

Article

Visual LLM-Assisted Metadata Schema Construction from Spreadsheets Through Structured Editor Operations

Wencong Chen, Xiaotong Zhang , Jie He  and Zhuoli Zhang

School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China

* Correspondence: zxt@ies.ustb.edu.cn (X.Z.); hejie@ustb.edu.cn (J.H.)

Abstract

Research data are often prepared in spreadsheets, but the meanings of their columns and values must still be translated into reusable metadata schemas. Even with a graphical editor, this translation requires repeated decisions about field names, types, properties, and hierarchy. Large language models (LLMs) can assist with these decisions, but workflows that return a complete JSON Schema make it difficult to inspect individual changes in a visual editor. We present an interactive visual LLM workflow for constructing metadata schemas from spreadsheets through structured editor operations. The workflow derives a schema draft from workbook information and represents both LLM suggestions and manual edits in the same operation model. Users can inspect, accept, or reject individual field-level operations in a graphical user interface (GUI) without reading or editing JSON Schema. The same mechanism supports updating an existing schema from a revised workbook while retaining user-approved changes. Experiments on six real-world datasets show that the operation-based workflow achieves type correctness and target-update accuracy above 97% across the evaluated models. Using the same LLM and datasets, it leaves fewer fields requiring correction than direct complete-schema generation or MetaConfigurator. A user study further shows that LLM assistance reduces the mean schema-construction time by 69.9%. The workflow gives researchers and data managers a practical way to create and maintain machine-readable metadata schemas for sharing spreadsheet data in accordance with the findable, accessible, interoperable, and reusable (FAIR) principles.

Keywords: metadata schema construction; spreadsheet data; large language models; graphical user interfaces; structured editor operations; human–LLM collaboration; schema evolution



Academic Editor: Fabio Grandi

Received: 17 July 2026

Revised: 22 August 2026

Accepted: 31 August 2026

Published: 3 September 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article

distributed under the terms and

conditions of the [Creative Commons](#)

[Attribution \(CC BY\) license](#).

1. Introduction

The FAIR principles call for scientific data to be findable, accessible, interoperable, and reusable [1]. For scientific data-sharing platforms, this requirement is often operationalized through structured and machine-readable metadata schemas. These schemas define the fields, units, constraints, and file references needed for data submission, validation, search, and reuse [2,3]. When schema creation is difficult, it becomes harder for repositories and communities to collect rich, standards-adherent metadata at scale [3,4]. Consequently, many scientific data platforms allow users or data managers to define their own metadata schemas, which often involves using JSON Schema.

JSON Schema is expressive, but it can be difficult for researchers with limited programming experience and, in some cases, even for experienced developers [5]. Users must

understand complex concepts and syntax such as primitive types, object properties, and array items. Empirical work reports that manual schema creation and maintenance are challenging and time consuming [4]. Graphical user interface (GUI)-based schema editors reduce part of this burden by turning schema-authoring tasks into form filling, diagram editing, or graphical configuration, such as Adamant [6], MetaConfigurator [7], Schema Playground [8] and pyJSON Schema Loader [9]. Figure 1 shows the MetaConfigurator interface, where many tasks that would otherwise require hand-written JSON Schema can be performed through diagram and GUI operations, supporting rapid schema construction. These tools demonstrate the value of graphical schema authoring. However, they still require understanding of the schema language [10].

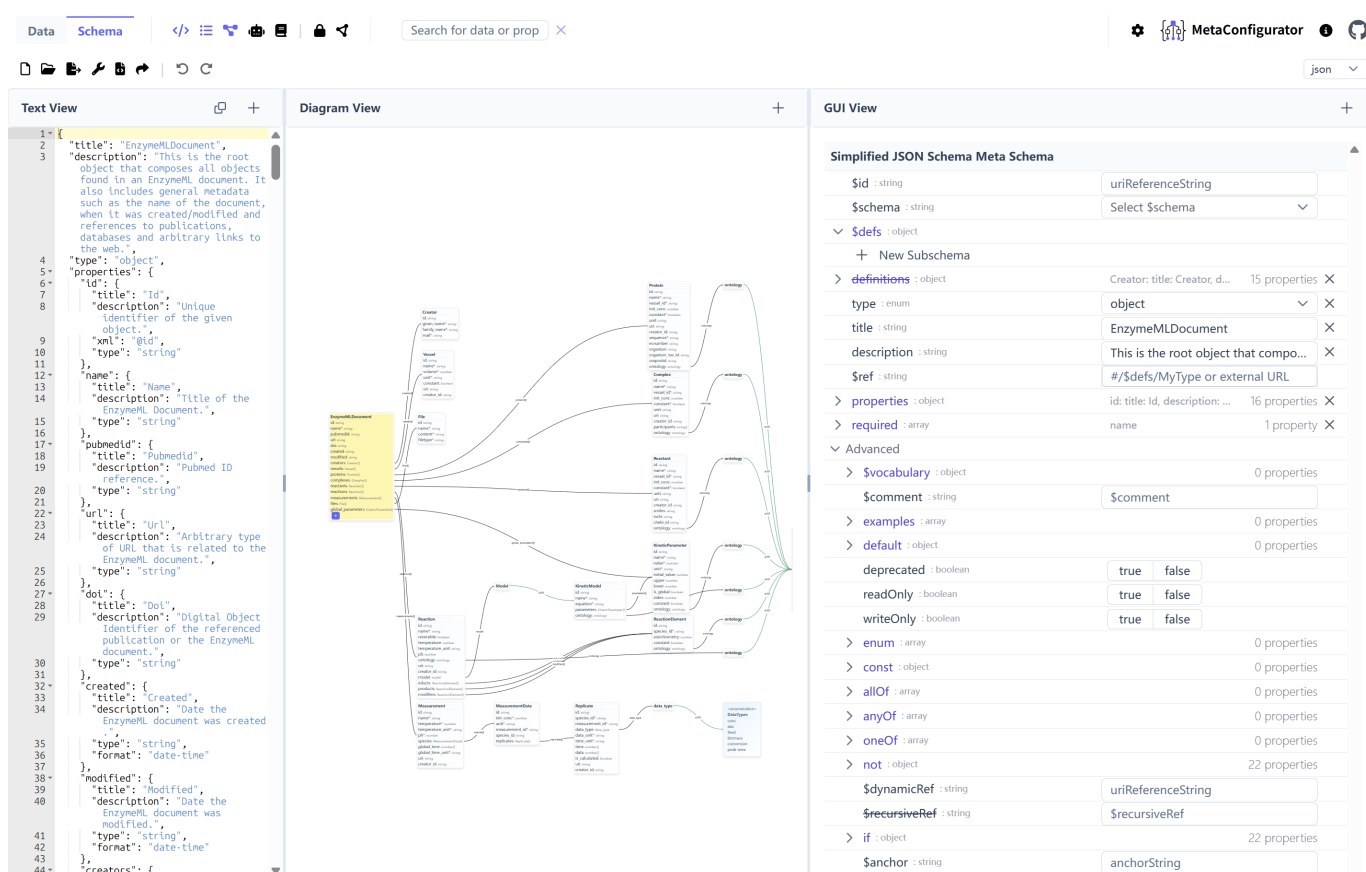


Figure 1. MetaConfigurator interface for JSON Schema authoring. The left panel shows the original JSON Schema, the middle panel presents the schema as a graph, and the right panel provides a GUI editor. Although some schema elements are presented to users and can be manipulated through the GUI, users must still understand JSON Schema keywords and their functions. The GUI primarily helps users understand the schema structure and reduces the need to write JSON Schema text manually.

Our previous work on the National Materials Data Management and Service (NMDMS) platform [11] provided a simpler GUI-based approach to schema construction. The platform introduced a graphical schema editor in which users construct metadata schemas through visual components without working with a schema language. Figure 2 shows the NMDMS schema editor, where schemas are also called templates. This usability was achieved by abstracting and simplifying JSON Schema concepts and by adding common research-data structures, such as a table structure for two-dimensional data that can be rendered as a table in the web interface, where users only need to specify the columns.

The platform has accumulated more than 20 million data records across materials, biological, and related research domains. At the same time, two practical limitations remain.

First, some concepts are still complex. For example, if a user wants a text field to accept multiple values, the editor requires the user to create an Array field and then place a String field inside it. This construction is equivalent to an array of strings in JSON Schema, and it also resembles programming-language array or list types such as `Array<String>` in Java. For many researchers with limited programming experience, however, it is more natural to understand “accepting multiple values” as an option of the text field that can be turned on rather than as a nested structure consisting of an array plus a string. Second, visual schema construction can still require substantial manual work. Another study on schema matching [12] reduced this effort by recommending existing schema fragments whose structure and context matched the user’s current editing intent. However, its benefit is limited for new deployments, small repositories, and specialized domains where relevant fragments are rare. In these cases, users may still need to start from a blank canvas or from a very incomplete schema.

Materials Genome Engineering Databases > Material Database > Edit Template

Template Outline

- Data ID*
- RNA-seq Metadata
 - RNA-seq Library Layout
 - RNA-seq Library Selection
 - RNA-seq Library Source
 - RNA-seq Library Strategy
 - RNA-seq Platform
 - RNA-seq System
 - Metadata File
- Raw RNA-seq Data
 - Files
 - File Description
- Processed RNA-seq Data
 - Data Process Protocols
 - Processed Files
- Biomaterial Metadata
 - Biomaterial Name
 - Biomaterial Protocols File
- Material Properties
 - General Properties
 - Chemical Properties
 - Physical Properties
 - Mechanical Properties
- Biological Properties
 - Age
 - Cell Strain

STRING

NUMBER

RANGE

CHOICE

IMAGE

FILE

ARRAY

TABLE

CONTAINER

GENERATOR

CONTAINER RNA-seq Metadata

STRING RNA-seq Library Layout

STRING RNA-seq Library Selection

STRING RNA-seq Library Source

STRING RNA-seq Library Strategy

STRING RNA-seq Platform

STRING RNA-seq System

Figure 2. NMDMS template editor for GUI-based schema construction. An asterisk (*) indicates a required field. Additional field settings are not shown.

The rapid progress of large language models (LLMs) creates an opportunity to reduce this remaining manual effort. However, asking users to describe an entire schema in natural language can still be burdensome, because they must specify the needed fields, types, constraints, and hierarchy. Data already accumulated by researchers provide a more concrete source of schema information, especially because these records are often the same data that will later be submitted and shared. Excel workbooks are a common carrier of such data in scientific work and are widely used in research for data entry, storage, and analysis [13]. Column headers and surrounding context suggest candidate field names and units, while column values indicate possible field types.

LLMs4SchemaDiscovery iteratively generates and refines a complete JSON Schema from domain specifications, scientific papers, and expert feedback, which is followed by ontology grounding [14]. The AI-assisted extension of MetaConfigurator supports natural-language schema creation and modification and displays validated results in a graphical schema editor [7,10]. MetaConfigurator supports spreadsheet data through comma-separated values (CSV) import and automatically infers an initial schema from the imported table. Although the two systems take different inputs, LLMs4SchemaDiscovery returns a complete schema, whereas MetaConfigurator returns either a complete schema or a replacement for a selected schema fragment.

Returning a complete schema or schema fragment creates two difficulties when users review changes through a GUI. First, users can correct, accept, or discard a generated sub-schema as a whole, but it cannot act on its field-level changes independently. In LLMs4SchemaDiscovery, expert feedback is used to generate the next complete schema rather than to approve or reject individual field changes.

Second, schema-level output complicates schema evolution. When an updated workbook requires several additions, deletions, renames, or property changes, a regenerated schema or sub-schema mixes the intended updates with content that should remain unchanged. The user must therefore inspect the regenerated content as a whole to identify the intended changes and determine whether previously accepted decisions have been preserved.

