Scientific Research Publishing

Journal of Computer and Communications, 2026, 14(11), 22-53
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

A Single-Logic and Minimal-Data-Unit Framework for Human-Computer Interaction in Data Analysis and Modeling

Mingyang Wei

Great Neck South High School, Lake Success, USA
Email: evanwei0330@gmail.com

How to cite this paper: Wei, M.Y. (2026) A Single-Logic and Minimal-Data-Unit Framework for Human-Computer Interaction in Data Analysis and Modeling. *Journal of Computer and Communications*, **14**, 22-53.
https://doi.org/10.4236/jcc.2026.1411002

Received: September 22, 2026
Accepted: November 2, 2026
Published: November 5, 2026

Copyright © 2026 by author(s) and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/

Open Access

Abstract

Contemporary data analysis systems combine large datasets, visual dashboards, machine learning models, and automated decision-support tools, but their analytical processes can be difficult to follow when operations are distributed across complex interfaces, hidden model structures, and multistage workflows. This paper proposes and partially instantiates a Single-Logic and Minimal-Data-Unit (SL-MDU) framework for human-computer interaction in data analysis and modeling. A Single Logic (SL) is a registered analytical operation that satisfies four testable atomicity criteria—declared input, one rule with exposed parameters, one inspectable output, and no intermediate result the task requires the user to check; a Minimal Data Unit (MDU) is the smallest data entity that is meaningful for the task at a declared granularity level. Relative to subset-flow, modular-composition, and provenance systems, SL-MDU contributes a prescriptive task-level contract: each result is accountable to an explicit (MDU, SL, parameters, output) record, rule outputs are kept distinct from human overrides, and a granularity policy decides which units are exposed. Following a design science research approach, the framework was abstracted from a working keyword-driven web implementation. We report code-level evidence of the implemented dispatch mechanism and a technical evaluation in a browser: on an illustrative 142-student × 4-indicator workload, the unmodified dispatcher executed 710 per-unit operations in a median of 30.5 ms, or 0.27 ms when the same rules were applied at cohort granularity, and per-unit execution scaled linearly to 2.7 s for 14,200 students. A heuristic evaluation by three evaluators (two developers and one professional tester) recorded 41 usability problems (mean severity 2.3 on a 0 - 4 scale), most severely the lack of undo and weak feedback on which unit an operation targets, together with 18 positive findings. Effects on user understanding, cognitive

load, and task performance are stated as a falsifiable hypothesis for a subsequent controlled study rather than claimed.

Keywords

Human-Computer Interaction, Visual Analytics, Visualization Provenance, Explainable AI, Human-in-the-Loop Machine Learning, Design Science Research, Interaction Traceability

1. Introduction

1.1. Background of Human-Computer Interaction in Data Analysis

Data analysis increasingly involves complex interactions among users, data, analytical models, and computational tools. Users work with dashboards, machine learning pipelines, automated recommendation systems, and explainable artificial intelligence (XAI) tools. A technically correct output is not sufficient if users cannot understand how it was produced or identify grounds for questioning it. Ali *et al.* [1] distinguish data explainability, model explainability, post-hoc explanation, and explanation assessment. This broader perspective matters because even an explainable model can be difficult to use when its surrounding interaction is poorly designed.

Visual analytics research places the human within the analytical process. Endert *et al.* [2] argue that users are part of the analytical loop rather than occasional sources of input. Sacha *et al.* [3] likewise show how interactive visualization can connect human reasoning with machine learning. However, interfaces organized around entire workflows or models may compress the connection between individual data units and analytical steps, making that connection difficult for users to inspect.

1.2. Motivation for Single Logic and Minimal Data Unit

This gap between system capability and users' ability to interpret its operation motivates the SL-MDU framework. Analytical interfaces may require users to issue many commands, navigate dashboards, or configure complex model parameters. Such interfaces can be efficient for experts, but their transparency may be limited when several operations are presented as a single action. The proposed framework makes two units explicit: Single Logic (SL) and Minimal Data Unit (MDU).

An SL is a single, repeatable analytical operation, such as filtering, comparing, thresholding, classifying, visualizing, or explaining, that meets explicit atomicity criteria (Section 3.2). Amar *et al.* [4] identify low-level analytic activities through which users make sense of information visualizations, providing a foundation for this decomposition. An MDU is the smallest meaningful data entity for a task at a declared level of granularity, such as a table cell, record, feature, time-series

point, cohort aggregate, or model output. Rehrl and Gröchenig [5] illustrate the value of data-centered activity analysis, while Yu and Silva [6] demonstrate subset-based dataflow for interactive tabular analysis in VisFlow. Section 3.6 states precisely what SL-MDU adds to these and to provenance systems.

1.3. Research Significance and Objectives

The SL-MDU framework shifts attention from analytical outputs alone to the interactions through which those outputs are produced. It asks whether a user can connect a meaningful data unit to an explicit operation and inspect the resulting change. This connection matters in domains where analytical results inform consequential decisions, including healthcare, finance, education, urban planning, and organizational management.

