NORC at the University of Chicago

REPORT
March 2026

Artificial Intelligence for Enhancing Data Quality, Standardization, and Integration Toolkit Documentation

Presented by:
NORC at the University of Chicago and Wolfram Research

Presented to:
National Center for Science and Engineering Statistics at the National Science Foundation

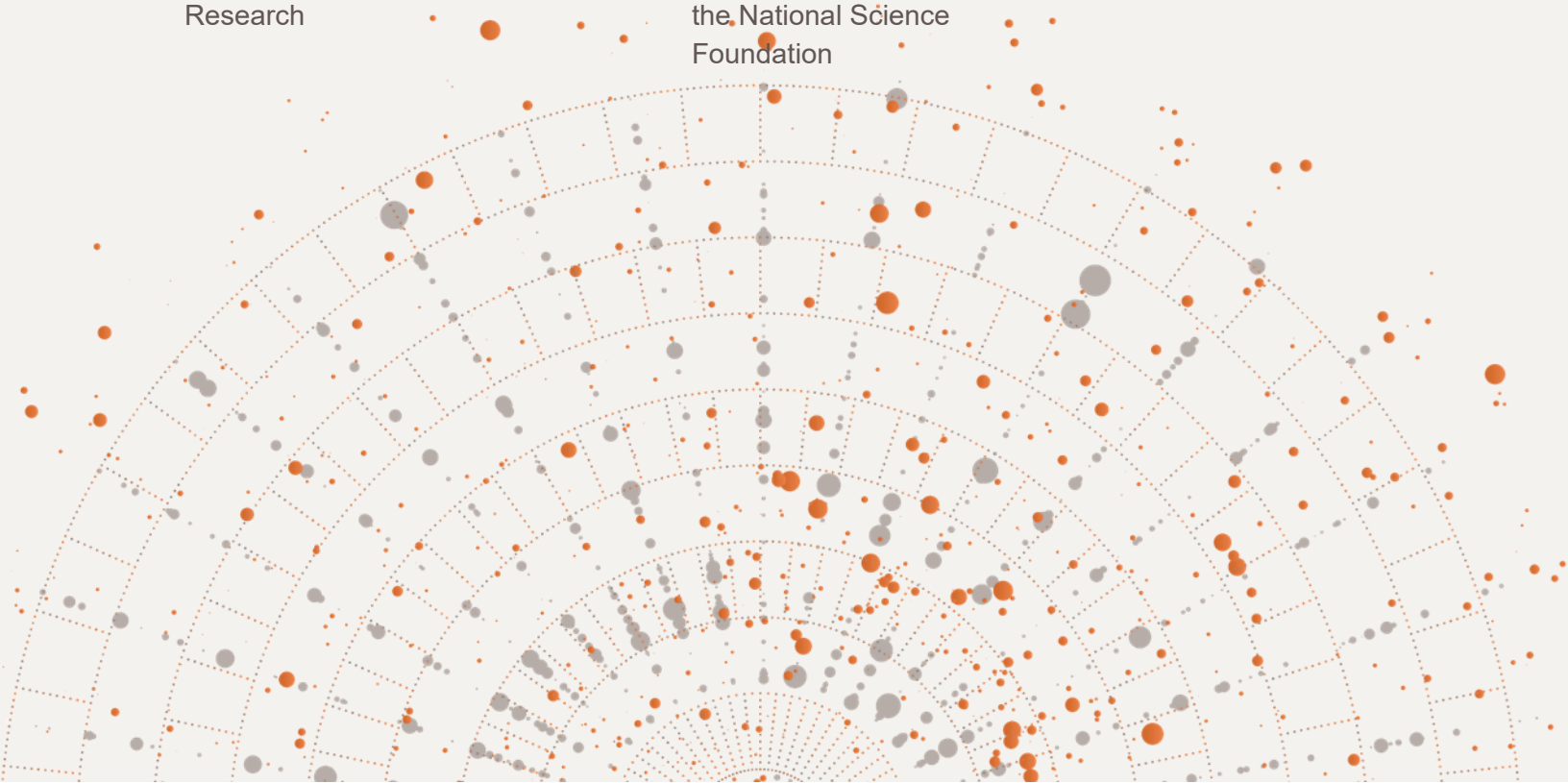


Table of Contents

List of Exhibits

List of Tables

America's DataHub Consortium (ADC), a public-private partnership, implements research opportunities that support the strategic objectives of the National Center for Science and Engineering Statistics (NCSES) within the U.S. National Science Foundation (NSF). These results document research funded through ADC and are being shared to inform interested parties of ongoing activities and to encourage further discussion. Any opinions, findings, conclusions, or recommendations expressed above do not necessarily reflect the views of NCSES or NSF. Please send questions to ncsesweb@nsf.gov. NCSES has reviewed this product for unauthorized disclosure of confidential information and approved its release (NCSES-DRN26-035).

AI-DQSI: General Overview of Toolkit

Background & Motivation

The AI-DQSI Toolkit was developed as part of a National Secure Data Service demonstration project[1] to explore how artificial intelligence can strengthen data quality, standardization, and integration (DQSI) activities across the federal statistical system. The project's Framework Plan deliverable identified persistent challenges faced by federal statistical agencies when working with survey, administrative, private sector, and geospatial data, especially when these sources must be combined for evidence building or program monitoring. Many of these sources suffer from limited documentation, inconsistencies in formatting or schemas across jurisdictions, and a lack of machine readability for data and documentation. By synthesizing insights from expert interviews, a literature scan, and tool reviews, the project team concluded that targeted AI capabilities could substantially reduce the manual burden required to explore, prepare, and integrate data. These insights motivated the development of a Toolkit designed to make such capabilities accessible, transparent, and aligned with the needs of a future National Secure Data Service (NSDS).

Federal statistical agencies increasingly rely on datasets drawn from diverse sources such as traditional surveys, administrative records, and non-traditional data such as GPS traces and web content, to build evidence and inform policy. These sources often arrive without common formatting or documentation, and many contain unstructured or semi-structured information that must be transformed before analysis. The resulting burdens include standardizing schemas, documenting variables, integrating across jurisdictions, and resolving missingness, all of which slow the production of high-quality statistics and create barriers to reuse. The AI-DQSI Toolkit was developed to address these practical needs by providing modular capabilities that assist with data preparation tasks essential to evidence building: documenting structure and content, organizing and labeling freeform text, harmonizing fields and values across files, imputing missing data transparently, and extracting structured information from documents. The tools apply AI where it adds clear value to tasks like semantic interpretation and summarization, while preserving analyst control, emphasizing human review, and supporting fitness-for-use decisions aligned with federal data quality expectations. In doing so, the Toolkit advances the broader goal of enabling a future data service environment in which agencies can more efficiently standardize, integrate, and assess data for statistical use, with attention to privacy, security, and responsible AI use.

General Guidance on Access and Use: The Toolkit can be deployed in two primary ways: (1) as a web application hosted in a secure environment for analysts, and (2) as containerized code that

[1] Artificial Intelligence for Enhancing Data Quality, Standardization, and Integration. America's Datahub Consortium. See: https://www.americasdatahub.org/award-ai-dqsi-24/

agencies can download and stand up in their own environments. Each tool is delivered as a containerized, browser-based application that can run in a local or secure hosted environment. Tools that incorporate large language models (LLMs) include a settings panel where users can supply and manage their own model credentials. The panel also displays processing metrics such as LLM utilization and estimated cost. This design enables organizations to control access to external AI services within their existing security posture, while allowing flexibility in model selection and configuration. Collectively, these design decisions align with the broader project goal of demonstrating how AI can support future NSDS workflows in a responsible and iterative manner.

Toolkit at a Glance

The AI-DQSI Toolkit contains five complementary tools that streamline documentation, structuring, harmonization, imputation, and extraction tasks in statistical workflows. Each of the tools, listed in Table 1, reflects a common challenge identified in the Framework Plan (e.g., insufficient metadata, inconsistent schemas, unstructured text, missing values, and essential information locked in documents) and provides a pathway to address that challenge efficiently.

A major strength of the Toolkit is its modularity; each tool performs a distinct function that fits into broader DQSI workflows. Tools provide detailed, auditable outputs such as harmonized tables accompanied by schema definitions, JSON metadata, cluster summaries with confidence scores, and imputation flags showing where values were replaced. At the same time, analysts retain substantial control: each tool exposes manual override options, editable groupings, customizable predictors for imputation, and parameter tuning for clustering or extraction. These capabilities allow users to apply AI-assisted methods without surrendering interpretability or methodological rigor.

Table 1. Summary of AI Toolkit tools, their core functions and AI/ML used

Tool Name	Description	Core Functions	AI/ML Used
Data Harmonizer	Align schemas and standardize values across multiple tabular datasets	Auto/manual schema generation; field grouping; value normalization; export of harmonized tables; schema JSON	LLMs (field grouping, target schema inference, optimization, and normalization suggestions)
Free Text Encoder	Structure free-text fields by clustering responses into themes or assigning them to defined categories	Unsupervised clustering; user-defined groups; optional LLM cluster summaries; confidence scoring	Local machine learning (clustering), optional LLM summaries
Metadata Extractor	Generate machine-readable metadata for undocumented tabular datasets	Table detection; variable name interpretation; type inference; representative values; summary statistics	LLMs (header interpretation, text-based inference)

Tool Name	Description	Core Functions	AI/ML Used
Missing Data Assistant	Assess and impute missing values with transparent diagnostics	Data profiling; simple imputation (mode/mean); imputation by chained equations using random forest; pre/post statistics; imputation flags	Machine learning/statistics (no LLMs)
Tabular Data Extractor	Extract structured answers or tables from semi-structured documents	Question answering; multi-document ingestion; optional Retrieval-Augmented Generation (RAG); confidence scoring; table generation	LLMs (extraction, QA); optional RAG

- The **Data Harmonizer** supports reconciliation of structurally similar datasets that differ in field naming, organization, or formatting. Unlike in traditional manual harmonization, the tool guides users through an iterative schema alignment workflow, enabling both automation and human correction. Its design emphasizes transparency, allowing analysts to review and refine proposed groupings or normalization decisions before generating a unified output.
- The **Free Text Encoder** addresses the frequent problem of free text variables that are rich in information but difficult to analyze or standardize at scale. By offering both automated clustering and user-defined categorization, the tool helps transform qualitative responses into structured variables suitable for downstream analysis, integration, or reporting. Its intended role is not to replace subject matter insight, but to accelerate exploratory analysis and assist in building defensible taxonomies.
- The **Metadata Extractor** provides an entry point for making sense of unfamiliar or poorly documented data. It provides a rapid, automated way to surface the structure of a dataset, including variable names, types, and example values, so that users can assess fitness for use, identify anomalies, and plan workflows. Because many federal statistical tasks rely on clear and complete metadata, this tool helps reduce the initial overhead often required to understand and validate new data sources.
- The **Missing Data Assistant** focuses on imputation, which is essential for minimizing bias and maintaining analytic integrity when working with imperfect datasets. Rather than enforcing a single approach, the tool gives analysts access to a spectrum of methods, from simple substitutions to advanced modeling. Its interface is designed to help users select a method, understand underlying assumptions, and document the effects of imputation through diagnostics and flags.
- Finally, the **Tabular Data Extractor** enables the transformation of narrative or semi-structured documents into analysis-ready tables. It is especially useful when statutory reports, public notices, or program descriptions contain information critical for integration or linkage. By pairing question-answering capabilities with an interface tailored to producing structured outputs, the tool helps analysts access information that would otherwise require significant manual review.

Together, these tools support the common components of data documentation, structuring, harmonization, imputation, and extraction in DQSI workflows across federal statistical agencies.

How the Tools Fit Together

The five tools can be used independently or in sequence as a multi-stage data preparation pipeline, with analysts carrying artifacts (for example, data dictionaries, grouped fields, and imputation flags) from one step to the next.

1. An analyst can start a workflow with the **Metadata Extractor**, which generates a baseline understanding of variables, field types, and representative values from raw files. This step provides a starting point for assessing data readiness, identifying anomalies, and planning harmonization work.
2. Next, the analyst can use the **Free Text Encoder** to structure any open-ended or descriptive fields found within the data. By clustering or categorizing these values, the tool enables qualitative fields to be treated as structured variables that can be used in harmonization or integration.
3. Once documentation and text encoding are complete, the **Data Harmonizer** can be used to align datasets. It allows users to define or accept automatically proposed schemas, group analogous fields, normalize values, and generate merged tables that represent consistent structures across all files.
4. After harmonizing schema and values, the resulting datasets may contain missing fields due to incomplete or inconsistent reporting. The **Missing Data Assistant** can help analysts impute missing values, apply advanced modeling where appropriate, and flag imputed entries for auditability.
5. In cases where source datasets do not contain all needed information, the **Tabular Data Extractor** provides the means to create structured tables from supplemental documents. Users can ask targeted questions and receive structured outputs suitable for linking back to harmonized datasets.

Figure 1. Example scenario illustrated for an analyst working with Chicago crime data

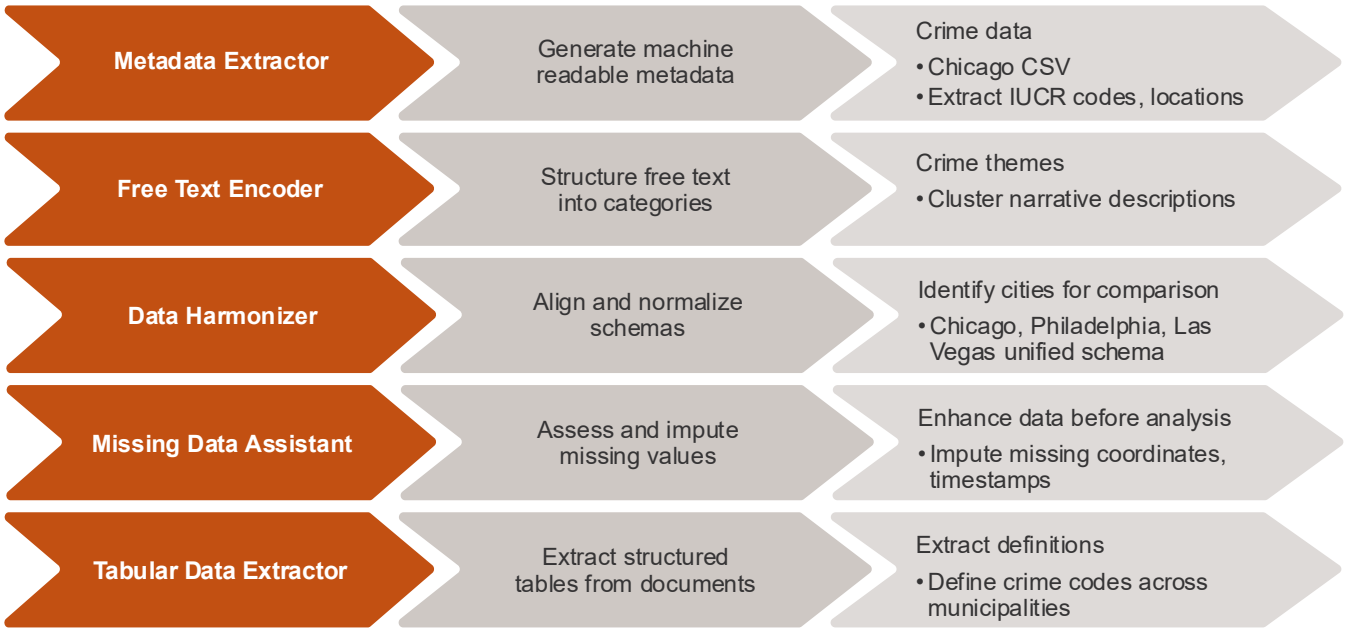


Figure 1 illustrates how the AI-DQSI Toolkit can be applied to a multijurisdictional data integration scenario. Imagine an analyst who must integrate crime data from Chicago, Philadelphia, and Las Vegas to support a multi-city safety analysis. Each dataset arrives with its own structure, field naming conventions, metadata gaps, and inconsistencies. The Toolkit provides coordinated tools that streamline this process from initial documentation through final harmonization and enhancement.

A compilation of all public datasets referenced throughout this report, and their web locations where available, can be found in Appendix B: Dataset Reference Table.

1. The analyst loads Chicago's Crimes – 2001 to Present dataset into the **Metadata Extractor**. The tool identifies dataset variables (IUCR codes, primary crime type, location description, date/time fields, district and beat information) and generates a machine-readable summary of each field. This automatically generated metadata gives the analyst an overview of the types of values that are present and where inconsistencies or ambiguities may appear. This same step can then be repeated for Philadelphia and Las Vegas, ensuring that all incoming data sources are documented to a consistent baseline before any integration occurs.
2. Many crime datasets include narrative or descriptive fields written in free text. To make this information usable in downstream integration and analysis, the analyst sends such fields (e.g., short case summaries from Chicago's incident reports) into the **Free Text Encoder**. In automated clustering mode, the tool groups narratives into interpretable themes such as theft, assault, narcotics activity, or domestic incidents. Alternatively, the analyst can specify custom categories. The Free Text Encoder then assigns each record to one or more categories, transforming qualitative descriptions into structured variables. These structured labels feed directly into later harmonization steps.
3. With metadata documented and text fields structured, the analyst now loads all three cities' crime datasets into the **Data Harmonizer**. Through a combination of AI-supported schema inference and user oversight, the tool proposes a unified set of variables that capture key crime attributes across jurisdictions. For example, "Beat" in Chicago may correspond to "District" in Philadelphia, even though the naming differs. Las Vegas data may encode location or crime-type fields differently, requiring grouping or renaming to achieve consistency. The tool also proposes value normalization (e.g., aligning timestamp formats or standardizing location types). The analyst reviews and adjusts these proposals, ensuring that the resulting schema reflects both conceptual equivalence and analytic validity. The output is a unified, city-agnostic crime dataset.
4. Once the data are harmonized, the analyst may find missing values in critical variables, such as coordinates, incident dates, or jurisdictional identifiers. The **Missing Data Assistant** provides a structured way to address these gaps. Analysts can choose among several imputation strategies, including simple imputation (for example, mode or median) and iterative multivariate methods based on chained equations. For example, if Chicago records frequently omit coordinates in certain years, the tool can infer missing values based on similar incidents. Importantly, the tool adds transparent imputation flags, enabling analysts and reviewers to track which values were estimated. A built-in profiler summarizes pre- and post-imputation statistics for each variable, allowing analysts

to evaluate distributional changes and assess the effects of the chosen method. Pre-imputation summaries can also be downloaded to incorporate in data documentation and for quality review.

5. Sometimes datasets do not contain all the contextual information needed for analysis. For example, when comparing multiple cities, an analyst may need to know which municipalities had specific policing policies in effect, such as officer training requirements, use-of-force policies, or community engagement practices, during a given time period. The **Tabular Data Extractor** can be used to extract such structured information from policy documents or multi-jurisdictional reports. Supplemental tables can then be fed back into earlier steps (e.g., Metadata Extractor for documentation or Data Harmonizer for integration) so the policy context can be joined to the underlying records for analysis.

In the final stage, the system produces a harmonized, metadata-rich, and fully imputed dataset that combines Chicago, Philadelphia, and Las Vegas crime records into a single analytic resource. Intermediate artifacts such as metadata files, schema definitions, structured text labels, imputation flags and summaries, and extracted tables are stored alongside exports to support auditing, quality review, and later reuse. The analyst can now proceed to statistical modeling, visualization, or reporting, confident that the data have been consistently documented, standardized, and enhanced.

Considerations for Using the Toolkit

Audience, Scope, and Exclusions

The Toolkit is intended primarily for analysts such as data scientists, statisticians, research methodologists, and program analysts responsible for preparing datasets for research, public-use files, or integration activities. While advanced users can take full advantage of parameter tuning, schema editing, custom grouping, or multivariate imputation, the tools are also accessible to entry-level analysts who need first pass structure or exploratory insights before deeper processing. By design, the Toolkit provides functionality that agencies routinely need when curating administrative or survey data for downstream statistical use. For example, the Data Harmonizer exports SQL statements for seamless loading into databases for advanced users, and the Free Text Encoder reports cluster quality metrics to help non-specialists interpret results.

The Toolkit is particularly well-suited for applications involving administrative data integration, where datasets representing similar concepts like crime incidents, building permits, or program participation must be harmonized across states, counties, or time periods. The Data Harmonizer excels in these scenarios. Analysts working with open-ended survey responses or descriptive administrative fields will benefit greatly from the Free Text Encoder, which helps create structured variables suitable for statistical analysis or integration. The Metadata Extractor provides value when analysts encounter datasets without documentation, and the Missing Data Assistant supports datasets with patchy or incomplete coverage.

Conversely, the Toolkit is less suitable for tasks involving unstructured image-based data, or workflows requiring advanced geospatial analytics such as raster processing, segmentation, or spatial deep learning. Tools relying on PDF or Excel parsing may also struggle with heavily formatted documents that do not resemble well-structured tables, such as those with merged header cells or multi-row column labels. Consequently, analysts working with complex documents may need to pre-clean or reorganize such inputs to optimize results.

This documentation provides performance recommendations and best practices for each tool. Operations relying on LLMs can be slow, especially when large datasets must be passed to a model. The tools' file size limits reflect these constraints, and although row-dense datasets scale well, column-heavy datasets may run slowly. Documentation therefore emphasizes performance recommendations and best practices, such as disabling LLM value normalization for large harmonization jobs or preferring simple CSV formats when possible.

Responsible Use of AI in the Toolkit

AI is used selectively across the Toolkit to support specific functions. Tools such as the Data Harmonizer, Free Text Encoder, Metadata Extractor, and Tabular Data Extractor rely on LLMs hosted in AWS Bedrock to perform schema extraction, semantic grouping, column name interpretation, narrative-to-table generation, and cluster summarization. By contrast, the Missing Data Assistant relies entirely on conventional statistical and machine learning methods, such as the random forest algorithm. This division ensures that LLMs are used only where they provide unique value, while core statistical tasks remain transparent and replicable.

Users are advised to minimize the amount of sensitive information sent to LLMs, rely on exemplar-based or metadata-based prompts rather than passing full datasets, and use secure AWS environments that enforce least privilege access. The documentation encourages human-in-the-loop review for all AI-generated outputs, especially where decisions could impact data quality, equity, or policy conclusions. Bias mitigation strategies, disclosure risk considerations, and model transparency are prominently featured to ensure that the tools support trustworthy, accountable statistical work.

This documentation emphasizes the importance of capturing assumptions, parameter settings, model versions, and human review steps throughout these workflows. Where available, tool-level profilers and progress logs should be used to record per-stage timings and summarize resource needs, supporting repeatability and performance tuning in subsequent runs.

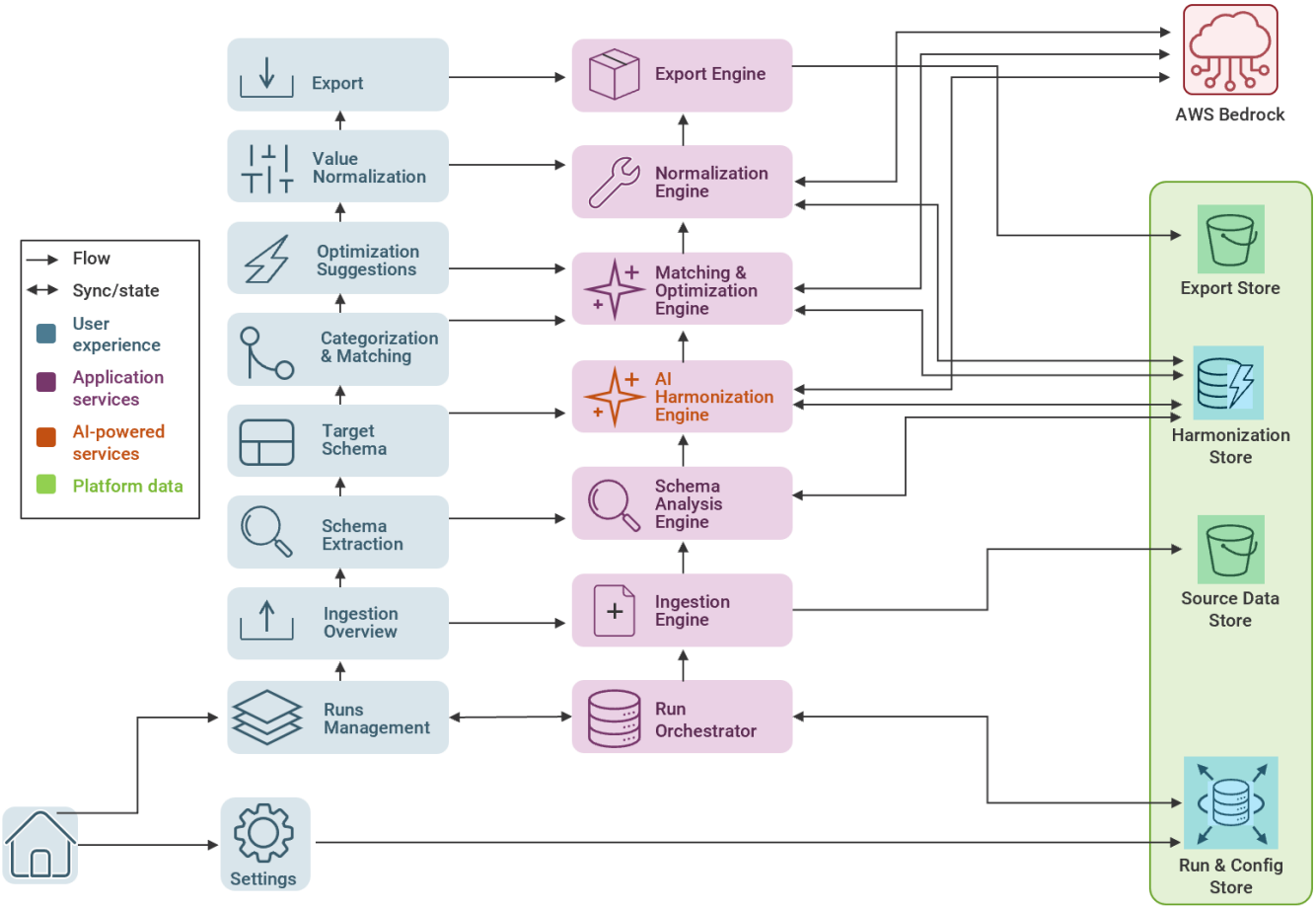
Data Harmonizer: Tool-Specific Documentation

The Data Harmonizer is designed to standardize heterogeneous tabular datasets that represent the same underlying concepts but differ in naming conventions, structure, and formatting. The tool provides an end-to-end pipeline for schema extraction, LLM-assisted field grouping and categorization, value normalization, and harmonized export, with persistent run management that preserves inputs, artifacts, and outputs per session. By leveraging a hybrid combination of AI-assisted semantics, embedding-based clustering, and human-guided review, the Data Harmonizer automates much of the tedious manual work involved in dataset integration while ensuring that analysts maintain full visibility and control over final decisions.

The tool addresses several pain points that commonly arise in harmonization workflows. In manual or rule-based approaches, analysts must painstakingly review field names and sample values to infer semantic equivalence, often relying on domain knowledge or inconsistent documentation. Heterogeneous datasets, such as crime reports from multiple cities, use varying terms to describe equivalent concepts, and these differences can cause analysts to misclassify fields or overlook subtle distinctions. Manual harmonization is time-consuming, error-prone, and difficult to repeat consistently. The Data Harmonizer overcomes these shortcomings by using LLM-powered semantic analysis to interpret field names and values, reducing errors linked to superficial name matching and providing a scalable approach that can handle diverse jurisdictions and domains.

Target Users, Use Cases, and Limitations: The primary users of the Data Harmonizer are data scientists, statisticians, and research methodologists responsible for preparing data for evidence-building, program evaluation, or integration into larger data infrastructures such as the envisioned National Secure Data Service (NSDS). The tool is particularly effective for multi-jurisdictional administrative data (e.g., crime datasets or building permit records from different municipalities) where files share conceptual structure but differ in schema and encoding. While powerful, the tool has limitations: it performs best on rectangular tabular data and does not currently support nested JSON structures, multi-table relational joins, or formats requiring extensive pre-cleaning, such as merged header rows. Whether the Data Harmonizer may be used with restricted data also depends on deployment context and agency governance policies.

Figure 2. Data Harmonizer architecture



Technical Details

Users interact with a sequence of clearly defined steps (file upload, schema extraction, target schema definition, categorization, optimization, value matching, and finalization) that guide them through the harmonization process.

Users can define a target schema via JSON, simple form, list, template, or LLM inference. When a target schema is defined, exports include only target-schema fields; otherwise, exports include the selected groups. The Settings panel allows users to enable or disable LLM features, select the specific Claude model, configure temperature and maximum tokens, and test and save the LLM connection for subsequent steps.

Supported Input

The Data Harmonizer supports CSV and Excel (XLS, XLSX) as first-class input formats for ingestion, preview, schema extraction, and harmonization. Other formats such as Parquet, SQL tables, or JSON are not supported and should be converted to CSV or Excel prior to ingestion.

While the tool is well-suited for ingesting tabular data originating from a wide range of programmatic and administrative sources, these datasets generally share a similar structural profile: a set of rows representing events, individuals, or transactions, and columns representing attributes of those entities. To produce high-quality results, the Data Harmonizer assumes that input files contain a header row and that each column name, even if cryptic or abbreviated, reflects a meaningful data attribute, as these names form the basis of semantic interpretation during schema extraction.

The tool benefits from the availability of sample values within each column because these values help the LLM interpret the semantics of each field. For best results, the user should ensure files have consistent encodings. The Harmonizer profiles each column (type hints, percentage of nulls, uniqueness, sample values) before grouping. Beyond the presence of headers and basic metadata, there are no strict prerequisites, although datasets with sparse information may yield poor automatic groupings.

In the Target Schema step, users have the option to upload accompanying field documentation (e.g., codebooks, data dictionaries, schema docs) to enhance matching. The user can provide one or more PDF, TXT, DOCX, HTML, MD, Markdown, JSON, or HTM documents that describe the data fields. The system uses this context to improve field grouping accuracy during categorization. Relevant sections are extracted from this documentation and added to the LLM prompt to improve field grouping decisions.

Logging and Reproducibility

The tool preserves inputs, intermediate artifacts, and outputs on a per-run basis using persistent run directories. Each harmonization run is stored in a dedicated folder that contains uploaded input files, schema extraction results, field groupings, optimization outputs, normalization plans, exported datasets, and a time-stamped progress log. These artifacts are written to disk and remain available across user sessions as long as the application is deployed with persistent storage for the run directory.

When a user logs out or closes the application, previously completed runs are not lost as long at the application persists. Run Management supports creating, loading, reviewing, and archiving runs, supporting exploratory in-the-loop workflows where analysts may revisit results over multiple sessions. Reproducibility is supported through the preservation of structured run artifacts (e.g., schema extraction files, group definitions, normalization rules, SQL exports). Users are strongly encouraged to download and archive final outputs and run bundles for long-term storage, sharing, or reuse.

Run persistence depends on retaining the underlying run directory; users operating the tool in containerized or unstable shared environments should ensure that the /runs directory is mounted to persistent storage. If the application is redeployed without preserving this directory, previously created

runs will no longer be accessible. The tool is best suited for interactive analyst workflows with reproducibility driven by saved artifacts and exports.

Model Architecture

System Architecture and Execution Environment

The Data Harmonizer runs as a Dockerized Streamlit application. The container publishes the application port and mounts host volumes for /data (input datasets) and /runs (artifacts and exports) so that files and generated outputs persist across container restarts. The architecture combines deterministic preprocessing, embedding-assisted similarity analysis, LLM-based semantic reasoning, and human review to harmonize schemas across heterogeneous datasets.

The system uses a hybrid methodology that combines LLM-based semantic interpretation, embedding-assisted similarity analysis, and human review. LLM-assisted features are configurable in the Settings panel where users select the model and adjust parameters such as temperature and maximum tokens that influence suggestion behavior.

LLM access is provided through AWS Bedrock. The primary reasoning model is typically Claude Sonnet with Claude Haiku used as a fallback for reliability. Temperature is configured at 0.7 and Top P is not explicitly provided in the request body. Model identifiers, prompt templates, and configuration parameters are stored in a versioned settings file so that experiments remain reproducible.

Data Ingestion and Dataset Profiling

During ingestion the application performs several preprocessing steps. It first generates a dataset manifest that records file level metadata including file names, row counts, column counts, and selected inputs. The system then builds a column-level profile for every dataset field, capturing inferred data type hints, null percentages, uniqueness ratios, and representative sample values. These metrics are written as structured run artifacts so analysts can review intermediate outputs and verify schema extraction before grouping and normalization.