Table 1 summarizes how the three systems represent and review LLM-generated schema changes.

Table 1. Comparison of the proposed workflow with LLM-assisted schema-discovery and schema-editing systems.

Feature	LLMs4SchemaDiscovery [14]	AI-assisted MetaConfigurator [7,10]	This Paper
Spreadsheet input	No; the system uses domain specifications and scientific text rather than spreadsheet input.	Indirect; a spreadsheet table is exported to CSV, imported into MetaConfigurator, and used to infer an initial schema before LLM-assisted refinement.	Yes; workbook layout, headers, sampled values, and column statistics are provided to the LLM as context.
GUI schema editing	No visual schema editor is reported.	Yes; generated schemas are validated and displayed in a combined graphical and textual editor.	Yes; users construct, review, correct, and update schemas in the graphical editor.
LLM result representation	A complete JSON Schema that is iteratively refined.	A generated schema or a replacement for a user-selected sub-schema.	The editor validates LLM-proposed field edits and compiles them into an ordered sequence of field-level operations.
Human review granularity	Expert feedback between schema-refinement iterations.	Users correct, accept, or discard the generated schema or selected replacement as a whole.	Each editor operation, such as Create, Rename, Move, Configure, or Delete, can be reviewed individually.
Selective acceptance or rejection	Not at the level of individual schema edits.	Not at the level of individual editor operations.	Yes; users can accept useful operations and reject unwanted ones.
Existing-schema updates	Iterative refinement from additional papers and feedback.	Natural-language modification of a user-selected schema region.	The workflow applies workbook-driven additions, deletions, renames, moves, and property changes to the existing schema as separate operations.
JSON Schema knowledge during review	Required for direct inspection of the generated JSON Schema.	The GUI reduces manual JSON Schema authoring, but users must still understand JSON Schema syntax to inspect and correct schemas.	None; users review changes through field controls and operation cards without viewing or editing JSON Schema.

We implement the workflow in a GUI editor, where users review schema changes without viewing or editing JSON Schema syntax. The LLM proposes field-level edits rather than a replacement schema, and the editor validates and compiles these proposals into atomic operations. An edit operation is a concrete GUI action, such as adding a field, renaming a field, changing a type, setting a unit, moving a field, or deleting a field. The

resulting operations are shown to users in a visual list. Users can apply or reject each operation independently. User edits are recorded in the same operation model, so manual and LLM-assisted changes can be interleaved and an existing schema can be updated from a newer Excel workbook. We also define several field properties to simplify schema modeling. For example, the Multiplicity property represents repeated values directly on a field, removing the need for a separate Array type. This makes schema composition more intuitive for users and gives the LLM fewer structural alternatives to consider when proposing edits. Figure 3 shows the operation review panel, where schema edits are presented as individual GUI operations.

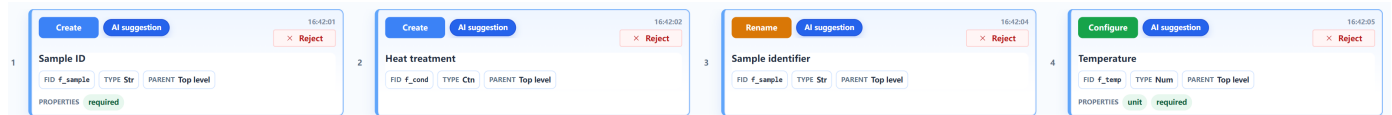


Figure 3. Operation review panel for AI-assisted schema editing. Each edit is shown as an operation with its target field, allowing users to inspect and selectively apply/reject the proposed changes. The numbers 1–4 indicate the order in which the operations were generated.

The main contributions of this paper are summarized as follows.

- We define a shared field-level operation model for LLM suggestions and manual edits together with editor-side validation and compilation.
- We develop an Excel-to-schema workflow in which deterministic code extracts workbook content and validates operations, while the LLM proposes field names, types, properties, and groupings; each generated field remains linked to its source column.
- We represent workbook-driven schema updates as explicit operations, allowing users to reject unwanted changes without discarding previously accepted parts of the schema.
- We evaluate schema construction and updates across public datasets and multiple LLMs, compare the operation workflow quantitatively with direct complete-schema generation and the MetaConfigurator workflow, and assess editing effort through an operation-cost analysis and a user study.

2. Methods

2.1. Schema Building

To further abstract the schema-building process and guide system design toward better usability, we have proposed SContainer in our previous work [15]. It is an abstract data model for GUI-based schema construction and data sharing. It mathematically unifies single-valued and multi-valued fields and separates data structures from constraints: only string, number, and container remain as core field types, while other data types are expressed as constraints or predefined structures built from the core types. Based on this model, we define the schema representation and editing operations used by both manual GUI actions and AI-assisted construction as follows.

2.2. Field Types and Properties

Each field has a name, a position, a field type, and a set of properties. Our schema model has three core types:

- String fields (Str), for names, labels, notes, identifiers, and other literal text;
- Numeric fields (Num), for measurements, counts, scores, numeric codes, and other values represented as numbers;

- Container fields (Ctn), for grouping related fields and expressing nesting. Conceptually, a container corresponds to a JSON object or dictionary. A complete data entry is always a container.

Field properties represent constraints. In a GUI editor, they are options that users can simply toggle on or off. They are separate from data types. This separation makes schema editing more intuitive: a constraint can be turned on or off without rebuilding the field structure. For example, repeated values are controlled by a Multiplicity property, so a single-valued field can become multi-valued through one property edit and vice versa. The same separation also allows more flexible combinations. A Choice constraint, for instance, can be applied not only to string fields but also to numeric fields when the valid values are a finite set of numbers. Table 2 summarizes the field properties used by our editor.

Table 2. Core field types and compatible field properties used by the visual editor.

Field Property	Compatible Types	Meaning in the Editor
Required	Str, Num, Ctn	The field must appear in the data.
Multiplicity	Str, Num, Ctn	The field accepts one value or multiple values. When Multiplicity is enabled, the field corresponds to an array in JSON. A repeated container corresponds to an array of JSON objects.
Choice	Str, Num	Values must be selected from a predefined set, corresponding to <code>enum</code> in JSON Schema.
Unit annotation	Num	A numeric value may carry a measurement unit, such as C, h, or wt. %.
File	Str	The field value is a file reference, such as a file name, path, or stored file id.
Image	Str	The field value is an image reference. It differs from a general file property because the system can display image previews.
Mutual exclusion (mutex)	Ctn	The direct child fields of the container are mutually exclusive. In one data entry, the user fills only the child field that applies and leaves the other alternatives empty. This is useful when the same recorded item can appear in different forms. For example, a measurement container can include an action string field and a value numeric field, and each data entry uses only one of them. It can also represent process-specific records, where different processing steps require different fields.

Other field types introduced in the original NMDMS [11] can be defined as combinations of core field types and properties, as shown in Table 3. As a result of the streamlined data model, Array and Choice no longer need to be separate field types; they become field properties that can be toggled on or off. Figure 4 demonstrates the new GUI editor based on this data model.

Table 3. Data types defined as combinations of core field types and properties.

Field Type	Core Type	Field Property or Predefined Structure
String	Str	No special property by default.
Number	Num	No special property by default; unit annotations may be added when needed.
File	Str	File property.
Image	Str	Image property.
Container	Ctn	No special property by default.
Range	Ctn	Predefined container with two fixed numeric child fields, used for interval values and values with uncertainty, such as lower–upper bounds or $a \pm b$.
Table	Ctn	Repeated container to represent two-dimensional tables. Its child fields are displayed as columns and they are prohibited from being containers or using Multiplicity.
Generator	Ctn	Container with the Mutual exclusion property; its direct child fields are mutually exclusive alternatives.

The same design leaves room for later extension. For example, a date–time field type could be represented as a numeric field with a timestamp property. Such extensions should be added only when they reduce real user effort in real schemas; otherwise, too many field types and properties would reintroduce the cognitive burden that the abstraction is meant to avoid.

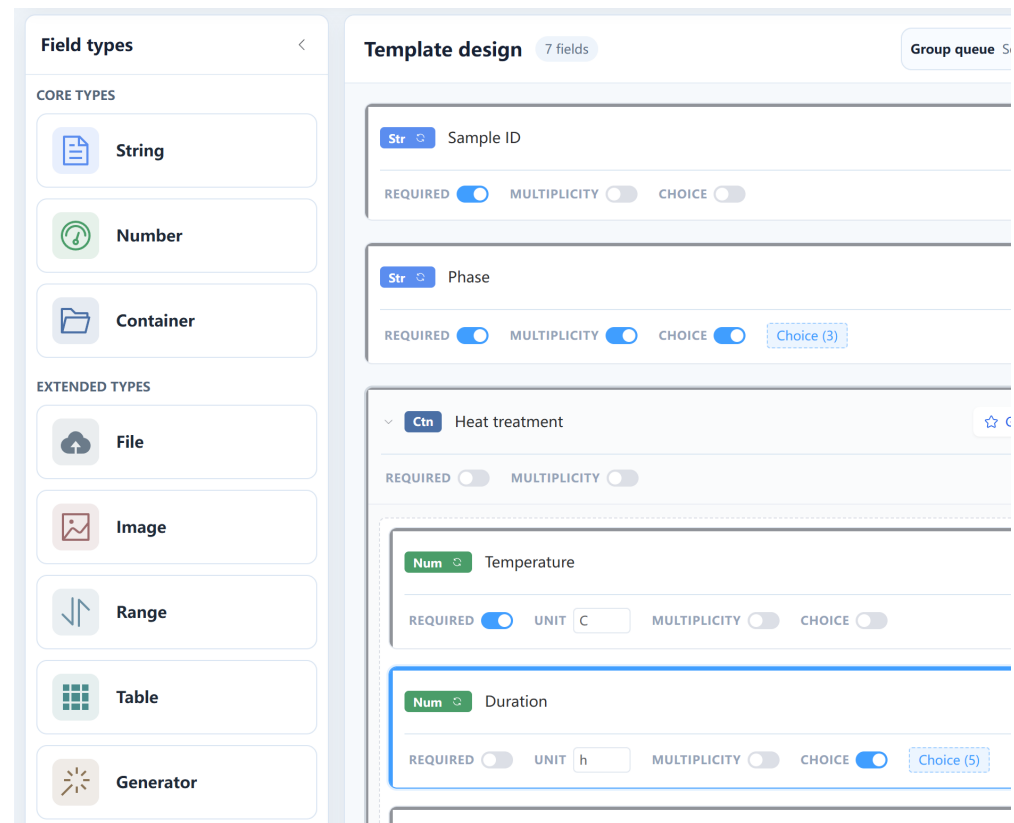


Figure 4. New GUI editor (prototype) with repeated values and choices represented as field options.

2.3. Schema Editing Operations

Schema construction can be viewed as executing an ordered sequence of operations. For example, suppose the sequence first creates a *Sample ID* field and then creates a *Temperature* field at the same level. Executing both operations produces a schema with both fields. If the first operation is rejected, the editor rebuilds the draft from the same history but skips that operation, so *Sample ID* is absent while *Temperature* remains. Later operations that depend on the rejected *Sample ID* field, such as renaming it or changing its requiredness, are skipped as well. In this sense, rejecting selected operations works as a fine-grained undo over fields or field properties rather than a rollback of the whole schema. This is useful when users want to undo changes to selected fields while preserving the rest of the schema.

The operation set is shared by manual editing and AI-assisted editing. Each schema edit made through the GUI is recorded as an operation, as shown in Figure 5.