The paper has five objectives:

O1. Define SL and MDU for human-computer interaction (HCI) in data analysis, with operational criteria for SL atomicity and a policy for choosing MDU granularity.

O2. Differentiate the resulting constructs from subset-flow, modular-composition, and provenance systems by an explicit construct mapping.

O3. Derive design principles and a five-layer reference architecture for SL-MDU interaction, including the access-control requirements of sensitive data.

O4. Instantiate the dispatch mechanism in a working reference implementation and document it with inspectable code excerpts and supplementary source files.

O5. Evaluate the instantiation: measure its performance on the illustrative workload, report a heuristic evaluation by three evaluators, and state a falsifiable hypothesis for subsequent user studies.

Dudley and Kristensson [7] emphasize that interaction with a machine learning model is as important as the underlying algorithm. Mosqueira-Rey *et al.* [8] identify inconsistent uses of the term human-in-the-loop (HITL), underscoring the need to specify how users participate. Gómez-Carmona *et al.* [9] treat users as active participants in interactive machine learning rather than passive recipients of results. SL-MDU makes this participation explicit by identifying the data unit and operation involved in each interaction.

1.4. Research Approach

This study follows a design science research (DSR) approach. Following Hevner *et al.* [10] and the process model of Peffers *et al.* [11], it centers on a purposeful artifact: a reference implementation whose construction and demonstration form part of the inquiry. The framework provides an artifact-grounded design theory, using the vocabulary of Gregor and Hevner [12], expressed through constructs (SL and MDU), a model (the five-layer architecture in Section 4), a method (the interaction cycle and granularity policy in Section 3), and implemented or illustrative instantiations (Sections 4 and 5). The artifact was developed first as a web-based environment for interactive analysis and interface construction; the frame-

work was then abstracted from that implementation and refined against the literature. Evaluation is organized using the Framework for Evaluation in Design Science (FEDS) [13]. This paper reports the first, artificial evaluation episode—technical performance measurement and a heuristic evaluation by three evaluators (Section 6)—and specifies the naturalistic episodes that follow.

1.5. Scope of Implementation Claims

To avoid restating caveats throughout, the status of every claim in this paper is fixed here, and later sections refer back to this statement. Three statuses are used.

Implemented. Keyword-based registration of operations; the common dispatcher that resolves a keyword to an ordered module chain and one entry function, with module and invocation caching; the user-editable bridge table and export registry; element-, cell-, row-, column-, and card-level selection; the three-tab element-empowerment window; timestamped current-state persistence per window; and a per-user file synchronization of seven configuration keys (Sections 4.2 - 4.5; Supplementary Material S1-S2).

Measured. Dispatch latency with and without caching, the cost of the illustrative 142 × 4 workload at unit and cohort granularity, its scaling to 14,200 students, table rendering, and trace size (Section 6.2; Supplementary Material S3). The scenario-specific operations used in these measurements were written for this revision and registered in the unmodified dispatcher; they are not a deployed student application.

Designed or illustrative. An append-only analytical event history, version comparison, the adaptive granularity policy (Section 3.4), multi-user access control (Section 4.7), the student risk-warning dashboard (Section 5.3), and all effects on users. No real student data were used, and no claim about user benefit is made before the studies specified in Section 6.4.

2. Literature Review

2.1. Interactive Data Analysis and Visual Analytics

Visual analytics provides a strong theoretical foundation for the SL-MDU framework, as it views data analysis as a joint process of computation and human reasoning. Andrienko *et al.* [14] argue that visual analytics can be viewed as model building, in which users construct mental and/or formal representations of reality by interacting with data. Sacha *et al.* [15] extend this with a knowledge-generation model in which insight arises from a constant back-and-forth between data, visualization, user understanding, and system interaction, consistent with Endert *et al.*'s [2] argument that the human belongs within the analysis. These accounts motivate a more explicit representation of the link between a selected data entity, the operation applied to it, and the resulting analytical state. SL-MDU complements them by organizing user actions around task-relevant data units and identifiable operations; it does not replace existing models of analytical reasoning.

2.2. Low-Level Analytical Operations and the Term Single Logic

Research on low-level analytical operations provides a foundation for SL. Amar *et al.* [4] identify ten activities in information visualization: retrieve value, filter, compute derived value, find extremum, sort, determine range, characterize distribution, find anomalies, cluster, and correlate. Brehmer and Munzner [16] organize such tasks into a multi-level typology that separates why a task is performed from how and on what. Gotz and Zhou [17] define an action tier of semantically meaningful, atomic analytic steps between low-level interface events and higher-level sub-tasks, and use it to capture insight provenance. Data-transformation tools such as Wrangler [18] represent each transformation as an editable operator in a script.