Metadata context files such as PDFs, TXT files, or data dictionaries may optionally be uploaded to provide background information. These documents are segmented using a manual character level sliding window with a chunk size of 500 characters and a 50-character overlap. Tables within metadata documents are flattened to plain text before segmentation. Chunk embeddings are generated using Amazon Titan Embed Text v2 with a dimensionality of 1024. Retrieval uses cosine similarity implemented through scikit learn and returns the top five contextual chunks when metadata is available.

Token usage is managed primarily through retrieval limits and output constraints. Claude Sonnet provides a context window of approximately 200,000 tokens. Retrieved metadata is capped at five chunks and truncated to 600 characters each. Output limits are typically 12,000 tokens for grouping

tasks and 4,000 tokens for other operations. Input prompts are further reduced by replacing long filenames with short aliases such as file1 or file2.

Semantic Schema Alignment and Post Processing

The schema alignment workflow centers on an LLM that interprets column names together with sample values to generate semantic descriptions of dataset fields. These interpretations allow the system to propose canonical field names that generalize across datasets even when terminology differs significantly. To reduce LLM workload and improve scalability, the system performs embedding-based similarity comparisons between fields. Similar fields are clustered into conceptual groups such as “temporal” or “geography” before canonical mapping begins.

Cluster assignment uses a combination of embedding similarity and historical cluster examples from previous runs. New fields may be attached to existing clusters or placed into new clusters when similarity thresholds are not met. Analysts can review cluster assignments in the interface and manually correct groupings before canonical schema generation proceeds. This human-centered step ensures that domain knowledge can override automated suggestions when necessary.

Prompt construction uses a single dynamic template rather than separate system and user prompts. The prompt injects the JSON representation of all dataset fields, defines grouping rules for identifying semantically equivalent columns, and instructs the model to output structured JSON containing groups of related fields and a confidence score between 0.0 and 1.0 for each group. If metadata context exists, the retrieved chunks are appended as supplementary context. Prompt templates are stored in Data-Harmonizer/llm/prompts.py.

Two independent confidence signals are recorded. The LLM outputs a self-reported confidence value for each semantic group, and the application computes the arithmetic mean across groups to produce a session-level confidence metric. Retrieval confidence is derived from cosine similarity scores between query and metadata chunks. These metrics remain in separate signals rather than being merged into a single value.

After semantic groups are produced, the system performs deterministic post-processing and validation. The tool proposes canonical names and mappings, records user edits for auditability, and applies schema alignment steps including renaming columns, reordering them to canonical order, inserting missing canonical fields, and removing unmapped fields. Value normalization rules can be defined per column, such as lowercase conversion, whitespace trimming, spaces to underscores, ISO-8601 date parsing, boolean conversion, numeric conversion, and categorical slugification. Transformations can be previewed on sample rows so analysts can validate expected changes before exporting the harmonized dataset.

All schema alignment outputs, including semantic field groupings, canonical mappings, user edits, and normalization rules, are preserved in the /runs volume as part of the run state and remain available when the user logs in again, provided the application is deployed with persistent run storage. Users can

reload these runs through the Runs Management interface or export a complete run bundle for long-term retention and reproducibility. A preview dataset is generated so users can verify schema alignment and value normalization before exporting the final harmonized dataset.

Performance Considerations

Evaluation

Internal NORC testing adopted an iterative evaluation methodology, drawing on a centralized dataset inventory and a shared space for documentation and issue logging. Testing included verification of expected field groupings, inspection of value transformation rules, and assessment of runtime performance. Per-step progress logs and structured artifacts were written to each run directory, allowing analysts to reconstruct processing steps, verify grouping and normalization decisions, and replicate outputs.

Evaluation of the Data Harmonizer centered on a diverse set of real-world datasets chosen to reflect the challenges of cross-jurisdictional and cross-domain integration. Crime datasets from Chicago, Philadelphia, Las Vegas, Washington DC (2021 and 2023), Madison, Cincinnati, Raleigh, and Scottsdale served as the primary evaluation domain. Additional test cases included building permit data from Los Angeles, New York City, and Chicago, to assess generalizability beyond crime. These datasets were selected intentionally to represent a mix of well-documented administrative sources and more challenging files with ambiguous metadata, reflecting real-world conditions that federal analysts routinely encounter. Crime data, for example, lacks a universal standard schema and exhibits large variations across cities, making it an ideal testbed for semantic schema matching approaches. The presence of specialized terminology (e.g., IUCR, NIBRS), diverse location field formats, and multiple date/time formats helped stress-test the LLM's semantic reasoning.

Performance Metrics

Output Quality: Output quality can be evaluated based on field mapping accuracy, the degree of schema coverage, and the correctness of canonical value normalization. Tests show that the tool performs well in identifying broad semantic groupings such as dates, locations, and offense categories but experiences variability in more ambiguous cases, particularly when fields share names but differ in meaning or when sparse columns provide insufficient context. The tool's JSON artifacts (e.g., schema_extraction.json, groups.json, normalization_plan.json) can be consumed by external QA scripts to compute mapping accuracy, schema coverage, and normalization correctness. The tool also produces transformation rules as an optional SQL export, which supports auditing.

Computational Performance: Processing time varies with dataset size, selected features (for example, LLM-assisted steps), and the breadth of field grouping and normalization. The interface provides progress indicators for each major stage. For larger inputs, individual steps can take several minutes.

Cost Guidance

Two workflows were tested in the Data Harmonizer to develop cost guidance. The first workflow focused on harmonizing municipal crime data from Chicago, Las Vegas, and Philadelphia; the second workflow featured harmonizing building permit data from Los Angeles, New York City, and Chicago. Details from each run are summarized in the table below.[2] These scenarios are intended to provide cost guidance by showing token usage for each run along with the total number of tokens, LLM call, computational runtime, and total cost. Session metrics are exposed for users in a tool panel to help monitor processing performance and costs.

[2] The tests in the table were run on March 12, 2026.

Table 2. Cost guidance for Data Harmonizer

Input File Name(s)	Input File Description(s)	Input File Size(s)	Run Parameters	Tool Runtime	LLM Version	Total Tokens	Total LLM Calls	Total Cost
Chicago 2025 Crime LVMPD NIBRS Crimes Philadelphia Crime	Crime data in Chicago in 2025; Las Vegas; and Philadelphia from municipal open data portals (3 CSVs)	Chicago: 12.1 MB; 52K rows Las Vegas: 20.4 MB; 160K rows Philadelphia: 19.3 MB; 134K rows	Temperature: 0.7 Max Tokens: 4,000 Target Schema: LLM Inference; Use Metadata Files to Improve Matching	Steps: Ingestion: 67 sec Schema extraction: 28 sec Target Schema: 12 sec Categorization & Matching: 55 sec Optimization: 15 sec Value normalization: 35 sec Export: 180 sec Total Computational Time: 102.62 sec LLM Call Time: 102.60 sec	Claude Sonnet 4.5 region us-east-1	39,343	16	$0.1624
LA Build Permits NYC Build Permits Chicago Build Permits	Building permit data (Data.gov) for NYC, LA, Chicago (3 CSVs)	NYC: 4.5 MB; 10K rows LA: 26.8 MB; 25K rows Chicago: 12.3 MB; 12K rows	Temperature: 0.7 Max Tokens: 4,000 Target Schema: LLM Inference	Steps: Ingestion: 9 sec Schema extraction: 2 sec Target Schema: 12 sec Categorization & Matching: 33 sec Optimization: 14 sec Value normalization: 66 sec Export: 63 sec Total Computational Time: 146.83 sec LLM Call Time: 146.81 sec	Claude Sonnet 4.5 region us-east-1	93,378	24	$0.3207

Limitations and Failure Modes

Common failure scenarios include:

- Headers are missing or improperly formatted. This can cause the tool to misinterpret fields or cluster unrelated columns together. Ambiguous or domain-specific field names may also lead to incorrect mapping suggestions, requiring users to intervene during schema review.
- LLM credentials are not configured or the connection test fails. In this scenario, LLM-dependent steps (for example, field grouping suggestions, target schema inference, optimization suggestions, and normalization suggestions) will be disabled, and the workflow proceeds with non-LLM features. In such cases, LLM-assisted features (grouping, suggestions, schema inference) will be unavailable.
- Value normalization may fail silently if the tool cannot detect consistent encoding patterns across datasets.

Overall, robustness to noise, typos, and inconsistent formatting varies across steps. While LLM-based semantic reasoning provides some resilience to messy naming conventions, the tool's ability to group fields correctly can degrade when sample values are sparse, non-representative, or heavily corrupted. Non-standard entries, such as free-text crime descriptions embedded in columns intended to be categorical, may also reduce accuracy. Testing revealed that some critical identifiers, such as case numbers, were occasionally omitted from canonical groups, highlighting the need for vigilant user review. The Data Harmonizer also does not perform semantic unit conversion (e.g., converting between measurement units). Such transformations should be performed in downstream processing.

Comparison with Traditional Approaches

Compared with manual harmonization in Excel or R, the Data Harmonizer offers significant efficiency gains, reducing the amount of time analysts spend scanning column names, identifying equivalences, and writing custom scripts for every new dataset. Manual processes are difficult to document and replicate, and they rely heavily on domain expertise that may not be consistently shared across teams. By contrast, the Data Harmonizer provides a structured workflow with auditable outputs, supporting greater reproducibility.

Rule-based tools such as OpenRefine are well-suited for deterministic transformations but lack the semantic reasoning capabilities required for cross-jurisdictional schema alignment. These tools often fail when field names or value encodings differ significantly across datasets, forcing analysts to encode vast numbers of brittle rules. The Data Harmonizer's integration of LLM-driven semantic understanding helps overcome these limitations by interpreting meaning rather than relying solely on string similarity, though user oversight remains critical to correct subtle misclassifications.

The tool's hybrid approach, which combines automated suggestions with user validation, offers a middle ground between fully manual and fully automated harmonization. While accuracy is not yet on par with manual work performed by highly experienced domain specialists, the tool accelerates the

process and lays the foundation for more scalable deployments. Its primary advantage is that it provides a repeatable, transparent, and documented workflow, rather than one-time manual scripting. Unlike general-purpose chat interfaces, the Data Harmonizer also integrates deterministic preprocessing, structured prompts, persistent artifacts, and auditable outputs, making it suitable for federal statistical workflows that require transparency and reproducibility.

Free Text Encoder: Tool-Specific Documentation

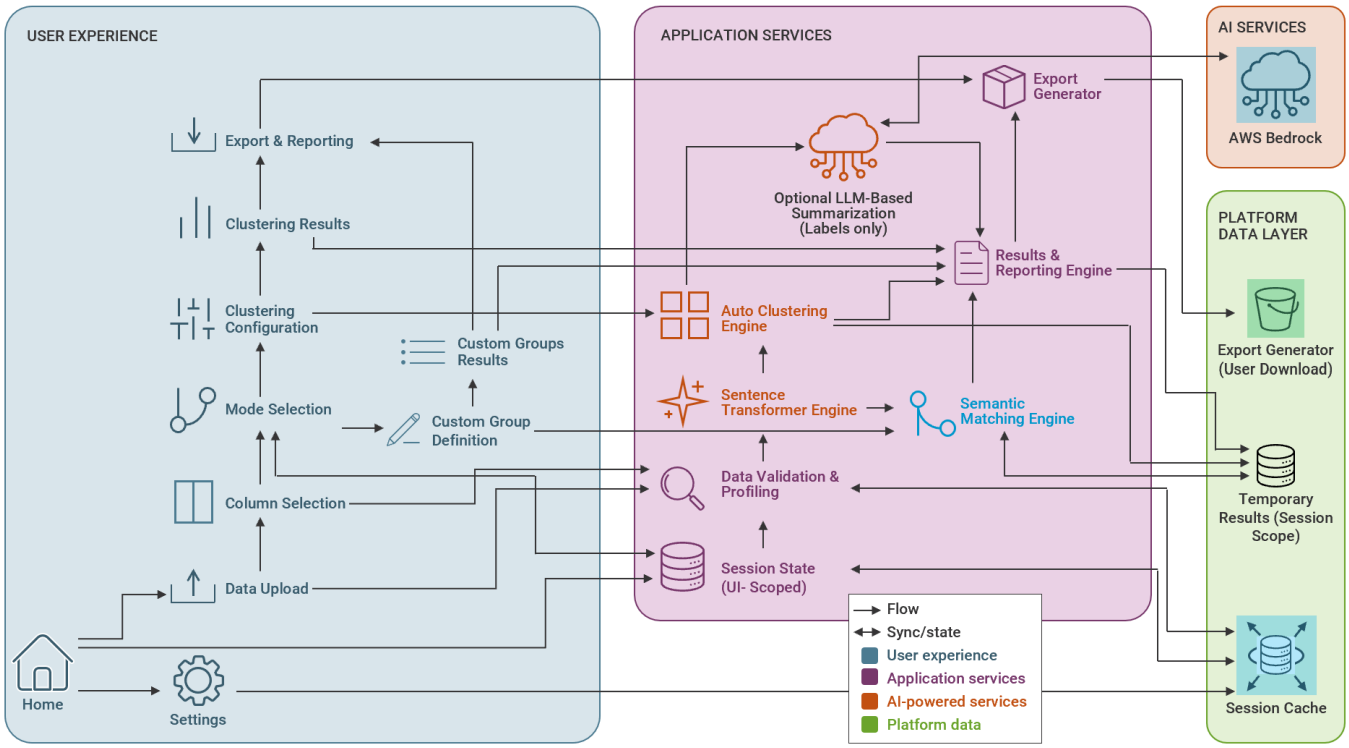
The Free Text Encoder is designed to transform unstructured textual fields within tabular datasets into structured, analysis-ready variables. Many administrative and survey datasets contain narrative fields such as open-ended survey responses, FOIA request descriptions, or permit work descriptions that traditionally require labor-intensive manual coding or rule-based categorization. This tool automates that process through two complementary modes: unsupervised clustering, which groups similar responses into emergent categories, and user-defined categorization, which assigns records to user-specified custom categories based on semantic similarity. The Free Text Encoder provides these two complementary workflows (automated clustering and custom semantic grouping) and supports configurable clustering algorithms, cluster-quality diagnostics, near-duplicate detection, interactive visualizations, and optional AI-generated cluster summaries.

The motivation for developing the Free Text Encoder arises from well-known limitations in traditional text coding workflows. Manual categorization is slow, error-prone, and inconsistent across coders, particularly when datasets contain thousands of responses. Rules-based approaches, such as simple keyword matching, often fail to capture the meaning of text that contains spelling variations, abbreviations, jargon, or multi-topic narratives. These methods also struggle to generalize across datasets from different organizations or jurisdictions. The Free Text Encoder addresses these problems by using statistical clustering techniques to detect natural groupings in the text and by applying semantic similarity models to ensure that records are matched to the correct category, even when wording varies significantly. This reduces analyst burden, improves reproducibility, and supports clearer documentation.

Target Users, Use Cases, and Limitations: The tool is intended for federal data scientists, statisticians, research methodologists, and program analysts who frequently handle narrative fields but need to convert them into structured variables for downstream analysis or integration. The Free Text Encoder is well-suited for open-ended survey responses, administrative request logs such as FOIA descriptions, program data fields containing action or work descriptions, and any dataset where text needs to be grouped into thematic categories. For exploratory analysis and automatic theme discovery, the user should start with Clustering mode; for hypothesis-driven coding against existing agency taxonomies, the user should use Custom Categories mode. Custom Categories mode operates on a single column at a time, and support for multi-column semantic fusion has not been implemented.

Tool performance can degrade when processing datasets with tens of thousands of records. In addition, users working with extremely short, noisy, multilingual, or domain-specific text may find that the clustering algorithms require experimentation to produce meaningful results.

Figure 3. Free Text Encoder architecture



Technical Details

The Free Text Encoder is delivered as a browser-based GUI built using Streamlit, which runs as a web application accessible via a local browser interface. Users upload datasets, configure clustering parameters, or define custom categories, and obtain structured outputs directly in the web interface. While configuration options are extensive, the interface is designed to guide analysts through both clustering and custom category assignments. Output files including cluster summaries, detailed assignment files, and analysis reports can be downloaded for integration into downstream workflows.

All processing occurs locally using CPU-only execution, and no GPU is required. LLM-based summarization is optional and requires API credentials. Minimum recommended system resources include multiple CPU cores and sufficient RAM for embedding generation and clustering.

Supported Input

The tool can ingest narrative data from a wide variety of sources, including administrative logs, survey datasets, program records, and private-sector datasets containing textual notes or comments. Typical use cases include open-ended survey response fields describing attitudes or experiences, administrative records where staff document actions or events, FOIA request descriptions that outline the requester's purpose, and building permit descriptions outlining the type of work performed. These

inputs provide rich qualitative information that becomes more analytically useful once transformed into structured categories. Whether the Free Text Encoder may be used with restricted or sensitive data depends on deployment context and agency governance policies.

The Free Text Encoder accepts CSV and Excel files (.csv, .xlsx, .xls). It automatically detects common encodings, including UTF-8 and Latin-1, and supports text, categorical, boolean, and numeric columns for analysis. The tool requires the presence of at least one column containing freeform text that the user intends to cluster or categorize. The tool assumes that each input file includes a header row, and that the target column is encoded as readable text without embedded markup or non-standard encodings. The tool performs automatic encoding detection and language detection, and will exclude entries with empty or invalid text during processing.

The tool performs better when text entries contain meaningful content (e.g., extremely short entries or single-word responses reduce the effectiveness of both clustering algorithms and semantic similarity scoring). While no explicit metadata beyond headers is strictly required, the presence of consistent formatting, absence of null-character encoding issues, and reasonable text length all improve result quality. Multicolumn analysis is supported in Clustering (all selected columns weighted equally); Custom Categories analyzes one text column at a time.

Input Limitations: The tool cannot process nested JSON structures or multi-value fields stored in complex formats without prior transformation. Very large datasets trigger automatic sampling for clustering, and extremely large or highly stylized files may impact performance. Multi-language support, maximum allowable text length per cell, and preprocessing of embedded markup remain unspecified and should be verified for specific use cases. Datasets larger than 10,000 rows trigger automatic sampling (default ~2,000 rows) for clustering; all records are still assigned after the model is trained.

Logging and Reproducibility

Logging and monitoring capabilities are minimal, although the tool provides progress feedback through continually updating progress bars and messages. Detailed system logs enhance usability for large-scale processing. As a prototype, the Free Text Encoder is best suited for exploratory and iterative analysis rather than fully automated production use, though it lays the groundwork for future enhancements that could support broader operational deployments. The results view includes Performance Metrics (total analysis time, cache stats, and LLM usage estimates) for each run.

The Free Text Encoder may optionally cache a small amount of run metadata locally to describe aspects of a run and improve performance between executions. This caching behavior is enabled by default but is fully configurable: users can disable caching or change the cache location via the Caching Configuration settings in the tool's settings.yaml file. However, this cached information does not constitute a full persistent run history, and users must export outputs to retain results long-term beyond the current workflow.

Model Architecture

Models and Rule Engines

The Free Text Encoder supports two primary analytical workflows: clustering analysis and custom semantic grouping. Both workflows rely on vector representations of text generated by sentence embedding models to measure semantic similarity between records.

In the clustering workflow, textual data is converted into 384-dimensional embeddings using a sentence transformer model. These embeddings are grouped using clustering algorithms including K-Means (Automatic and Advanced modes) and HDBSCAN. To support exploration and interpretation, dimensionality reduction techniques such as UMAP or t-SNE project the high dimensional embedding space into an interactive two-dimensional representation.

Once clusters are formed, analysts may optionally enable LLM summarization. When credentials are configured, the system generates human-readable cluster descriptions consisting of a short name and a concise explanation of the dominant theme within each cluster. Clustering itself remains entirely embedding-based, while LLMs are used only for post-processing interpretation. This hybrid approach combines automated grouping with human-readable explanations, so analysts can quickly understand the themes present in the data.

The application operates directly on tabular data rather than using a Retrieval-Augmented Generation pipeline. Each row of the dataset is converted into a single synopsis string that represents the row's content. Columns are processed according to their weight and data type, then concatenated using the "|" separator. The resulting string is truncated at 500 characters, using word boundary-aware truncation so that text is cut at the last whitespace before the limit and an ellipsis is appended. Rows are processed independently and no overlap or cross-row chunking occurs.

Data Preparation and Preprocessing

Input CSV and Excel files are loaded directly into Pandas dataframes, preserving the original row and column structure. To construct embedding-ready text, columns with a weight ratio below 0.1 are excluded from the synopsis string. Remaining values are converted to normalized text representations according to their data type. For example, boolean values are converted to "Yes" or "No". The formatted values are concatenated into the synopsis string used for embedding, while the original relational dataframe remains in memory for downstream processing.

Several preprocessing operations occur before embeddings are generated. The system performs encoding auto-detection for text columns and language detection when multiple languages may be present. It also generates synopsis strings for embedding input and can perform optional sampling when datasets are large. For datasets exceeding 10,000 rows, the system samples 2,000 rows for the initial clustering phase and assigns remaining rows to the nearest cluster centroid afterward to control memory usage.

Near-duplicate detection may occur as a preprocessing stage. In these cases, highly similar text entries are identified using cosine similarity and grouped before clustering. The cosine similarity threshold for near-duplicate detection is 0.92, although the precise algorithmic implementation is not publicly disclosed.

Embeddings and LLM-generated summaries may also be cached when enabled, allowing faster re-execution of the pipeline.

Embedding Generation and Similarity Computation

The primary embedding model is the *SentenceTransformers all MiniLM L6 v2* encoder, which runs locally on CPU and produces 384-dimensional vectors. Input text is truncated to 500 characters before encoding. Internally, the embedding model processes text in batches of 32, while the surrounding application processes rows in batches of 500.

After generation, embedding vectors are L2 normalized. Similarity between vectors is computed using cosine similarity, implemented as the dot product between normalized vectors. All similarity calculations occur in memory using NumPy operations. The system does not use FAISS or any persistent vector database, because clustering requires the entire embedding matrix to be resident in memory.

If the primary embedding model fails, the system automatically falls back to paraphrase *MiniLM L3 v2*. When local models are unavailable, the pipeline may optionally call AWS Bedrock embedding models such as *Titan v2* or *Cohere v3* depending on configuration.

For custom category assignment, similarity scores are computed between response embeddings and analyst-defined group descriptions. The system returns the single best matching group along with a confidence score derived from cosine similarity. Assignments are accepted only if the similarity score meets the configured threshold. Confidence values are therefore derived entirely from embedding similarity rather than LLM output.

Clustering, Visualization, and Validation

Clustering operates on the embedding vectors using either K-Means or HDBSCAN. K-Means is implemented using the scikit learn library and supports both Automatic and Advanced configuration modes. HDBSCAN provides density-based clustering that can detect irregular cluster shapes and identify noise points.

Dimensionality reduction methods including UMAP or t-SNE may be used to project embeddings into two-dimensional space for visualization or exploratory analysis. The clustering algorithm itself relies on Euclidean distance, particularly in the HDBSCAN implementation.

If HDBSCAN fails to produce clusters or labels more than 50 percent of the dataset as noise, the result is discarded and clustering is re-executed using K-Means as a fallback.

After clustering completes, the system performs several validation and analysis operations. Clustering quality metrics including the Silhouette score and the Davies Bouldin score are calculated. The system then generates cluster-level summaries, extracts top keywords, assigns responses to clusters, and produces visualizations of cluster structure.

An overall cluster quality grade ranging from A+ to F is also generated. This composite score penalizes clusters that are extremely small or highly imbalanced and provides user recommendations for improving clustering quality.

LLM-Based Cluster Summarization

Large language models are used only after clustering has completed and are limited to cluster naming and summarization. The clustering algorithm itself does not rely on LLM inference and remains entirely embedding-based.

The prompt instructs the model to analyze clusters of survey responses and return a short descriptive name consisting of three to five words, along with a one to two sentence summary capturing the cluster's main theme or sentiment. Responses are returned in JSON format containing the cluster name and summary. The model is not asked to produce confidence scores.

Cluster summaries are generated in batches of twenty clusters per request. Temperature is set to 0.3 to encourage stable outputs, and the Top P parameter is not configured.

When AWS Bedrock models such as Claude Sonnet 4.5 or Claude Haiku are used, the available context window is approximately 200000 tokens. Each LLM call is capped at a maximum output length of 4000 tokens.

The pipeline enforces a hard limit of fifteen LLM calls per execution run, allowing summaries for up to 300 clusters. If this limit is reached, remaining clusters remain unlabeled and the pipeline proceeds without generating additional summaries. Dynamic prompt truncation is not implemented.

If an LLM API request fails, the affected cluster remains without a generated summary while processing continues for the remaining batches.

The system architecture separates embedding generation, clustering, and LLM summarization into distinct stages. Embedding generation relies on a local sentence transformer encoder running on CPU. Clustering uses established machine learning libraries implementing K-Means and HDBSCAN, while dimensionality reduction may optionally use UMAP or t-SNE.

Cluster summarization uses externally hosted large language models accessed through configured credentials. Because deployments may differ across environments, explicit model identifiers are stored in configuration files rather than hardcoded in the pipeline. This approach supports reproducibility, configuration management, and governance review.

To maintain reproducibility, a fixed random seed of 42 is applied to stochastic operations including dataset sampling, dimensionality reduction, and K-Means fallback clustering. The summarization stage uses a low temperature of 0.3 to reduce output variability while still allowing minor linguistic variation. Embedding failures cascade through multiple fallback stages beginning with the primary local model, followed by a smaller local fallback model, and finally optional Bedrock based embeddings if configured.

Performance Considerations

Evaluation

Evaluation of the Free Text Encoder has focused on datasets that reflect common federal analytical tasks involving free-text fields. The Austin Cultural Centers Open Response survey dataset served as a representative case for coding narrative survey responses. Administrative datasets used for evaluation include the Chicago FOIA Request Log (Transportation category) and Chicago Building Permits data, where long-form descriptions of work performed could be grouped into thematic categories. These datasets expose diverse vocabulary, varying text lengths, and a mix of informal and formal writing styles, making them suitable for stress-testing clustering and category assignment.

NORC's internal testing approach relied on iterative validation using a shared dataset inventory, with testers examining both the coherence of cluster results and the accuracy of category assignments. Evaluation outputs include clustering visualizations, cluster-level quality metrics, near-duplicate flags, confidence scores for semantic assignments, and performance statistics including embedding cache usage and total runtime. Special attention was given to whether clusters aligned with domain expectations – for example, whether FOIA requests clustered meaningfully into groups related to video requests, traffic violations, or permit inquiries.

Performance Metrics

Output Quality: Several types of output quality indicators are available. Cluster coherence metrics such as Silhouette and Davies-Bouldin scores help analysts judge the internal consistency of machine-generated clusters. Clustering quality is assessed using Silhouette scores (-1 to 1) and Davies-Bouldin scores (0 to infinity). The tool also reports cohesion and separation metrics, cluster size distribution, and a composite letter grade indicating overall cluster quality.

Computational Performance: Performance depends on dataset size, column selection, and clustering algorithm choice. The tool includes automatic sampling for large datasets, batch-based embedding generation, and caching of embeddings and summary results to speed up subsequent runs. Memory usage is influenced by dataset size and selected visualization settings.

Cost Guidance

Three files were tested in the Free Text Encoder to develop cost guidance. The first file was a set of responses to a community survey about cultural centers in Austin. The second, larger file was a set of FOIA request logs regarding traffic incidents in Chicago. The third and largest file was a dataset of building permits issued by the City of Chicago. For all three files, different tool parameters were tested. Details from each run are summarized in the table below.[3] These scenarios are intended to provide cost guidance by showing token usage for each run, along with the total number of LLM calls, computational runtime, and total cost. These session metrics are exposed for users in a tool panel to help monitor processing performance and costs.

[3] The tests in the table were run on March 12, 2026.

Table 3. Cost guidance for Free Text Encoder

Input File Name	Input File Description	Input File Size	Run Parameters	Tool Runtime	LLM Version	Total Tokens	Total LLM Calls	Total Cost
Austin Cultural Centers Survey Data	Responses to a community survey conducted when auditing Austin cultural centers (1 CSV)	862 rows and 7 columns (268 KB)	Automatic clustering mode on "response" column Algorithm: K-Means Automatic Generate AI Summaries	Total computational time: 40.29 seconds LLM call time: 21.98 seconds	Claude Sonnet 4.5, region us-east-1	2,849 tokens	1 call	$0.0221
Austin Cultural Centers Survey Data	Responses to a community survey conducted when auditing Austin cultural centers (1 CSV)	862 rows and 7 columns (268 KB)	Custom categories mode on "response" column; 5 categories Assignment Mode: Exclusive Minimum Confidence Threshold: 0.30	13.7 seconds	Not applicable – no AI-based summarization performed, so no LLMs used			
Chicago Transportation FOIA Requests	Logs of FOIA requests made regarding Chicago traffic incidents (1 XLSX)	15,834 rows and 5 columns (1.98 MB)	Automatic clustering mode on "response" column Algorithm: K-Means Automatic Generate AI Summaries	Total computational time: 371.97 seconds LLM call time: 4.72 seconds	Claude Sonnet 4.5, region us-east-1	564 tokens	1 call	$0.0036
Chicago Transportation FOIA Requests	Logs of FOIA requests made regarding Chicago traffic incidents (1 XLSX)	15,834 rows and 5 columns (1.98 MB)	Automatic clustering mode on "response" column Algorithm: HDBSCAN Minimum Cluster Size: 8 Minimum Samples: 5 Generate AI Summaries	Total computational time: 382.43 seconds LLM call time: 11.15 seconds	Claude Sonnet 4.5, region us-east-1	1,348 tokens	1 call	$0.0102
Chicago Building Permit Data	Descriptions of authorized Chicago	121,619 rows and 115	Automatic clustering mode on "work_description" column	Total computational	Claude Sonnet 4.5,	4,591 tokens	1 call	$0.0294

Input File Name	Input File Description	Input File Size	Run Parameters	Tool Runtime	LLM Version	Total Tokens	Total LLM Calls	Total Cost
	building permits (1 CSV)	columns (89.6 MB)	Algorithm: K-Means Automatic Generate AI Summaries	time: 397.39 seconds LLM call time: 29.52 seconds	region us-east-1			
Chicago Building Permit Data	Descriptions of authorized Chicago building permits (1 CSV)	121,619 rows and 115 columns (89.6 MB)	Automatic clustering mode on “work_description” column Algorithm: K-Means Advanced Number of Groups (K): 12 Generate AI Summaries	Total computational time: 457.74 seconds LLM call time: 15.27 seconds	Claude Sonnet 4.5, region us-east-1	2,736 tokens	1 call	$0.0173

Limitations and Failure Modes

Common failure scenarios include:

- Encoding detection resolves most text-format irregularities, but heavily corrupted files may still fail during ingestion.
- Clustering quality may degrade when text entries are extremely short, repetitive, dominated by noise, or include many numeric symbols. Clustering interpretability can also degrade when narrative fields include multiple unrelated topics.
- Very large datasets may result in longer processing times or reduced visualization fidelity due to sampling or point-limiting settings. For large datasets, clustering algorithms may run slowly or appear to stall without clear progress indicators.
- In Custom Categories mode, low-confidence assignments or excessive unassigned records may occur if the user-specified categories are either too broadly or too narrowly defined.
- Duplicate detection exists but behaves inconsistently under heavy duplication.

Together, these limitations indicate the need for preprocessing or careful parameter tuning when the user is working with messy or heterogeneous text fields.

Comparison with Traditional Approaches

Traditional approaches to text coding rely on manual review, where analysts read and categorize responses by hand, or rule-based systems that apply keyword or regex-based classification. While manual coding can be precise when conducted by trained specialists, it is slow, labor-intensive, and difficult to scale beyond small datasets. Rule-based systems offer reproducibility but lack the semantic understanding necessary to handle linguistic variation or emerging vocabulary. OpenRefine and similar tools support deterministic transformations but do not provide semantic grouping or clustering.

The Free Text Encoder improves upon these methods by combining machine learning-based clustering with human interpretation of cluster outputs. It also allows analysts to define custom groupings while leveraging semantic similarity models to assign records consistently. Unlike manual coding, the tool is scalable to tens of thousands of records, and unlike purely rule-based systems, it adapts to variations in language use. Its hybrid design maintains transparency and auditability by exposing confidence scores, quality metrics, and LLM summaries. Unlike general-purpose chatbot interfaces, the Free Text Encoder also provides a structured, scalable workflow with quantitative quality diagnostics, reproducible embeddings, and explicit review by analysts, which are capabilities not available in standalone LLM tools.