Operations identify fields by field IDs rather than by field names, because names can be changed by later operations and different fields may have the same name. Such an ID is written as *fid*. The root of the schema is written as *root*, and a parent location is written as *p*, where $p \in \{\text{root}\} \cup F_{\text{ctn}}$ and F_{ctn} is the set of container-field IDs. The Create operation assigns a newly created field an ID denoted by fid_{new} , as shown in Equation (1):

$$\text{fid}_{\text{new}} := \text{Create}(p, j, k, c, \theta). \quad (1)$$

This operation executes the following steps: create a new field under parent location p , insert it at sibling position j , display it with name k , assign it core type $c \in \{\text{Str}, \text{Num}, \text{Ctn}\}$, and attach the compatible property set θ . The property set θ contains the enabled properties listed in Table 2. Omitted properties take default values, and θ_0 denotes the default property set.

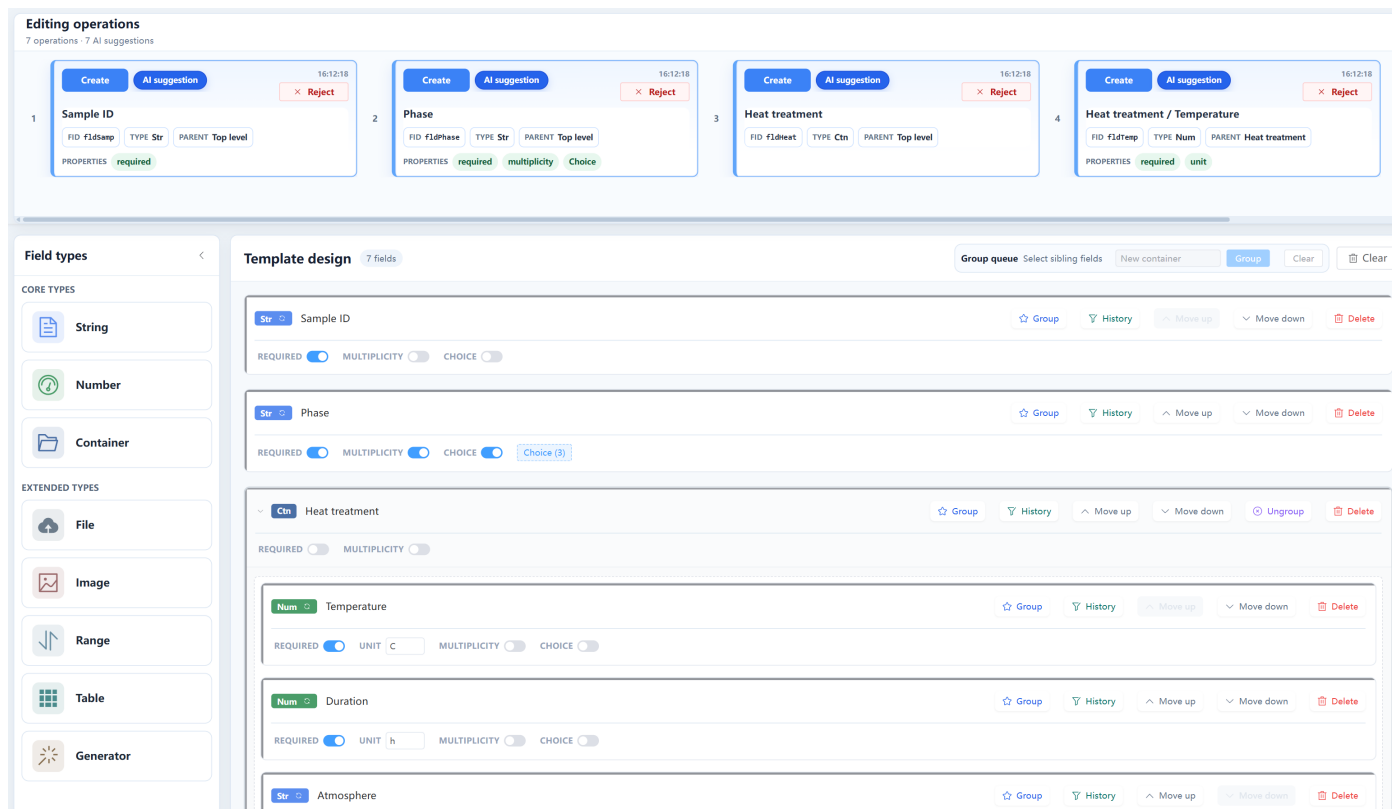


Figure 5. Operation records in the GUI editor. Every edit is recorded as an operation.

For example, the two operations are written in Equation (2):

$$\begin{aligned} fid_p &:= \text{Create}(\text{root}, 1, \text{Processing}, \text{Ctn}, \theta_0), \\ fid_t &:= \text{Create}(fid_p, 1, \text{Temperature}, \text{Num}, \{\text{unit} : \text{C}\}), \end{aligned} \quad (2)$$

The first operation creates a container named Processing as the first top-level field and assigns it fid_p . The second operation then creates the first child field Temperature under fid_p , assigns it type Num, attaches the unit annotation C, and assigns the new field fid_t . The same notation is used for the remaining operations: j is a sibling position, k is a display name, c is a core type, and θ is a property set. Property changes use Configure and Unconfigure. Table 4 lists the supported operations and their effects. For each operation, the system verifies its executability: referenced fields must exist, field properties must be compatible with the field type, etc. If the check fails, the operation has no effect and the system records the error.

Figure 6 demonstrates this selective review. The current schema shown in the editor is obtained by executing the operations that are currently accepted in the operation history. Rejected operations are not deleted from the history; they are simply excluded from the current draft. Accepting a rejected operation again places it back into the executed sequence, which works as a selective redo for that operation. In this example, rejecting the creation operations for Sample ID and for the Temperature field under Heat treatment is equivalent to temporarily removing these two fields from the draft. If editing is represented only as complete schema states, this kind of review is harder to support. The user can roll

back to an earlier schema state and redo later states, but it cannot directly turn off or restore one rename, move, or property change while keeping the other accepted changes.

Table 4. Editor operations and their effects.

Operation Form	Effect
$fid_{new} := Create(p, j, k, c, \theta)$	Create a new field under parent p , insert it at sibling position j , assign it the display name k , core type c , property set θ , and the new id fid_{new} .
$Name(fid, k)$	Rename field fid to display name k .
$Move(fid, p, j)$	Move field fid under parent p at sibling position j .
$Type(fid, c)$	Change the core type of field fid to $c \in \{Str, Num, Ctn\}$.
$Configure(fid, s, x)$	Set property s of field fid to value x .
$Unconfigure(fid, s)$	Clear property s of field fid , or restore its default value.
$fid_{new} := Group(\langle fid_1, \dots, fid_m \rangle, p, j, k, \theta)$	Create a new container named k under parent p , assign it property set θ and id fid_{new} , and place the selected sibling fields $fid_1, \dots, fid_m$ inside it.
$Ungroup(fid)$	Remove container field fid and promote its direct children to the parent of that container.
$Delete(fid)$	Delete field fid and its descendants.

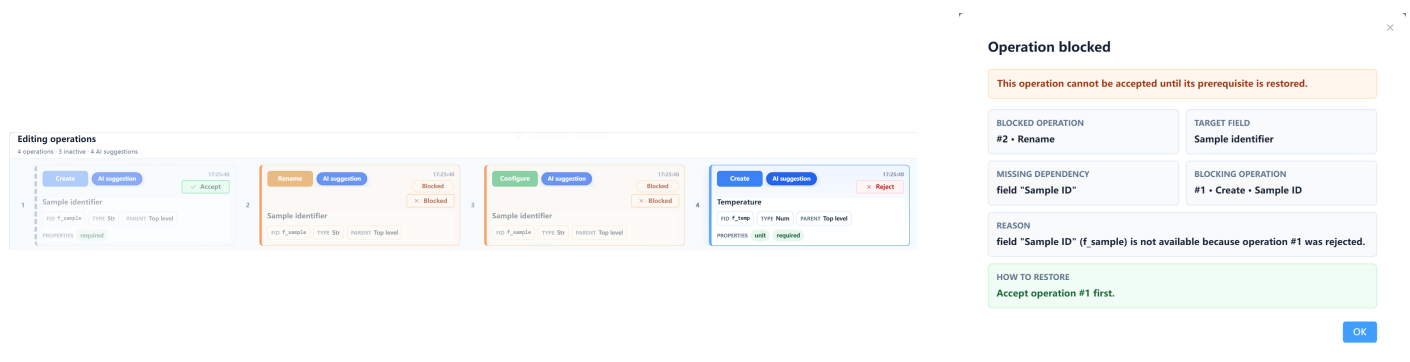


Figure 6. Selective review of schema editing operations. The numbers identify the operations and indicate the order in which they were generated. Rejected operations remain visible in the history but are not executed in the current schema, and they can be accepted again later. Left: rejecting the creation of `Sample ID` blocks its dependent `Rename` and `Configure` operations while leaving the independent `Temperature` operation active. Right: clicking a `Blocked` button opens a structured explanation of the missing dependency and how to restore it.

Operation dependencies are resolved during schema reconstruction. If rejecting a prerequisite removes a referenced field, dependent operations are not applied and are marked as `Blocked`. The dialog shown in Figure 6 identifies the missing field and rejected prerequisite and indicates which operation must be accepted to restore the dependency.

The current prototype serializes LLM planning and manual editing. A modal progress window blocks schema editing while an LLM request is processed. Once the resulting editor operations have been added to the history, users can resume editing and review those operations.

For larger schemas, the editor supports search by field name, hierarchical path, or field type. After a field is located, the history can be filtered to show the associated operations; for a container, these include operations on its descendant fields.

The editor also provides a schema outline that presents the field hierarchy as an expandable tree. The locate control beside each tree node moves the editor to the corresponding field and highlights it without changing the operation-history filter. In Figure 7,

the locate control for the DateTime container brings that container and its Date and Time children into view.

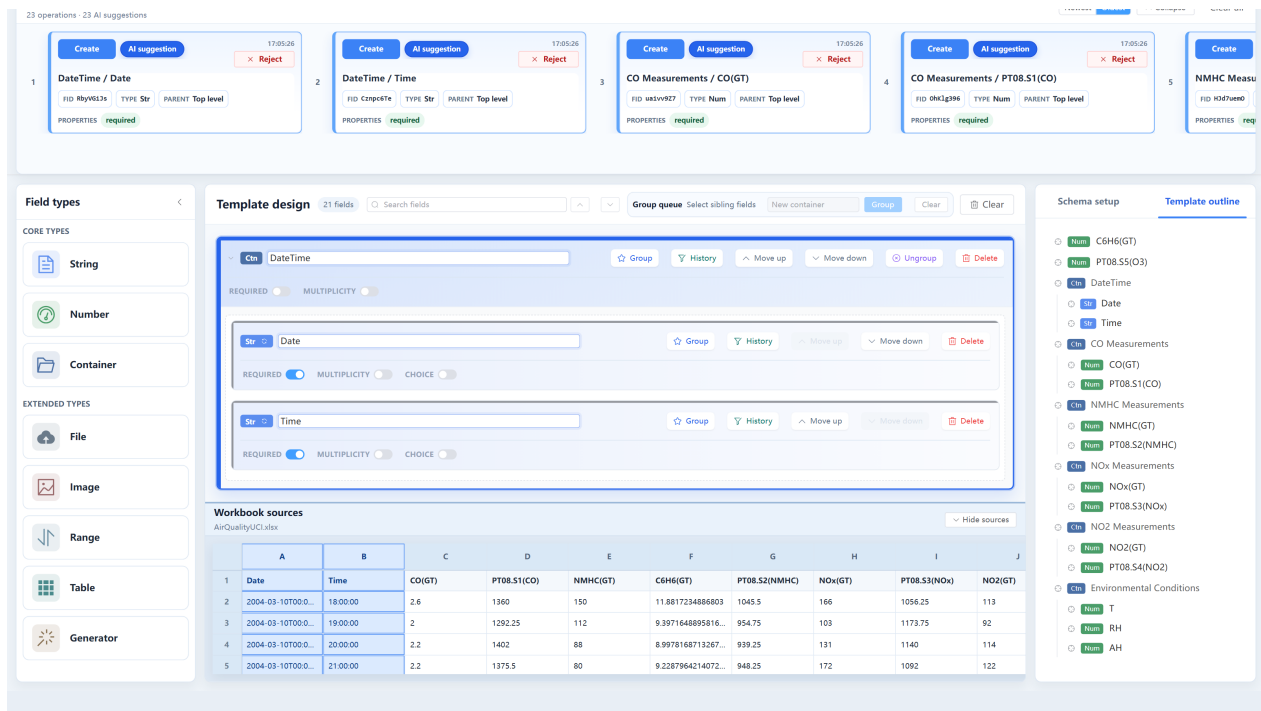


Figure 7. Schema outline for navigating a large schema. The outline shows the field hierarchy as a tree, and the locate control for DateTime brings the corresponding container into view and highlights it in the editor.