The literature therefore already offers several near-synonyms—analytic task, action, operation, and operator. We introduce a separate term because none of them carries the property SL-MDU needs. Analytic task and action are descriptive categories used to classify or log what users do; operator denotes an implementation unit whose granularity is left to the tool author. Single Logic denotes a prescriptive contract that an operation must satisfy before it is exposed to users: one declared input, one rule with visible parameters, and one inspectable output, tested by the criteria in Section 3.2. Every SL is an action in the sense of Gotz and Zhou, but not every action or operator is an SL.

The inspectable-operation idea also draws on interactive machine learning. Sarkar's technical report [19] argues for analytical modelling interfaces that support judgment, revision, and interpretation. Peer-reviewed work supports the same requirements: Kulesza *et al.* [20] show that explanations should be sound, complete, and actionable so that users can correct a learner, and Amershi *et al.* [21] show that users need to see and steer the effect of each input on the system's behavior. Accordingly, an SL should expose what data it acts on, what rule and parameters it applies, and what result it produces.

2.3. Human-in-the-Loop and Interactive Machine Learning

Mosqueira-Rey *et al.* [8] distinguish active learning, interactive machine learning, and machine teaching, and note that the human role is often discussed generically rather than as a concrete operating procedure—labeling, error correction, model inspection, and feature selection are all lumped together as "human input". Dudley and Kristensson [7] show that the structure and behavior of the interface are as crucial as algorithmic performance. Gómez-Carmona *et al.* [9] reconceptualize the user as an active participant. SL-MDU makes this participation concrete: the user controls the pairing of data units with operations, rather than merely reacting after the system runs.

2.4. Explainable AI and Visual Explainability

SL-MDU is closely related to explainable AI because one of its goals is interpretability. Ali *et al.* [1] stress that beyond accuracy, data, model, and post-hoc explain-

ability and explanation evaluation are all needed for trust. Alicioglu and Sun [22] show that visual analytics can explain complex model behavior, and Yuan *et al.* [23] show that visual-analytics techniques apply before, during, and after modeling—to features, records, training examples, predictions, error cases, and feature contributions. SL-MDU adds interaction-level traceability: which data unit was operated on, which operation and parameters were applied, and how the result changed. It does not solve black-box opacity in deep learning.

2.5. Modular Dataflow and Unit-Based Systems

Modular and dataflow systems provide practical precedents. VisComposer supports the composition of visualization pipelines from programmable modules [24], while VisFlow provides a subset-flow model in which subsets of data items flow between filtering, selection, and visualization nodes, with brushing, linking, and embedded visualization [6]. Both make analytical activity visible as a network of operations. Progressive visualization is also relevant because fine-grained interaction can be expensive: partial or approximate results can support reasoning while computation continues [25].

2.6. Visualization and Data Provenance

The provenance layer of SL-MDU (Section 4.4) overlaps substantially with two provenance literatures, which this section makes explicit. In visualization, VisTrails [26] captures the history of a visualization workflow as a version tree of changes to its pipeline, so that any earlier specification and its parameters can be recovered and compared; Freire *et al.* [27] survey such workflow provenance. Graphical histories [28] record the sequence of visualization states for revisiting and communication, and insight provenance [17] records the analytic actions that led to a finding. Ragan *et al.* [29] organize these approaches by what is recorded (data, visualization, interaction, insight, rationale) and why (recall, replication, action recovery, collaboration, presentation, meta-analysis), and Xu *et al.* [30] survey the analysis of interaction logs and provenance.

In databases, Buneman *et al.* [31] distinguish why-provenance, the set of source tuples that justify an output, from where-provenance, the locations from which an output value was copied; Cheney *et al.* [32] add how-provenance, which records how inputs were combined. The W3C PROV data model [33] expresses provenance as entities, the activities that use and generate them, and the agents responsible. These notions map directly onto SL-MDU: an MDU is an entity, an SL application is an activity, the user is an agent, and the trace of a classification (steps t1 - t4 in Section 5.3) is the why-provenance of its label. SL-MDU does not contribute new provenance semantics or capture techniques; it adopts these notions and specifies them at the grain of the user's analytical task.

2.7. Positioning within Design Science Research

The contribution is best understood from a design science perspective. March and

Smith [34] distinguish construction and evaluation, as well as constructs, models, methods, and instantiations. SL-MDU addresses these artifact types through its definitions, architecture, interaction cycle and granularity policy, and reference implementation. Hevner [35] describes three interacting research cycles: relevance connects the research to its problem environment, rigor grounds it in existing knowledge, and design connects artifact construction with evaluation. Gregor and Hevner [12] guide the presentation of the artifact-grounded contribution, and FEDS [13] provides the evaluation vocabulary used in Section 6.

3. Conceptual Framework

3.1. Single Logic

An SL is a repeatable analytical operation that is exposed to the user as one step. Examples include filtering records, comparing a value with a baseline, thresholding, grouping, summarizing a distribution, transforming a variable, classifying a case, and inspecting a model output. Its purpose is to make each analytical step identifiable and inspectable, not to imply that complex analysis has become intrinsically simple. The concept extends the low-level activities of Amar *et al.* [4] from a task taxonomy to an interaction contract with three elements: a declared input, a declared rule with its parameters, and an inspectable output [7] [20]. In the reference implementation, each registered operation has a named entry point (Section 4.5). Registration is an execution boundary; whether a registered operation is also an analytical SL is decided by the criteria below.