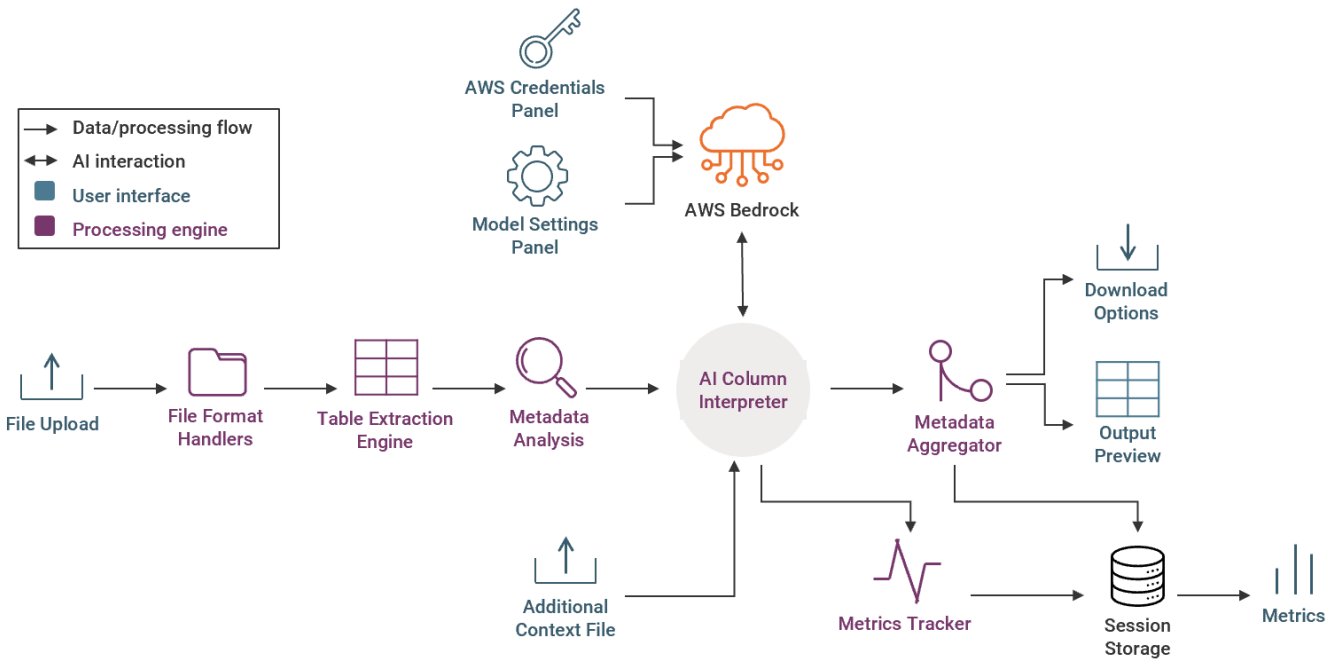
Metadata Extractor: Tool-Specific Documentation

The Metadata Extractor provides analysts with an automated way to generate structural and semantic metadata for tabular datasets and documents that contain embedded tables. Its core function is to detect tables, extract their structure, infer column types, compute representative statistics, and produce a consolidated, machine-readable data dictionary. The tool supports CSV and Excel inputs and includes optional AI-assisted expansion of abbreviated or coded column names into clearer descriptions, providing a rapid first-pass overview when no codebook is available.

The Metadata Extractor was created to address pain points that arise when analysts receive partially documented or undocumented datasets. Manual inspection of raw CSV files is extremely time-consuming, and errors are common when analysts attempt to reconstruct variable definitions by hand. Rule-based scripts can extract column names but rarely provide additional context, such as inferred data types, representative values, or semantic interpretations. The Metadata Extractor automates these steps by applying table-detection logic and AI-assisted header interpretation, significantly reducing the time needed to assess dataset structure and improving reproducibility across analysts and projects.

Target Users, Use Cases, and Limitations: The tool is intended for federal data scientists, statisticians, data stewards, researchers, and analysts responsible for dataset intake, documentation, and preparation. It is particularly useful when working with administrative or program datasets provided by external partners, open data portals, or public reports. The Metadata Extractor provides value whenever a dataset must be evaluated for fitness for use or integrated into a broader evidence-building workflow. But the tool has limitations: complex spreadsheet structures like multirow headers, merged cells, and pivot table formats may not be parsed correctly. Processing performance also diminishes for large or column-dense files, and the tool's reliance on LLM-based interpretation means users should review outputs carefully to catch semantic misclassifications.

Figure 4. Metadata Extractor architecture



Technical Details

The tool allows users to upload files, trigger metadata extraction, preview detected tables, and export results.

The tool makes use of LLMs for header interpretation and expansion of column names; when this feature is enabled, the tool requires AWS Bedrock credentials. However, basic metadata extraction (i.e., table detection, type inference, and identification of representative values) does not rely on LLMs, and thus does not require a connection to any external compute resources. The prototype has a limit of 1 GB per input file, and users should expect slower performance as file size approaches this boundary.

Supported Input

The tool ingests data from a range of sources, including municipal crime reports, tabular public records, program documentation, and public regulatory notices. These data sources tend to exhibit substantial variation in formatting, documentation quality, and table layout, making automated metadata extraction particularly beneficial. The tool helps analysts quickly understand the structure of unfamiliar datasets and identify variables that may require further cleaning, recoding, or harmonization.

The Metadata Extractor tabular inputs in CSV, TSV, Excel (XLS, XLSX) formats. Other formats such as Parquet, SQL exports, or JSON are not supported and should be converted to CSV, TSV, or Excel prior to ingestion.

Certain prerequisites improve extraction quality and stability. The first row of each table must contain headers to ensure accurate column identification and type inference. Encodings such as UTF-8 and consistent delimiters help ensure accurate parsing. Embedded tables must be visually identifiable with consistent rows and columns; narrative text formatted to “look like” a table may not be detected. Sufficiently populated columns are also beneficial, because representative values and type inference depend on observing a range of actual data points.

Users also have the option to upload accompanying documentation (e.g. survey instruments or descriptions of variables) that provide additional context on the dataset. The user can provide a maximum of one such TXT or JSON file. Relevant information is extracted from this documentation and added to the LLM prompt to enrich the metadata generated by the tool.

Input Limitations: Complex Excel files with merged header cells, rotated text, or nested column groups can lead to inaccurate metadata extraction. Complex or rotated headers and highly stylized layouts may require manual review and correction in the exported data dictionary.

Logging and Reproducibility

The application indicates when a previously analyzed file is served from cache and provides an inline preview of the generated PDF report for immediate inspection. Detailed diagnostic logs are not exported, and failed extractions should be rerun after adjusting file preparation steps. The tool generates JSON artifacts for schema and metadata extraction, but does not provide detailed diagnostic logs, error recovery, or autosave for partial runs. The Metadata Extractor does not persist run artifacts or metadata files; users must download generated outputs to retain results.

Model Architecture

The Metadata Extractor analyzes tabular datasets and generates structured metadata describing the detected tables and their columns. After tables are identified, the system performs column-level analysis to understand the structure of the dataset. Column headers are extracted and reconstructed when necessary.

Preprocessing Steps

Before metadata analysis begins, the system performs preprocessing to ensure that the dataset can be parsed reliably and analyzed consistently. This stage includes automatic file encoding detection, header extraction, detection of multiple tables within a file, and sampling of representative values from each column. When a file is uploaded, the system computes a hash of the file contents. This hash allows the application to detect whether the same file has already been processed during the current session. If so, cached results may be reused to accelerate subsequent runs.

For each detected table, the system extracts a small set of representative values from each column. Up to five non-empty values are randomly sampled from each column. These values assist with type

inference and, when language model assistance is enabled, help interpret abbreviated or ambiguous column names by providing contextual examples.

Real-world datasets often contain irregular formatting, embedded metadata rows, or multiple tables within a single file. Rather than assuming a fixed structure, the system applies heuristic rules to identify valid tabular regions.

Candidate tables are identified by examining structural patterns such as empty rows, empty columns, and consistent blocks of data. Columns containing only empty values are treated as potential separators between tables. Once candidate regions are identified, the system analyzes rows near the beginning of each region to determine the most likely header row.

A valid header row is expected to contain descriptive text rather than numeric values or date like values. The rows following the header are examined to confirm that they contain data consistent with a tabular structure. When datasets contain multi-row headers, fragmented header components, or merged header cells spanning multiple columns, the system reconstructs column names; it does so by expanding merged header cells and combining relevant header fragments, so that each column receives a single coherent name that accurately reflects the meaning of the column.

Metadata Analysis

Once the table structure and column headers are established, the system performs metadata analysis for each table and column. At the table level, the system records structural information such as the total number of rows and columns.

Each column is evaluated to determine its likely data type. Columns are categorized as numeric or categorical based on the observed values. Numeric columns are analyzed to compute descriptive statistics including mean, minimum value, maximum value, and standard deviation. Categorical or textual columns are analyzed to determine unique values and their frequencies. To prevent excessive output, the number of reported unique values may be capped. Columns containing only missing or empty values are excluded from further analysis. The result of this stage is a structured metadata representation describing the characteristics of each table and its columns.

LLM-Assisted Column Name Interpretation

Many datasets use abbreviated or coded column headers such as qty, amt, or dept_id. To improve readability, the system optionally uses an LLM to expand these abbreviations into descriptive column names.

This capability is implemented using Anthropic Claude models accessed through AWS Bedrock. The model receives the original column names along with representative sample values from each column. These examples help the model infer the meaning of abbreviated column headers. If the user provides

additional contextual information about the dataset, that information is also included to help the model interpret domain-specific terminology.

Using these inputs, the model expands abbreviations, resolves ambiguous terms, and produces clear, human-readable column names. The response is returned as a JSON object mapping the original column names to their expanded forms. The system retains both versions of the column name. The original name is preserved for reference, while the expanded name is used in generated metadata reports.

If the LLM feature is disabled or valid AWS credentials are not available, the system falls back to heuristic parsing and type inference without performing semantic column name expansion.

LLM Prompt Template

The LLM is used exclusively for column name expansion and interpretation. The prompt instructs the model to expand abbreviations and standardize column names while preserving their meaning.

The model receives the extracted column names along with sample values from each column. If the user provides additional context about the dataset, this information is included to help interpret domain-specific abbreviations. The prompt instructs the model to expand acronyms, convert symbols and shortened units to full natural language forms, merge fragmented header text when necessary, and remove redundant wording while preserving meaning.

The model must return only a JSON object mapping original column names to expanded names. No additional text or explanation is allowed in the response. For example, the model may return a response such as:

```
{
  "qty": "Quantity",
  "dept_id": "Department Identifier",
  "avg_temp_c": "Average Temperature Celsius"
}
```

The system includes partial determinism controls but is not fully deterministic. Column name expansion uses controlled generation parameters, including a temperature value of 0.8 and a maximum token limit of 10,000. The slightly elevated temperature allows the model to interpret abbreviations flexibly while maintaining semantic consistency.

The Anthropic Bedrock interface does not allow both temperature and Top P to be specified simultaneously. Because temperature is explicitly defined in the request, the model uses the default Top P value of 0.9 provided by the API.

Representative column values are selected through random sampling. Up to five non-empty values are sampled from each column. Because sampling is random and language model generation is

probabilistic, repeated executions on the same dataset may produce slightly different outputs. In practice, the semantic interpretation of column names typically remains consistent across runs.

Post-Processing and Metadata Output

After metadata analysis and optional column name interpretation are completed, the system consolidates the results into a structured data dictionary. Each column entry includes the original column name, the interpreted or expanded column name, the inferred data type, and the associated statistics or categorical value summaries.

The system generates several export formats for documentation and analysis. PDF and Excel reports present the metadata in a readable format suitable for documentation. A CSV file stores the metadata in tabular form for programmatic use.

The PDF report includes information such as the number of tables detected, column names, inferred data types, unique value summaries, and statistical metrics for numeric fields. Within the application interface, the generated report can be previewed directly, and users can download the outputs without repeating the analysis.

The system includes explicit mechanisms for handling common failure scenarios and irregular datasets. Language model responses are validated to ensure that they contain valid JSON. If the model returns additional text or malformed output, the system attempts to extract the JSON object from the response. If extraction fails, the column expansion step is halted, and the user is notified.

Errors occurring during AWS Bedrock invocation, such as invalid credentials, model access failures, or communication errors, are captured and displayed through the application interface. CSV parsing errors are also handled explicitly. When parsing fails, the system attempts to identify the problematic line in the file and reports the line number along with the raw contents of that line to assist users in diagnosing formatting issues.

Because table detection relies on heuristic analysis, individual tables that cannot be processed successfully are skipped rather than causing the entire workflow to fail. Unsupported file types are rejected immediately. Only CSV, TSV, XLS, and XLSX inputs are accepted.

Performance Considerations

Evaluation

Evaluation of the Metadata Extractor has focused on a diverse set of datasets that reflect realistic federal analytical workflows. Crime datasets from Washington, DC (2021 and 2023) were used to test CSV and Excel table parsing, type inference, and metadata generation. These files often contain many columns with mixed types, providing a useful stress test for both detection and semantic interpretation.

Per-run metadata artifacts and summary reports can be used to verify header interpretation accuracy and type inference consistency across datasets, providing a basis for internal QA.

Testing revealed several strengths and areas for improvement. For tabular files, the Metadata Extractor successfully generated variable inventories and type summaries, though type inference sometimes mislabeled date fields as categorical or misinterpreted all text columns. Performance for large documents was also slower than ideal, and certain expected tables were missed entirely depending on the layout.

NORC's internal testing process emphasized qualitative review of extracted metadata. Analysts compared tool outputs against the expected structure of known datasets, inspected representative values, and verified type inference accuracy. They also evaluated the usability of the exported data dictionary and how well the tool supported rapid dataset familiarization. Per-run outputs include a PDF report with table-level and column-level details, a CSV summary with separate Summary and Details sections, and an Excel workbook with Summary and Details sheets. These artifacts enable reviewers to verify column interpretations, type inference, and statistical calculations across multiple tables and pages. This approach aligns with real-world intake workflows, where analysts must determine if a dataset is usable and what cleaning or harmonization steps will be needed.

Performance Metrics

Output Quality: Although the tool does not expose detailed quantitative performance metrics, certain quality dimensions are inherent to metadata extraction. Output quality can be evaluated by confirming the interpreted column names against the original headers, checking the correctness of inferred types, and verifying that the reported counts and statistics in the exports match expectations for each table. When the tool successfully identifies representative values and meaningful type classifications, analysts can rely on the data dictionary for preliminary assessments without manually inspecting the raw dataset. Key quality indicators include header interpretation accuracy, correctness of inferred data types, and completeness of table detection; these metrics can be derived from exported artifacts.

Computational Performance: Performance varies across input types. For straightforward CSV and Excel files with moderate row counts, extraction is typically fast. However, large or wide datasets increase processing time due to the need to scan representative values across many columns. Processing time varies with file size and number of tables. Caching reduces repeat processing time for previously analyzed files, and progress indicators are shown during extraction and report generation.

Cost Guidance

Two datasets were tested in the Metadata Extractor to develop cost guidance.[4] The first dataset contained data on crime incidents in the District of Columbia in 2021, while the second dataset contained American Community Survey data for 2024, and a supplementary file with a list of unabbreviated variable

[4] The tests in the table were run on March 12, 2026.

names. These scenarios are intended to provide cost guidance by showing token usage for each run along with the total number of LLM calls, computational runtime, and total cost. Session metrics are exposed for users in a tool panel to help monitor processing performance and costs.

Table 4. Cost guidance for Metadata Extractor

Input File Name	Input File Description	Input File Size	Run Parameters	Tool Runtime	LLM Version	Total Tokens	Total LLM Calls	Total Cost
DC Crime Data 2021	Data on crime in the District of Columbia in 2021 (1 CSV)	28,320 rows and 25 columns (7.6 MB)	Default	6.68 seconds	Claude Sonnet 4.6, region us-east-1	2,108 tokens (input and output combined)	1 call	$0.0101
ACS 2024 Data	American Community Survey data for 2024 (and list of variable names as supplementary information) (1 XLSX, 1 TXT)	84,401 rows and 50 columns (24.2 MB; supplementary file is 2.41 KB)	Supplementary TXT file included containing list of variable names	34.03 seconds	Claude Sonnet 4.6, region us-east-1	4,276 tokens (input and output combined)	1 call	$0.0257

Limitations and Failure Modes

Common failure scenarios include the following.

- Complex header structures (common in financial spreadsheets or multi-level administrative reports) may not be detected correctly. Columns with merged headers, nested labels, or rotating text often confuse table parsers.
- If AWS credentials are missing or invalid, LLM-based header interpretation is skipped, and the tool proceeds with heuristic parsing.
- Sparse or irregular data, inconsistent encodings, and mixed format columns can also lead to type inference errors or incomplete metadata extraction. For example, text fields containing a mix of dates, numbers, and codes may be misclassified as categorical.
- Documents containing stylized tables may produce no detectable tables at all.

Users should carefully review interpreted names, types, and statistics in the exported reports, particularly for files with unconventional encodings, mixed data, or highly abbreviated headers.

Comparison with Traditional Approaches

Traditional metadata extraction processes usually involve manual inspection of datasets or the creation of ad hoc scripts to parse column names and summarize field characteristics. While these approaches can yield high-quality results for small datasets, they do not scale well and are prone to inconsistencies between analysts.

Rule-based tools such as spreadsheet formulas or Python scripts can automate some aspects of metadata extraction but typically lack semantic interpretation. These tools do not understand abbreviated or cryptic header names, nor can they infer the meaning of value formats beyond basic type detection.

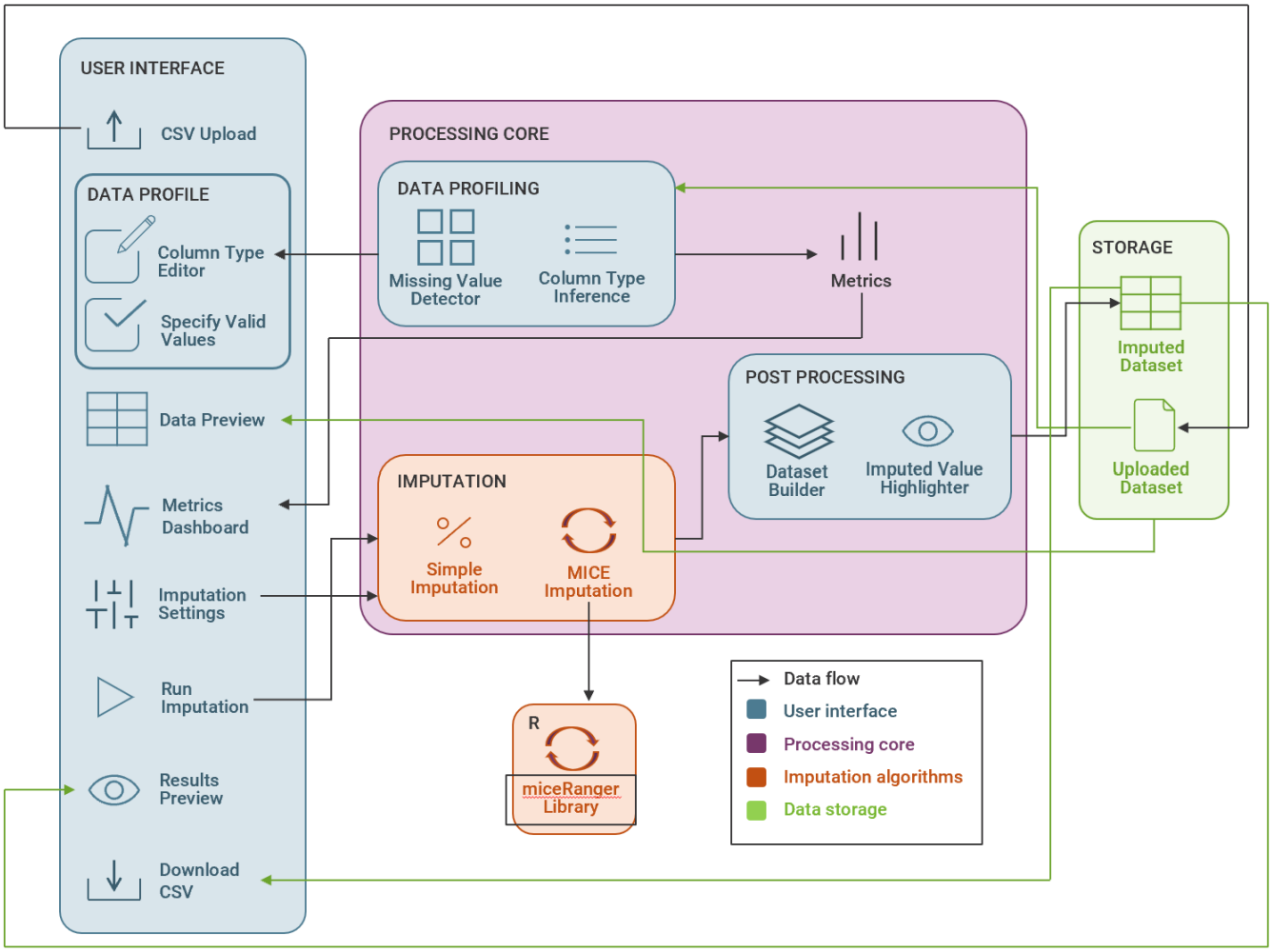
The Metadata Extractor improves on these approaches by integrating table detection, type inference, semantic interpretation, and data dictionary generation into a unified workflow. While the tool still requires careful oversight and human review of output, especially when handling complex formatting, it significantly reduces the burden of initial data familiarization and provides a repeatable process that analysts can apply across diverse datasets.

Missing Data Assistant: Tool-Specific Documentation

The Missing Data Assistant is a structured imputation tool for rectangular tabular datasets with incomplete values. It is designed to help analysts identify missingness, review variable types, select an imputation method, and export a completed dataset with traceable indicators showing where imputations occurred. The tool is intended to reduce bias and information loss that can result from dropping incomplete records or applying overly simplistic replacement methods without review. It does not use a large language model. Missingness handling, variable processing, and value generation are performed using conventional statistical and machine learning methods, which makes the workflow more auditable and easier to validate in settings where methodological traceability is important.

Target Users, Use Cases, and Limitations: The tool is intended for federal data scientists, statisticians, research methodologists, and program analysts who need defensible, scalable imputation workflows. Typical use cases include imputation in survey microdata, multijurisdictional administrative files (e.g., transportation operations, crime incidents), and program datasets containing both numeric and categorical variables. As a prototype, the tool can be constrained by very large or column dense files, preview rendering limits (e.g., Pandas Styler), and long runtimes for multivariate approaches; column type inference may also require manual overrides in edge cases.

Figure 5. Missing Data Assistant architecture



Technical Details

Supported Input

The tool is designed for survey data, administrative records, and other program files with a mix of numeric and categorical variables that frequently contain gaps. These sources often exhibit Missing at Random (MAR)-style relationships, where missingness depends on observed variables and model-based imputation can improve quality relative to simple replacement methods.

The tool supports CSV inputs for ingestion and imputation. Excel files should be converted to CSV before use. Each input file must include a header row, and encodings should be consistent, with UTF-8 recommended. Columns are interpreted as numeric, categorical, or datetime, and analysts can override type detection before running imputation. CSV parsing may fail for unusual separators or encodings,

and very wide tables can slow computation or prevent preview rendering. Nested JSON and multi-table joins are not supported and should be resolved upstream into a single rectangular table.

Logging and Reproducibility

The tool emphasizes output traceability rather than full system telemetry. Each run produces a completed dataset, and imputation flags that distinguish observed values from imputed values at the cell level. It also generates pre- and post-imputation summaries, so analysts can compare key descriptive properties before and after missing-value treatment.

These outputs support QA and review by making it possible to identify where changes occurred and to assess whether the imputation process introduced noticeable shifts in distributions or category frequencies. However, the current implementation does not maintain comprehensive operational logs such as step-level timing, memory usage, or detailed run histories. Reproducibility therefore depends primarily on preserving the input file, the selected variable types, any invalid-value rules, and the chosen imputation settings used for the run. For multivariate imputation methods that rely on stochastic algorithms (e.g., random forests), users can specify a random seed to reproduce identical results across runs with the same data and settings.

The Missing Data Assistant does not write intermediate or final files to disk. All outputs exist only within the user's session unless explicitly downloaded.

Model Architecture

The Missing Data Assistant uses a staged processing architecture that separates user interaction, analytical processing, and output generation. At a high level, the architecture includes data ingestion, data profiling, imputation configuration, model execution, post-processing, and storage of the original and completed datasets. The user interface supports CSV upload, data preview, column-type review, specification of valid or invalid values, selection of imputation settings, execution of the run, results preview, and export. This design exposes key decision points directly to the analyst, particularly column typing, because imputation behavior depends on whether a field is treated as numeric, categorical, or datetime.

After upload, the tool parses the file into a single tabular structure and generates an initial preview. It then profiles each column to identify missing cells, compute column-level missingness metrics, infer variable classes, and prepare the metadata needed for imputation. Analysts can override inferred types and specify invalid placeholder values that should be treated as missing before imputation begins. This step is important because many operational datasets use blank strings, sentinel codes, or labels such as "Unknown" rather than explicit null values.

The imputation layer supports two pathways. The first is a simple rule-based pathway intended for lower-risk cases or columns with limited missingness. In this mode, numeric variables can be filled using a summary statistic, and categorical variables can be filled using the most common observed

class. This approach is fast and easy to interpret, but it does not preserve relationships across variables and is best treated as a baseline or operational fallback. The second pathway is multivariate imputation, which estimates missing values using relationships among variables in the dataset rather than treating each column independently. In the current implementation, this workflow uses chained-equation imputation with random forest models through the miceRanger backend. In practice, this means the tool iteratively updates incomplete variables using patterns present in the observed data rather than relying only on one-column replacement rules. Multivariate imputation methods rely on established R packages (miceRanger) accessed locally via rpy2; no external R services are used.

The post-processing layer reconstructs the completed dataset, generates per-cell imputation flags, and produces pre- and post-imputation summaries for review. These outputs are intended to make imputed values visible rather than blending them into the final dataset without distinction. The storage layer supports this process by retaining both the uploaded dataset and the imputed output so the system can support preview and export of both versions. Overall, the tool functions as an orchestration layer around conventional imputation methods and combines structured preprocessing, explicit user controls, standard estimation methods, and review-oriented outputs to support a transparent workflow for handling missing data.

Performance Considerations

Evaluation

Initial testing used mixed domains to explore the tool's behavior on numeric and categorical data: NYC MTA Bus Speeds (continuous durations and counts), NYC Ferry Ridership (numeric counts with categorical routes), and Las Vegas NIBRS crime (categorical code recovery). These datasets offered varied missingness patterns and realistic predictor relationships. Transportation and crime domains are broadly relevant to federal work and contain mixed feature types with interpretable relationships (e.g., routes, times, codes). They also expose edge cases—such as sparse predictors or heavy categorical cardinality—that test robustness.

Evaluation compared imputed outputs to known values by withholding a subset of complete data and computing numeric error (for example, RMSE or MAE) and categorical recovery (for example, accuracy or F1). The imputed files, flags, and summary statistics were archived alongside the original data to support independent QA.

Early tests highlighted practical issues: preview rendering limits in large tables, the need for progress bars, and the need for clearer guidance on what constitutes "missing" (e.g., NA, empty strings, whitespace, special sentinels). Simple mode imputation proved to have limited analytic value in several categorical scenarios, whereas multivariate approaches recovered plausible values (e.g., codes) when sufficient predictors existed.

Performance Metrics

Output Quality: Numeric imputation quality should be assessed using Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE), while categorical recovery should be evaluated using accuracy or F1. Distributional similarity checks (such as Kolmogorov-Smirnov tests) help identify shifts in central tendency or variance introduced by imputation. The tool currently provides pre-/post- summaries but does not compute these metrics automatically.

Computational Performance: Runtime increases with dataset size, missingness patterns, and method complexity. Multivariate imputation trains models iteratively for each incomplete feature. Two files were tested in the Missing Data Assistant to develop cost guidance. The first file was a CSV of New York City MTA Bus Speeds. The second, larger file was a log of New York City ferry ridership. Both files contained numeric, categorical, and date types with missing values. Details from each run are summarized in the table below.[5] These scenarios are intended to provide cost guidance by showing computational runtime under different scenarios using different sized files and imputation strategies. Runtime is the only applicable metric exposed in the tool's final step and is reported in seconds.

Table 5. Performance metrics for Missing Data Assistant

Input File Name	Input File Description	Input File Size	Run Parameters	Tool Runtime	LLM Version	Total Tokens	Total LLM Calls	Total Cost
MTA Bus Speeds	NYC Metro bus speeds (2020 – 2024) from Data.gov with removal at random (1 CSV)	4.4 MB; 71,907 rows	Simple Imputation with Missing Threshold of 5% (trip_type: Mode; total_operating_time: Median) miceRanger with 1 column to impute (trip_type) and 2 predictors (route_id; total_mileage); Max Iterations: 10; Seed: 42	0.10 seconds (Simple); 36.16 seconds (miceRanger)	Not applicable (no LLMs used)			
NYC Ferry Ridership	NYC ferry ridership for 2025 from Data.gov with removal at random (1 CSV)	48.1 MB; 1M rows	Simple Imputation with Missing Threshold of 20% (Route: Mode; Boardings: Median) miceRanger with 2 columns to impute (Route; Boardings) and 4 predictors (Hour; Direction; Stop; TypeDay)	0.71 seconds (Simple); 2.39 hours (miceRanger)	Not applicable (no LLMs used)			

[5] The tests in the table were run on March 30, 2026.

Limitations and Failure Modes

Common failure scenarios include:

- Data types may be incorrectly inferred (for example, numeric stored as text); in such a situation, imputation may fail or produce implausible values.
- Insufficient predictors for multivariate methods, as sparse predictors can weaken multivariate estimates.
- Long runtimes when many features have high missingness.
- Preview rendering can fail on large tables; when this occurs, analysts should proceed with imputation and download/export the results, rather than relying on in-app visualization.
- Multivariate methods can overfit when sample sizes are small or relationships are weak. Analysts should inspect imputation flags and pre-/post- summaries to detect implausible shifts and adjust predictors, thresholds, or methods as needed.

Comparison with Traditional Approaches

Handcrafted imputation delivers maximal control but is slow, non-transparent, and difficult to reproduce across teams and time. The tool centralizes method selection and produces standardized artifacts (i.e., imputation flags, datasets summaries, exported outputs) that make QA and governance more reliable.

Scripts offer determinism but are often brittle across heterogeneous datasets and provide limited diagnostics or method guidance. The tool improves usability by exposing multiple strategies in one interface, visualizing missingness, and standardizing outputs for downstream use.

Overall, the tool's hybrid design—from quick baseline methods to richer multivariate options—supports tradeoffs between speed and accuracy. Structured outputs (imputation flags and summaries) deliver auditability and facilitate peer review, compared to simple "mean fill" workflows.