2.4. AI-Assisted Schema Construction

The field model and operation set above also define the action space for AI assistance. Rather than treating the LLM output as a replacement schema, our workflow asks the LLM to propose edits against a starting schema T_0 . The editor validates and compiles these proposals into the same operation space used for manual edits, and it executes the accepted editor operations to build the draft. For construction from scratch, T_0 is an empty schema, which is denoted by $T_\emptyset$.

The LLM input includes the current schema T_0 , the user-provided schema name, and the user-provided schema description. Together, the name and description may record the domain, scope, and purpose of the schema for later schema users, and they may also contain domain knowledge such as abbreviations or measurement conventions. External context may also be provided by the user, including detailed requirements such as user-specified fields or grouping requirements, or reference material such as an Excel workbook. These inputs help the LLM infer candidate field names, types, properties, and hierarchy.

Based on these inputs, the LLM outputs a list of edit proposals. Each proposal states the intended edit, the affected field or placement, and any new values such as field name, type, unit, or other properties. For example, the LLM can propose adding a Temperature field under Heat treatment. After validation and compilation, the resulting editor operations are shown in the same operation history as manual edits, which lets users review AI-assisted changes through the GUI.

Before execution, the editor validates each LLM proposal. The checks ensure that the proposal is complete, requests a supported edit, refers to an existing target when needed, and obeys the field-type and property rules. Table 5 summarizes these checks. Each LLM proposal is processed sequentially and can emit one or more editor operations. When a validation check fails, the processing of that proposal stops and a diagnostic is reported for

that proposal. Operations already emitted before the failure, together with operations from valid independent proposals, remain in the partial draft. A later proposal that depends on a field not created because of an earlier error is also omitted. The partial draft remains editable and can serve as the starting schema for a later update.

Table 5. Validation checks applied before AI-proposed operations are executed.

Check Point	What Is Checked	Example Error
Operation form	Required information is present for the proposed edit.	A create operation has no display name.
Supported edit and type	The edit action is supported and the field type is Str, Num, or Ctn.	An unknown action or a non-core field type.
Target and placement	Referenced fields exist, parent fields are containers, and sibling names remain valid.	A missing parent field or a duplicate sibling name.
Property compatibility	Field properties match the field type and field-control rules.	A unit annotation on a container field.

Accepted operations are executed to construct an editable schema draft in the GUI. From the interface perspective, this draft is presented in the same way as a manually edited schema: both are shown through operation records, as illustrated in Figure 5. Users can then continue editing the draft through the same operations and reject inappropriate proposed changes. The updated schema can also be sent back to the LLM as the new starting schema, so user edits and LLM edits can be combined in an iterative workflow. Figure 8 summarizes the complete workflow.

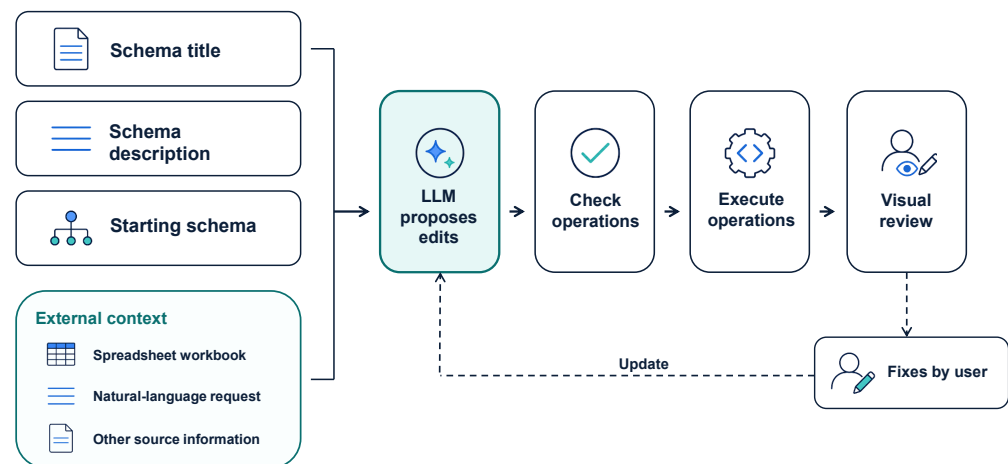


Figure 8. Generic workflow for LLM-assisted visual schema editing.

2.4.1. Excel Workbooks as LLM Input

In many scientific workflows, a tabular Excel workbook already contains an implicit schema. A column header provides a candidate field name, and the observed values in that column indicate the field type and field properties. The simplest way to obtain the schema from an Excel workbook is to send the workbook to an LLM and ask it to infer the schema. However, this formulation has practical and methodological limitations. First, real workbooks may contain many sheets, columns, and rows. Although some commercial or tool-augmented LLMs can read workbook files directly and extract relevant information, this assumption does not hold for many locally deployed LLMs. In a local deployment, the system usually needs to parse the workbook first and provide the LLM with a compact text representation of the relevant workbook content. Second, even when an LLM can access the workbook, some information should be obtained by deterministic parsing rather than semantic inference. For example, header names and column order can be read more

reliably by fixed parsing rules than by asking an LLM to rediscover the layout. If such rule-based layout information is mixed with semantic inference in one opaque generation step, layout-reading errors become difficult to distinguish from errors in field naming, type selection, unit recognition, or grouping.

We therefore do not treat the workbook itself as the direct LLM input. Instead, the system first extracts column-level layout information, such as column names and positions, together with sampled cell values. These elements can be read directly from the workbook by fixed parsing rules. The extracted information is used as the external context for the LLM planner. The planner can then focus on semantic and domain-related choices such as standardizing column header names, choosing field types, recognizing units, deciding whether observed values support Choice values or Multiplicity, and grouping semantically related fields into containers.

2.4.2. Extracting Workbook Information

Extraction produces one column description for each column. A column description records the column location, header text, leading cells in the same column, representative sampled cells from later rows, simple counts such as row count, non-empty count, and distinct-value count, and the observed distinct values when a column has only a small number of different values. These counts are computed over the configured extraction range, which is limited to 200 rows per worksheet in this paper, while the sampled cells provide a compact preview. The counts reveal missingness and low cardinality within that range but may miss rare mixed-type or outlier values, particularly beyond the row limit. Such cases still require user review; richer type-distribution and outlier analyses remain future work.

Table 6 provides an example sheet excerpt: headers use shorthand, units are embedded in labels, and some cells contain punctuation such as semicolons to separate multiple values. The header row contains samp#, phase(s), HT T(C), soak-hr, atm/gas, and SEM pic. After extraction, column E is recorded with header atm/gas, leading cells such as E2=Ar and E3=vacuum, later sampled cells such as E11=vacuum, and observed distinct values such as Ar and vacuum. Table 7 shows the full description for column E. Based on the extracted workbook information, the LLM may interpret column C as a temperature field with unit C, column E as an atmosphere field with a Choice property, and column F as an image field because its header is SEM pic and its values include image file names such as s01_sem.jpg.

Table 6. Example workbook fragment for illustrating column extraction.

	A	B	C	D	E	F
	samp#	phase(s)	HT T(C)	soak-hr	atm/gas	SEM pic
2	S-001	FCC; BCC	900	2	Ar	s01_sem.jpg
3	S-002	FCC	850	1	vacuum	s02_sem.jpg
4	S-003	BCC; mart.	780	3	N2	s03_sem.jpg
5	S-004	FCC; Laves	920	4	Ar	s04_sem.jpg
6	S-005	BCC	800	2	N2	s05_sem.jpg
7	S-006	FCC; B2	870	1	vacuum	s06_sem.jpg
8	S-007	mart.	760	3	Ar	s07_sem.jpg
9	S-008	FCC	840	2	N2	s08_sem.jpg
10	S-009	BCC; B2	910	4	(blank)	s09_sem.jpg
11	S-010	FCC; BCC	890	2	vacuum	s10_sem.jpg
12	S-011	Laves	930	5	N2	s11_sem.jpg
13	S-012	FCC	810	1	Ar	s12_sem.jpg

GenAI/LLM use note: This synthetic workbook fragment was generated by ChatGPT 5.6 and is used only to illustrate column extraction; it was not used in the evaluation.

Table 7. Column description extracted for column E in Table 6.

Extracted Information	Column E Example	Use in LLM Planning
Workbook position	Sheet Sheet1; column E	Indicate where the field should appear in the schema draft and preserve the workbook order.
Header text	Cell E1; raw text atm/gas	Suggest a field name such as Atmosphere and the semantic meaning of the gas environment.
Leading cells	E1=atm/gas, E2=Ar, E3=vacuum, E4=N2, . . .	Give early row context so the LLM can connect the header with concrete observed values.
Sampled cells	E11=vacuum, E12=N2, E13=Ar	Indicate that the field value is textual and provide candidate gas labels beyond the leading rows.
Value summary	Twelve data rows; eleven non-empty cells; one blank cell; three distinct values: Ar, vacuum, N2	Suggest that atm/gas may use a Choice property, and that Required should not be enabled because at least one row is blank.

Encoded or ambiguous headers such as Col1 or Var_A do not prevent column extraction or field creation. Sample values may still indicate a string or numeric type. However, when the header, values, and surrounding columns provide no semantic cues, the LLM may retain the original header and may be unable to infer either a descriptive field name or a unit not specified in the workbook. The resulting schema remains editable, but these fields require user review.

After the schema draft is generated through the workflow, the editor also shows where each generated field came from in the workbook. Figure 9 shows the draft after the atmosphere field has been created from column E. Selecting the schema field highlights the original workbook column; selecting column E in the workbook preview highlights that column and the corresponding schema field. The user can therefore move from a generated field back to its source column, or from a workbook column to the generated field, when inspecting why the field name, type, and properties were suggested.

