Version": 1,
      "Name": "std_fn_direct_move"
    },
    {
      "Function": "uuid-iso8601",
      "Version": 1,
      "Name": "std_fn_iso8601"
    }
  ]
}
```


For statistical programmers, this structure mirrors the specification authoring web GUI:

- reusable joins/filters become studyDatasetOperations,
- the mapping grid becomes sdtmMinus,
- cross-cutting derivations become sdtmDerivations,
- and the function registry is captured in transformationFunctions.

The engine does not infer behavior. It executes exactly what is declared in this specification, using only governed functions.

TOOL USE FOR AUTOMAPPING

The web GUI offers several tools to assist with automapping specifications:

- **reuse mappings** (import from other studies via spec shopping),
- **import from file** (validated MUS specifications),
- **prefill based on user-uploaded rules**, and
- **predict with AI** (Bedrock based suggestions).

REUSE (“SHOP”) MAPPINGS

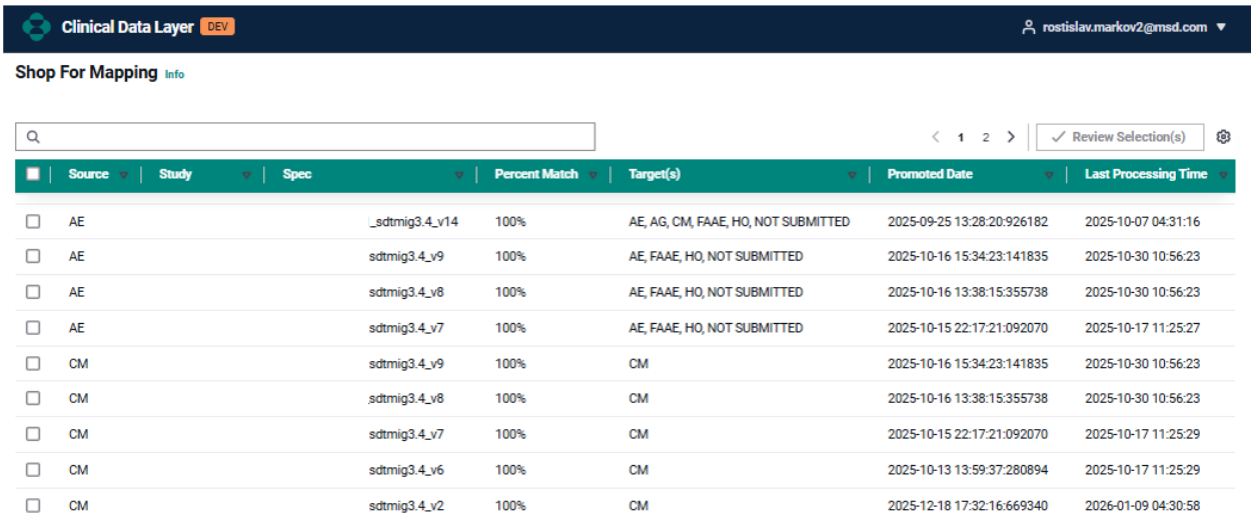
Users can “shop” for existing mappings, which automates the discovery and reuse of mappings from promoted specifications for prior studies with similar data profile. For a selected source form, users see:

- matching reference studies/specs,
- similarity scores,
- common variables,
- SDTMIG version and promotion time.

Users can:

- filter by SDTMIG version and minimum score,
- sort by score,
- inspect overlap details,
- select reference specs whose mappings they want to reuse.

When users apply reuse, the current specification gets updated with the retrieved mappings. This yields 60–80% baseline mapping reuse for standardized studies before AI is invoked.



Source	Study	Spec	Percent Match	Target(s)	Promoted Date	Last Processing Time
AE		_sdtmig3.4_v14	100%	AE, AG, CM, FAAE, HO, NOT SUBMITTED	2025-09-25 13:28:20:926182	2025-10-07 04:31:16
AE		sdtmig3.4_v9	100%	AE, FAAE, HO, NOT SUBMITTED	2025-10-16 15:34:23:141835	2025-10-30 10:56:23
AE		sdtmig3.4_v8	100%	AE, FAAE, HO, NOT SUBMITTED	2025-10-16 13:38:15:355738	2025-10-30 10:56:23
AE		sdtmig3.4_v7	100%	AE, FAAE, HO, NOT SUBMITTED	2025-10-15 22:17:21:092070	2025-10-17 11:25:27
CM		sdtmig3.4_v9	100%	CM	2025-10-16 15:34:23:141835	2025-10-30 10:56:23
CM		sdtmig3.4_v8	100%	CM	2025-10-16 13:38:15:355738	2025-10-30 10:56:23
CM		sdtmig3.4_v7	100%	CM	2025-10-15 22:17:21:092070	2025-10-17 11:25:29
CM		sdtmig3.4_v6	100%	CM	2025-10-13 13:59:37:280894	2025-10-17 11:25:29
CM		sdtmig3.4_v2	100%	CM	2025-12-18 17:32:16:669340	2026-01-09 04:30:58

Figure 7. Reuse (“shop”) mappings screen.

IMPORT/EXPORT OF SPECIFICATIONS

To support reuse across studies and environments and enable integration with external AI agents, the web GUI allows programmers to **export** and **import** mapping specifications as JSON files. From the run history, a programmer can download the exact published specification (MUS file) that was used for a given SDTM run. This makes it easy to:

- reapply a successful mapping pattern to a new study,
- share specifications with collaborators, or
- archive a regulatory “snapshot” of the transformation logic.

Programmers can upload an external or previously exported specification into the tool. Before any mappings are committed, the system performs automated checks:

- syntax and structure (is this a valid specification?),
- function references (do the functions exist in the current registry and are they approved?),
- SDTM domains/variables (do they match the configured SDTM-IG version?),
- source references (do the source tables/variables exist for this study?).