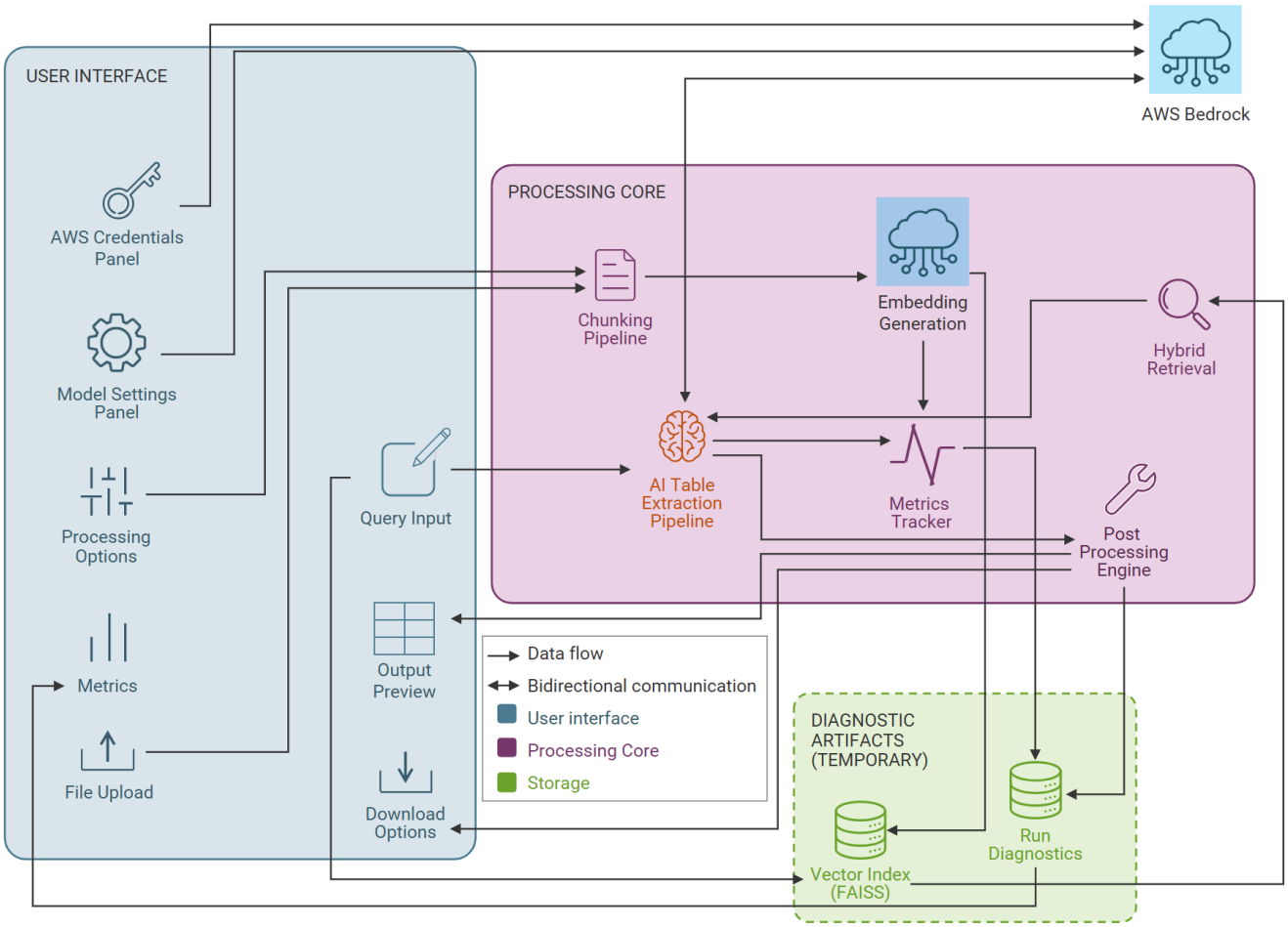
Tabular Data Extractor: Tool-Specific Documentation

The Tabular Data Extractor converts semi-structured or unstructured documents in PDF or TXT format into structured tables suitable for analysis (JSON/CSV/XLSX). It uses a Retrieval-Augmented Generation (RAG) pipeline to ground an LLM's answers in the source content: documents are parsed and chunked, embedded into a local vector store, retrieved semantically against a user's question, and then passed to the LLM to produce a structured result. Users pose targeted prompts (e.g., "List California entities and recognition dates") and receive tables accompanied by confidence scores and, where available, downloadable exports.

Many high-value data points live inside narrative reports, statutory notices, or transcripts where manual table creation is slow, error-prone, and hard to reproduce across analysts. Rule-based scrapers are brittle in the face of layout variation, multipage tables, and domain vocabulary shifts. The tool addresses these pain points by combining semantic retrieval with structured prompting, returning consistent tabular outputs that can be audited, versioned, and integrated downstream (e.g., into harmonization or analytical pipelines).

Target Users, Use Cases, and Limitations: The tool is intended for federal data scientists, research methodologists, program analysts, and data stewards who need to extract structured fields from documents at scale. Typical use cases include Federal Register notices (entities, effective dates, eligibility criteria), legislative transcripts (bill numbers, sponsors, committees), and transportation fact sheets (locations, ridership).

Figure 6. Tabular Data Extractor architecture



Technical Details

Supported Input

The tool is designed for narrative and semi-structured documents commonly used across federal workflows: legislative transcripts (e.g., H.R. 1), Federal Register notices (e.g., Indian Entities), transportation fact sheets (Amtrak state summaries), and other agency publications with lists, enumerations, and embedded tables. These sources stress the pipeline's ability to retrieve relevant context and to produce accurate field values across long, mixed format texts.

The tool supports PDF and TXT documents with extractable text (it is unable to directly process PDF files that consist of scanned images). Results are displayed in the interface and can be exported to XLSX or CSV. Structured exports are standardized and downloadable directly from the interface.

Documents must contain an accessible text layer; image-only or scanned PDFs should be OCR-processed before they are uploaded, as OCR is not included in the tool. In other words, the tool does not have the built-in capacity to process PDFs that consist of scanned images. Clean encodings and clear sectioning improve chunking and retrieval quality. For large files, consistent headings and section markers help the system form stable chunk boundaries.

Logging and Reproducibility

Users should retain the prompt text and retrieval parameters with each export to support cost review and reproducibility. The system also records processing metrics during execution. Token counts are estimated using the tiktoken tokenizer during embedding generation and are tracked as part of those metrics. Reproducibility is further supported by fixed retrieval and scoring settings, including the hybrid retrieval weighting, candidate pool size, and final top-k selection. To reduce redundant computation across repeated runs, both the chunking output and the FAISS retriever are cached in session state using the file's MD5 hash and, for the retriever, the embedding model ID. Reprocessing the same file can therefore reuse prior artifacts rather than repeating the full preprocessing and retrieval setup.

The Tabular Data Extractor writes lightweight diagnostic files in the application's root directory during execution. These files may include chunking metadata, similarity scores, and a FAISS vector index used to support retrieval and debugging. They are intended to facilitate debugging in the event of suboptimal results, and are overwritten on subsequent runs; these artifacts are not intended to serve as a persistent long-term record of prior runs.

Model Architecture

The Tabular Data Extractor is designed to extract structured tabular information from unstructured documents such as PDFs and TXT files. The system combines semantic retrieval with LLM reasoning to identify relevant document segments and convert them into structured key value outputs. A local vector store is used to store document embeddings. Embedding models encode both document chunks and user queries. An LLM generates structured results constrained by templated prompts.

The pipeline begins with document ingestion and preprocessing, followed by semantic retrieval and structured generation. After documents are parsed and segmented into chunks, embeddings are generated and stored in a vector index. At query time, the user prompt is embedded using the same embedding model and compared with stored vectors to retrieve the most relevant document chunks. These retrieved chunks are inserted into a structured prompt template that constrains the LLM to produce machine-readable JSON output. The response is then parsed into tabular results and displayed in the interface along with confidence metrics.

The architecture is modular and configurable. Index configuration, retrieval parameters, and prompt templates can be tuned depending on the corpus size and domain requirements. Flat FAISS indexes work well for smaller collections where exact similarity search is feasible, while inverted file or graph-based indexes can be used when approximate search is required for faster retrieval in larger datasets.

Document Parsing and Chunking

During preprocessing, documents are parsed using file-specific libraries capable of extracting text from PDFs and TXT files. PDF processing uses plain text extraction rather than Markdown conversion, because plain text extraction is more consistent across document types and avoids the reliability issues observed with unstructured tables during Markdown conversion

The primary chunking mechanism uses LangChain's *RecursiveCharacterTextSplitter*, which recursively segments text using a hierarchy of logical separators including paragraph breaks (\n\n), line breaks (\n), table column separators (|), and word boundaries. This recursive approach attempts to preserve larger semantic units such as paragraphs before splitting into smaller fragments when necessary.

The default configuration uses a chunk size of 1,500 characters with an overlap of 300 characters between adjacent chunks. The overlap ensures that contextual information spanning chunk boundaries remains available during retrieval.

Before chunking, the system scans extracted text for labeled references such as "Table 3", "Figure 8.2", "Appendix B", and "Equation 3.1(ii)" and replaces them with unique random identifiers. This tagging step preserves consistent cross-chunk linking of references during retrieval. The tagger covers a broad set of label types, including Table, Figure, Image, Chart, Graph, Diagram, Appendix, Equation, Section, and Algorithm. Before context is sent to the LLM, the identifiers are restored to their original human-readable labels.

There is no separate table serialization or Markdown table extraction step in the current implementation of the tool. Tables are retained as plain text within the standard recursive chunks. For queries that require broader table context, the LLM context slider can be used to expand the context window and capture a wider surrounding passage.

The final chunk set therefore consists of a single set of recursive text chunks derived from plain text-extracted pages.

Semantic Retrieval and FAISS Index Configuration

Each document chunk is converted into a vector embedding using the AWS Bedrock Titan embedding model *amazon.titan-embed-text-v2:0*. Embeddings are generated through the LangChain *BedrockEmbeddings* integration, and each chunk is embedded individually. The resulting vectors are converted into NumPy float32 arrays and normalized using L2 normalization prior to indexing. Normalization ensures that cosine similarity can be computed efficiently using inner product distance.

Embedding generation is executed in parallel using a *ThreadPoolExecutor* with max_workers = 10, which allows multiple chunks to be embedded concurrently. Token counts for each chunk are estimated using the *tiktoken* tokenizer (with *cl100k_base* vocabulary) during the embedding process and are tracked as part of the processing metrics.

The vector database uses FAISS with the following configuration. The index type is *faiss.IndexFlatIP*, and vector normalization is applied using *faiss.normalize_L2(vectors)*. The distance strategy used by LangChain is cosine similarity, and the embedding dimension is 1024, which corresponds to the Titan embedding model output dimension. Because vectors are normalized before indexing, the inner product search performed by IndexFlatIP behaves as cosine similarity.

The retrieval system uses a hybrid retrieval strategy that combines dense semantic search with lexical search. Dense similarity is computed through the FAISS index by comparing query embeddings with stored document embeddings. Lexical similarity is computed using the BM25Okapi algorithm, which calculates relevance based on term frequency, inverse document frequency, and document length normalization. Raw BM25 scores are normalized to a 0 to 1 range using sigmoid normalization centered around the mean score.

The final ranking score is calculated using the hybrid scoring formula:

$$\text{HybridScore} = (1 - \alpha) \times \text{LexicalScore} + \alpha \times (1 - \text{DenseScore})$$

where $\alpha = 0.5$, giving 50% weight to dense semantic similarity and 50% weight to BM25 lexical similarity. The dense score from FAISS is inverted before combining, because FAISS returns a distance-style score in this pipeline configuration.

Retrieval occurs in two stages. First, an intermediate candidate pool of 20 chunks is retrieved from the hybrid search system. After hybrid search, the top 20 candidates are passed to a Cohere Rerank v3.5 model (*cohere.rerank-v3-5:0*) via AWS Bedrock. The reranker scores each candidate against the query, and the top 5 results are returned as the final retrieved set inserted into the prompt context used by the LLM.

Prompt Template and LLM Configuration

The system uses templated prompts to ensure that the language model returns structured machine-readable outputs. Prompt templates are stored externally and loaded dynamically at runtime. Each template contains system instructions that constrain the model to use only the information present in the retrieved context and to return a strictly formatted JSON object.

The JSON structure must be directly convertible into an HTML table. Each key in the JSON object represents a column header and each value must be a list of strings. Each index position across the lists represents a single row in the resulting table. All lists must have the same length, even if only one item is present. If a requested field is not found in the context, the system instructs the model to return an empty string within a list (*[""]*) so that list lengths remain consistent. If none of the requested information is found in the provided context, the model must return the JSON *object {"note": ["Answer not found in the given context"], "generation_confidence_score": ["0"]}*.

The prompt includes explicit output rules requiring the response to begin with { and end with }, and forbids any explanations, preamble, or text outside the JSON object. The prompt also requires a

"*generation_confidence_score*" key in the JSON object. Its value must be a list containing exactly one string from 0 to 100, scored according to the following rubric: 90 to 100 for all fields found clearly and unambiguously, 70 to 89 for most fields found with minor ambiguity, 50 to 69 for some fields found with significant gaps, and 0 to 49 for high ambiguity, contradictions, or mostly missing fields.

The prompt template contains two runtime variables. The *{context}* variable contains the retrieved document chunks or the full document in non-RAG mode, and the *{input}* variable contains the user's question. These variables are inserted into a LangChain *ChatPromptTemplate*, which produces the final prompt passed to the model.

The system typically runs inference using AWS Bedrock Claude models. The LLM configuration parameters include temperature of 0.2, max tokens of 20000, and Anthropic API version as *bedrock-2023-05-31*. AWS Bedrock's Anthropic interface does not allow both temperature and top_p to be controlled simultaneously in the request. Because the application specifies a temperature value, the top_p parameter is omitted from the request body, and the model uses the default top_p = 0.9 defined by the Anthropic API.

Before the prompt is sent, UUID label tags in the retrieved context are restored to their original human-readable labels such as "Table 3" or "Figure 2.1". If the model returns an invalid JSON, the pipeline retries the call up to 3 times before raising an error.

These constraints help ensure that the model produces deterministic outputs that can be reliably parsed into structured tables.

Token Budgeting, Confidence Scoring, and Failure Handling

Token budgeting in the system is handled indirectly through chunk size limits and retrieval constraints rather than explicit prompt truncation. During embedding generation, token counts are estimated using the *tiktoken* tokenizer with the *cl100k_base* vocabulary. The number of tokens processed during embedding generation is recorded as part of the system's processing metrics.

Prompt size is controlled primarily through retrieval limits. The system inserts only the top 5 retrieved chunks into the prompt context, and chunk size can also be adjusted through a UI slider, with a default value of 1000 words. These limits ensure that the constructed prompt remains within the context window of the underlying model.

The exact context window depends on the LLM used through AWS Bedrock. Claude Sonnet 4.6 supports a context window of approximately 1,000,000 tokens, while Claude Haiku 4.5 supports approximately 200,000 tokens. Because these context limits are large, the implementation does not include explicit sliding window or dynamic truncation mechanisms, and instead relies on limiting the number of retrieved chunks.

After the LLM produces its output, the JSON response is parsed and converted into a structured table. The system computes three confidence metrics: retrieval confidence, generation confidence, and total

confidence. Retrieval confidence is calculated as the average reranker relevance score of the retrieved chunks. If k represents the number of retrieved chunks and score_i represents the reranker score for each chunk, retrieval confidence is computed as the mean of those scores.

Generation confidence is derived from the model's self-reported "*generation_confidence_score*" field in its JSON output. The value is expected to be a 0 to 100 integer encoded as a string inside a one item list, for example *["85"]*, and is normalized to a 0 to 1 float by the pipeline. If the key is missing or cannot be parsed, the pipeline falls back to a default of 0.5.

The final combined score is calculated using the formula:

TotalConfidence = β × RetrievalConfidence + γ × GenerationConfidence

where β = 0.5 and γ = 0.5. For the non-RAG path, retrieval confidence is set to 1.0 since no retrieval step is involved. The primary confidence score surfaced to the user is the generation confidence, while retrieval confidence and total confidence support internal ranking and evaluation.

The pipeline also includes several safeguards for handling errors and edge cases. If the retrieval stage returns no chunks, the system does not invoke the LLM and instead immediately returns a hardcoded response indicating that no relevant content was found, with retrieval confidence, generation confidence, and total confidence all set to 0.0. If the model produces invalid JSON output, the system retries the Bedrock API call up to 3 times. On each attempt, it cleans formatting artifacts such as Markdown code block markers before attempting JSON parsing again. If all attempts fail, the pipeline raises an error and surfaces it to the user.

The system also detects scanned PDFs that contain images rather than machine-readable text. Before document processing begins, the system samples up to 10 pages and measures extracted text length to determine whether the PDF is image-based. If a scanned document is detected, processing stops immediately and the user is notified, and is informed the file must be replaced with a text-based PDF.

Errors that occur during file processing, chunking, embedding generation, retrieval, or LLM inference are handled at the file level where possible, so that remaining files can continue processing, while global errors such as a failure to connect to AWS Bedrock are treated as fatal. If any individual chunk fails during embedding generation, the embedding step fails for that file and processing continues for the remaining files. Chunking output and the FAISS retriever are cached using the file's MD5 hash and embedding model ID so repeated processing of the same file can skip redundant work. No explicit timeouts are implemented for individual retrieval or LLM inference steps.

Multi-File Query Feature

The multi-file query feature allows the system to process multiple uploaded documents and consolidate their extracted results into a single unified output table. It is enabled via a checkbox in the sidebar and is only active when more than one file is uploaded. When multi-file query is enabled, each file is processed independently through the full pipeline, including chunking, retrieval, and LLM extraction.

Each file produces its own JSON answer structured as a column-oriented dictionary. These per-file results are collected and stored together before any consolidation step. Once all files have been processed, their individual JSON results are passed to a second LLM call along with a user-provided consolidation prompt. This prompt is entered separately from the per-file extraction question and is intended to specify the desired output structure, such as which columns should appear in the final combined table and how data from different files should be aligned.

The consolidation LLM call uses the following parameters: temperature of 0.0, max tokens of 100000, and the same Bedrock Claude model selected in the sidebar. The model is instructed to normalize and align the per-file results into a single flat column-oriented JSON object, which is then rendered as a single unified HTML table.

When only one file is uploaded, the multi-file query option is automatically disabled regardless of the checkbox state. In this mode, per-file results are rendered directly into separate per-file HTML tables without any consolidation step, each preceded by the file name as a heading. In multi-file mode, the final output is a single combined table exported as CSV or Excel. In single-file mode, each file gets its own table section in the output, with separate sheets per file in the Excel export.

Performance Considerations

Evaluation

Evaluation covered diverse sources: legislative transcripts (H.R. 1), Federal Register notices (Indian Entities), Amtrak Illinois 2023–2024 fact sheets, and California State Assembly Daily Journals. These materials include long narrative sections, embedded tables, and enumerations requiring consistent semantic retrieval and structured extraction. Selected sources reflect common federal workflows where structured fields are buried in narrative – ideal for testing the RAG pipeline's fidelity. Legislative and regulatory content also varies in vocabulary and layout, making them good stress tests for chunking, retrieval, and structured prompting.

Accuracy varied by prompt specificity and document structure: some prompts yielded correct structured answers (speaker, date, committees), while others returned wrong counts or empty tables, especially in multifile scenarios (e.g., combining 2023/2024 locations). In the initial version of the tool, confidence scores needed tooltips and consistent placement; and progress indicators lagged behind actual completion in certain runs.

NORC testers used scenario-driven prompts, toggled LLM context size and RAG on/off, and verified outputs against source documents. Evaluation verified each extracted field against the source text and recorded confidence thresholds used for filtering, with side-by-side comparison of single-document runs and multi-document runs where applicable.

Table 6. Sample prompts for Tabular Data Extractor

Sample prompts: Single PDF (Amtrak data from Illinois in 2023)	**Sample prompts: Multiple PDFs** (Amtrak data from Illinois in 2023 and 2024)
• "Create a table of all stations and improvements mentioned in the document." • "Extract all station names and summarize the changes described for each." • "Provide a table with: Station | Improvement | Year Completed." • "Identify all capital projects listed and return: Project | Description | Location | Status."	• "Identify stations that appear in both documents and list their improvements side-by-side." • "Combine all entries from all PDFs into a single table with: Station | Improvement | Year | Source Document." • "Extract all unique projects mentioned in the documents with their associated year." • "Combine all entries from all PDFs into a single table with: Station | Improvement | Year | Source Document."

Performance Metrics

Output Quality: Quality should be assessed using field-level accuracy against the source, coverage of all requested fields, and consistency across runs. Confidence scores should correlate positively with correctness; thresholds used to filter low-confidence rows should be documented in the run notes.

Computational Performance: End-to-end latency reflects parsing, chunking, embedding, retrieval, and LLM inference. Large documents and higher retrieval top-k values increase runtime. The interface displays progress during each stage, and users should record stage timings when troubleshooting or comparing prompts.

Cost Guidance

Two workflows were tested to develop cost guidance.[6] The first uses a Congressional transcript text file to retrieve information about the speaker who introduced a given bill; the second uses two PDF reports of service changes at the state level for Amtrak in 2023 and 2024. These scenarios use different input file types and sizes; the first scenario uses a single file for a simple query while the second uses multiple files for a more complex query. These scenarios illustrate differences in tool runtime, tokens, LLM calls, and total cost when working with a single file versus multiple files, enabling multi-file querying, and adjusting the LLM context size.

[6] The tests in the table were run on March 12, 2026.

Table 7. Cost guidance for Tabular Data Extractor

Input File Name	Input File Description	Input File Size	Run Parameters	Tool Runtime	LLM Version	Total Tokens	Total LLM Calls	Total Cost
Congressional Transcripts, HR 1	Congressional Bills 103th Congress, H.R. 1 Introduced in House from the U.S. Government Printing Office (1 TXT file)	73 KB; 1,338 lines	LLM Context Size: 2,000 Use RAG	12 sec	Claude Sonnet 4.6 region us-east-1	Tokens Used (LLM): 3,881 Tokens Used (Embeddings): 16,431	4	$0.0199
Amtrak, Illinois 2023 and 2024	Amtrak in Illinois Fiscal Year 2023 & 2024 summary reports (2 PDF files)	IL 2023: 923 KB; 11 pages IL 2024: 488KB; 7 pages	LLM context size: 10,000 Use RAG Use multi-file query	55.32 sec	Claude Sonnet 4.6 region us-east-1	Tokens Used (LLM): 16,845 Tokens Used (Embeddings): 9,262	8	$0.0859

Limitations and Failure Modes

Observed failure scenarios include:

- Ambiguous prompts leading to incorrect field values or no response. To avoid this, the user should provide field-specific prompts to reduce ambiguity.
- Empty tables when combining documents across editions or years. The user should ensure that prompts are refined to target specific tables, sections, or date ranges.
- For long documents, the user should experiment with different chunk sizes and provide field-specific prompts to reduce ambiguity. Large documents are split into smaller chunks before extraction; users can set the maximum size of each document chunk, i.e. how much text the LLM can retrieve as surrounding context. While smaller chunk sizes reduce cost and reduce the risk that the LLM will be overloaded by the amount of retrieved text, some relevant context for the query may be lost if the chunk size is too small.

Comparison with Traditional Approaches

Manual tabulation offers high fidelity but is slow, non-scalable, and inconsistent across analysts. The tool automates discovery and tabularization, returns structured outputs with confidence metrics, and enables faster iteration and audit trails suited to federal data curation workflows.

Deterministic parsers can be efficient on fixed layouts but are brittle when formats or vocabulary change. The tool's approach, combining RAG with LLMs, adapts to varied phrasing by retrieving

semantically relevant context before generation, offering better resilience across heterogeneous documents.

The tool's modular pipeline (consisting of: parse; chunk; embed; retrieve; prompt; extract) and its user-facing interface options allow analysts to ask questions and receive structured tables rapidly. This reduces manual curation time and supports downstream integration with harmonization and analysis tools.

Checklist of Best Practices

This checklist summarizes how to apply the five tools in a responsible way that maximizes the utility, objectivity, and integrity of data.

Cross-Cutting Best Practices

The following set of best practices are applicable across all five tools.

Transparency and Reproducibility

- Document exact input datasets used, including but not limited to file names, versions/vintages, dates of retrieval, and counts of variables and records (if any filtering was done, this should include counts both before and after).
- Capture context of tool run, including but not limited to tool name, tool version/build, Large Language Model and embedding versions used, any parameters that were set by the user (in particular, note random seeds when they are used), and output quality metrics.
- Ideally document all of this information in a lightweight one-page document for each run.
- Maintain data lineage that tracks changes over time across files. This includes storing change logs and commit IDs for Github or any other version control platform.
- Export and save artifacts produced by tools.

Keeping Humans in the Loop

- Define the decision points where human review will take place, even before a run of the tool is initiated.
- If the tool is working with a subset of data, subsetting/sampling should be informed by human expertise and domain knowledge, to ensure that rare subgroups and edge cases are included.
- Document who signs off on or approves decisions and tool output, and on what basis.
- Treat output as exploratory until it has been validated by human review.
- Ensure that human reviewers know how to correctly interpret the quality metrics displayed by the tool.
- Acceptable quality thresholds should be set up front, even before the tool is run, and the final values achieved should be documented.

Privacy and Security

- Only input the minimum necessary data into the tool – include only the variables that are necessary for a particular task.
- Mask direct identifiers where feasible.

- Scrub output files for potential inadvertent disclosures, such as personal identifiers included in extracted tables or metadata.
- Scrub LLM prompts and artifacts to ensure they exclude confidential content or are appropriately redacted.
- Purge temporary files and cache directories after quality checks, beyond what is required to be retained for reproducibility.
- Securely store all output files, intermediate files, logs, and artifacts in approved locations with suitable levels of encryption.

Bias and Fairness

- Identify sensitive attributes (e.g., race/ethnicity, gender, geographic location) in input data which may lead to disparities in model performance.
- Evaluate tool's performance across subgroups and investigate disparities in results.
- When disparities are observed, consider mitigation or re-evaluate suitability of tool use. For example, the user might only use a tool to classify certain subgroups and use traditional approaches to classify the subgroups where it performs poorly.
- If fairness audits are run separately, such as by using other software libraries, record these audits and decisions in the documentation for that tool run and retain outputs along with artifacts from the tool itself.

Efficiency

- When working with large datasets, work in stages. Conduct initial profiling and exploration with a small representative sample to tune parameters before scaling to larger batches of data.
- Consider how best to subset columns and rows. For example, perhaps rows can be sampled but all columns need to be retained for inference.
- Adjust tool parameters thoughtfully to improve performance (see following recommendations for individual tools).
- Cache reusable computations (such as embeddings or document chunking) and artifacts across runs, to avoid recomputing unnecessary steps.
- Monitor runtime, memory, and LLM token usage; log these metrics in the run documentation so that future teams can estimate cost and plan resource capacity.

Tool-Specific Best Practices

The following section lays out some more detailed best practices that apply when using each individual tool.

Best Practices for Data Harmonizer

Transparency and Reproducibility

- Make use of the tool's functionality for outputting SQL code. This can be referenced to serve as documentation of the data transformation logic, the final field names, and so on.
- Save final approved set of mappings and any generated SQL code.
- Log original names of input columns.
- If the user made manual edits to the tool's automatically proposed schema (e.g., renaming fields), or if a target schema template was used, document this.

Keeping Humans in the Loop

- Identify high-importance fields (such as key identifiers) beforehand and ensure that mappings for these fields are manually reviewed and approved.
- Utilize the tool's interactive interface to review, edit, or reject the suggestions for canonical field names.
- Spot-check output files to verify that data transformations have taken place as expected.
- Track if any necessary fields were left unmapped.

Privacy and Security

- Redact any personally identifying information values from examples when documenting the run.

Bias and Fairness

- Inspect mappings for codes and categories to ensure that important subgroups are not collapsed or merged, erasing nuance.
- Document rationale when consolidating categories, so that users can reverse or refine decisions.

Efficiency

- For large datasets, infer dataset schema based on a subset of data (containing all columns but a sample of rows) rather than running full-scale harmonization within the tool. Use the generated SQL code to carry out harmonization in a database for better performance.

Best Practices for Free Text Encoder

Transparency and Reproducibility

- If preprocessing is carried out on input data before it is uploaded to the tool (e.g., converting to lowercase, removing stop-words, stemming and lemmatization), record these steps along with the run parameters.

- Document exactly which parameter selections were made – e.g. the LLM version, the clustering algorithm used, which columns were used to determine the groupings, and the minimum confidence threshold set.
- Download and save copies of the clustering results and report files.

Keeping Humans in the Loop

- Manually review the clusters before they are finalized – both the cluster labels and a random sample of members.
- Leverage the LLM-based cluster summary feature to help determine whether output clusters provide useful and representative coverage of the thematic areas of interest and align with analytic questions.
- Screen for incoherent or mixed clusters; if unsatisfactory, re-tune parameters (e.g., desired number of clusters, confidence threshold).
- Review quality metrics displayed in the tool, such as Silhouette score, to determine output quality; refer to in-tool explanations to assist with interpretation.
- If human review is not feasible or results are low-quality, treat output as exploratory only, and refrain from operational deployment.

Privacy and Security

- Remove or mask personally identifying information in datasets before encoding free text. Do not upload sensitive variables if not being encoded.
- Govern access to model artifacts, as embeddings can be a source of data leakage.
- If defining custom categories, avoid including sensitive information in category descriptions.

Bias and Fairness

- Review clusters to ensure that subgroups are suitably represented. Adjust parameters (e.g., number of clusters, confidence threshold) if needed.
- Pay particular attention to text that is written in less common languages, dialects, or styles of expression (e.g., slang), to ensure that this has been correctly classified.
- Consider what is the suitable assignment mode for custom categories – i.e. if records can be members of multiple clusters.
- Review clusters for offensive or stigmatizing language in cluster names and summaries/descriptions.

Efficiency

- For large datasets, consider uploading records in batches, with batches being randomly assigned to minimize skew.
- Only upload the columns that contain the necessary identifiers and the variable(s) to be encoded.

Best Practices for Missing Data Assistant

Transparency and Reproducibility

- Record imputation method and hyperparameters selected (e.g., which imputation algorithms were selected for each column), and ensure that this is documented separately for every variable being imputed.
- Document random seed, if one is set.
- Download and save copies of the results, including the imputation flags that indicate which values were imputed by the tool
- In addition to the dataset itself, save the data profile before and after imputation.

Keeping Humans in the Loop

- Consider beforehand, based on domain knowledge, whether data is missing not at random (i.e. the likelihood of data being missing is systematically related to the value of that data itself), in which case it will not be suitable for automatic imputation with this tool.
- When imputing critical variables, require review by subject matter experts, and consider manual edits based on domain knowledge.
- Review the descriptive summaries of the dataset (variable distributions and summary statistics) both before and after imputation, to compare and determine whether the dataset has changed in ways that are unintended or seem statistically unreasonable (e.g., variance inflation).
- Define guardrails – e.g., if an imputed value is outside a certain plausible range, note this and require manual review.

Privacy and Security

- Limit inclusion of direct identifier variables.

Bias and Fairness

- Investigate patterns in output, such as if one subgroup sees systematically larger imputed values.
- Avoid imputing protected attributes unless this is explicitly required and approved; if so, document the rationale and chain of approval.

Efficiency

- Consider starting with simpler imputation methods and only escalating to more complex methods if needed.
- Only carry out imputation for columns with substantial missingness.
- When working with large datasets, consider carrying out imputation in batches.

Best Practices for Metadata Extractor

Transparency and Reproducibility

- When downloading the generated metadata, assign it a version number or label that can be matched to the input dataset, to identify which specific version of the input corresponds to which version of the output.
- If manually amending or annotating the output, log these modifications.

Keeping Humans in the Loop

- Consider beforehand if input files are suitable. The tool is not optimized to work with especially complex nested tables that have a significant degree of stylized formatting or lack clearly defined header rows. It can still be useful for a quick exploration of such files, but manual validation should be carried out to ensure that all columns have been detected and accounted for.
- Ensure that all fields in the input file are accounted for in the generated output.
- Spot-check field definitions for correctness based on domain knowledge.
- Amend or annotate metadata when automated inferences are uncertain or incorrect.

Privacy and Security

- Ensure that metadata does not expose sensitive values (such as example records with personally identifying information); avoid uploading such sensitive variables to the tool if not strictly necessary.

Bias and Fairness

- Ensure that definitions for demographic variables are complete and up-to-date.

Efficiency

- Again, note that the tool is not optimized to work with complex nested tables that have a significant degree of stylized formatting or lack clearly defined header rows; consider manual review in these cases.

Best Practices for Tabular Data Extractor

Transparency and Reproducibility

- Document all parameter selections. This includes the exact prompts, LLM context size, and whether or not Retrieval-Augmented Generation is used.
- Document any post-processing steps.

Keeping Humans in the Loop

- Test multiple versions of prompts. The tool takes a natural-language question as a prompt (e.g. “What stations did each Amtrak route serve in 2023?”), and variations of question wording will affect the level of detail in the output. The user should experiment with different prompt iterations if the initial output is unsatisfactory.
- Ensure that keywords in the prompt are directly relevant to the content of the input file(s).
- When crafting the prompt, clearly specify the desired output columns and any other requirements for structuring the results.
- When conducting manual spot-checks, identify critical variables/values and ensure that these are prioritized for checking against input pages/documents.

Privacy and Security

- If personally identifying information is included in the extraction, ensure that sensitive variables are appropriately redacted before output is shared or released.
- Ensure that source documents and intermediate cached files (e.g., the vector embeddings produced during RAG) are all stored with suitable access controls.

Bias and Fairness

- Especially when working with multiple input files, ensure that all sources are covered in the extraction and that no part of the input has been skipped. If anything has been skipped, document these omissions.
- If giving examples of desired output in the prompt, make sure that these examples cover all subgroups (e.g., if data for certain race/ethnicity categories should be present in the output, specify that all of these categories are expected).

Efficiency

- Be mindful of input file size and length, as well as the total number of input files. Tool performance varies with the length of the input document; for example, in long documents, data may not be extracted after a certain point. Comparison across many files is also impeded by LLM token limits.
- The LLM context size is the maximum word limit that determines the chunk size for document splitting. The value to which this is set therefore affects the amount of context the tool retains in producing an answer.
- Using the RAG option can improve accuracy and efficiency for large documents, as the document will be converted into stored chunks and only the most relevant chunks will be retrieved. However, this means that if relevant context is not retrieved, the accuracy of the response will be negatively affected.
- Specify desired output columns explicitly in prompts, to reduce the need for post-processing.