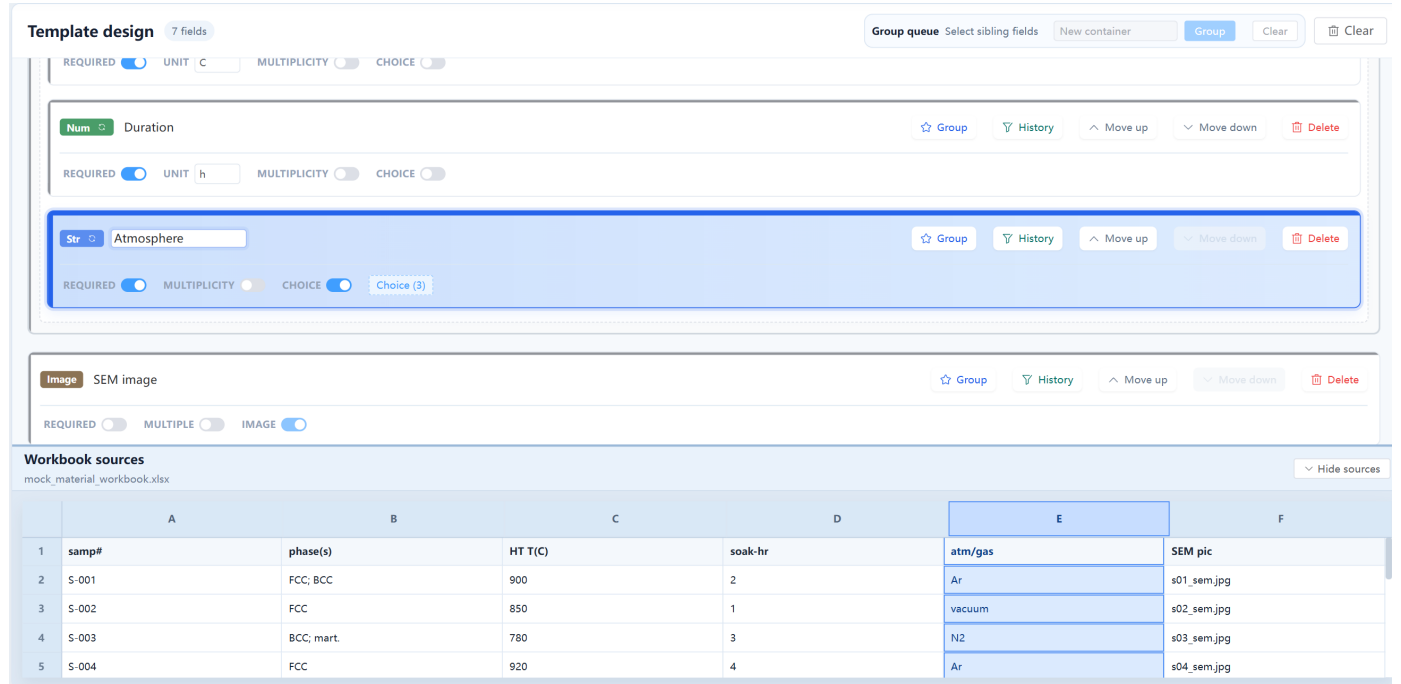


Figure 9. Workbook source review in the GUI editor. Column E in the workbook preview is linked to the generated Atmosphere field, making the correspondence between the raw workbook column and the schema field visible during review.

To illustrate the complete review process, we use ten columns from the Air Quality workbook distributed through the University of California, Irvine (UCI) Machine Learning Repository (Table 8). The LLM creates numeric fields for pollutant measurements, string fields for Date and Time, and a DateTime container for the latter two fields. Figure 10 shows the resulting operations and draft. Both interface figures use the field-history filter: Figure 10 displays operations associated with DateTime, whereas Figure 11 displays those associated with C6H6 (GT).

Table 8. Four sample rows from ten columns of the UCI Air Quality workbook.

Date	Time	CO (GT)	PT08.S1(CO)	NMHC (GT)	C6H6 (GT)	PT08.S2(NMHC)	NOx (GT)	PT08.S3(NOx)	NO2 (GT)
2004-03-10	18:00	2.6	1360	150	11.88	1045.50	166	1056.25	113
2004-03-10	19:00	2.0	1292.25	112	9.40	954.75	103	1173.75	92
2004-03-10	20:00	2.2	1402	88	9.00	939.25	131	1140	114
2004-03-10	21:00	2.2	1375.50	80	9.23	948.25	172	1092	122

The figure displays a software interface for editing operations and schema design. It includes sections for 'Editing operations' (showing 3 of 21 operations), 'Field types' (listing CORE and EXTENDED types like String, Number, Container, File, Image, Range, Table, and Generator), 'Template design' (showing a draft for DateTime with sub-fields Date and Time), and 'Workbook sources' (displaying a table of data from the UCI Air Quality dataset).

	A	B	C	D	E	F	G	H	I	J
1	Date	Time	CO (GT)	PT08.S1(CO)	NMHC (GT)	C6H6 (GT)	PT08.S2(NMHC)	NOx (GT)	PT08.S3(NOx)	NO2 (GT)
2	2004-03-10T00:00...	18:00:00	2.6	1360	150	11.8817234886803	1045.5	166	1056.25	113
3	2004-03-10T00:00...	19:00:00	2	1292.25	112	9.3971648895816...	954.75	103	1173.75	92
4	2004-03-10T00:00...	20:00:00	2.2	1402	88	8.9978168713267...	939.25	131	1140	114
5	2004-03-10T00:00...	21:00:00	2.2	1375.5	80	9.2287964214072...	948.25	172	1092	122

Figure 10. LLM-generated operations and schema draft for the UCI Air Quality example. The operation history is filtered to DateTime, showing the creation of the Date and Time string fields and the operation that groups them in the DateTime container.

The draft correctly assigns units to T, RH, and AH, but it omits units for five reference-measurement fields. During review, the user retains the generated Create and Group operations and adds the missing units through manual Configure operations. Figure 11 illustrates this correction for C6H6 (GT); the same procedure is applied to CO (GT), NMHC (GT), NOx (GT), and NO2 (GT). The PT08.S2(NMHC) field remains unitless because the dataset documentation specifies no unit.

Beyond single-row headers, the extractor supports merged and multi-level headers that form a recognizable rectangular region. It records merged-cell ranges and assigns each data column a header path that combines the shared higher-level header with its leaf header. Table 9 illustrates this structure.

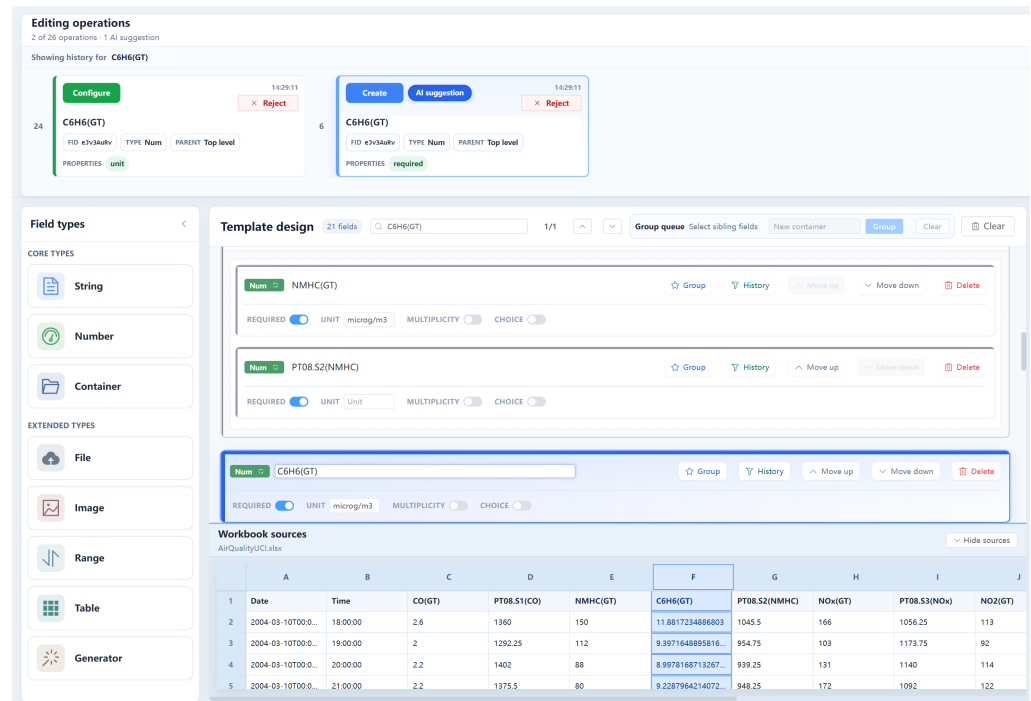


Figure 11. Schema after partial manual review of the same example. The filtered view shows the generated Create operation for C6H6(GT) and the manual Configure operation that adds the missing unit microg/m3; the corresponding workbook column is highlighted below.

Table 9. Example worksheet fragment with merged and multi-level headers.

	A	B	C	D	E	F
1	Heat Treatment			Mechanical Test		
2	Sample ID	Temperature (°C)	Time (h)	Atmosphere	Hardness (HV)	Yield Strength (MPa)
3	S01	850	2	Ar	280	640
4	S02	900	1.5	N2	295	670
5	S03	750	3	vacuum	260	610

In Table 9, ranges B1:D1 and E1:F1 define the shared headers Heat treatment and Mechanical test. For each data column, the LLM input includes its location, leaf header, sample values, full header path, and covering merged range. The shared header can then guide the grouping of related columns.

The visual schema can be exported as JSON Schema Draft 2020-12. Multiplicity is represented as an array, and a Generator is represented as an object with oneOf constraints over its mutually exclusive child fields. Required and Choice properties are exported using the corresponding standard keywords. Each export is validated against the Draft 2020-12 meta-schema, which checks syntax and keyword structure but not domain-specific conventions. Integration with ontologies and standard schema libraries remains future work.

3. Evaluation

To evaluate the Excel-to-schema workflow, we use six public tabular datasets spanning materials science, chemistry, environmental science, engineering, and medicine. A separate case study evaluates the workflow on a large multi-worksheet workbook. The evaluation

also includes a user study comparing manual construction with LLM-assisted review in the same visual editor.

3.1. Datasets and Setup

Table 10 lists the datasets used in the evaluation.

Table 10. Public tabular datasets used in the main evaluation.

Dataset	Workbook Content
Mg twin nucleation [16]	Grain-level microstructure features, Schmid factors, misorientation statistics, and twin-nucleation labels in Mg alloys.
OBELiX solid electrolytes [17]	Lithium solid-state electrolyte composition, lattice-parameter, ionic-conductivity, publication-reference, and Crystallographic Information File (CIF) metadata fields.
Concrete compressive strength [18]	Concrete mixture components, age, and compressive-strength measurements.
Air quality [19]	Gas-sensor responses, reference gas concentrations, and environmental measurements.
Environmental Protection Agency Risk-Screening Environmental Indicators (EPA RSEI) chemical table [20]	Chemical identifiers, toxicity values, source fields, flags, and physicochemical properties.
Cirrhosis survival [21]	Clinical variables for cirrhosis patient survival prediction.

The six datasets were selected because they are public, span multiple scientific domains and spreadsheet sizes, and document their workbook fields and, where available, units, value types, and allowed values.

The evaluation uses a locally deployed Qwen3.6-27B LLM as the primary schema-planning component. The LLM proposes schema-editing operations from the extracted workbook descriptions, which the editor validates and compiles. We also evaluate the workflow with GPT-5.6-So1 and DeepSeek-V4-Flash on the six main datasets.

Qwen3.6-27B is run with temperature 0 and chain-of-thought output disabled. Responses were required to be strict JSON and were processed using the validation and compilation procedure described in Section 2.4.

The client ran on a Windows workstation with an AMD Ryzen 9 9950X CPU and 128 GB RAM; Qwen3.6-27B ran on a server with an Intel Xeon Platinum 8558P CPU, 1 TB RAM, and an NVIDIA H20 GPU.