Only mappings that pass validation are eligible to be applied. The rest are reported back with clear error messages. This allows teams to safely reuse and adapt existing transformation logic, whether authored by humans or by external AI agents, without bypassing standards, function governance, or study-specific constraints.

GENERATIVE AUTOMAPPING

AI-assisted automapping leverages a prompting pipeline to a large language model. This inference pipeline provides context including study metadata, source variable names and descriptions, available transformation functions, and model response formatting rules. The model returns structured mapping proposals, which are written into the mapping database (which backs the web GUI) as draft suggestions. We are currently working on an embedding-based pipeline that generates a study profile based on a set of files such as EDC exports. This context will be used to further ground model responses (RAG pattern). In all cases, AI suggestions remain candidate mappings: they must be reviewed, edited, and approved by programmers before they become part of a promoted specification.

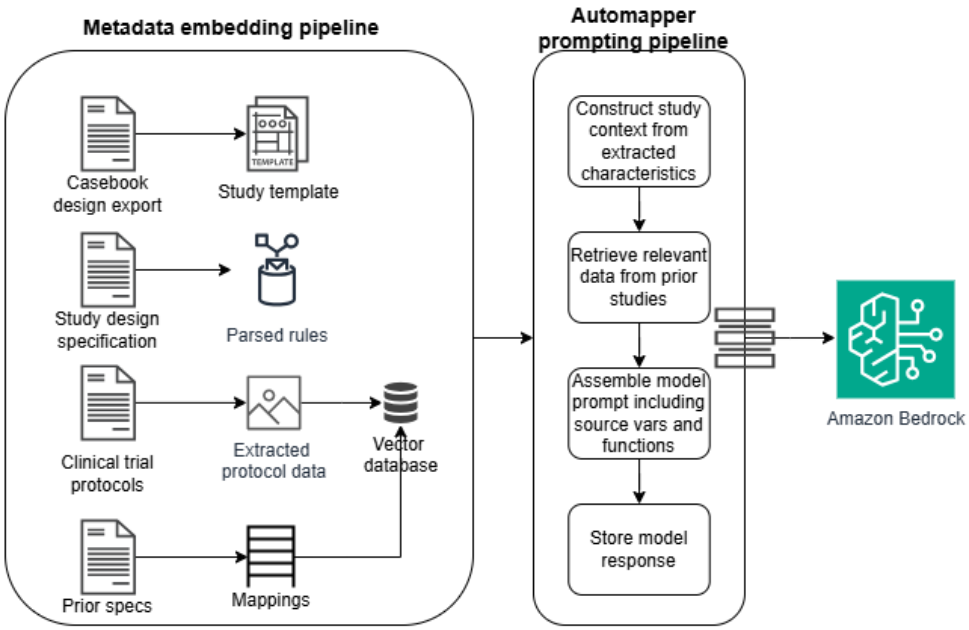


Figure 8. Generative automapping pipeline design.

DISCUSSION

GOVERNANCE COMPLIANCE

The design enforces governance at multiple layers:

- function registry controls what code can be used,
- MUS schema and validation enforce structural and reference integrity,
- dataset operations are restricted to explicit patterns or validated SQL,
- AI suggestions are constrained to approved functions and MUS structure,
- imports go through staged validation and transactional commit, and
- observability provides transparent execution traces and metrics.

Specifications are configuration artifacts, not executable code. They are versioned, auditable, and serve as lineage anchors for compliance (including masking and change impact analysis).

BASELINE AND EXPECTED BENEFITS

We benchmark the architecture against a target of approximately 4,000 SDTM updates annually. Initial experience on early releases and pilot studies indicates:

- baseline mapping reuse in the range of 60–80% for standardized studies,
- automation rates approaching ~90% for selected use cases when combining reuse and deterministic execution,
- substantial reductions in manual effort and turnaround time, with projected savings on the order of 32,000 hours per year at full scale,
- near “day 1” SDTM availability as an aspirational goal once processes and adoption are mature.

These figures represent a mix of observed results on initial cohorts and projected impact when the platform is applied consistently across the portfolio. We expect AI assisted mapping, especially with RAG based grounding, to further reduce the “long tail” of manual effort.

CONCLUSION

SDTM automation does not require sacrificing transparency for efficiency. We create a compliant, scalable, **glass-box** architecture by combining:

- deterministic dataset operations and SQL/dataframe modeling,
- a reuse-first specification design and spec shopping,
- governed, versioned transformation functions,
- constrained AI augmentation via structured prompt context and (soon) RAG,
- parameterized derivations for complex logic,
- robust import/export with validation, and
- full engine observability from orchestrator to per-domain outputs,

AI enhances mapping design within clear boundaries. Governance remains embedded in code. Code is version-controlled and test-backed. Execution is deterministic and observable. The specification becomes the

intelligence layer of clinical transformation, while the engine ensures reproducible SDTM delivery across the study portfolio.

ACKNOWLEDGMENTS

The authors would like to thank the BARDS Statistical Programming Leadership and the Management team at Merck & Co., Inc. including Amy Gillespie, Janakiraman Gopinath, and Janet Low for their support. The authors would also like to thank Principal Scientists Donna Hyatt, Proma Debnath, Sangeeta Sama, and Srinivas Malipeddi for their contributions to the presented framework. We extend our gratitude to AWS Professional Services for their invaluable collaboration, which has significantly enhanced the development of the presented framework and its implementation.

CONTACT INFORMATION

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

Author Name: Jacques Lanoue
Company: Merck & Co., Inc.
Address: 351 N Sumneytown Pike, North Wales, PA 19454
Email: jacques.lanoue@merck.com
Website: https://jobs.merck.com/us/en/mrl-locations