- If input has complex structure or formatting that leads to poor results, consider reverting to manual extraction instead. Likewise, if the required information is scattered across multiple sections that are far apart in the document, the tool's automated performance may be poorer than manual extraction.
- Note that the tool is not well-suited for deep contextual reasoning beyond what is stated directly in the retrieved text, or for questions about structural metadata (e.g., the number of rows, columns, or pages in the document itself).

Guidance on Privacy and Ethical Concerns

This deliverable summarizes key privacy, security, and ethical concerns that may arise when using AI-powered tools in federal statistical agency workflows, especially tools that support data quality, standardization, and integration. The guidelines and key considerations in this deliverable draw on the results of framework development over the course of the AI-DQSI project – including the lessons learned from conversations with federal statistical agencies – to delineate key privacy and ethical concerns that surface when working with AI-powered tools in general, and the AI-DQSI toolkit specifically.

Deployment Context

It is worth noting that the considerations described below depend heavily on where and how the tools are deployed.

If the tools are **publicly deployed on the NSDS website**: No confidential, restricted, or sensitive data should ever be used. Users must assume that all inputs and outputs are visible to unauthenticated users. Only synthetic, public, or fully de-identified and disclosure-proofed data should be processed by the tools. Ideally, the tools would detect and warn or prohibit uploads of certain file types or schemas that can be inferred/are known to contain sensitive information (e.g. schemas with an “SSN” field).

If the tools are **deployed internally within an agency’s secure environment:** Tools may work with data protected by CIPSEA, FERPA, HIPAA, or other legal requirements, if and only if this is permitted under the appropriate data governance protocols. Additionally, the considerations described in the following section apply in full and must be built into all aspects of tool usage, including: the access controls for the tools; tool audit logs; encrypted storage; and tool data retention and disposal policies.

Potential Privacy and Security Concerns

Risk of Adversarial Attacks

There is the possibility of adversarial attacks that attempt to extract sensitive information or manipulate outputs. This includes attacks that target the model itself or the surrounding workflow (prompts, retrieval sources, tool calls, or post-processing). Specifically, they may seek to infer sensitive attributes, recover them from embeddings or intermediate outputs, extract memorized training or prompt content, or re-identify records. They may also interfere with the tool directly by manipulating tool decisions (e.g., through prompt injection), thus inducing the system to produce incorrect or biased outputs. Finally, they may compromise logs or temporary processing artifacts.

General mitigation strategies for public deployment. In a public deployment environment, it is essential to operate under the assumption that all inputs are untrusted. Any user may upload data or

attempt to manipulate the system, and the platform must therefore treat every submission as potentially malicious or improperly formatted.

Public deployments should also be understood as environments in which adversaries may intentionally probe the system to uncover weaknesses, extract information, or induce incorrect behavior. Because of this, administrators and system designers must be fully aware of these risks and ensure that appropriate countermeasures are in place. This includes anticipating attempts to bypass validation rules, overwhelm the system, or inject content designed to manipulate downstream processing. Public-facing tools should be built with robust input sanitization, strong validation controls, and protective monitoring to detect unusual patterns of use, which may indicate that adversaries are intentionally testing system boundaries.

General mitigation strategies for internal deployment. Internal deployments require a more nuanced and layered set of protections, because they often operate on sensitive, legally protected, or confidential data. In this context, access should follow role-based or attribute-based controls, ensuring that users can view and manipulate only the minimum amount of data necessary for their specific job functions. This principle limits the blast radius of accidental disclosure and mitigates insider risks.

All models and supporting services should be run within agency-managed secure compute environments, such as secure enclaves or other accredited infrastructures. Sensitive data should not be sent to public endpoints unless the agency has reviewed and approved contractual, technical, and legal safeguards that meet its compliance requirements. To further protect sensitive information, both input and output filtering should be applied to detect and block fields that may contain PII or sensitive attributes.

Systems should maintain logging and active monitoring to identify anomalous queries, unusual usage patterns, or suspicious access attempts. Rate-limiting can provide an important additional safeguard against automated scraping, extraction, or brute-force probing. Duties related to data access, model operations, and final data release decisions should be separated, to ensure that no single individual gains unilateral control over sensitive workflows.

Finally, internal deployments should undergo pre-deployment red teaming and regular reassessments to test for vulnerabilities such as prompt injection, data extraction attempts, model and output manipulation, or other emerging attack vectors. These exercises help identify weaknesses before they are exploited, and ensure that the system evolves as new threats and techniques arise. Routine testing and review help maintain the integrity and confidentiality of internal workflows, supporting the overall security posture of the agency.

Table 8. Tool-specific risks: Adversarial attacks

Tool	Potential Risks
Data Harmonizer	• Adversaries could exploit example values or schema previews to infer sensitive fields. • The SQL code produced by the tool could reveal structure of protected databases if exported unsafely.
Free Text Encoder	• Text embeddings can leak information about original raw text. • Cluster summaries could reveal personally identifying information.
Missing Data Assistant	• Imputation models could learn patterns that indirectly reveal sensitive attributes (e.g. imputed race from correlated variables). • The models' stored training matrices could be extracted if not secured.
Metadata Extractor	• Metadata sometimes contains sensitive contents (such as field names identifying protected data systems).
Tabular Data Extractor	• Extracted tables may inadvertently retain personal identifiers from PDFs or scanned documents. • Maliciously crafted PDFs may exploit model parsing/prompting components.

Tool-specific mitigations. Mitigation strategies for these risks are discussed more fully for individual tools in the Checklist of Best Practices, but a brief summary is that users should take care to sanitize all input files, including blocking or restricting any data that contains personally identifiable information. They should also ensure that data previews, generated summaries, and other automated outputs do not expose personal information, even inadvertently. Finally, agencies should limit access to intermediate files, logs, and temporary artifacts, as these often contain sensitive details that must be protected throughout the workflow.

Potential Data Misuse or Leakage

Even without adversarial attacks, during the process of tool usage, input datasets may be inadvertently disclosed or shared with people who should not have access. Alternatively, these datasets may end up being used for purposes for which they were not originally intended (e.g., for administrative, regulatory, or enforcement actions, rather than strictly for statistical purposes). Additionally, unauthorized users may upload data to a tool in ways that bypass or strain existing governance processes.

Separate from user access issues, data may also be unintentionally stored in logs, cached in temporary files, or exposed through exported reports or summary files generated by the tools.

General mitigation strategies for public deployment. In a public deployment environment, leakage risk is exceptionally high because the system must assume that any user may upload any content, including sensitive or improperly de-identified data. Hence, the tools should not accept any input that could produce sensitive intermediate data, such as row-level person-level records, spreadsheets with

identifier-like column names, or PDFs containing administrative forms. Beyond refusing sensitive data, public deployments should also incorporate automated screening and file-type restrictions, blocking uploads of formats commonly associated with confidential information.

General mitigation strategies for internal deployment. In internal deployments, mitigation focuses on ensuring that all data processing complies with the legal and privacy protections attached to each dataset. Agencies should maintain clear data governance processes that track the respective applicable laws and consent requirements (such as FERPA for education records, HIPAA for health data, and CIPSEA restrictions ensuring datasets are used solely for statistical purposes), and these protocols must be consistently enforced throughout the workflow. Where appropriate, agencies may consider using privacy-preserving record linkage or other statistical privacy techniques to reduce the risk of disclosure during data integration or matching tasks. Tool users should provide only the variables necessary for a given task, redacting direct identifiers and reducing granularity whenever possible to limit exposure of sensitive information.

Additionally, systems should strictly limit the content stored in application logs, prompts, cached files, and telemetry data, since these intermediate artifacts often contain traces of sensitive fields. Agencies should implement retention schedules that ensure these artifacts are stored only as long as required and then securely deleted using approved disposal methods. Together, these measures help to protect sensitive information throughout the tool's end-to-end lifecycle.

Table 9. Tool-specific risks: Potential data misuse or leakage

Tool	Potential Risks
Data Harmonizer	• The harmonized datasets can expose underlying raw identifiers if not redacted. • The generated SQL code can reveal structure of protected databases.
Free Text Encoder	• Free text routinely contains unstructured personal information; accidental clustering or summaries may echo back sensitive content.
Missing Data Assistant	• Imputation flags and intermediate matrices may reveal structural details of restricted datasets.
Metadata Extractor	• Some metadata fields (like example values) may include personally identifying information or protected attributes.
Tabular Data Extractor	• Extracted tables may include sensitive information such as names, addresses, dates of birth, or administrative IDs.

Tool-specific mitigation. Mitigation strategies for these risks are discussed more fully for each tool in the Checklist of Best Practices, but in brief, users should ensure that each tool is supplied only with the minimum data necessary for the task at hand, reducing exposure of sensitive information wherever possible. Access to intermediate files, logs, and other temporary artifacts should be tightly restricted, and any temporary files that are not essential for documentation or reproducibility should be securely deleted once processing is complete.

For public deployments in particular, systems should block the upload of any files that may contain PII or other sensitive content, since public interfaces cannot safely process or store such data.

Potential Ethical Concerns

Algorithmic Bias

As a result of algorithmic bias, AI-powered tools have often displayed differential performance across different subgroups/sensitive populations, which may lead to unfair or discriminatory policy outcomes.

Table 10. Tool-specific concerns: Algorithmic bias

Tool	Potential Concerns
Data Harmonizer	• Category mappings may collapse minority or legacy subgroups.
Free Text Encoder	• Textual embeddings and clustering often under-represent less common dialects, languages, or expression styles. • LLM summaries may mischaracterize subgroups.
Missing Data Assistant	• Imputation error can vary across demographic subgroups, potentially reinforcing inequalities in derived statistics.
Metadata Extractor	• Incorrect inference of variable types or categories could disproportionately affect small or under-represented groups.
Tabular Data Extractor	• Extraction quality may vary for forms/documents used in smaller programs or geographic areas; inconsistent coverage of different input layouts can bias downstream analyses.

Mitigation strategies. End users should rely on each tool's output quality metrics, as documented in this report, when reviewing results. Users should manually evaluate whether the tool's outputs are consistent and accurate across different subgroups – for example, by checking for parity in error rates or verifying that mappings behave similarly across demographic categories. This review helps determine whether the tool's outputs are suitable for inclusion in statistical products or whether further refinement is needed.

In addition, end users should ensure that any pre-processing steps that could affect input data quality (such as imputation, masking, or general data cleaning) are clearly captured in their documentation. This transparency not only supports reproducibility but also allows future analysts to understand how such steps may influence outcomes.

Where outputs appear questionable, end users should manually adjust thresholds, revise problematic mappings or labels, and record these decisions for future reference. Such interventions are often necessary to ensure that the outputs align with domain knowledge and support high-quality statistical reporting.

Finally, end users should consider conducting fairness audits or applying bias correction techniques using external tools or software, especially when the stakes of downstream statistical products are high. These additional checks help identify disparities that may not be apparent from the tool's built-in diagnostics and support a more rigorous assessment of model performance across varied populations.

Output Quality Issues

Tool output quality issues may lead to the dissemination of misinformation. In particular, when end users lack the specialized domain knowledge to detect issues or errors in the tool output, this may lead to the proliferation of outdated information, "hallucinations" (fake data), inaccuracies, misattributed sources, misleading summaries, and inconsistent output in response to identical prompts.

Table 11. Tool-specific concerns: Output quality issues

Tool	Potential Concerns
Data Harmonizer	• The suggested mappings fields may be incorrect, leading to the incorrect conflation of concepts or categories.
Free Text Encoder	• LLM summaries may misinterpret and misrepresent the contents of clusters. • Clustering may group unrelated text in an incoherent way.
Missing Data Assistant	• Overaggressive imputation can distort distributions and create statistically implausible values.
Metadata Extractor	• Automatically generated variable descriptions may be wrong or incomplete, leading to misunderstandings of what variables signify that propagate in downstream analytical use.
Tabular Data Extractor	• Extracted tables may introduce transcription errors, formatting inconsistencies, or misaligned rows, producing incorrect data that is used in analyses.

Mitigation strategies. For internal or official statistical uses, any high-impact outputs should undergo review and sign-off by subject matter experts who can identify errors, misinterpretations, or gaps in the generated content. When the necessary domain expertise is not available, users should adopt conservative defaults, require additional validation steps, and avoid disseminating outputs that have not been thoroughly vetted for accuracy and consistency. Users should recognize that complex models cannot always provide a complete mechanistic explanation for exactly why a particular result was produced, and should not make this assertion when presenting results, or treat such outputs as authoritative without validation.

To support high-quality outputs, users should also document every aspect of how the tool was configured during a given run. This includes noting the model name and version, all prompts used, parameter settings, retrieval sources and their versions, quality control checks applied, and any post-processing procedures. For models that incorporate stochastic components, users should be sure

to record parameters that influence this random variation, such as model temperature, random seed values, and any other generation settings. Capturing this information helps to ensure that unexpected or inconsistent outputs can be traced back to specific configurations and evaluated for their reliability.

Appendix A: Frequently Asked Questions

This FAQ addresses common questions raised during demonstrations and early use of the AI-DQSI Toolkit, particularly around data sensitivity, model behavior, reproducibility, and deployment considerations.

Question	Answer	Applicable Tool(s)	Deployment Dependent?	Section
Can these tools be used with restricted or confidential data?	Yes, when deployed in a secure, agency-controlled environment that complies with applicable legal and governance requirements. Public or demonstration deployments should only use synthetic, public, or fully de-identified data.	All tools	Yes	Deployment Context; Privacy and Security
Does the toolkit send data to the open internet?	No. The tools do not browse the internet or retrieve external data. When LLMs are used, they only process data explicitly provided during the tool run (e.g., uploaded data, documents). LLMs may rely on their general pre-training to interpret language, but they do not access external content beyond the configured model endpoint.	LLM-enabled tools	No	Model Architecture; Responsible Use of AI
Where does the LLM “look” when interpreting data?	The LLM processes only uploaded datasets, extracted metadata, or retrieved document chunks provided by the tool. It relies on general model training, not “live” external data sources.	LLM-enabled tools	No	Model Architecture
Why not use ChatGPT or another general-purpose LLM instead?	The toolkit provides task-specific preprocessing, deterministic steps, structured outputs, cost controls, and auditability that are not available in general-purpose chatbot interfaces. These components comprise a set of guardrails that help users to more easily get appropriate, high-quality results.	All tools	No	Comparison with Traditional Approaches
How do the tools control cost compared to ad hoc LLM use?	The tools preprocess and structure inputs so that only targeted, relevant information is sent to the	LLM-enabled tools	No	Cost Guidance; Model Architecture

Question	Answer	Applicable Tool(s)	Deployment Dependent?	Section
	LLM. LLMs are only invoked for steps where they have particular value over deterministic methods. These strategies reduce token usage and avoid unnecessary LLM calls.			
Are the tools optimized for specific LLMs?	Yes. Prompts and workflows are authored and tuned for specific model behaviors to improve reliability and output quality. Model switching is possible but may impact performance.	LLM-enabled tools	No	Model Architecture
Can users switch LLMs?	In some tools, yes, subject to deployment permissions. However, prompts re optimized for particular models, so switching may affect consistency, cost, or accuracy.	LLM-enabled tools	Yes	Model Architecture; Settings
How do the tools address LLM non-determinism?	The toolkit combines LLM outputs with deterministic preprocessing, constrained prompts, validation rules, and human review to improve consistency across runs.	All tools	No	Logging and Reproducibility
Are results reproducible?	Partially. The tools preserve run artifacts such as schemas, mappings, normalization rules, parameters, and outputs to support review and reuse, but users are responsible for exporting or saving results for long-term retention. Because some steps use Large Language Models, exact reproducibility is not guaranteed; however, deterministic preprocessing, constrained prompts, and human review are used to promote stable and interpretable results.	All tools (especially Data Harmonizer)	Yes	Logging and Reproducibility
What information is saved between sessions?	This varies by tool. The Data Harmonizer persists full run artifacts by default, provided the application is deployed with persistent storage. The Free Text Encoder optionally caches small amounts of run metadata by default, which is fully configurable and does not constitute saved runs. The Tabular Data Extractor	Tool-specific	Yes	Tool-Specific Logging and Reproducibility

Question	Answer	Applicable Tool(s)	Deployment Dependent?	Section
	writes small, temporary diagnostic files that are overwritten on subsequent runs. The Metadata Extractor and Missing Data Assistant do not save run artifacts; users must download outputs to retain results. See tool-specific documentation for details.			
Who controls data retention and deletion?	Data retention is governed by deployment configuration and organizational policy. Administrators may delete files, clear volumes, or destroy containers as needed.	All tools	Yes	Deployment Context
How are these tools intended to fit into real analytical workflows?	The toolkit supports upstream tasks (data ingestion, documentation, structuring, harmonization, imputation, and extraction) that prepare data for downstream statistical analysis.	All tools	No	How the Tools Fit Together

Appendix B: Dataset Reference Table

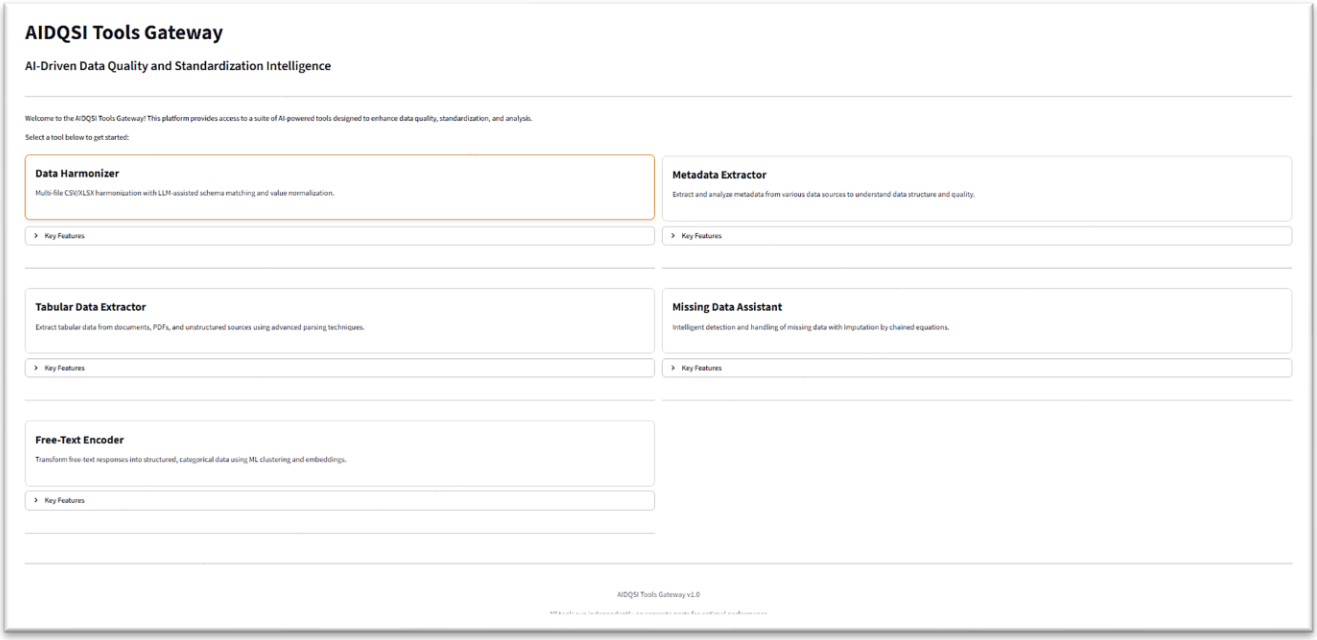
Dataset Name	Dataset Link	Tool Dataset Was Used to Develop/Test
Chicago 2001 to Present Crime Data	https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2/about_data	Data Harmonizer
Las Vegas Crime Data	https://opendata-lvmpd.hub.arcgis.com/datasets/ef9c4bbbdc804e65a6a6b8ce7dc86107_0/about	Data Harmonizer
Philadelphia Crime Data	https://cityofphiladelphia.github.io/carto-api-explorer/#incidents_part1_part2	Data Harmonizer
Madison Crime Data	https://data-cityofmadison.opendata.arcgis.com/datasets/cityofmadison::police-incident-reports/explore	Data Harmonizer
Cincinnati Crime Data	https://data.cincinnati-oh.gov/safety/Reported-Crime-STARS-Category-Offenses-on-or-after/7aqy-xrv9/about_data	Data Harmonizer
Raleigh Crime Data	https://data-ral.opendata.arcgis.com/datasets/24c0b37fa9bb4e16ba8bcaa7e806c615_0/explore?location=35.797271%2C-78.624284%2C9.38&showTable=true	Data Harmonizer
Scottsdale Crime Data	https://data.scottsdaleaz.gov/datasets/COS-GIS::police-incident-reports/about	Data Harmonizer
Los Angeles Building Permit Data	https://data.lacity.org/A-Prosperous-City/LA-BUILD-PERMITS/xnhu-aczu/about_data	Data Harmonizer
New York City Building Permit Data	https://data.cityofnewyork.us/Housing-Development/DOB-Permit-Issuance/ipu4-2q9a/about_data	Data Harmonizer
Chicago Building Permit Data	https://data.cityofchicago.org/Buildings/Building-Permits/ydr8-5enu/about_data	Free Text Encoder, Data Harmonizer
Austin City Cultural Centers Audit Community Survey Data	https://catalog.data.gov/dataset/city-cultural-centers-audit-community-survey-open-response-data	Free Text Encoder
Chicago Transportation FOIA Requests Log	https://data.cityofchicago.org/FOIA/FOIA-Request-Log-Transportation/u9qt-tv7d/about_data	Free Text Encoder
District of Columbia Crime Data 2021	https://opendata.dc.gov/datasets/DCGIS::crime-incidents-in-2021	Metadata Extractor, Data Harmonizer

Dataset Name	Dataset Link	Tool Dataset Was Used to Develop/Test
District of Columbia Crime Data 2023	https://opendata.dc.gov/datasets/DCGIS::crime-incidents-in-2023	Metadata Extractor, Data Harmonizer
American Community Survey 2024	Dataset created internally by NORC, but same data can be obtained through the Census API	Metadata Extractor
New York MTA Bus Speeds	https://data.ny.gov/Transportation/MTA-Bus-Speeds-2020-2024/6ksi-7cxr/about_data	Missing Data Assistant
New York City Ferry Ridership	https://data.cityofnewyork.us/Transportation/NYC-Ferry-Ridership/t5n6-gx8c/about_data	Missing Data Assistant
Congressional Transcripts	Dataset created internally by NORC, but same data can be obtained through the Congress.gov API	Tabular Data Extractor
Federal Register: Indian Entities	https://www.federalregister.gov/documents/2024/01/08/2024-00109/indian-entities-recognized-by-and-eligible-to-receive-services-from-the-united-states-bureau-of	Tabular Data Extractor
Amtrak State Fact Sheets	https://www.amtrak.com/state-fact-sheets	Tabular Data Extractor
California State Assembly Daily Journals	https://clerk.assembly.ca.gov/legislative-publications/daily-journals	Tabular Data Extractor