3.2. Schema Construction Accuracy

The first experiment evaluates schema construction from an empty schema. For each workbook, the generated schema is compared with the dataset reference schema using the five metrics in Table 11. Generated and reference fields are matched one-to-one by normalized field name. Type, property, source, and parent-placement scores require agreement with the corresponding reference field.

Container fields are excluded from the leaf-field coverage, type, property, and source scores. Instead, we evaluate their organization through the placement of their leaf fields: a leaf is correct when it is top-level or nested in the same way as its reference field and, if nested, has the same sibling fields. This score compares the generated schema with the dataset's reference hierarchy, so it may mark another reasonable grouping as incorrect. Container names themselves are not scored. Boolean-like columns are evaluated within the editor model because the editor has no Boolean core type; fixed binary values such as Yes/No or 0/1 are represented as Choice values on string or numeric fields in both the generated and reference schemas.

For the cross-model comparison, all three models receive the same extracted workbook descriptions and prompts and use the same validation and scoring rules. Each model starts from an empty schema, and no dataset documentation or reference schema is included in the prompts. Each model is run three times on every dataset, and the results are reported

as the mean $\pm$ sample standard deviation (SD). DeepSeek-V4-Flash uses temperature 0 with thinking disabled.

Table 11. Metrics used to compare generated schema drafts with the reference schema for each dataset.

Metric	Definition
Field coverage	Percentage of expected leaf fields matched one-to-one by normalized name. A generated name that appends unit text supported by the source header or reference unit is treated as matched but still requires a name correction.
Type correctness	Percentage of matched leaf fields whose exported string or numeric type equals the reference type.
Property correctness	Percentage of matched leaf fields whose normalized Required, Unit, Choice, Multiplicity, File, and Image properties equal the reference properties.
Source correctness	Percentage of matched leaf fields whose editor source reference equals the corresponding workbook column reference.
Parent placement	Percentage of matched leaf fields with the same top-level or nested status as the reference and, when nested, the same sibling fields.

The Qwen3.6-27B construction results for the six main datasets are shown in Table 12. All outputs are valid and compile without validation errors. Field coverage and source correctness are both $100.0\% \pm 0.0\%$. The overall type, property, and parent-placement correctness are $97.6\% \pm 0.3\%$, $81.1\% \pm 1.3\%$, and $98.2\% \pm 3.2\%$, respectively. The remaining mismatches mostly concern units, finite-value constraints, field grouping, and the types of identifiers or sparsely populated columns.

Table 12. Qwen3.6-27B schema-construction accuracy on the six main datasets. Percentage values are the mean $\pm$ sample SD over three independent runs.

Case	Fields	Covered	Coverage	Type	Property	Source	Parent
Mg twin nucleation	31	31	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$83.9 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$
OBELiX solid electrolytes	28	28	$100.0 \pm 0.0\%$	$86.9 \pm 4.1\%$	$85.7 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$
Concrete strength	9	9	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$
Air quality	15	15	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$66.7 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$
EPA RSEI chemical	80	80	$100.0 \pm 0.0\%$	$99.2 \pm 0.7\%$	$85.4 \pm 2.9\%$	$100.0 \pm 0.0\%$	$95.8 \pm 7.2\%$
Cirrhosis survival	20	20	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$55.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$
Overall	183	183	$100.0 \pm 0.0\%$	$97.6 \pm 0.3\%$	$81.1 \pm 1.3\%$	$100.0 \pm 0.0\%$	$98.2 \pm 3.2\%$

Table 13 compares the schema-construction results of the three evaluated models on the same six workbooks.

Table 13. Cross-model comparison on the same six workbooks. Values are the mean $\pm$ sample SD over three independent runs. The final row gives the mean of the six dataset results. All models achieve complete field coverage and source correctness; a field with one or more name, type, property, or parent-placement errors is counted once under the Fields Requiring Correction column.

Case	Type Correctness			Property Correctness			Fields Requiring Correction		
	Qwen	GPT	DeepSeek	Qwen	GPT	DeepSeek	Qwen	GPT	DeepSeek
Mg twin nucleation	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$83.9 \pm 0.0\%$	$89.2 \pm 13.0\%$	$83.9 \pm 0.0\%$	5.0 ± 0.0	3.3 ± 4.0	31.0 ± 0.0
OBELiX solid electrolytes	$86.9 \pm 4.1\%$	$100.0 \pm 0.0\%$	$89.3 \pm 0.0\%$	$85.7 \pm 0.0\%$	$76.2 \pm 16.5\%$	$78.6 \pm 6.2\%$	7.0 ± 0.0	14.7 ± 4.6	28.0 ± 0.0
Concrete strength	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	0.0 ± 0.0	3.0 ± 5.2	9.0 ± 0.0
Air quality	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$66.7 \pm 0.0\%$	$82.2 \pm 30.8\%$	$93.3 \pm 0.0\%$	5.0 ± 0.0	6.0 ± 3.5	15.0 ± 0.0
EPA RSEI chemical	$99.2 \pm 0.7\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$85.4 \pm 2.9\%$	$90.8 \pm 5.1\%$	$77.9 \pm 2.9\%$	15.0 ± 6.6	41.0 ± 1.7	80.0 ± 0.0
Cirrhosis survival	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$100.0 \pm 0.0\%$	$55.0 \pm 0.0\%$	$70.0 \pm 17.3\%$	$45.0 \pm 0.0\%$	9.0 ± 0.0	8.0 ± 6.9	20.0 ± 0.0
Mean across datasets	$97.7 \pm 0.6\%$	$100.0 \pm 0.0\%$	$98.2 \pm 0.0\%$	$79.4 \pm 0.5\%$	$84.7 \pm 13.8\%$	$79.8 \pm 1.3\%$	6.8 ± 1.1	12.7 ± 3.8	30.5 ± 0.0

The editor compiled every GPT-5.6-Sol and DeepSeek-V4-Flash response into executable operations without validation errors, and all models achieved $100.0\% \pm 0.0\%$ field coverage and source correctness. The main model-dependent differences concern field properties and hierarchy. Averaged across the six datasets, GPT-5.6-Sol achieved $100.0\% \pm 0.0\%$ type correctness, $84.7\% \pm 13.8\%$ property correctness, and $80.2\% \pm 3.0\%$ parent-placement correctness. DeepSeek-V4-Flash achieved $98.2\% \pm 0.0\%$ type correctness and $79.8\% \pm 1.3\%$ property correctness but created no containers, so every field required a parent-placement correction. Thus, the workflow runs successfully with all three models, while schema quality remains dependent on the model's ability to infer properties and grouping.

Table 14 examines the main error patterns in the three Qwen3.6-27B runs.

Table 14. Main error patterns observed across the three Qwen3.6-27B schema-construction runs.

Error Pattern	Frequency Across Runs	Reference Information and Workbook Content	Interpretation
Missing units	26 fields in each run	The units are specified in the dataset documentation but do not appear in the workbook headers or values supplied to the model.	Units that do not appear in the workbook cannot be recovered reliably.
Incorrect type for columns without values	Two fields in run 1; none in runs 2–3	The columns contain no non-empty values. Their type is specified only in the dataset documentation; it is not in the Excel workbook.	The model therefore cannot determine their type reliably from the workbook.
Identifier columns treated as numbers	Three fields in every run	The column names contain ID, indicating that the values are identifiers, but all observed values are numbers.	The model treats these columns as numeric fields because their values are numbers despite the ID in their names.
Allowed-value lists	Four missing lists in run 1; seven extra lists in each of runs 2 and 3; one incomplete list in every run	The extracted preview contains recurring categorical values, but it does not establish that every observed set is complete. For one field, only 0 occurs in the workbook range, whereas the dataset documentation lists three valid directions.	The model does not consistently distinguish observed examples from a complete list of allowed values.

The errors have three distinct sources. First, the missing units do not appear in the workbook and therefore cannot be inferred reliably without additional domain information. Second, some columns contain no values from which to infer a type, while other columns are labeled as identifiers but contain only numbers. For the latter, the model treats the columns as numeric fields despite the word ID in their names. Third, the allowed-value errors arise because observed categories do not show whether the list is complete. Qwen alternates between omitting allowed-value lists required by the reference and treating sampled values as exhaustive. In these cases, the extractor identifies the correct columns and headers; the errors occur when the model infers field types and properties from incomplete or ambiguous workbook information.

3.3. Comparison with Direct Complete-Schema Generation and MetaConfigurator

To evaluate whether the operation workflow produces more accurate schemas with fewer fields requiring correction, we compare it with direct complete-schema generation and MetaConfigurator on the same six datasets using Qwen3.6-27B. In the operation workflow, model suggestions are represented as validated field-level editor operations. In the direct complete-schema workflow, the model receives the same extracted workbook descriptions but returns a complete JSON Schema. The two workflows use the same model settings.

For spreadsheet input, MetaConfigurator imports CSV data and deterministically infers an initial schema without using an LLM. Its LLM-assisted Schema Modify function can then revise this existing schema through a natural-language request; the LLM does not generate the schema directly from the CSV data. For the comparison, we export the same rectangular workbook region to CSV and import it into MetaConfigurator. We then use Schema Modify to revise the inferred schema, including the imported-column mapping and the same extracted workbook descriptions in the request. MetaConfigurator is run with its default CSV-import and schema-inference settings.

All three workflows use Qwen3.6-27B with the same decoding settings, exclude dataset documentation and reference schemas from the model input, and receive no manual correction before scoring. Each workflow is run three times on each of the six datasets. An output is invalid if it cannot be parsed, does not conform to Draft 2020-12, or does not describe a record object. We apply the same construction-accuracy rules used for Table 12 and report the measures available for all three workflows: field coverage, type, property, and parent-placement correctness, together with the number of fields requiring correction. Property correctness covers Required, Unit, Choice/enum, and Multiplicity/array. An identifier recorded in MetaConfigurator's import mapping may establish a field match, but a changed name still counts as an error.

Table 15 reports the results for the three workflows. All outputs are valid record schemas, and all three workflows achieve 100% field coverage on the same six datasets used in the main experiment. Type correctness is also similar. The operation workflow achieves 80.3% property correctness and 98.2% parent-placement correctness compared with 14.8% and 56.3% for the direct complete-schema workflow and 36.6% and 0.0% for MetaConfigurator. MetaConfigurator retains a flat schema, so every matched field requires a parent-placement correction. On average, the operation workflow leaves 42.7 fields requiring correction compared with 167.7 for the direct complete-schema workflow and 183.0 for MetaConfigurator.

Table 15. Schema-construction results for the three workflows. Each workflow was evaluated in three independent repetitions on the same six datasets. For each repetition, the results were aggregated across the six datasets; the table reports the mean $\pm$ sample SD over the three repetitions.

Workflow	Coverage	Type	Property	Parent	Fields to Correct
Operation workflow	100.0 $\pm$ 0.0	97.8 $\pm$ 0.0	80.3 $\pm$ 0.0	98.2 $\pm$ 3.2	42.7 $\pm$ 4.6
Direct complete schema	100.0 $\pm$ 0.0	97.3 $\pm$ 0.0	14.8 $\pm$ 0.0	56.3 $\pm$ 12.8	167.7 $\pm$ 0.6
MetaConfigurator	100.0 $\pm$ 0.0	96.7 $\pm$ 0.0	36.6 $\pm$ 0.0	0.0 $\pm$ 0.0	183.0 $\pm$ 0.0

Table 16 reports property correctness and the number of fields requiring correction for each dataset. The operation workflow has the highest property correctness and leaves the fewest fields requiring correction on all six datasets. All three workflows achieve 100% field coverage on every dataset.