3.2. Operationalizing Atomicity

Defining an SL as "atomic at the level of the user's task" is circular unless atomicity can be tested. We therefore make the task explicit and define atomicity relative to it. A task specification T names the user role, the question to be answered, and the decisions the user is expected to verify or revise. An operation O is an SL for T if and only if it satisfies the four criteria in **Table 1**. Criteria C1 - C3 are checkable from the operation's registration and signature; C4 is checkable against T. Two tests follow from them: the split test (an operation that fails C2 or C4 must be decomposed) and the merge test (an operation whose output is never meaningful to the user in T should be absorbed into the SL that consumes it). Because each criterion yields a yes/no answer for a given T, a claim that an operation is an SL can be falsified, and two designers applying the criteria to the same T should reach the same decision.

Three negative examples show how the criteria reject operations. 1) Too coarse: a single "compute risk" operation that filters attendance, compares the midterm score with the cohort median, thresholds assignment completion, and classifies the result fails C2, because three cut-offs are hidden, and fails C4, because the tutoring task in Section 5.3 requires each flag to be inspected and possibly excused. It must be split into steps t1 - t4. 2) Too fine: "parse the attendance string into a number" satisfies C1 - C3 but fails the merge test, since its output is never mean-

ingful to a tutor; it belongs inside the attendance filter. 3) Registered but not an SL: the implemented keyword 导出 HTML (export HTML) is a valid execution unit of the dispatcher but has no MDU input and no analytical rule, so it is an interface command rather than an SL. Conversely, the filter 考勤筛选 registered in Listing 2 (Section 4.5) satisfies all four criteria for the tutoring task: its input is one attendance MDU, its rule is a comparison with an exposed operator and threshold, its output is one Boolean flag that becomes an input of the classification SL, and it computes nothing the tutor must inspect separately.

Table 1. Atomicity criteria for a Single Logic relative to a task specification T.

Criterion	Definition	Test and failure example
C1 Declared input	The operation consumes MDUs of one declared kind and level (e.g., one indicator of one student, or one indicator column of the cohort), identified before execution.	Can every input be named before the operation runs? Fails if the operation fetches further data internally, e.g., a "risk" operation that silently reads login logs.
C2 One rule, exposed parameters	The output is determined by one rule whose parameters are all visible and editable at the interface.	Is there an internal choice that changes the output and that a user in T could want to vary independently? If so, split. Fails if a tolerance or cut-off is hard-coded.
C3 One inspectable output	The operation produces one output whose relation to its inputs can be stated as (input, rule, parameters) → output, and which can serve as an MDU downstream.	Can the result be stated in that form? Fails if the operation returns several unrelated results or only mutates hidden state.
C4 No required intermediate	T does not require the user to inspect, verify, or revise any intermediate result produced inside the operation.	Does T ask the user to check a value computed inside O? If so, split. Merge test: if O's output is never meaningful to the user in T, absorb it.

Atomicity is therefore task-relative but not arbitrary. The same computation can be an SL for one task and not for another—a cohort-level classification is an SL for a program manager who reviews group sizes, but not for a tutor who must verify each flag. The criteria make that dependence explicit instead of leaving it to the designer's judgment.

3.3. Minimal Data Unit

An MDU is the smallest amount of data that is meaningful to select, process, visualize, interpret, or explain in a specific task. "Meaningful" implies that an MDU is not necessarily the smallest physical fragment but the smallest entity that still carries analytical value for the task [5]. In a spreadsheet an MDU may be a cell, row, column, or variable; in text, a phrase, sentence, or topic label; in a time series, a time-stamped value, an interval, or a trend segment; in model interpretation, a feature contribution, a prediction case, or an error instance [23]. Data-centered activity analysis [5] and VisFlow's subset-flow model [6], in which users interact with selected data units rather than the dataset as a whole, support this idea. Because the meaningful unit changes with the question, every MDU is defined at a level of a granularity hierarchy, and the choice of level is governed by the policy in Section 3.4.

3.4. Granularity Policy: Choosing the MDU Level

Cognitive load is the condition under which the framework is usable at all: a dataset may contain millions of potential MDUs, and exposing them indiscriminately would overwhelm working memory [36]. SL-MDU therefore specifies a granularity policy. It builds on the principle of overview first, zoom and filter, then details on demand [37] and on hierarchical aggregation [38], and adds a rule for when the system should recommend descending a level. The policy is specified here as a design contribution; it is not yet implemented (Section 1.5).

Levels. Each dataset is organized into an ordered hierarchy of MDU levels. For tabular data about people, the default hierarchy is L0 cohort aggregate (e.g., the distribution of attendance), L1 segment (e.g., students with two or more flags), L2 entity (one student's profile), L3 indicator value (one indicator of one student), and L4 source evidence (raw attendance events or one tutor note). SLs are registered with the level at which they consume and produce MDUs, which C1 already requires.