Appendix C: Screenshots of Tools

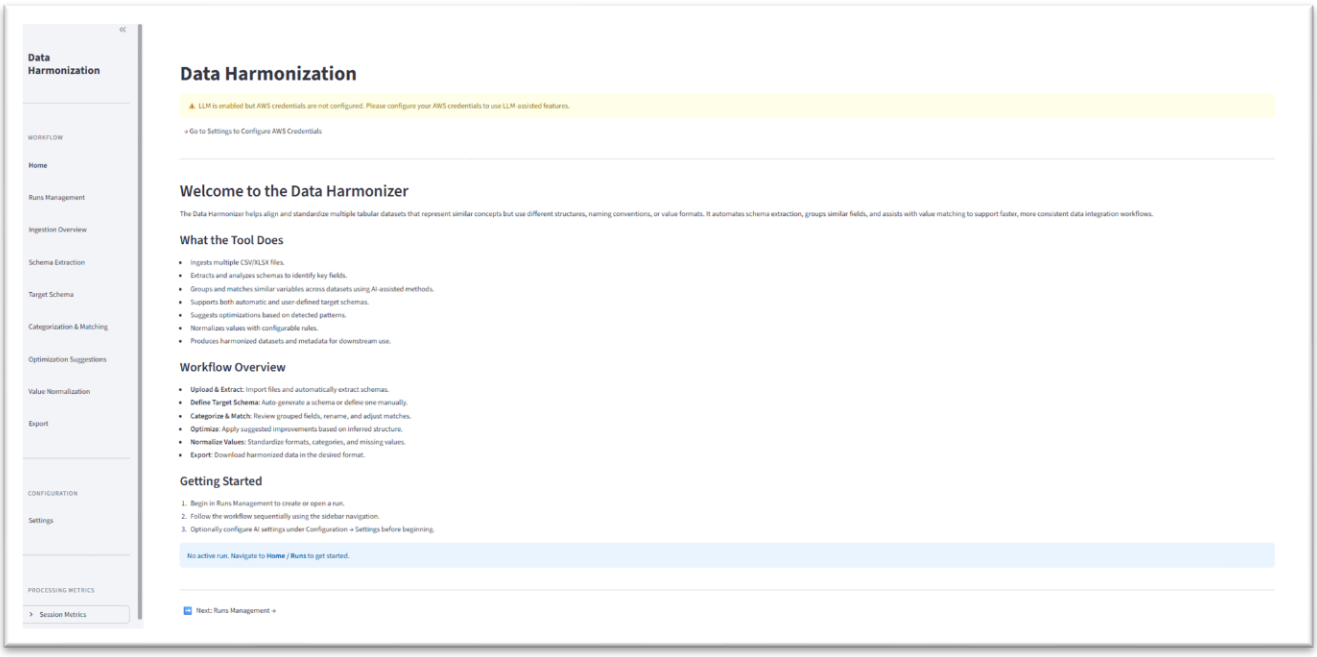
Toolkit Main Page

AI-DQSI Tools Gateway: Home Page

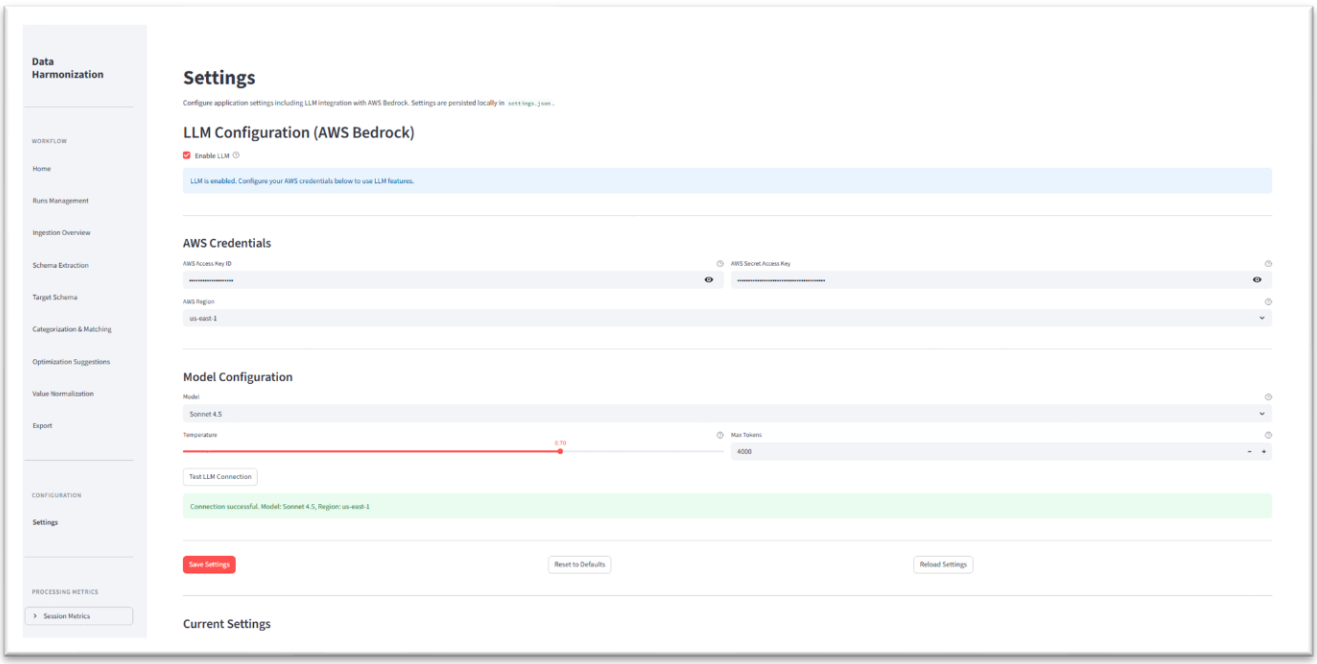


Data Harmonizer

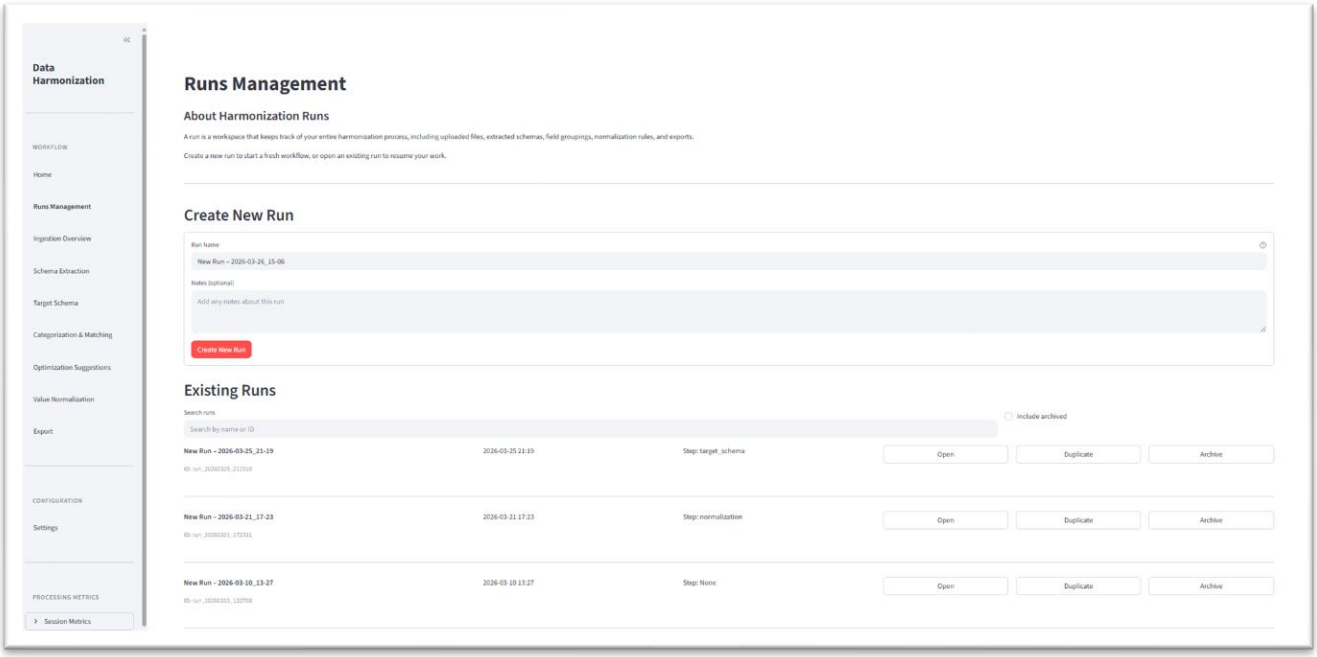
Data Harmonizer: Home Page



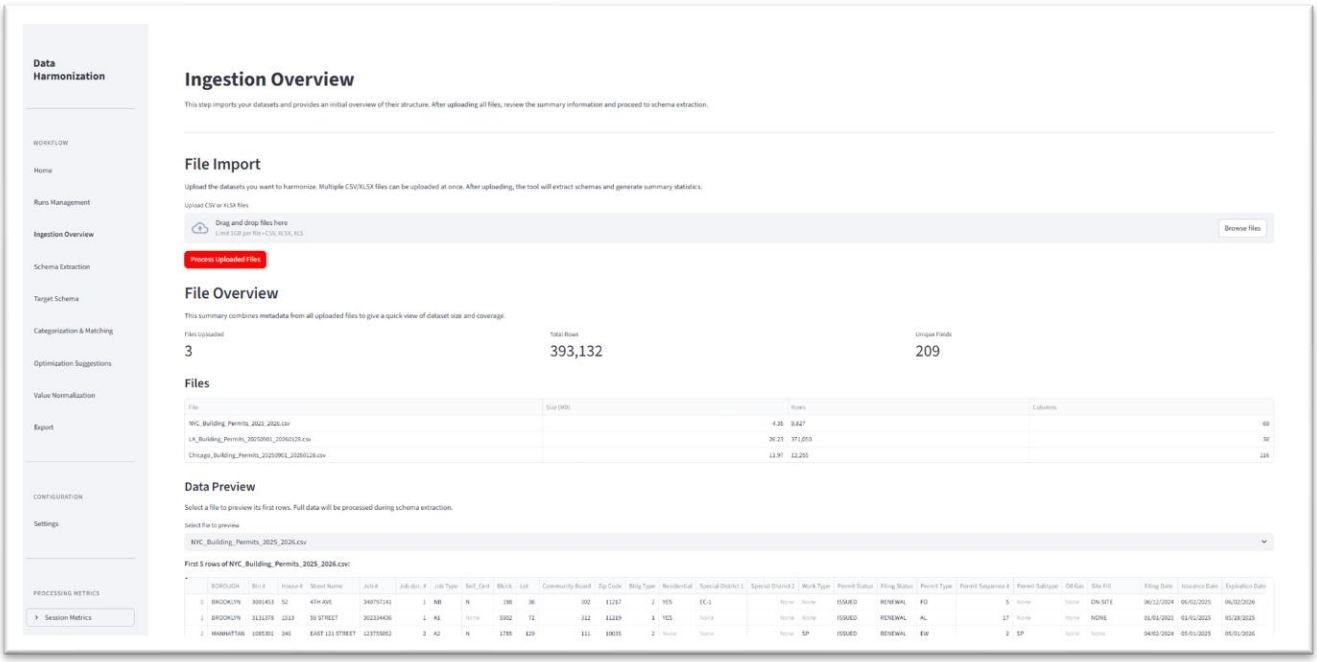
Data Harmonizer: Settings



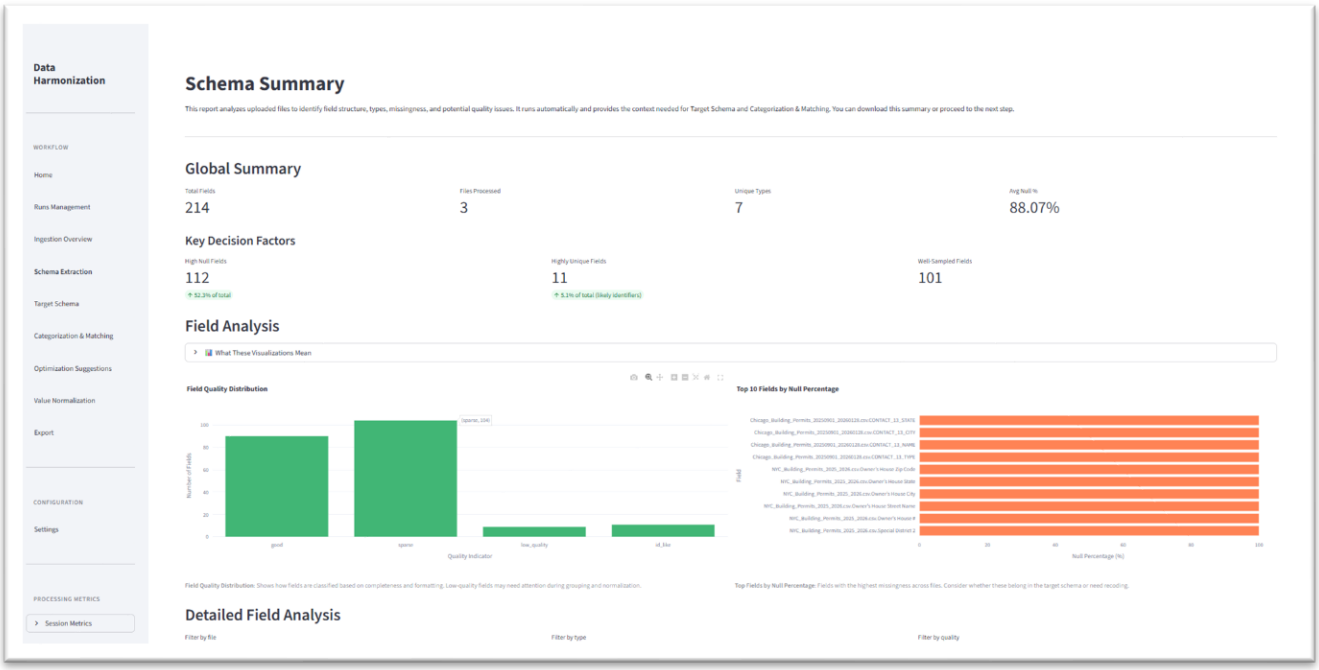
Data Harmonizer: Runs Management



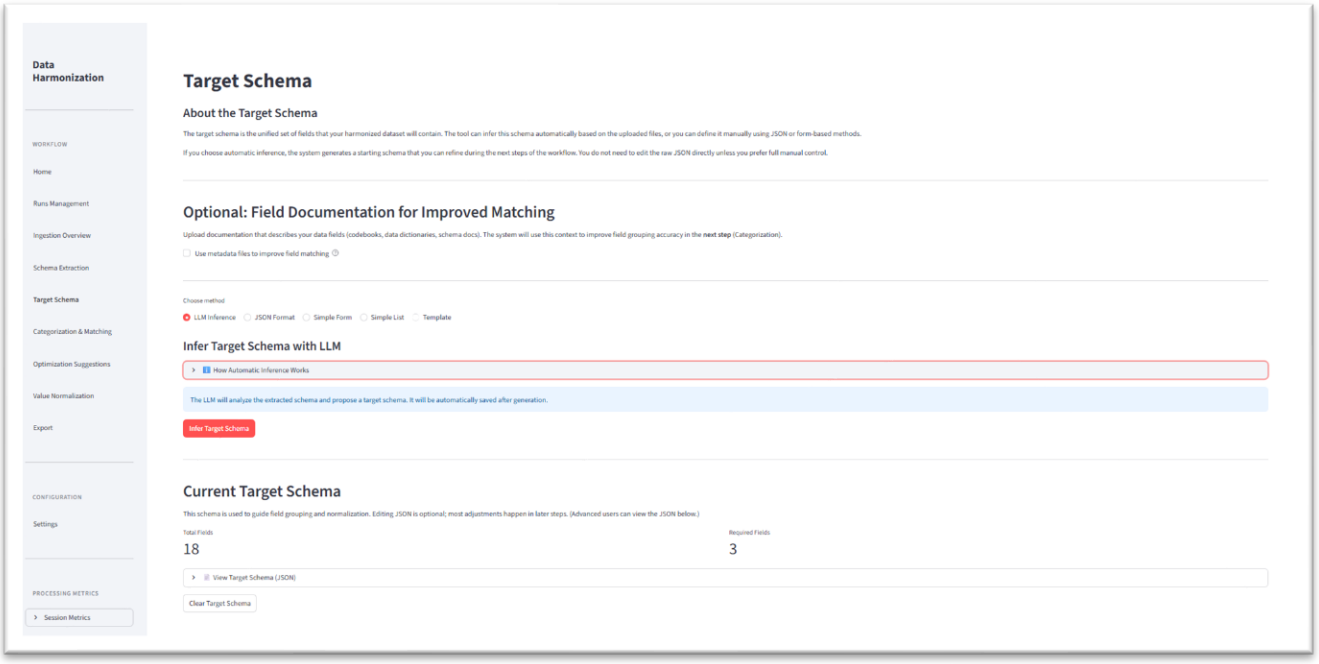
Data Harmonizer: Ingestion Overview



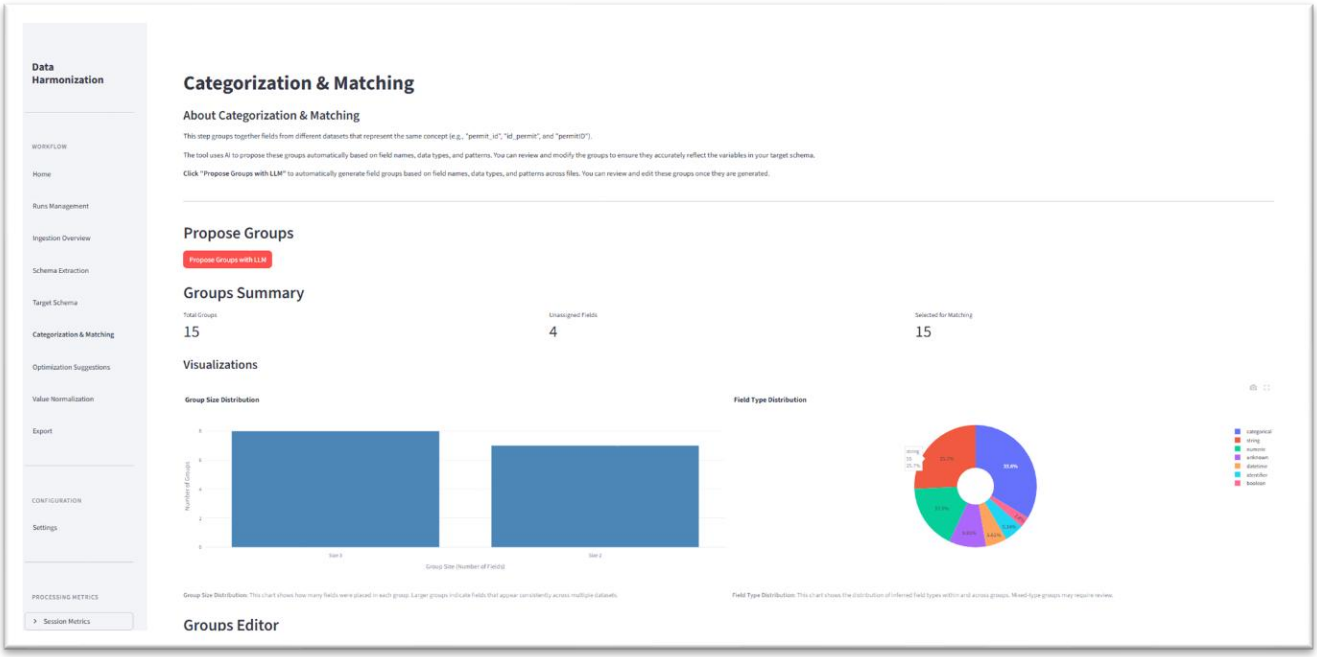
Data Harmonizer: Schema Extraction



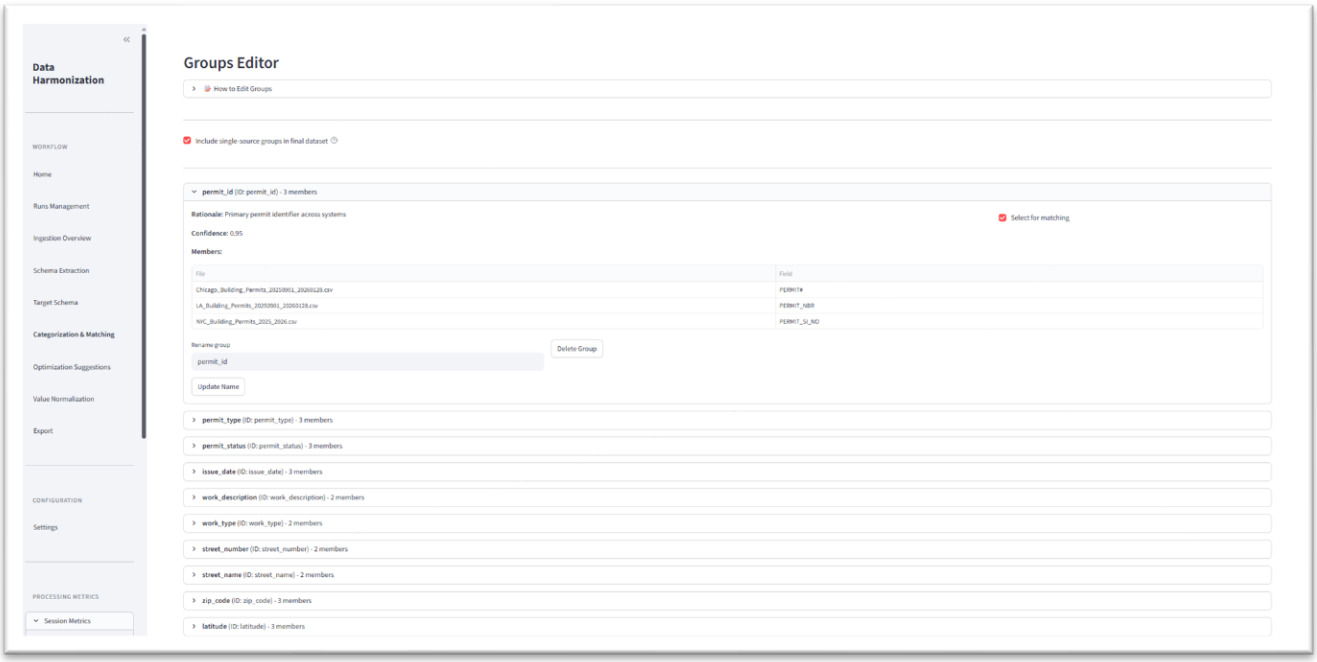
Data Harmonizer: Target Schema



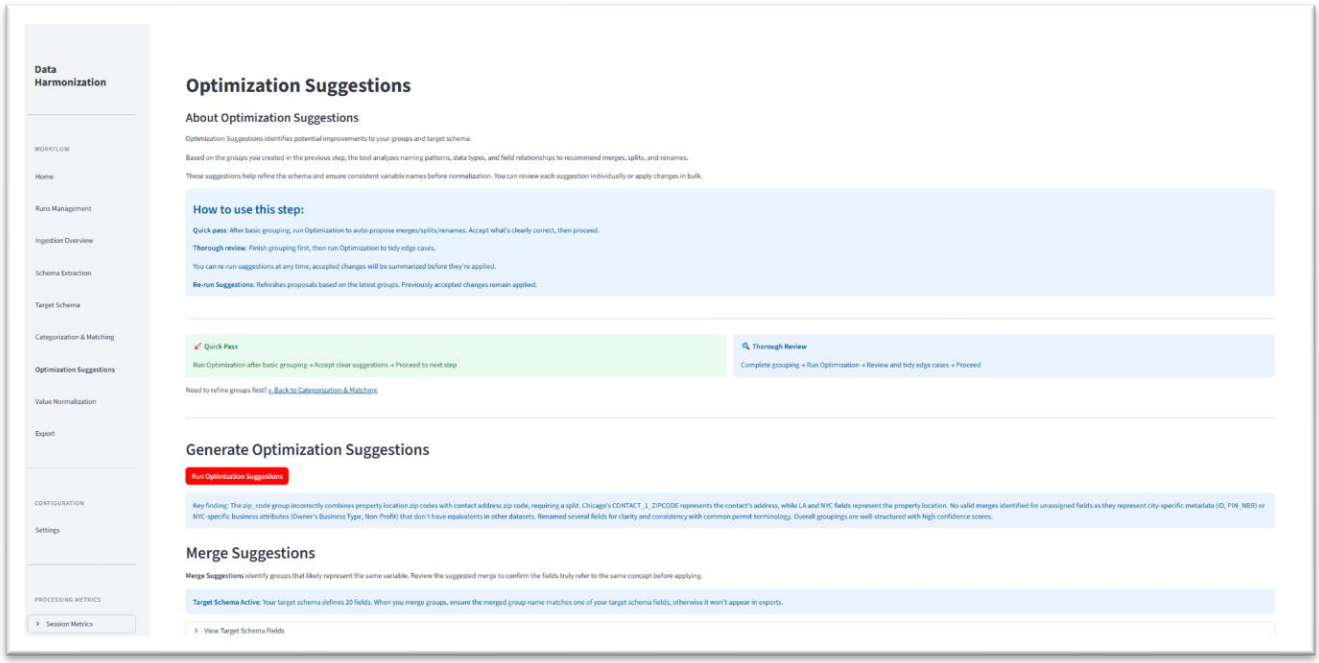
Data Harmonizer: Categorization & Matching (page 1/2)



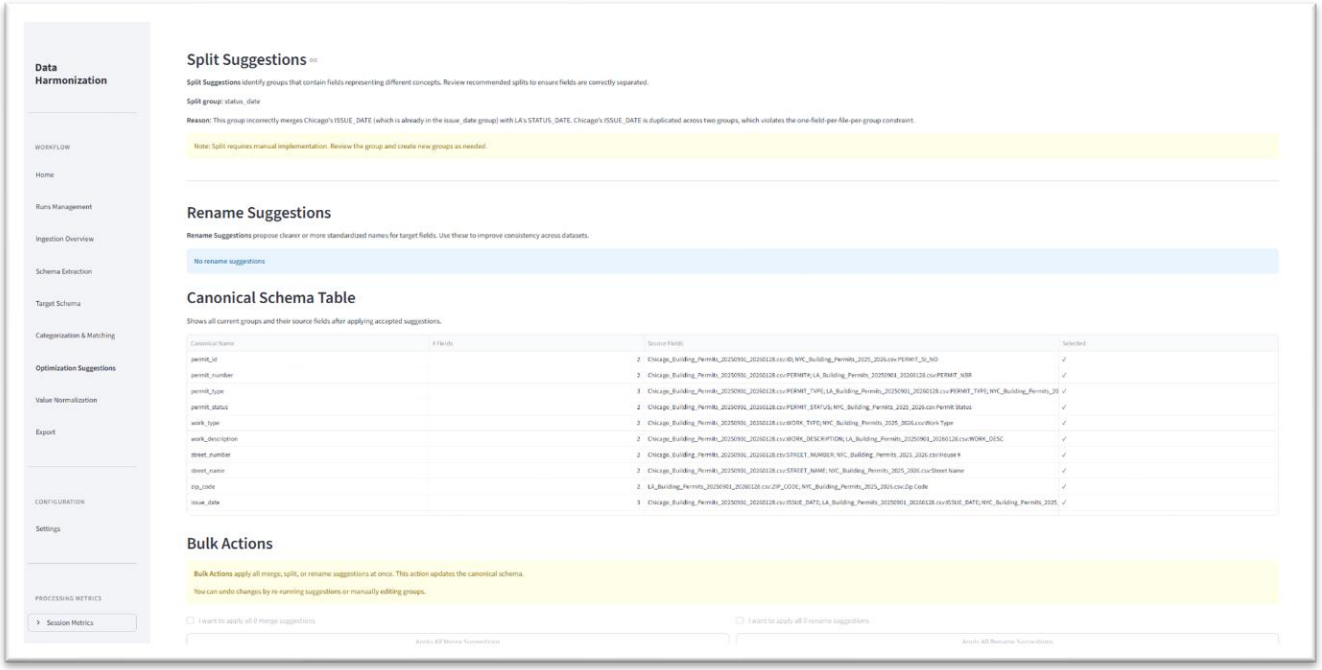
Data Harmonizer: Categorization & Matching (page 2/2)



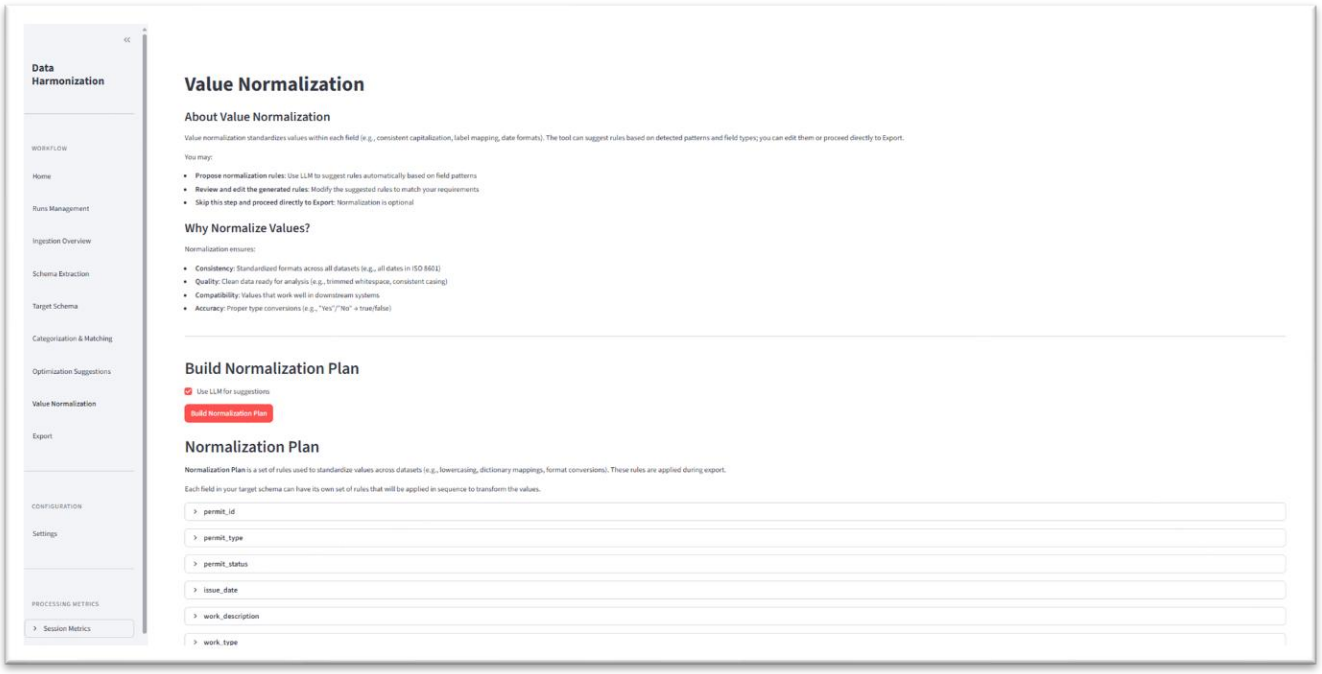
Data Harmonizer: Optimization Suggestions (page 1/2)



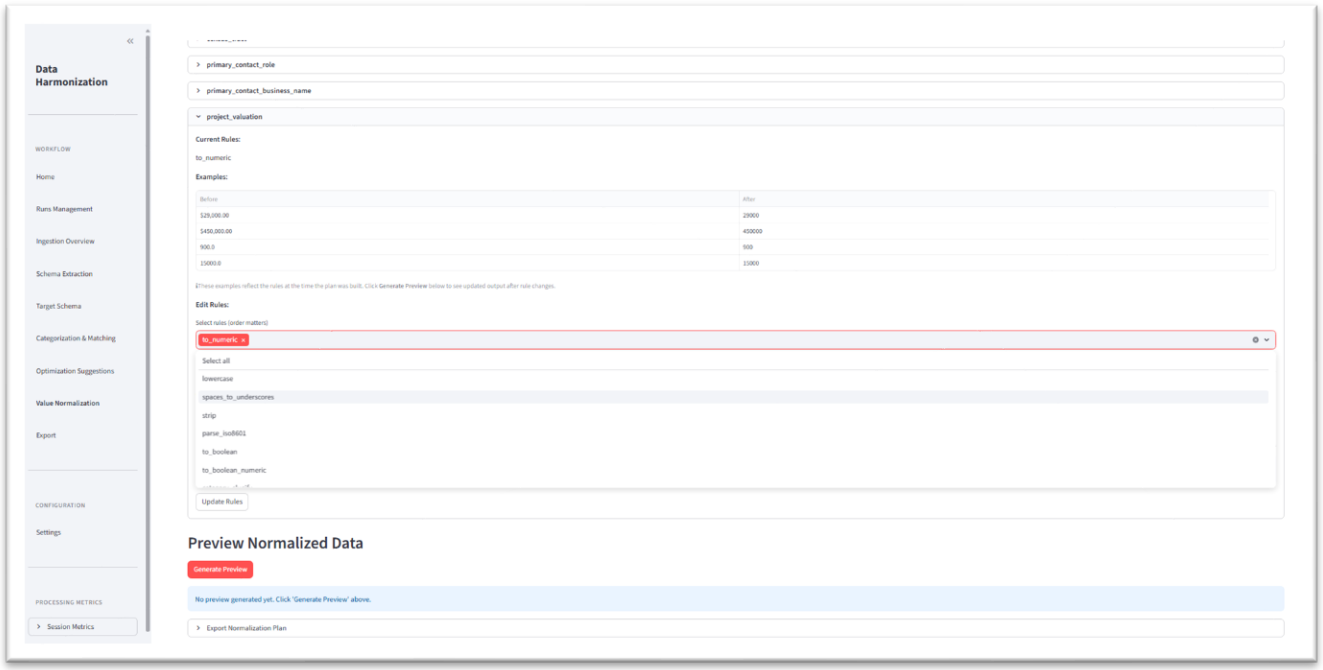
Data Harmonizer: Optimization Suggestions (page 2/2)



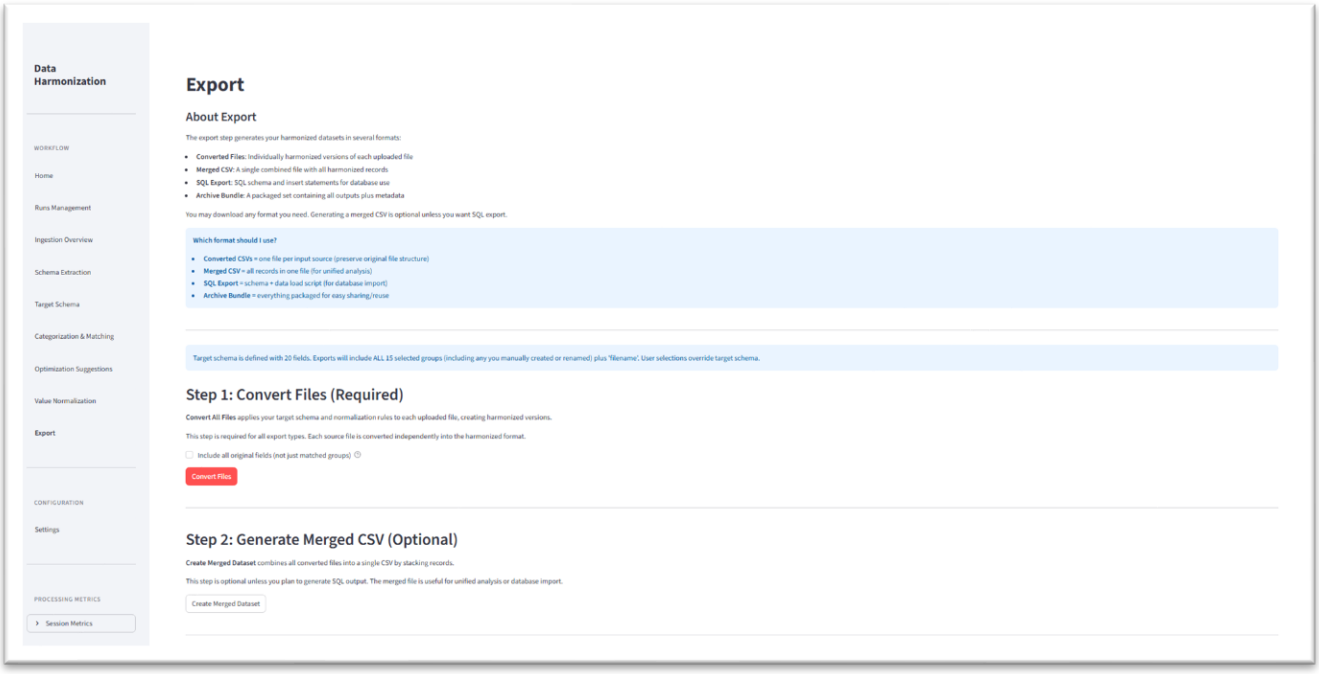
Data Harmonizer: Value Normalization (page 1/2)



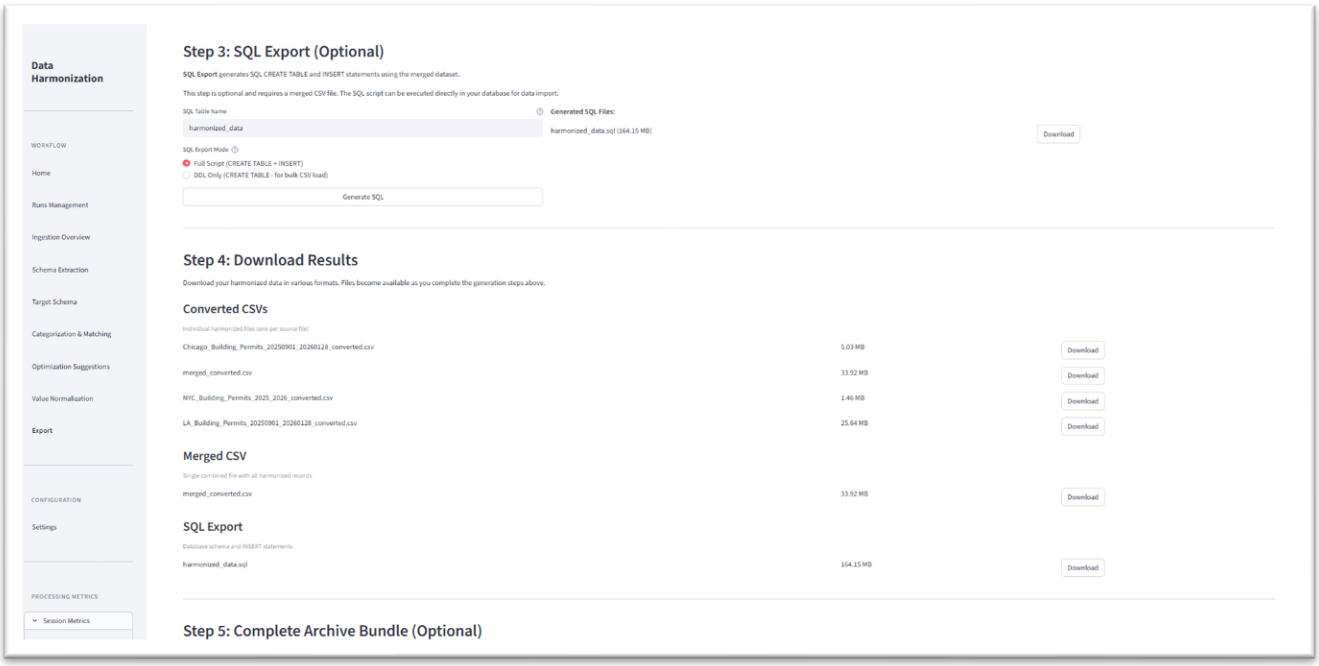
Data Harmonizer: Value Normalization (page 2/2)



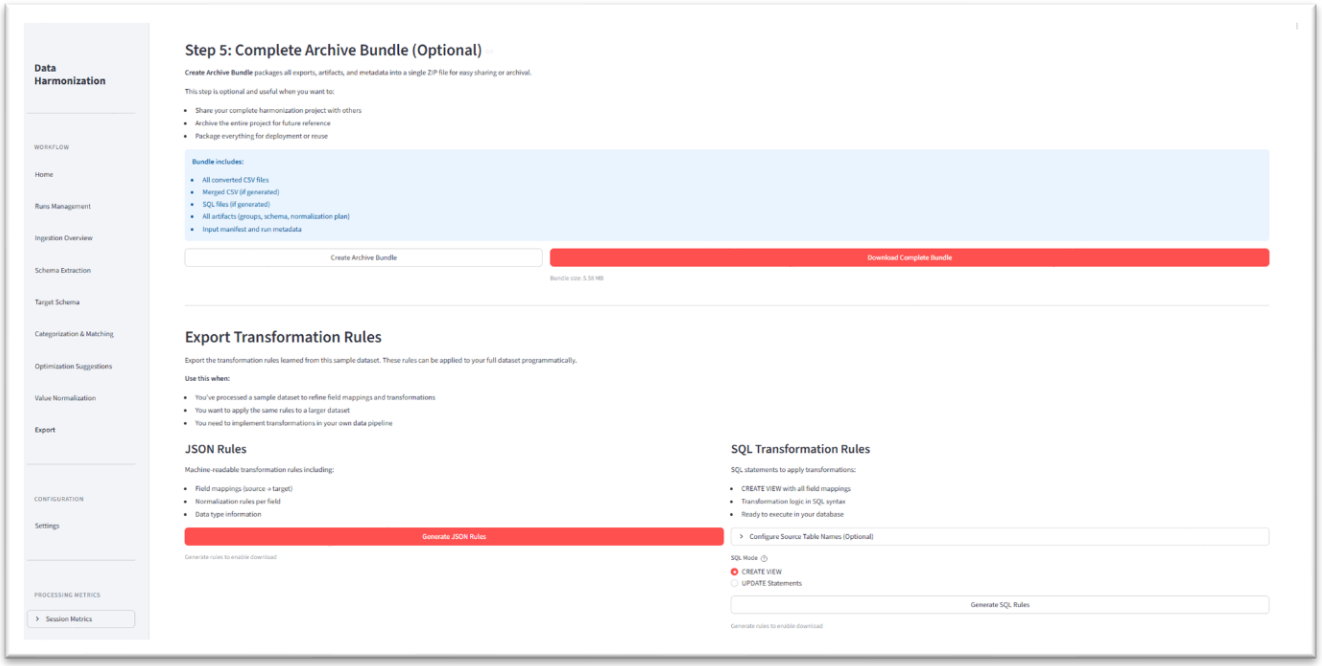
Data Harmonizer: Export (page 1/4)