All three workflows recover the expected fields but differ in property accuracy and grouping. The operation workflow represents suggestions as validated field- or group-level operations, whereas direct complete-schema generation returns the entire schema. MetaConfigurator's CSV importer creates a flat initial schema, and its LLM function modifies that schema only through a subsequent natural-language request. The resulting schemas remain flat and omit many semantic properties; adding groups and correcting the remaining properties would require manual editing or further LLM interaction.

Table 16. Per-dataset schema-construction results for the three workflows. Property denotes mean property correctness across three repetitions; Fields to correct is reported as mean $\pm$ sample SD.

Dataset	Measure	Operation Workflow	Direct Complete Schema	MetaConfigurator
Mg twin nucleation	Property	83.9%	12.9%	67.7%
	Fields to correct	5.0 ± 0.0	28.0 ± 0.0	31.0 ± 0.0
OBELiX solid electrolytes	Property	85.7%	25.0%	82.1%
	Fields to correct	7.0 ± 0.0	22.0 ± 0.0	28.0 ± 0.0
Concrete strength	Property	100.0%	11.1%	22.2%
	Fields to correct	0.0 ± 0.0	9.0 ± 0.0	9.0 ± 0.0
Air quality	Property	66.7%	40.0%	20.0%
	Fields to correct	5.0 ± 0.0	10.7 ± 0.6	15.0 ± 0.0
EPA RSEI chemical	Property	83.8%	6.3%	11.3%
	Fields to correct	16.7 ± 4.6	80.0 ± 0.0	80.0 ± 0.0
Cirrhosis survival	Property	55.0%	20.0%	45.0%
	Fields to correct	9.0 ± 0.0	18.0 ± 0.0	20.0 ± 0.0

3.4. Efficiency Estimate

In addition to accuracy, the efficiency gain from AI assistance is also important. In our editor, both the manual construction and post-editing of an LLM-generated draft are represented by editor operations. Once the final schema has been fixed, we can compare the operations needed to build it from an empty schema with the operations needed to edit the LLM-generated draft into the final schema and quantify the difference.

For each dataset, we use the schema draft produced in the preceding experiment as the starting schema T_0 . We then manually edit this draft until its fields match the field information described in the dataset documentation; the resulting schema is denoted by T^* . We then compute the minimum number of editor operations for two paths: constructing T^* from an empty schema, and editing the LLM-generated draft into T^* . Assigning a cost to each editor operation gives a quantitative estimate of how much editing is saved by starting from the LLM-generated draft compared with constructing the same schema manually from scratch. Table 17 lists the operation costs used for this comparison. These costs are simplified and abstract. For example, a field creation costs 3 because the user must choose the field type, choose its position, and enter the field name. A configured property costs 3 because the user must select the field, enable the property, and enter or choose the property value.

Table 17. Operation costs.

Editor Operation	Cost	Cost Basis
Create(p, j, k, c, θ)	3	Choose the type, position, and field name.
Name(fid, k)	2	Select the field and enter the new name.
Type(fid, c)	2	Select the field and choose another type.
Configure(fid, s, x)	3	Select the field, enable the property, and set the value.
Unconfigure(fid, s)	2	Select the field and clear the property.
Move(fid, p, j)	2	Select the field and choose the new position.
Group($\langle fid_1, \dots, fid_m \rangle, p, j, k, \theta$)	$m + 1$	Select m fields and enter the container name.
Ungroup(fid)	1	Remove one container layer.
Delete(fid)	1	Delete one field or container.

Manual cost is computed from the path from an empty schema to T^* . Post-edit cost is computed from the path from the LLM-generated draft to T^* . Their ratio expresses manual-construction cost as a multiple of post-editing cost. To assess sensitivity to the assigned weights, we use a manual-favoring setting that assigns a weight of 1 to Create and 5 to Configure, Rename, Type, Move, and Unconfigure. Table 18 reports the ratios under both settings.

Table 18. Operation-cost results and sensitivity analysis for the six main datasets. Manual cost and Post-edit cost use the default weights in Table 17. The manual-favoring setting assigns a weight of 1 to Create and 5 to Configure, Rename, Type, Move, and Unconfigure. Group, Ungroup, and Delete retain their default weights. The ∞ symbol indicates that no post-editing operations were required.

Case	Manual Cost	Post-Edit Cost	Default-Weight Ratio	Manual-Favoring Ratio
Mg twin nucleation	157	18	8.7×	3.9×
OBELiX solid electrolytes	147	20	7.4×	2.6×
Concrete strength	62	0	∞	∞
Air quality	88	15	5.9×	3.0×
EPA RSEI chemical	426	27	15.8×	7.4×
Cirrhosis survival	132	27	4.9×	2.9×

Changing to the manual-favoring weights does not change the conclusion. The five finite manual-to-post-editing ratios range from 4.9× to 15.8× with the default weights and from 2.6× to 7.4× with the manual-favoring weights. The Concrete draft has an infinite ratio because it requires no post-editing. Post-editing therefore has the lower weighted operation cost for every dataset under both settings.

3.5. User Study

To determine whether the proposed workflow reduces schema-construction time in practice, we conducted a between-groups user study with ten computer science graduate students. The participants were randomly assigned in equal numbers to the two conditions: five used only the manual controls of the proposed GUI, and five used the LLM-assisted workflow.

All participants constructed the same UCI Air Quality schema from an empty schema. They received a common specification listing the target schema’s 15 data fields, six containers, field types, units, and the required settings.

In the manual condition, participants created and configured every field through the GUI. In the assisted condition, participants invoked Qwen3.6-27B once, waited for the resulting editor operations, and then reviewed and corrected the draft through the same GUI. Timing began when each participant started the task—before the Qwen request in the assisted condition—and ended when the completed schema satisfied the supplied checklist.

As shown in Table 19, LLM assistance reduced the mean end-to-end completion time from 380.0 to 114.4 s, which is a 69.9% reduction. The manual-to-assisted time ratio is 3.32× between the Air Quality operation-cost ratios obtained with the manual-favoring weights (3.0×) and the default weights (5.9×) in Table 18. The ratios are similar in scale; the completion-time measurement also includes GUI interaction, keyboard input, model generation, and variation among participants. An exact permutation test based on the difference in mean completion time showed a statistically significant difference between the two groups ($p = 0.0079$).

Table 19. User-study completion times.

Condition	Participant	Completion Time (s)
Manual	M1	422
	M2	389
	M3	329
	M4	359
	M5	401
	Mean ± SD	380.0 ± 36.5

Table 19. *Cont.*

Condition	Participant	Completion Time (s)
LLM-assisted	A1	91
	A2	118
	A3	129
	A4	125
	A5	109
	Mean $\pm$ SD	114.4 $\pm$ 15.1

3.6. Existing-Schema Update Accuracy

When columns are added, removed, or changed in a later version of a workbook, its existing schema must be updated accordingly. We evaluate whether the proposed workflow can apply these changes to an existing schema. For each of the six datasets, the starting schema is the manually corrected schema T^* used in the operation-cost analysis. From the original workbook, we create the four modified workbook variants listed in Table 20, giving 24 modified-workbook cases for each model.

We evaluate update accuracy from two perspectives: changed workbook columns should produce the expected schema updates, while fields for unchanged columns should remain unchanged. In the Changed variant, each dataset has four targets: revised headers require field renaming, and newly inserted blank cells require removal of the required property.

Table 20. Modified workbook variants used for evaluating updates to an existing schema.

Variant	Workbook Change	Expected Schema Update
Removed	Remove four existing columns	Remove the corresponding fields
New	Add four new columns	Add fields for the new columns
Changed	Modify four existing columns by renaming their headers or inserting blank cells	Rename the corresponding fields or unset Required
Combined	Apply the same Removed, New, and Changed edits together	Apply all corresponding schema updates

A target update is correct only when the field's existence, name, type, properties, and source match the expected updated schema; parent-container placement is not included.

Table 21 lists the accuracy of the workflow's modifications to an existing schema, considering only the update targets and excluding fields that do not need to change. Qwen3.6-27B reaches 100.0% $\pm$ 0.0% accuracy in all four variants.

Table 21. Accuracy of intended schema updates for modified workbooks, which is reported as the mean $\pm$ sample SD over three independent runs.

Dataset	Removed	New	Changed	Combined
Mg twin nucleation	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
OBELiX solid electrolytes	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
Concrete strength	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
Air quality	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
EPA RSEI chemical	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
Cirrhosis survival	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
All	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%

Target-update accuracy alone does not show whether the rest of the existing schema remains stable. The second aspect is therefore whether fields corresponding to unchanged workbook columns remain untouched. Table 22 evaluates this preservation rate for non-target fields. For each update variant, we compare all existing fields that are neither removed-column targets nor changed-column targets. A field is counted as preserved only when its name, type, properties, and parent container remain unchanged.

Table 22. Preservation rate for existing fields that are not intended update targets, which is reported as the mean $\pm$ sample SD over three independent runs. Any change to a field's name, type, properties, or parent field is counted as not preserved.

Dataset	Removed	New	Changed	Combined
Mg twin nucleation	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
OBELiX solid electrolytes	80.6 $\pm$ 2.4%	89.3 $\pm$ 0.0%	87.5 $\pm$ 0.0%	75.0 $\pm$ 0.0%
Concrete strength	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
Air quality	100.0 $\pm$ 0.0%	86.7 $\pm$ 0.0%	93.9 $\pm$ 5.2%	100.0 $\pm$ 0.0%
EPA RSEI chemical	96.9 $\pm$ 0.8%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	96.8 $\pm$ 0.8%
Cirrhosis survival	93.8 $\pm$ 0.0%	96.7 $\pm$ 2.9%	93.8 $\pm$ 0.0%	91.7 $\pm$ 0.0%
All	95.0 $\pm$ 0.0%	96.9 $\pm$ 0.3%	97.1 $\pm$ 0.4%	93.8 $\pm$ 0.4%

Overall, the preservation rates are 95.0% $\pm$ 0.0%, 96.9% $\pm$ 0.3%, 97.1% $\pm$ 0.4%, and 93.8% $\pm$ 0.4% for the Removed, New, Changed, and Combined variants, respectively.

For the cross-model comparison, all three models start from the same manually corrected schema and receive the same modified workbook, prompts, validation procedure, and scoring rules. Each model is run three times on every update case, and the results are reported as the mean $\pm$ sample SD. Table 23 summarizes the target-update accuracy and preservation rates of the three models.

Table 23. Cross-model existing-schema update results. Values are the mean $\pm$ sample SD over three independent runs of the 24 cases from the six main datasets.

Model	Measure	Removed	New	Changed	Combined
Qwen3.6-27B	Target-update accuracy	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
	Preservation	95.0 $\pm$ 0.0%	96.9 $\pm$ 0.3%	97.1 $\pm$ 0.4%	93.8 $\pm$ 0.4%
GPT-5.6-Sol	Target-update accuracy	100.0 $\pm$ 0.0%	97.2 $\pm$ 2.4%	100.0 $\pm$ 0.0%	95.4 $\pm$ 4.0%
	Preservation	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%
DeepSeek-V4-Flash	Target-update accuracy	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	97.2 $\pm$ 0.0%
	Preservation	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%	100.0 $\pm$ 0.0%

All three models preserve most or all non-target fields, but their target-update accuracy differs. Qwen3.6-27B performs every intended update in all three runs. GPT-5.6-Sol shows variation in the New and Combined variants, while DeepSeek-V4-Flash consistently misses two targets in the Combined variant.

3.7. Large-Workbook Case Study

We evaluate the World Development Indicators (WDI) workbook separately as a large-workbook case study using Qwen3.6-27B. The 79.9 MB workbook contains six rectangular worksheets with a total of 130 source columns and up to 842,972 rows in one worksheet. The extractor retains all worksheets and columns but examines at most 200 rows per worksheet. Table 24 summarizes the construction, update, and operation-cost results.