Default and budget. The interface opens at the coarsest level that answers the task question, and the number of simultaneously exposed units is bounded by a configurable budget B (for example, 20 rows). When a descent would exceed B, the units are ranked by a task-specific relevance score, such as the number of active flags and the margin to the nearest threshold, and the remainder is shown as one aggregate MDU ("n further students").

Descent triggers. The system recommends moving from level k to k + 1 only when (T1) an SL at level k produces an exception—a threshold crossing, rule firing, or outlier—in which case only the affected units are expanded; (T2) the user questions or overrides an output, in which case only the why-provenance of that output, *i.e.*, the MDUs that contributed to it, is expanded; or (T3) the user explicitly drills down. The view ascends when the review of the expanded units is closed.

Control and accountability. Recommendations never prevent the user from choosing another level. Every change of level is recorded as a provenance event, so that the trace shows the level at which a decision was taken, and levels are also the unit of access control (Section 4.7). Table 2 summarizes the policy; Section 5.3.4 applies it to the student scenario, and Section 6.2 reports the computational cost of executing SLs at unit rather than cohort level.

Table 2. Granularity policy: MDU levels, default exposure, and descent triggers (design specification).

Level	MDU (student scenario)	Exposed when
L0 Cohort	Distribution of each indicator; counts per rule label	Default for every task that starts from the whole cohort.
L1 Segment	Students with ≥2 active flags; a tutor group	T1: a cohort-level SL fires; the segment is ranked and capped at budget B.
L2 Entity	One student's four indicators and label	T1 for a ranked student in the segment, or T3 (explicit selection).
L3 Indicator	One indicator value with its rule, parameters, and flag (t1 - t3)	T2: the user questions the label; only its contributing flags are expanded.
L4 Evidence	Raw attendance events; one tutor note	T2 during an override; subject to the stricter access rule for notes (Section 4.7).

3.5. Logical Structure of SL-MDU Interaction

The interaction cycle is straightforward: the user selects an MDU at the current level, applies an SL, inspects the output, and then accepts or revises the result. Figure 1 connects the user, interface, data unit, operation, output, and feedback. Each cycle exposes the link between a selected unit and the operation applied to it, allowing users to inspect how an outcome was produced. This approach places users within the analytical process [2], supports feedback and refinement in human-centered machine learning [3], and makes their role in HITL systems explicit [8].

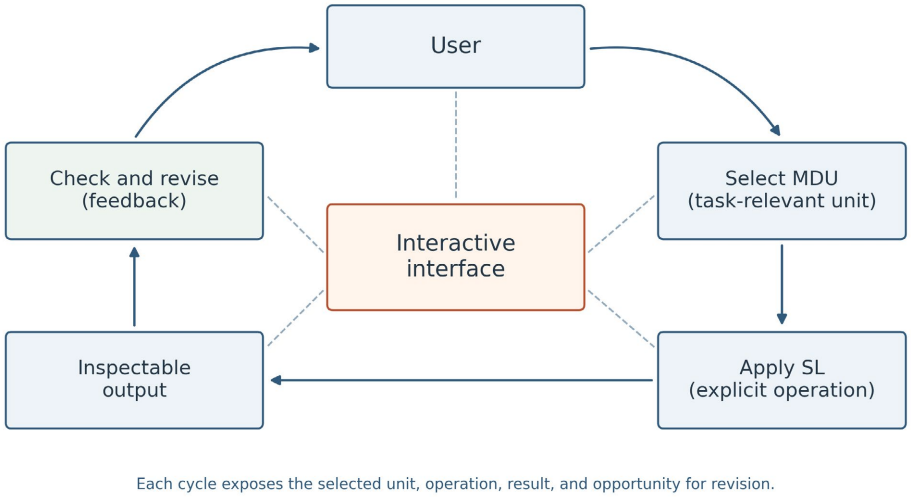


Figure 1. Conceptual SL-MDU interaction cycle. Each cycle pairs one selected MDU with one SL and exposes the result for revision.

3.6. What SL-MDU Adds to Prior Systems

Table 3 maps SL-MDU constructs to the closest constructs of subset-flow, modular-composition, transformation-script, and provenance systems and states, for each, what is shared and what differs.

The residual gap is not any single mechanism: dataflow execution, modular composition, and provenance capture all exist. It is the absence of a design-level contract that binds them at the grain of the user's decision. Specifically, SL-MDU contributes (i) a testable definition of which operations may be exposed as steps (C1 - C4), where prior systems leave module granularity to the tool author; (ii) a unit of accountability—the (MDU, SL, parameters, output) record—that can be read as the why-provenance of a decision by its human reviewer; (iii) the separation of rule output from human override as distinct provenance activities, which decision-support accountability requires [1]; and (iv) a granularity policy that links the level at which units are exposed to cognitive load and access control. Claims (i) and (iii) are illustrated by the implementation and scenario; claims (ii) and (iv) are design specifications whose value is to be tested by the hypothesis in Section 6.4.