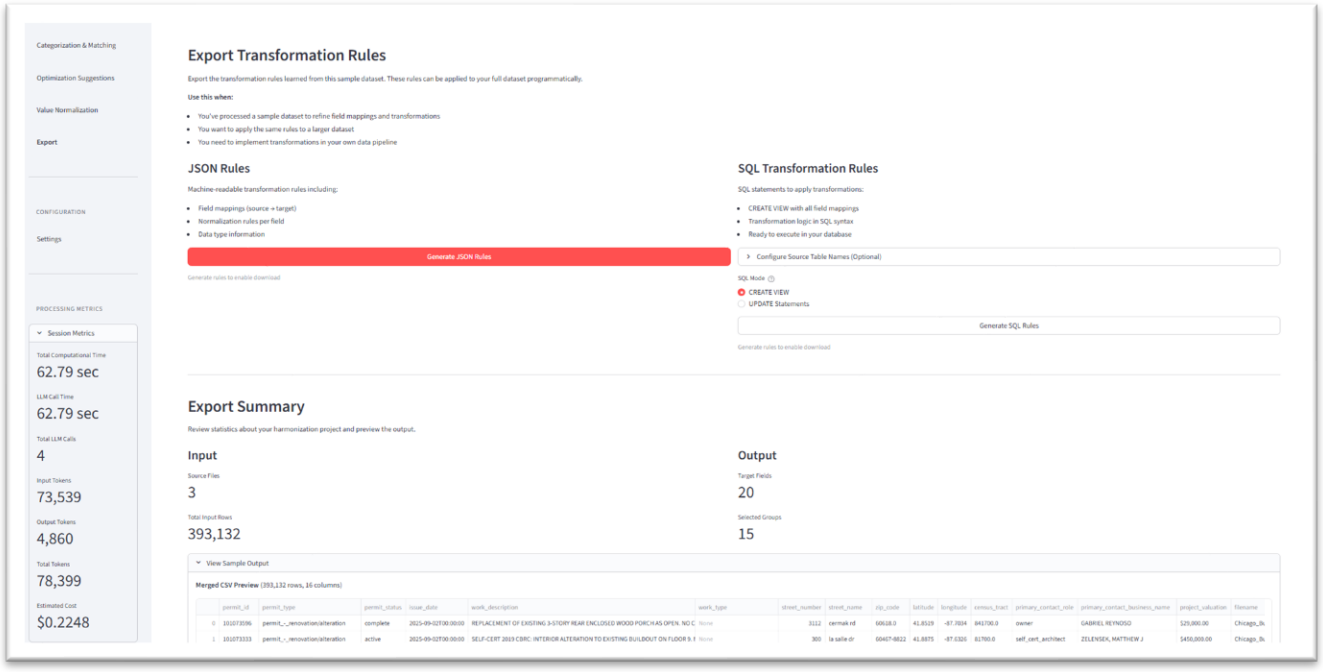
Data Harmonizer: Export (page 2/4)



Data Harmonizer: Export (page 3/4)

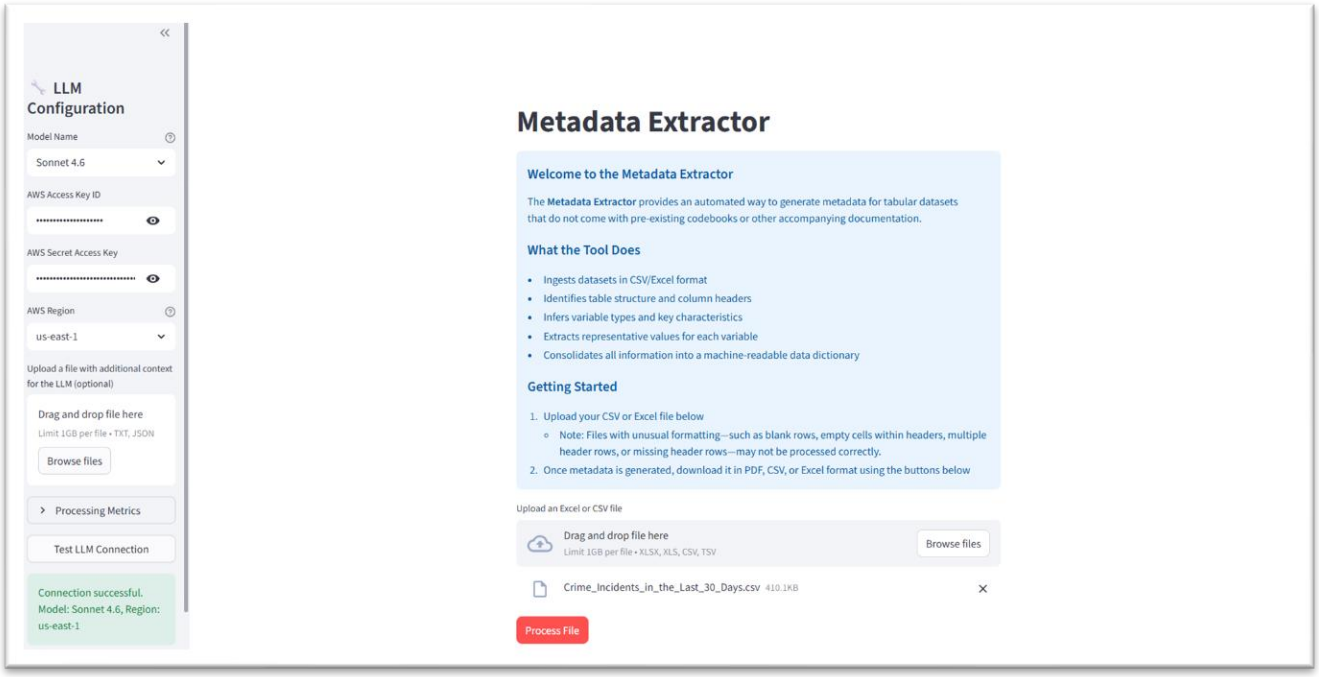


Data Harmonizer: Export & Session Metrics (page 4/4)

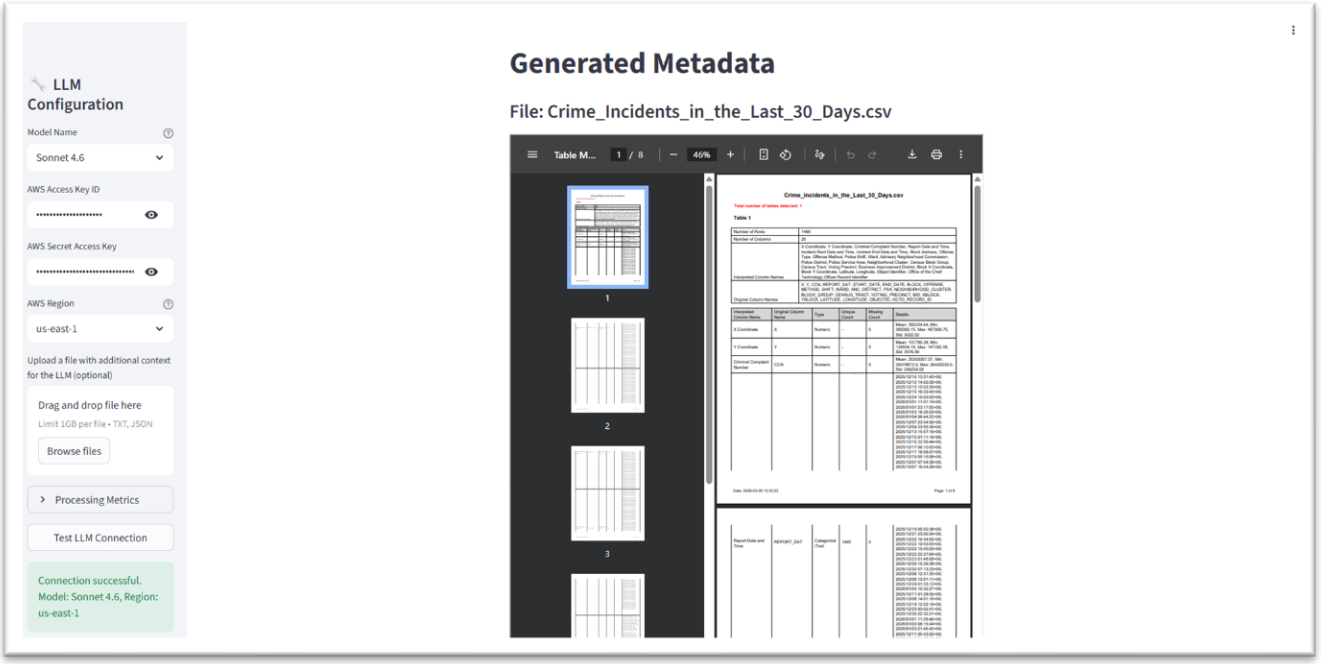


Metadata Extractor

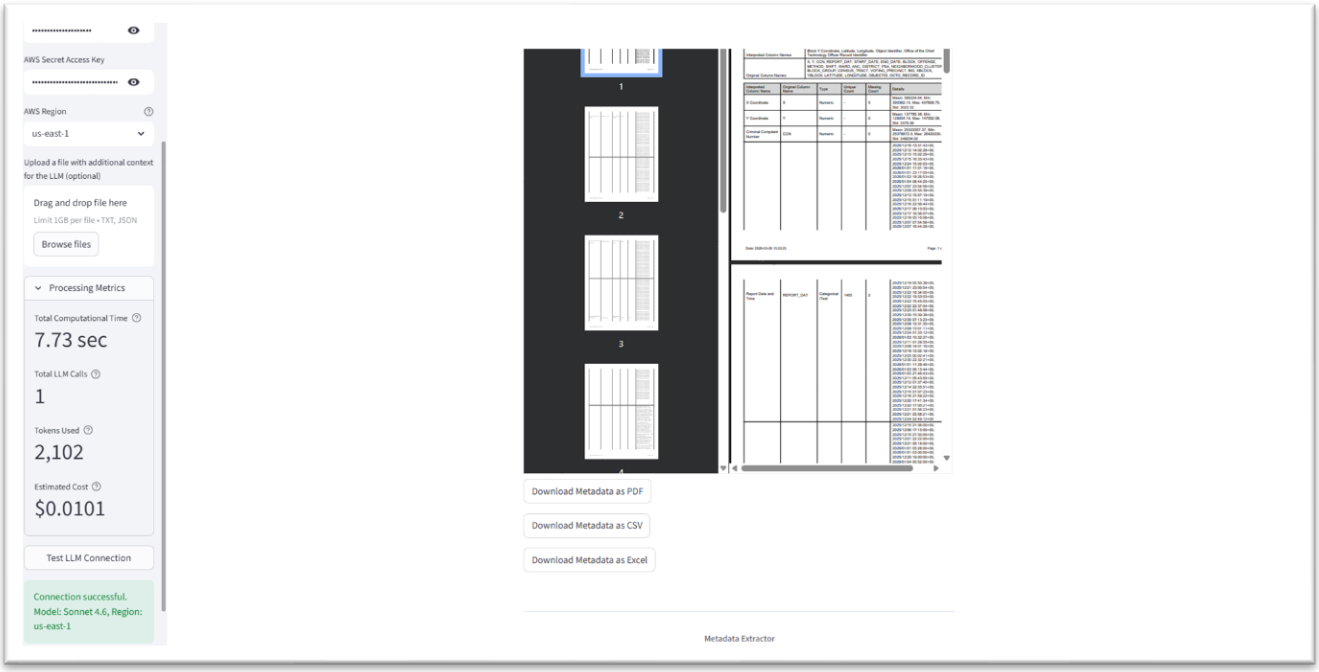
Metadata Extractor: Home Page



Metadata Extractor: Generated Metadata (page 1/2)

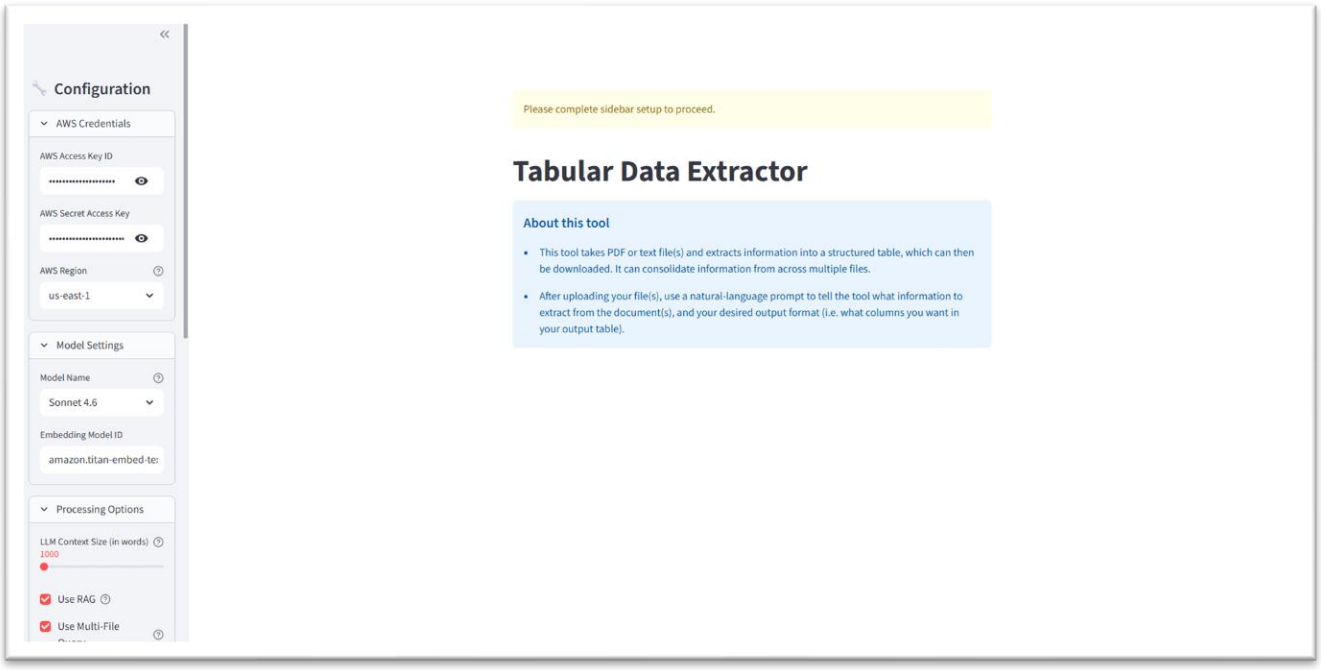


Metadata Extractor: Generated Metadata & Processing Metrics (page 2/2)

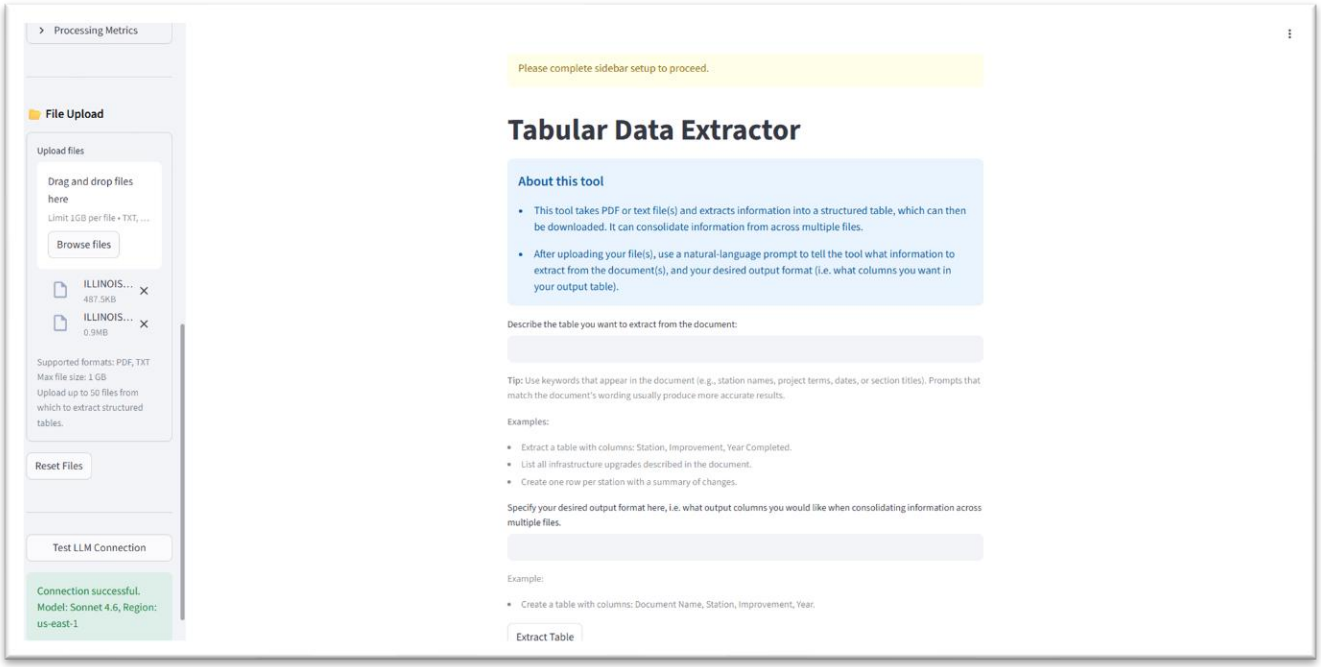


Tabular Data Extractor

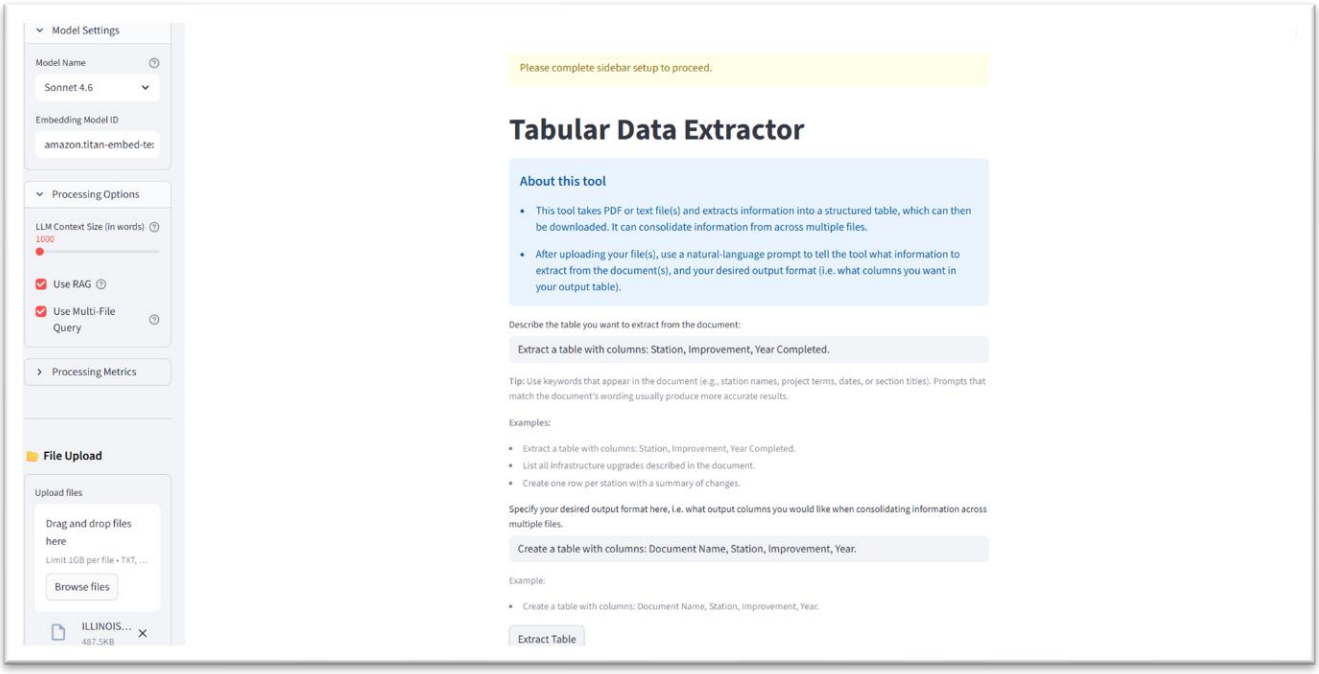
Tabular Data Extractor: Home Page



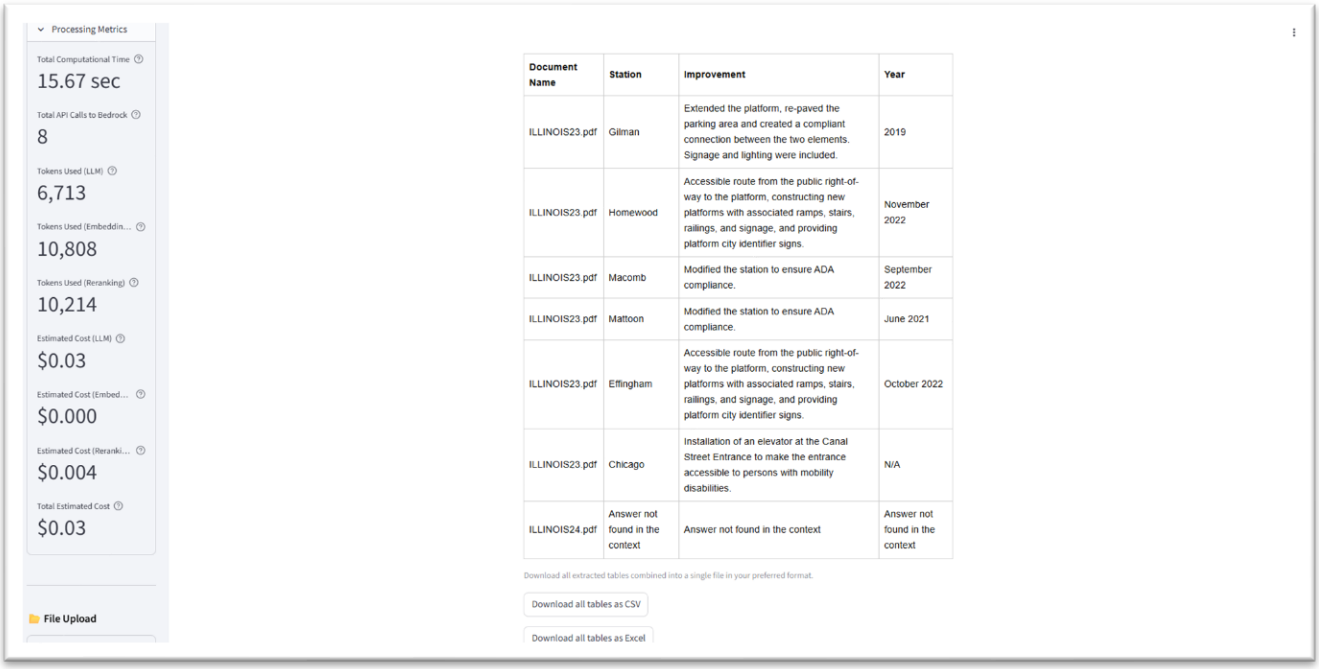
Tabular Data Extractor: File Upload



Tabular Data Extractor: Extract Table

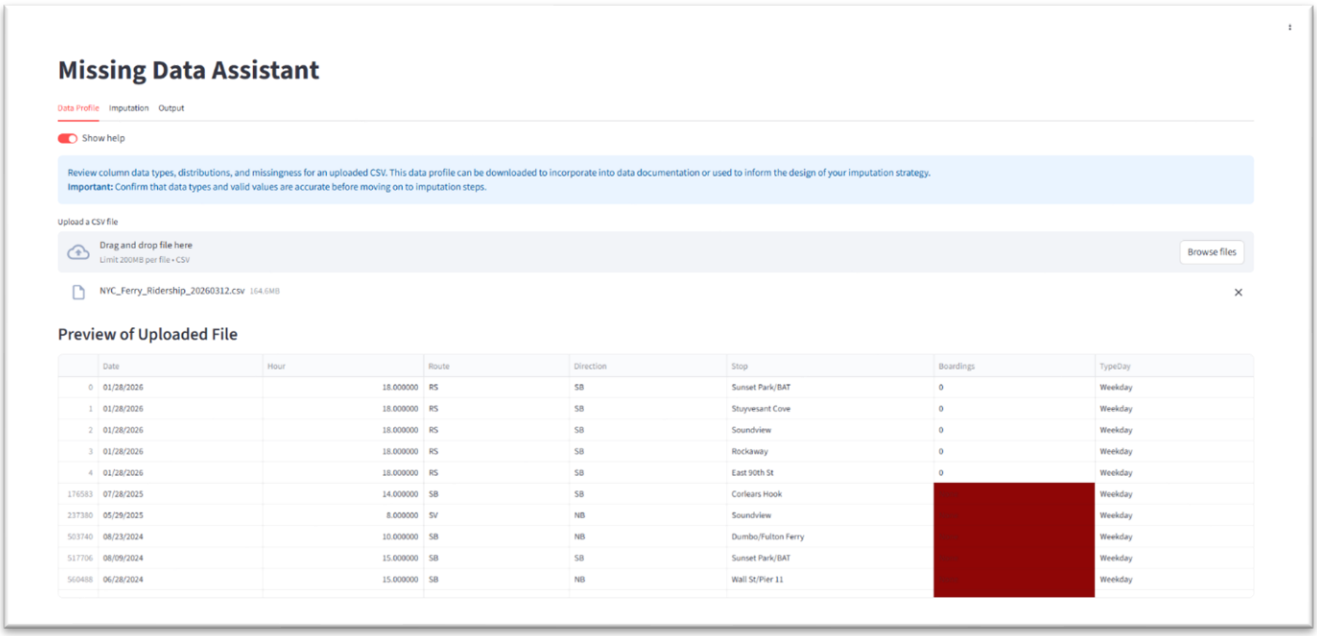


Tabular Data Extractor: Output Preview & Processing Metrics

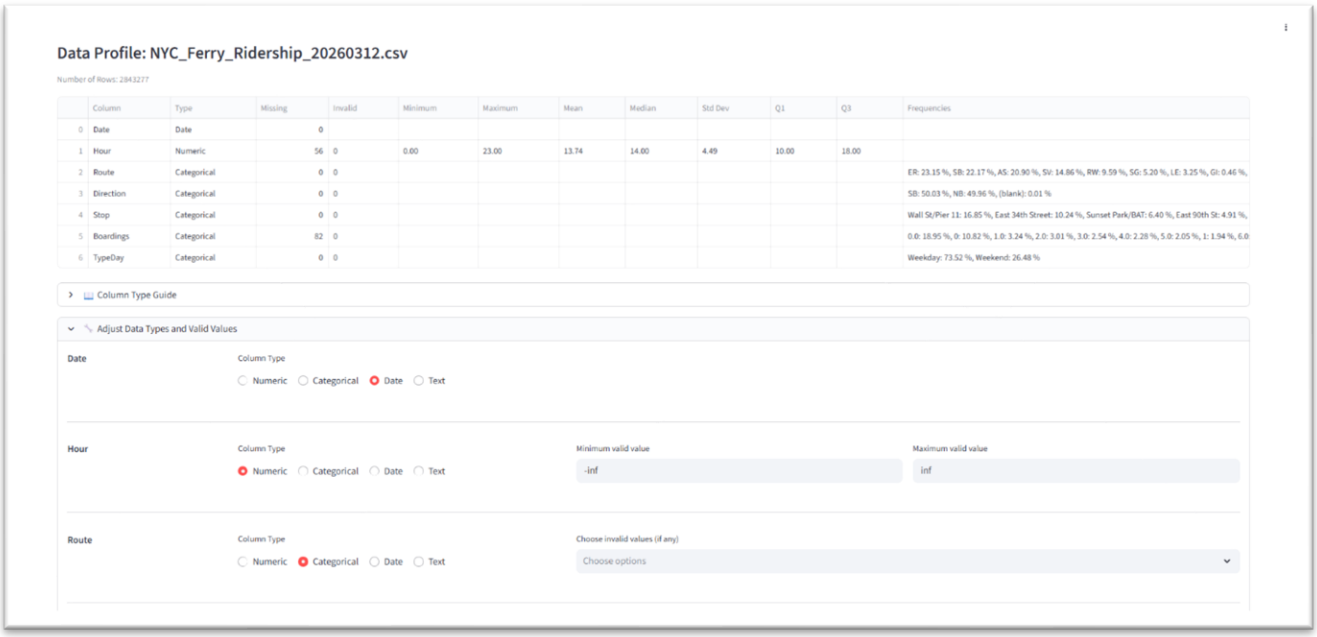


Missing Data Assistant

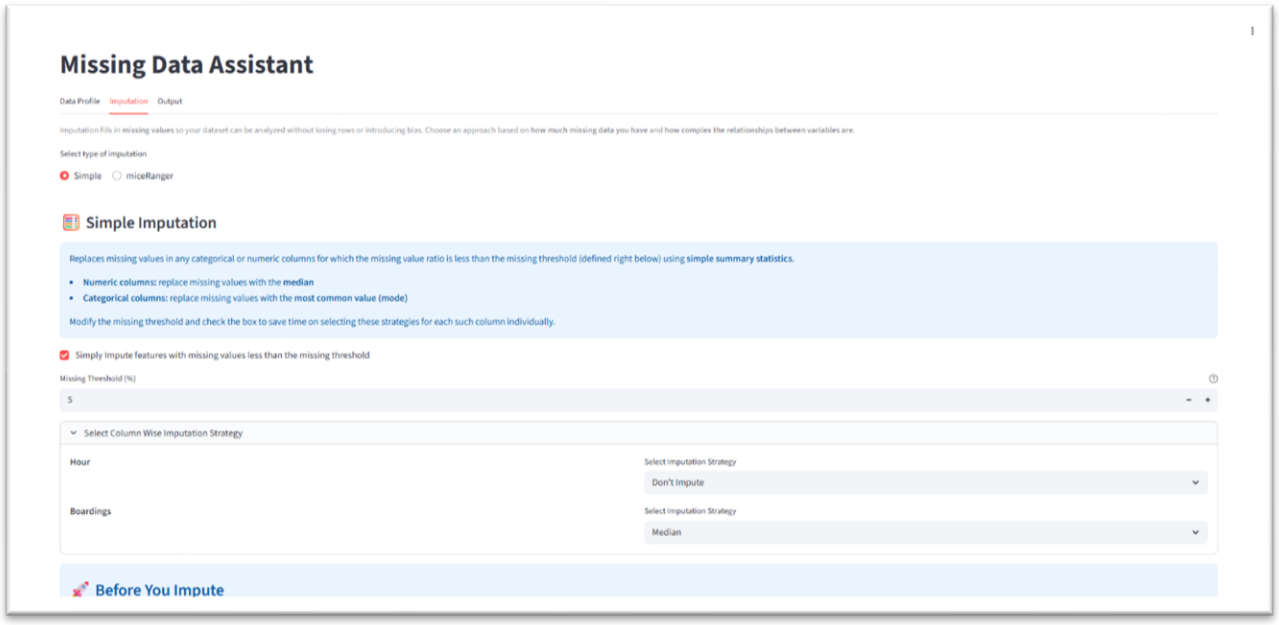
Missing Data Assistant: Home Page - Data Profile (page 1/2)