Table 24. Results for the WDI large-workbook case study. Construction timing and operation counts are reported as mean $\pm$ sample SD over three independent runs.

Evaluation	Measure	Result
Workbook	Extraction time	128.2 s
Construction	Runs	3; all completed successfully
	Field coverage, type correctness, and source correctness	100.0%, 100.0%, and 100.0%
	Property correctness	96.2%
	LLM planning time	197.7 $\pm$ 7.2 s
	Editor operations	137.3 $\pm$ 0.6
Update	Target-update accuracy	Removed 100.0%; New 100.0%; Changed 50.0%; Combined 83.3%
	Non-target preservation	100.0% for all four variants
Operation cost	Manual construction/post-editing	510/19
	Manual-to-post-editing ratio	26.8 $\times$ with default weights; 8.1 $\times$ with manual-favoring weights

All three construction runs produce valid schemas containing correctly typed and sourced fields for all 130 columns. Post-editing also retains a lower operation cost than manual construction under both the default and manual-favoring weights.

The WDI workbook contains columns with the same names on different worksheets, and these columns link related information across sheets. Because the current workflow does not model cross-worksheet relationships, it creates them as independent schema fields and appends worksheet names to distinguish them. In three mapping and footnote worksheets, the header detector treats the first data row as part of the header region for ten columns, although their final names and types remain correct. The WDI runs produced operation histories containing 137–138 operations, which require users to filter the history when reviewing a particular field. The current GUI links workbook columns and schema fields in both directions. When a user selects a column in the workbook view, the schema editor moves to and highlights the corresponding field; the user can then filter the operation history to show only operations for that field. These functions provide basic support for reviewing long histories, but the prototype still requires further UI/UX design, such as a clearer grouping and summarization of related operations. Operation-level representation and LLM assistance alone do not remove the review burden for large schemas.

4. Discussion

Although the workflow produces accurate editable drafts, it remains an assistant rather than a replacement for the user. Its performance depends on LLM capability, workbook quality, and whether the workbook headers and values provide enough semantic cues for field naming, typing, property selection, and grouping.

Several aspects of the workflow remain to be developed, including support for richer schema structures, more effective human–AI review of irregular and large workbooks, and links to established vocabularies and ontologies.

Richer schema semantics. The current schema model does not support recursive schemas or dynamic if–then–else branches. Future work should extend the visual schema model to represent these structures. Cross-worksheet relationships are another limitation: the workflow does not infer primary keys, foreign keys, or 1: N and N : M cardinalities. A relationship-aware extension should propose these links for user confirmation.

Robust and collaborative human–AI editing. The extractor handles contiguous tables with one or more header rows, but it has difficulty with unmerged notes, headers that begin on different rows, and multiple tables within one worksheet. Future work should improve

extraction for these layouts and ask users to confirm ambiguous cases. Large schemas can also produce long operation histories, motivating operation grouping and summarization. Collaborative editing should allow multiple users to review the same schema, record who accepted or rejected each operation, and resolve conflicting edits.

Schema interoperability and ontology alignment. Different datasets may use different field names for the same scientific concept, making their schemas difficult to combine. Future work could match generated fields to terms in controlled vocabularies, domain ontologies, or established schemas. For each suggested match, the editor should show the original workbook field, the proposed standard term, and its definition so that the user can accept or reject the mapping. Accepted mappings could then be stored in the schema, allowing differently named fields from separate datasets to be recognized as representing the same concept.

5. Conclusions

This paper presents an AI-assisted visual schema editing workflow for constructing scientific metadata schemas from Excel workbooks. Instead of treating the LLM result as a complete replacement schema, the method represents the result as a series of editor operations in the same operation space as manual editing. These operations can be accepted or rejected individually in a pure GUI editor, enabling a field-level review of AI-assisted schema changes without requiring users to read or edit JSON Schema syntax.

The evaluation shows that the workflow can operate with different LLMs, although the quality of inferred properties and grouping remains dependent on the selected model. Compared with direct complete-schema generation and MetaConfigurator, representing suggestions as editor operations produces drafts that require less correction and gives users direct control over individual changes. The operation-cost analysis and user study further indicate that starting from an LLM-generated draft can reduce the effort required to construct a schema.

The large-workbook case study shows that the workflow can process multiple worksheets and large numbers of rows and columns, but it also defines the current scope of the approach. Cross-worksheet relationships are not inferred, irregular header layouts remain difficult, and long operation histories increase the burden of review. These limitations motivate relationship-aware schema modeling, more robust workbook extraction, and an improved organization of operations for large collaborative projects.

By reducing the effort required to prepare and maintain reusable metadata schemas, the workflow helps lower a practical barrier to FAIR scientific data sharing in spreadsheet-based research workflows.

Author Contributions: W.C.: conceptualization, methodology, software, visualization, and writing—original draft preparation. X.Z.: conceptualization, resources, writing—review and editing, supervision, project administration, and funding acquisition. J.H.: resources, writing—review and editing, supervision, and project administration. Z.Z.: validation and writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Advanced Materials-National Science and Technology Major Project under Grant No. 2024ZD0607700. The APC was funded by the Advanced Materials-National Science and Technology Major Project under Grant No. 2024ZD0607700.

Institutional Review Board Statement: In accordance with Article 32 of the Measures for Ethical Review of Life Science and Medical Research Involving Human Subjects (China, 2023), this study was exempt from ethical review because it involved no medical or health-related intervention, sensitive personal information, or commercial interests.

Informed Consent Statement: Informed consent for participation was obtained from all subjects involved in the user study.

Data Availability Statement: The public source datasets are available from the repositories cited in the Evaluation section. All data along with the code used in the evaluation is available in the project GitHub repository: <https://github.com/scontainer-paper/llm-schema-editor> (accessed on 22 August 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wilkinson, M.D.; Dumontier, M.; Aalbersberg, I.J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.-W.; da Silva Santos, L.B.; Bourne, P.E.; et al. The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* **2016**, *3*, 160018. [CrossRef] [PubMed]
2. Batista, D.; González-Beltrán, A.; Sansone, S.A.; Rocca-Serra, P. Machine actionable metadata models. *Sci. Data* **2022**, *9*, 592. [CrossRef] [PubMed]
3. Musen, M.A.; O'Connor, M.J.; Schultes, E.; Martínez-Romero, M.; Hardi, J.; Graybeal, J. Modeling community standards for metadata as templates makes data FAIR. *Sci. Data* **2022**, *9*, 696. [CrossRef] [PubMed]
4. Broekhuis, S.B.; Zaytsev, V. Improving JSON Schema Inference by Incorporating User Inputs. In Proceedings of the CEUR Workshop Proceedings, Namur, Belgium, 21–22 November 2024; Volume 3941, pp. 146–159.
5. Attouche, L.; Baazizi, M.A.; Colazzo, D.; Falleni, F.; Ghelli, G.; Landi, C.; Sartiani, C.; Scherzinger, S. A Tool for JSON Schema Witness Generation. In Proceedings of the 24th International Conference on Extending Database Technology (EDBT 2021), Nicosia, Cyprus, 23–26 March 2021; pp. 694–697. [CrossRef]
6. Chaerony Siffa, I.; Schäfer, J.; Becker, M.M. Adamant: A JSON schema-based metadata editor for research data management workflows. *F1000Research* **2022**, *11*, 475. [CrossRef] [PubMed]
7. Neubauer, F.; Bredl, P.; Xu, M.; Patel, K.; Pleiss, J.; Uekermann, B. MetaConfigurator: A User-Friendly Tool for Editing Structured Data Files. *Datenbank-Spektrum* **2024**, *24*, 161–169. [CrossRef]
8. Cano, M.A.; Tsueng, G.; Zhou, X.; Xin, J.; Hughes, L.D.; Mullen, J.L.; Su, A.I.; Wu, C. Schema Playground: A tool for authoring, extending, and using metadata schemas to improve FAIRness of biomedical data. *BMC Bioinform.* **2023**, *24*, 159. [CrossRef] [PubMed]
9. Plathe, N.; Becker, M.M.; Franke, S. pyJSON Schema Loader and JSON Editor: A tool for file-based metadata management. *SoftwareX* **2024**, *28*, 101945. [CrossRef]
10. Neubauer, F.; Uekermann, B.; Pleiss, J. AI-assisted JSON Schema Creation and Mapping. In Proceedings of the 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Grand Rapids, MI, USA, 5–10 October 2025; pp. 79–83. [CrossRef]
11. Gong, H.; He, J.; Zhang, X.; Duan, L.; Tian, Z.; Zhao, W.; Gong, F.; Liu, T.; Wang, Z.; Zhao, H.; et al. A repository for the publication and sharing of heterogeneous materials data. *Sci. Data* **2022**, *9*, 787. [CrossRef] [PubMed]
12. Yin, N.; Liang, J.; Guo, X.; Jiang, X.; He, J.; Zhang, X. Semi-automatic construction of heterogeneous data schema based on structure and context-aware recommendation. *Sci. Data* **2025**, *12*, 190. [CrossRef] [PubMed]
13. O'Connor, M.J.; Hardi, J.; Martínez-Romero, M.; Somasundaram, S.; Honick, B.; Fisher, S.A.; Pillai, A.; Musen, M.A. Ensuring adherence to standards in experiment-related metadata entered via spreadsheets. *Sci. Data* **2025**, *12*, 265. [CrossRef] [PubMed]
14. Sadruddin, S.; D'Souza, J.; Poupaki, E.; Watkins, A.; Babaei Giglou, H.; Rula, A.; Karasulu, B.; Auer, S.; Mackus, A.; Kessels, E. LLMs4SchemaDiscovery: A Human-in-the-Loop Workflow for Scientific Schema Mining with Large Language Models. In *Proceedings of the The Semantic Web: 22nd European Semantic Web Conference, ESWC 2025; Lecture Notes in Computer Science*; Springer: Cham, Switzerland, 2025; Volume 15719, pp. 244–261. [CrossRef]
15. Chen, W.; Zhang, X.; He, J.; Xu, C.; Wan, Y.; Gong, H.; Zhuang, Y. SContainer: A document data model for GUI-based schema building in the sharing of generic scientific research data. *Inf. Syst.* **2026**, *141*, 102744. [CrossRef]
16. Yang, B.; Vassilev-Galindo, V.; Llorca, J. Dataset used in the publication entitled “Application of machine learning to assess the influence of microstructure on twin nucleation in Mg alloys”. *npj Comput. Mater.* **2024**, *10*, 26. [CrossRef]
17. Therrien, F.; Abou Haibeh, J.; Sharma, D.; Hendley, R.; Mungai, L.W.; Sun, S.; Tchagang, A.; Su, J.; Huberman, S.; Bengio, Y.; et al. OBELiX: A curated dataset of crystal structures and experimentally measured ionic conductivities for lithium solid-state electrolytes. *Digit. Discov.* **2026**, *5*, 910–918. [CrossRef]
18. Yeh, I.C. Concrete Compressive Strength [Dataset]. *Cem. Concr. Res.* **1998**, *28*, 1797–1808. [CrossRef]
19. Vito, S. Air Quality [Dataset]. *Sens. Actuators B Chem.* **2008**, *129*, 750–757. [CrossRef]

20. United States Environmental Protection Agency. RSEI Data Dictionary: Chemical Data. 2026. Available online: <https://www.epa.gov/rsei/rsei-data-dictionary-chemical-data> (accessed on 14 June 2026).
21. Dickson, E.; Grambsch, P.; Fleming, T.; Fisher, L.; Langworthy, A. Cirrhosis Patient Survival Prediction [Dataset]. *Hepatology* **1989**, *10*, 1–7. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.