Table 3. Construct mapping between SL-MDU and prior systems.

Prior work	Central construct	SL-MDU counterpart	Shared/what SL-MDU adds
VisFlow subset flow [6]	Subsets of data items flowing between filter, selection, and visualization nodes	Subset ↔ MDU; node ↔ SL	Shared: selection-centered interaction on tabular data. Adds: the unit is defined by task meaning at a declared level (Section 3.4), not by whatever subset flows; each result carries an (MDU, SL, parameters, output) record rather than only the current subset.
VisComposer [24]	Programmable modules composed into visualization pipelines	Module ↔ SL library row	Shared: modular, reusable operations. Adds: module boundaries are not SL boundaries; criteria C1 - C4 decide which operations are exposed to users, and registration binds one keyword to one entry function.
Wrangler [18]	Transformations as an editable script of operators	Operator sequence ↔ SL workflow	Shared: each step is inspectable, editable, and re-executable. Adds: extends beyond data transformation to comparison, classification, explanation, and human review, with a per-unit trace for decisions.
VisTrails [26] [27]; **graphical histories** [28]	Version tree of pipeline changes; history of visualization states	Provenance layer: SL application and revision records	Shared: recording operations and parameters to revisit earlier states. Adds: the record unit is the task-level (MDU, SL) pair on data values, not the pipeline specification, and a rule output is stored separately from a human override. The current implementation lacks the versioned history that VisTrails provides.
Insight provenance [17]; **provenance types** [29] [30]	Semantic, atomic analytic actions captured from user activity	SL ≈ action	Shared: an atomic, meaningful analytic step. Adds: actions are identified post hoc from logs; an SL is a prescriptive contract that an operation must meet before it is offered.
Why/where provenance [31] [32]; **PROV** [33]	Witnesses and source locations of outputs; entities, activities, agents	MDU ↔ entity; SL ↔ activity; user ↔ agent	Shared: which inputs justify an output. Adds: applies these notions at the interface for a human reviewer and models an override as a separate activity attributed to an agent. No new provenance semantics are claimed.

4. System Architecture and Design

This section maps the SL-MDU constructs to the reference implementation. The implementation is a keyword-driven web environment that loads ES modules directly at runtime without a separate bundling step, and every registered user operation passes through one common dispatcher. Implementation status follows Section 1.5.

4.1. Design Principles of SL-MDU Interaction

Analytical capability is useful only when users can understand and control the relevant operations. Four design principles guide SL-MDU interaction.

Modularity. Each SL represents one reusable operation. In the reference implementation, a keyword identifies an entry function and an ordered chain of modules loaded on demand, which makes the operation's execution boundary explicit; the criteria in Section 3.2 determine whether that boundary is also an analytical one [7].

Transparency. The selected input, operation rule and parameters, and output

should be inspectable, rather than only the final result [1]. The implementation names operations and declares callable exports in registries, which makes execution identifiable but is not by itself an analytical explanation.

User control. Users select MDUs, apply operations, and accept, reject, or revise results [9]. The operation library and bridge table are user-editable (Section 4.5). Changing an existing operation's parameters and writing a new operation are distinct activities; the latter requires implementation knowledge.

Progressive feedback. Fine-grained interaction should provide timely feedback, with partial or incremental results where appropriate [25]. The implementation reduces avoidable loading work through on-demand module loading, deduplication, and lazy panel content; Section 6.2 measures its effect.

4.2. The Five-Layer Reference Architecture

The framework organizes these principles into five functional layers. Table 4 pairs each layer's intended responsibility with the corresponding mechanisms in the reference implementation. The distinction is particularly important for provenance: timestamped state supports recovery, but is not a complete history of analytical events.

Table 4. Five-layer SL-MDU architecture and its current implementation.

Layer	Design responsibility	Mechanisms and current scope
Data	Store raw data, metadata, and configuration used to address MDUs and operations; enforce access to MDUs.	Browser-local key-value store for seven configuration keys, mirrored to per-user backend files; a script registry declares callable exports. Single-tenant, without authentication or concurrency control (Section 4.7).
Operation	Manage the SL library, rules, parameters, and input-output relationships.	User-editable bridge table mapping each keyword to an ordered module chain and one entry function; registries and a common dispatcher (Listings 1 - 3).
Interaction	Support selection, execution, revision, and feedback at the current granularity level.	Keyword-bound UI controls, configurable multi-tab windows, Web Components, and a sidebar card builder. Granularity is chosen manually.
Provenance	Associate selected units, operations, parameters, outputs, overrides, and level changes in an inspectable history.	Timestamped per-window state, persistent and scratch stores, cross-window events, and an invocation cache. No append-only analytical history.
Output	Present visualizations, results, and explanation views.	Workspace canvas, interactive table engine, chart applications, and HTML export. No unified chronological explanation panel.