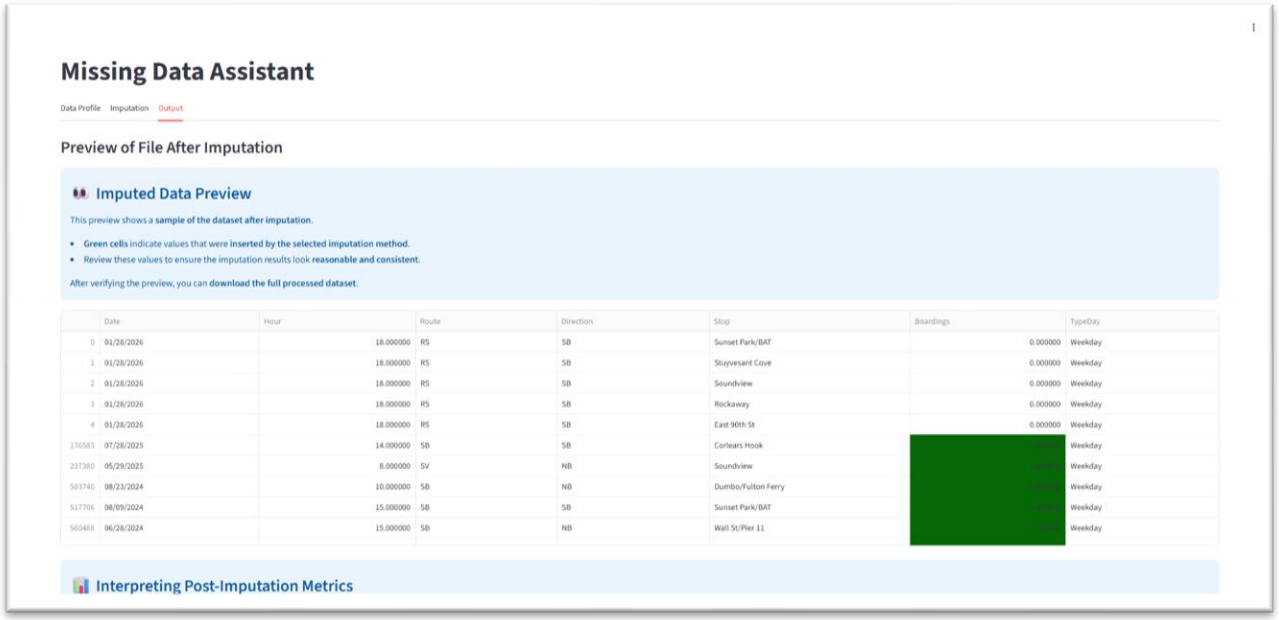
Missing Data Assistant: Home Page – Data Profile (page 2/2)



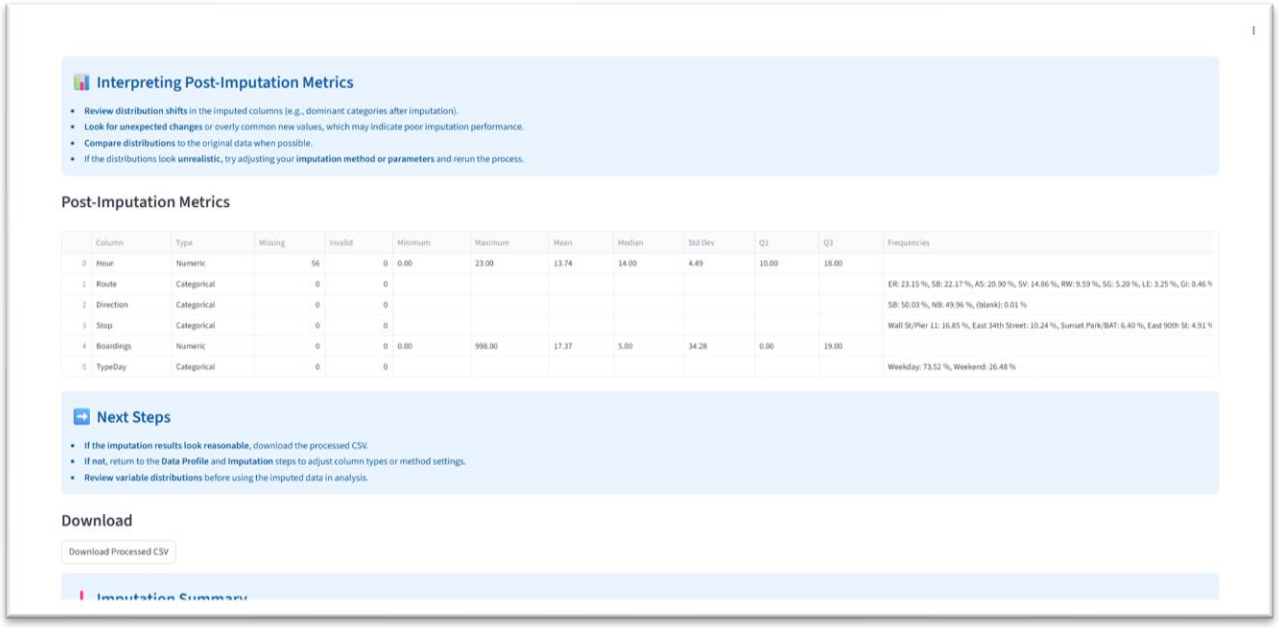
Missing Data Assistant: Imputation – Simple Imputation



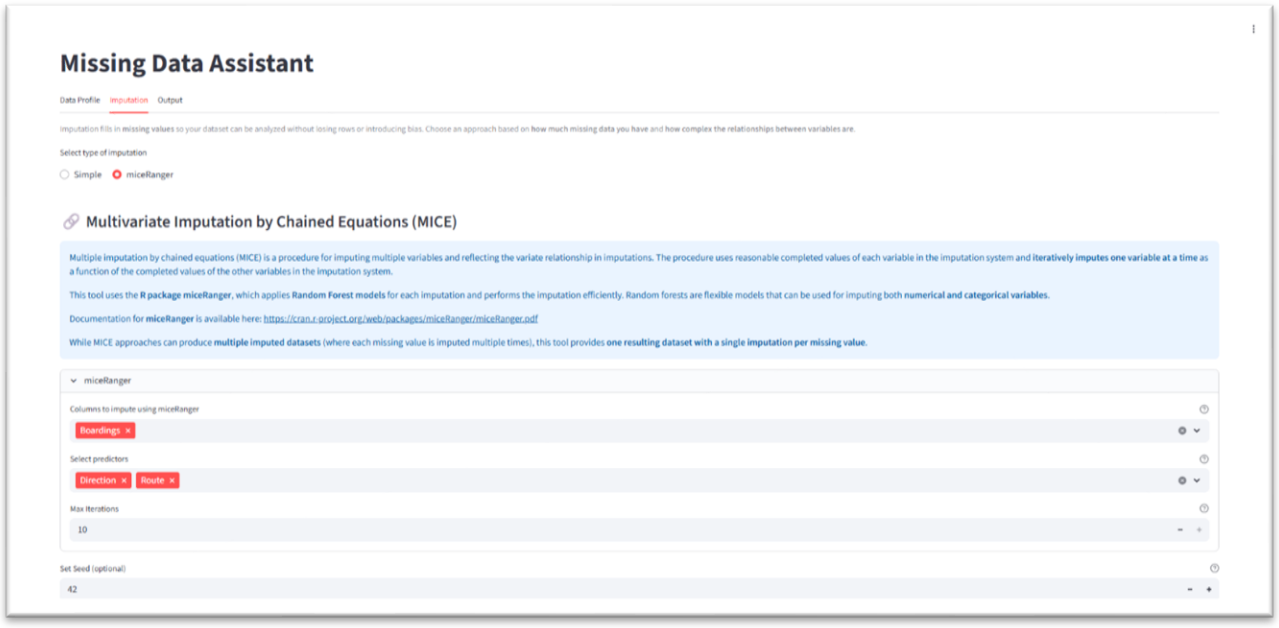
Missing Data Assistant: Output – Simple Imputation (page 1/2)



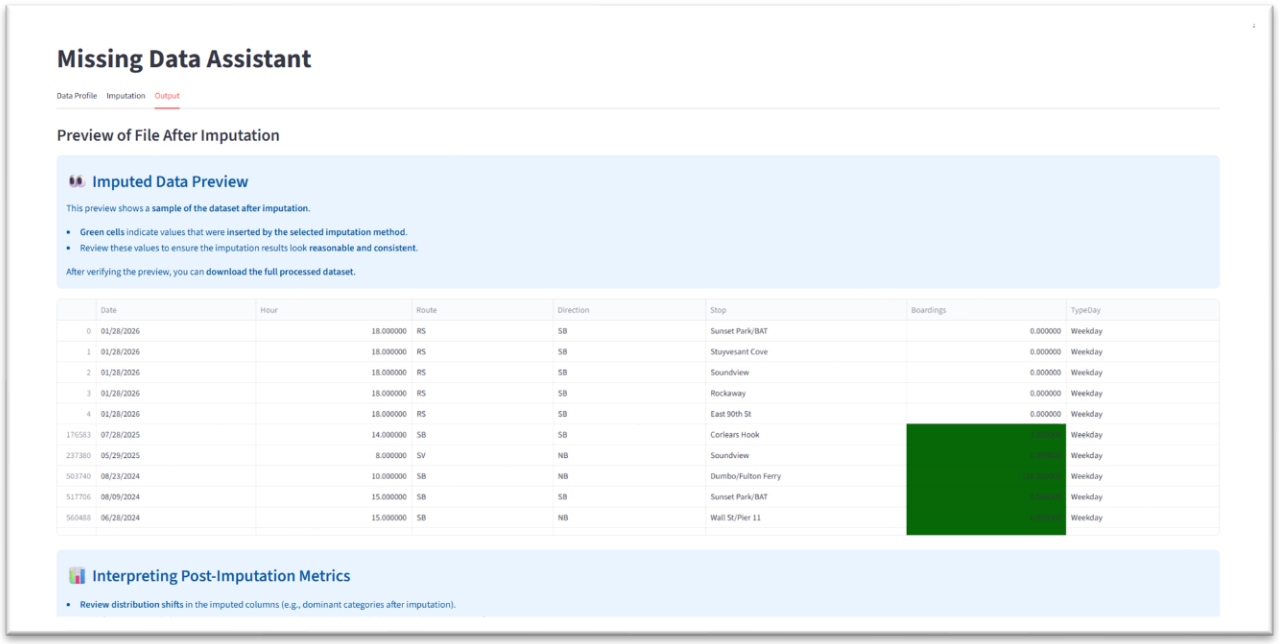
Missing Data Assistant: Output – Simple Imputation (page 2/2)



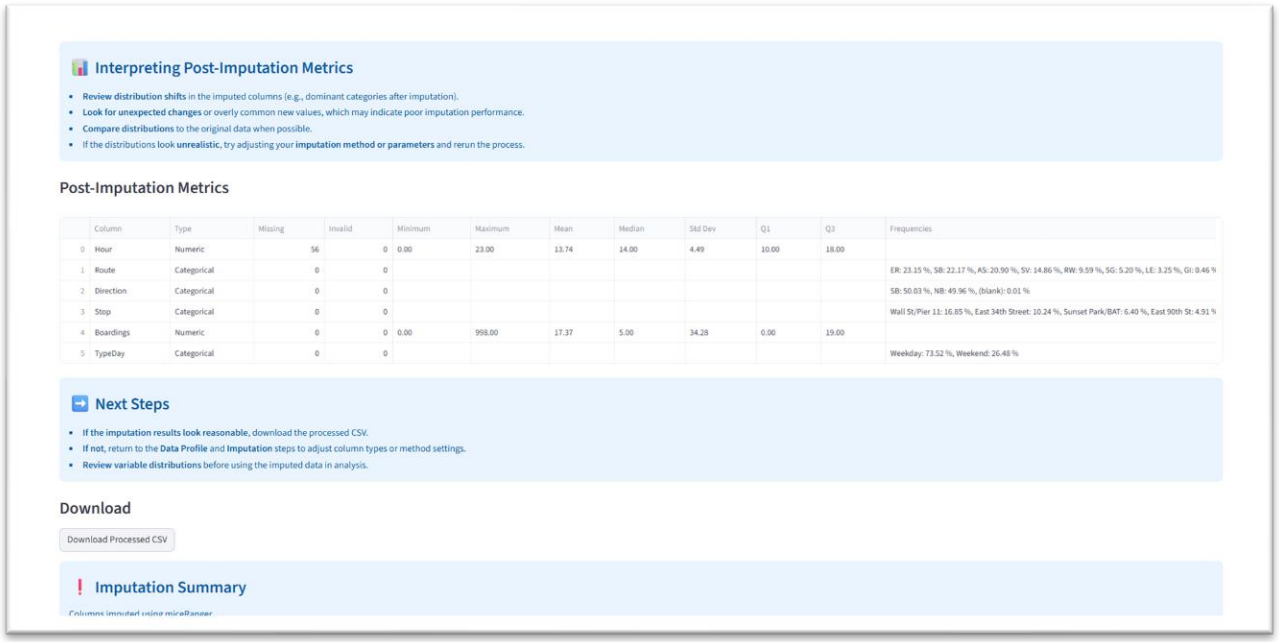
Missing Data Assistant: Imputation – miceRanger



Missing Data Assistant: Output – miceRanger (page 1/2)

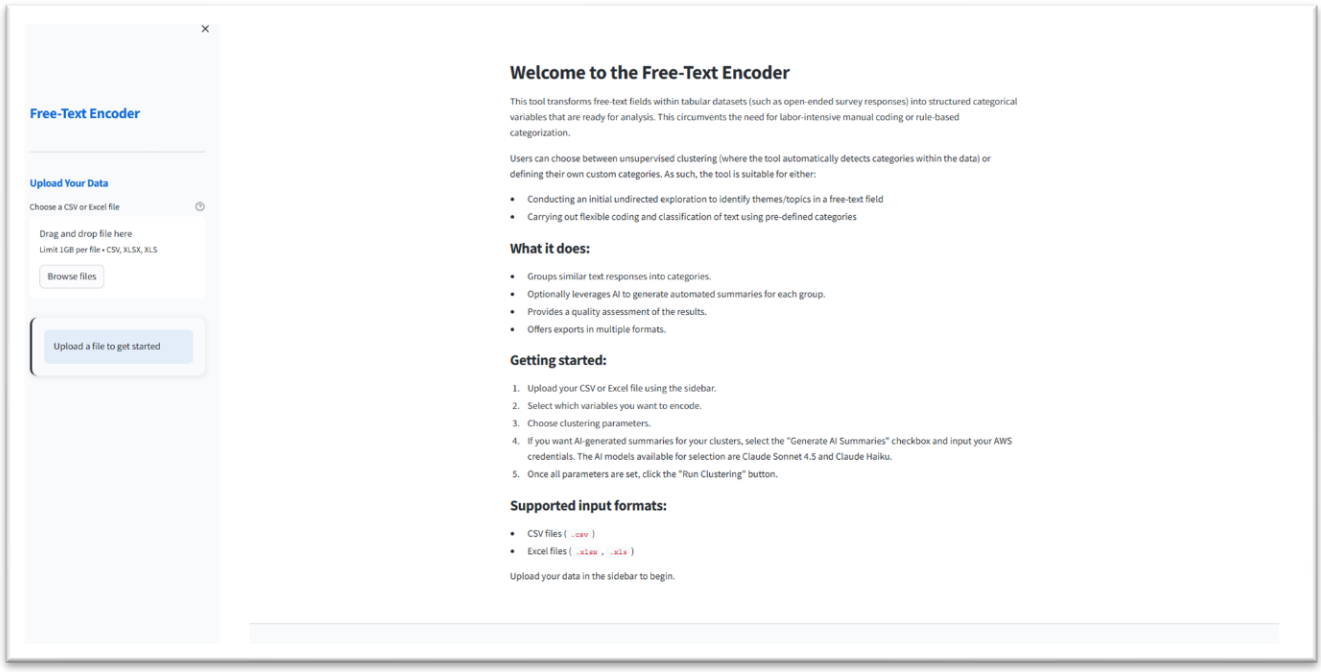


Missing Data Assistant: Output – miceRanger (page 2/2)

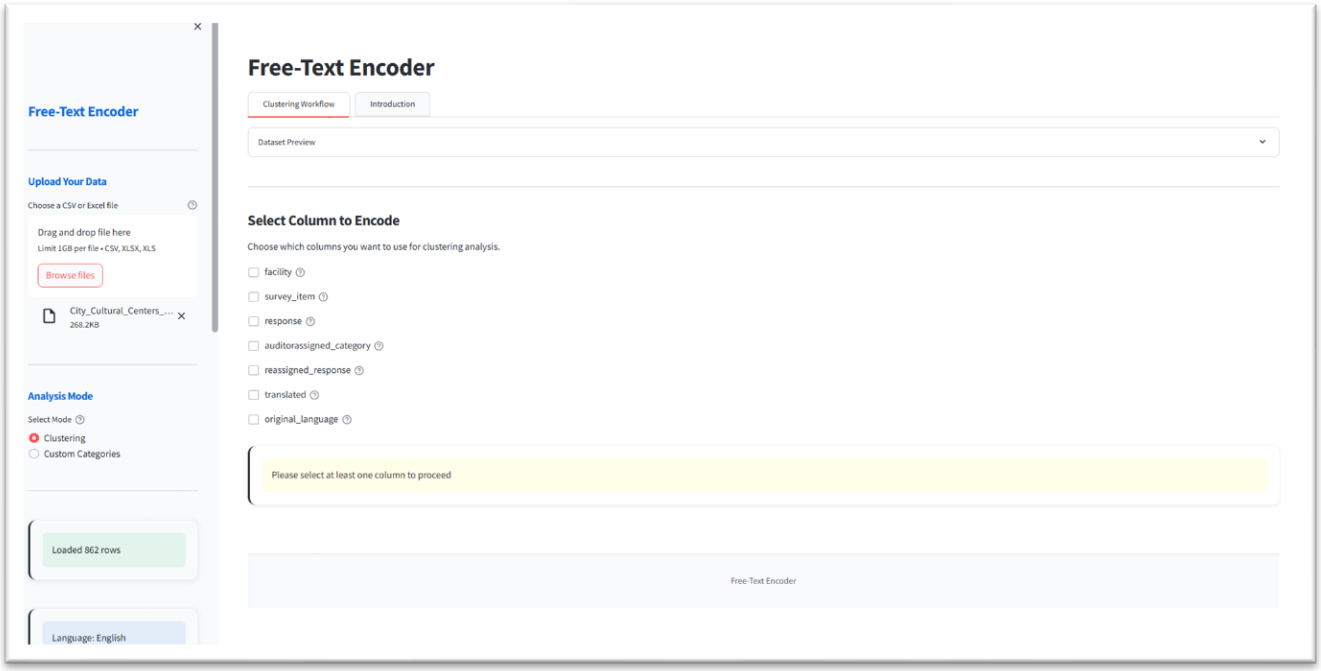


Free-Text Encoder

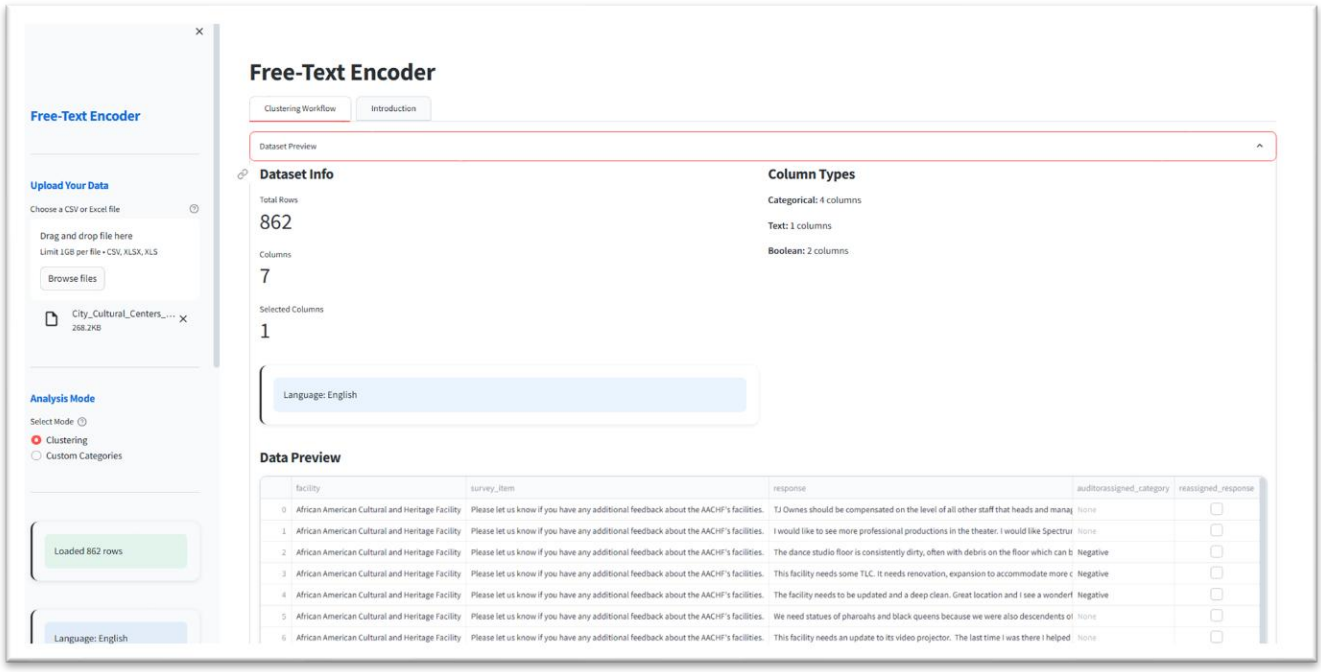
Free-Text Encoder: Home



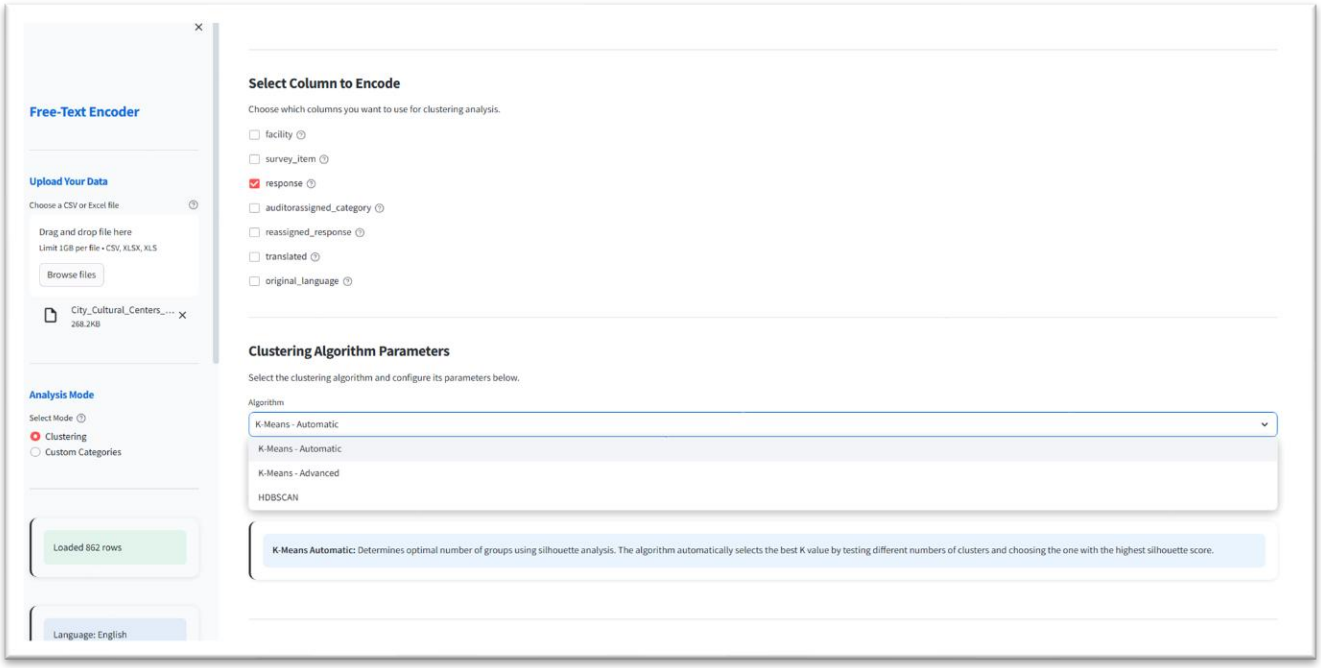
Free-Text Encoder: Home – Upload Your Data



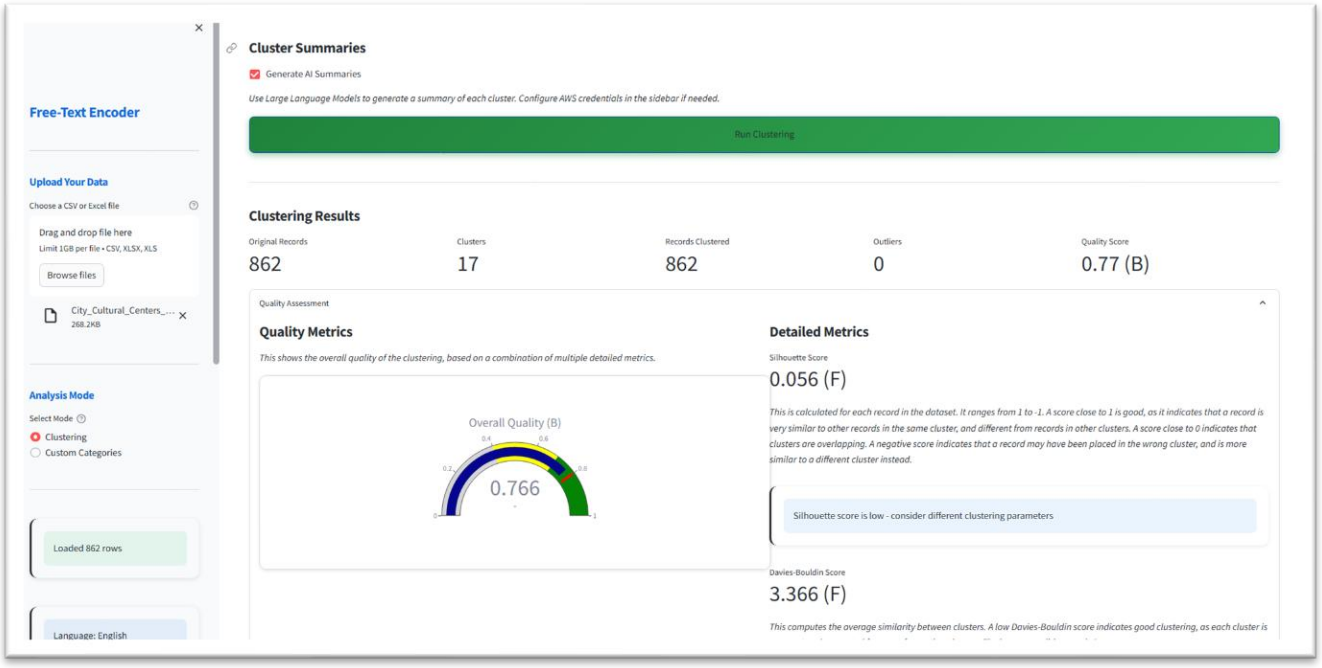
Free-Text Encoder: Clustering Workflow – Dataset Preview (page 1/6)



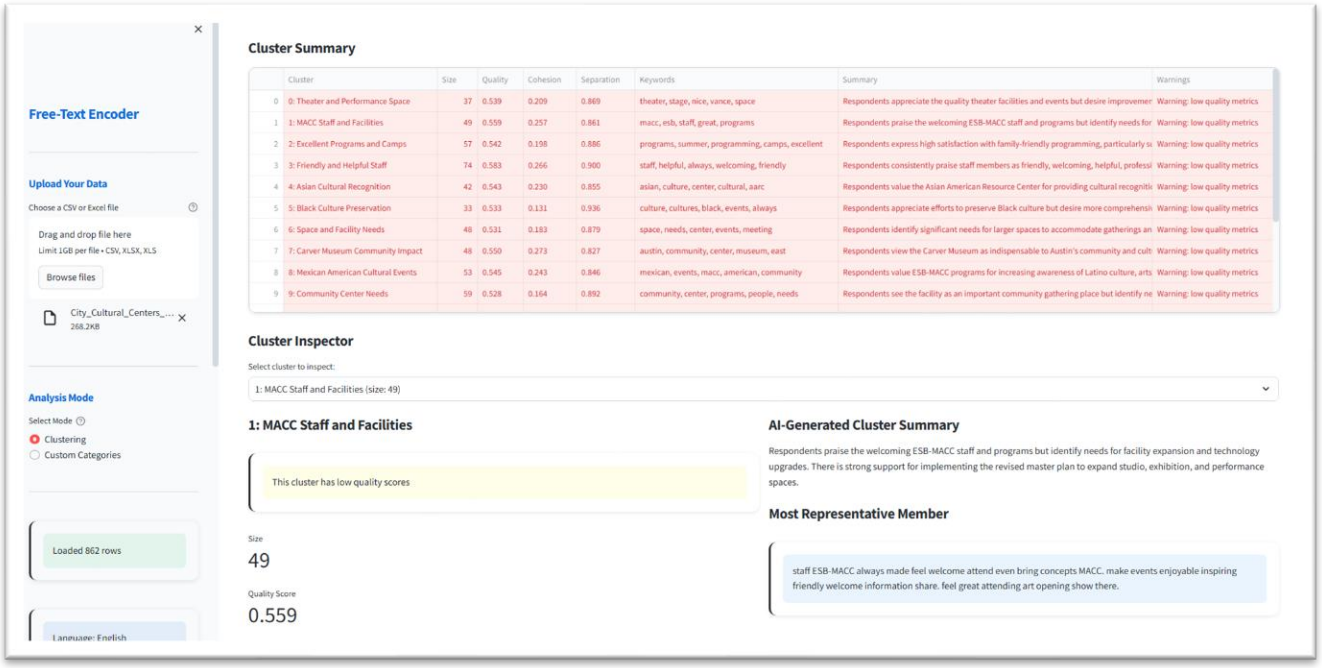
Free-Text Encoder: Clustering Workflow – Select Column to Encode (page 2/6)



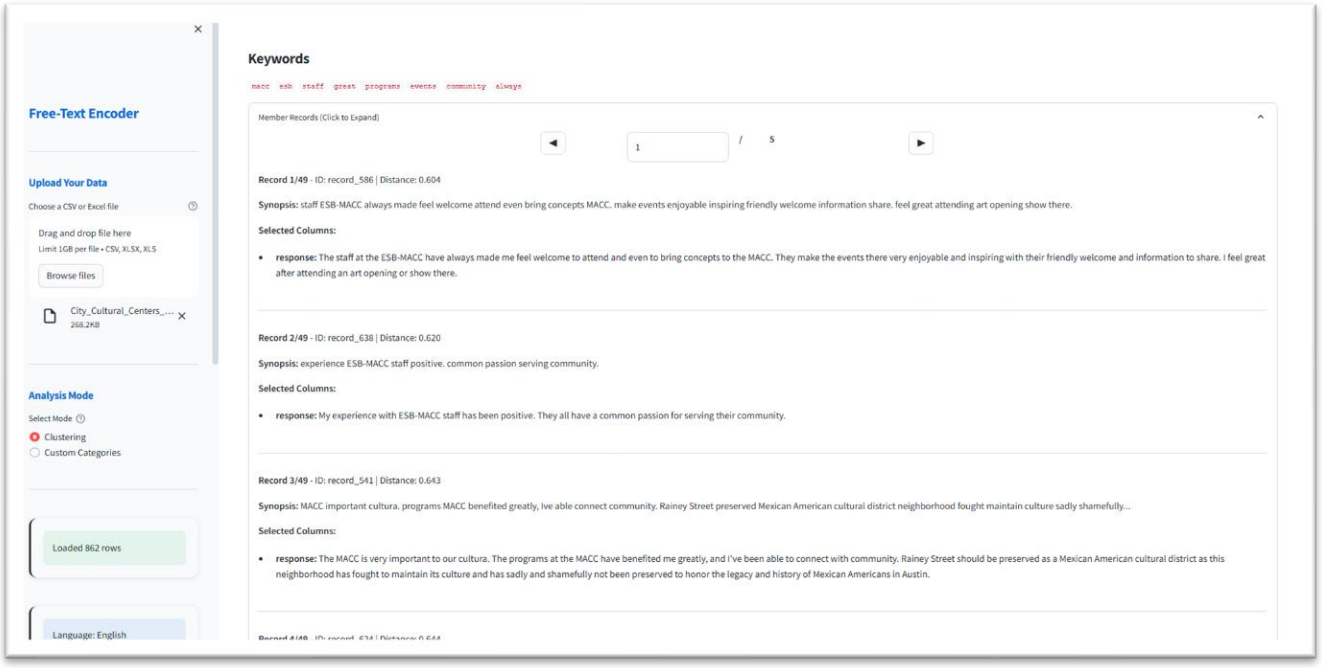
Free-Text Encoder: Clustering Workflow – Clustering Results (page 3/6)



Free-Text Encoder: Clustering Workflow – Cluster Summary (page 4/6)



Free-Text Encoder: Clustering Workflow – Keywords (5/6)



Free-Text Encoder: Clustering Workflow – Export Results & Processing Metrics (6/6)