4.3. Interface Design and User Workflow

The interface comprises five functional elements: a data panel for selecting MDUs; an operation panel grouping SLs by purpose; a workflow canvas for arranging operations; an output panel for charts, tables, and model summaries; and an explanation/history panel for inspecting inputs, operations, results, and revisions. These are design roles; the current implementation provides selection controls, configurable operation pages, a workspace, tables, charts, and persisted window

state [6] [24]. Figure 2 shows the implemented execution path with the actual system elements—the control identifier, handler, bridge-table row, registry entry, dispatcher, and the three tabs of the element-empowerment window—each labeled with its source file and line. Figure 3 shows the running window at each stage of the cycle.

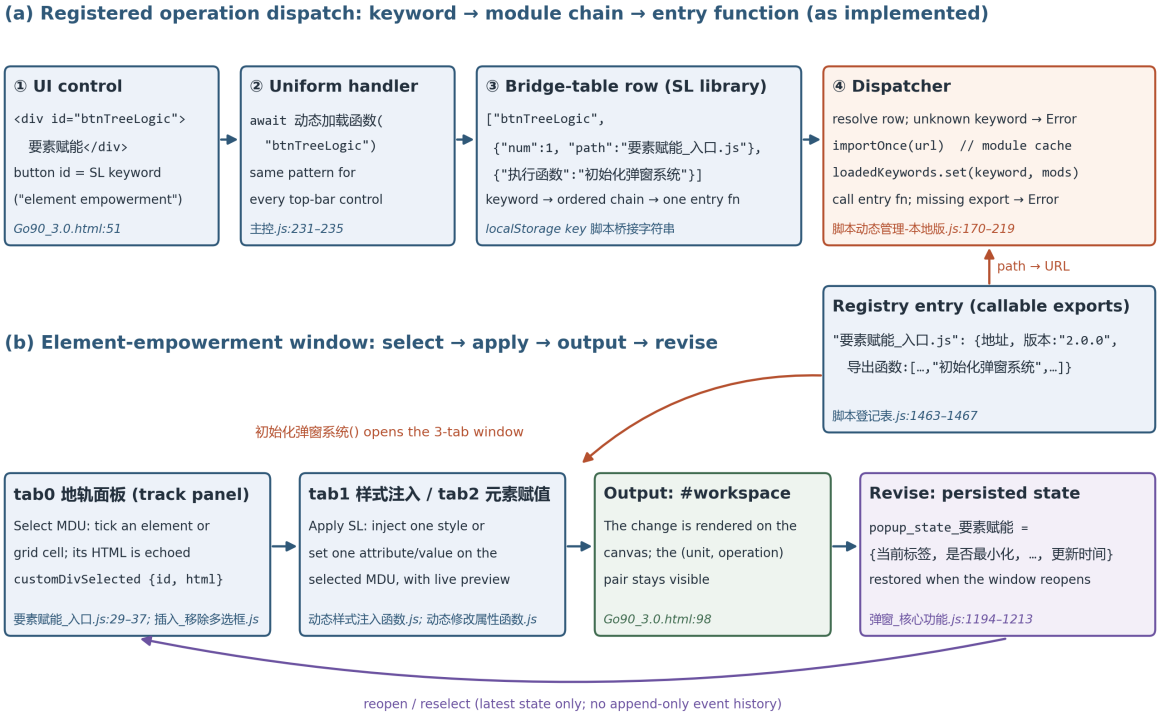


Figure 2. Implemented execution path. (a) A top-bar control whose identifier is the SL keyword is resolved through the bridge table and export registry by the common dispatcher. (b) The entry function opens the three-tab element-empowerment window that realizes select → apply → output → revise; revision restores the latest persisted state only. Chinese identifiers are the literal names in the source; English glosses are given in the boxes.

4.4. Explainability and Interaction Traceability

Explanation should be integrated into interaction. At the design level, the provenance layer records five kinds of event, expressed in PROV terms [33]: an MDU selection (entity used), an SL application with its parameters (activity), the output produced (entity generated), a human revision or override (a separate activity attributed to the user as agent), and a change of granularity level. The why-provenance [31] of any output is then the set of MDUs reachable backwards through these records, which is exactly what a reviewer needs to answer "why was this student flagged?". The reference implementation records timestamped per-window state, separate persistent and scratch stores, and operation identifiers; these support inspection of the current state and revision in progress but not an append-only record of every event. Section 6.2 measures the storage cost that such a record would have for the illustrative workload. SL-MDU addresses traceability at the

interaction level; historical replay and version comparison require provenance mechanisms of the kind VisTrails provides [23] [26].

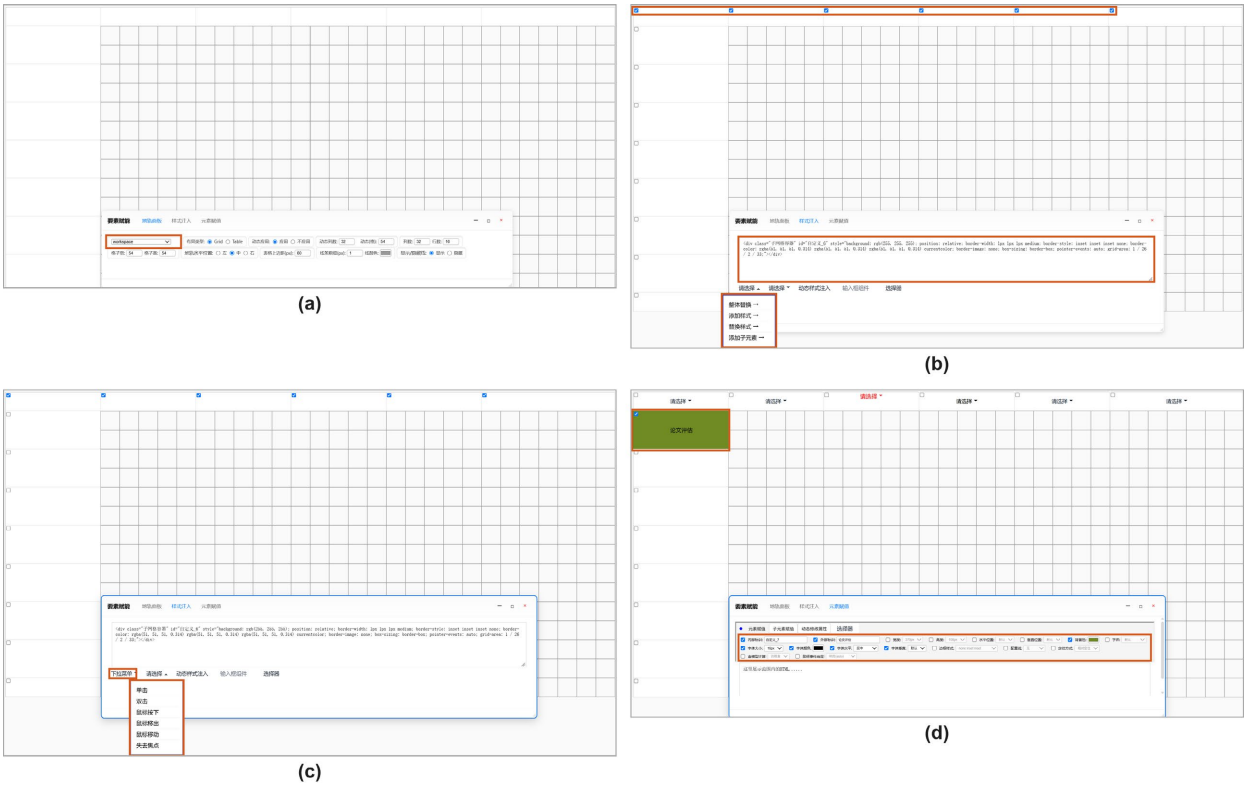


Figure 3. Screenshots of the implemented element-empowerment cycle; outlined regions mark the selected unit, the operation, and the result. (a) Track panel (地轨面板): a target container is chosen (workspace) and a grid layout is created on it. (b) Style injection (样式注入): the checkboxes of the selected elements are ticked, the HTML of the target MDU is echoed in the preview, and the operation menu offers whole replacement, add style, replace style, and add child element. (c) The same tab with the event menu (click, double-click, mouse down, mouse out, mouse move, blur) for binding an interaction to the selected unit. (d) Element assignment (元素赋值): per-property switches set the identifier, label, font, and background of one selected cell, and the result appears immediately on the canvas (green cell).

4.5. Reference Implementation

The reference implementation uses a common dispatcher. Given a keyword and arguments, it reads the bridge table, resolves the keyword to an ordered chain of modules and a single entry function, loads the modules through a module cache, stores the loaded chain in an invocation cache keyed by the keyword, and invokes the entry function. An undefined keyword or a missing export raises an error, so every executable operation is registered and named. Listing 1 gives the schema of the bridge table and the export registry; Listing 2 shows one SL of the student scenario registered and invoked through the dispatcher; Listing 3 abridges the dispatcher itself. MDUs can be addressed as a workspace element, a table cell, row or column, a card, or one style or attribute of an element, and the three tabs of the element-empowerment window—target selection, style operations, and attribute/value operations—are a concrete instance of selection, operation, and visual revision (**Figure 3**).

Listing 1. Schema of the bridge table (SL library) and the export registry, with rows from the implementation.

```
BridgeTable := [ Row, ... ]              // localStorage key 脚本桥接字符串 ("script bridge string")
Row         := [ keyword,                // SL name; also the id of the UI control
                 { num: int, path: string }, ...,   // module chain, executed in ascending num
                 { 执行函数: string } ]  // "execute function": the single entry function
Registry    := { filename: { 地址: URL, 版本: semver, 导出函数: [string, ...] } }
                                         // address, version, exported functions

// rows from 基础结构/脚本/动态导入路径.json (seed of the bridge table)
["btnTreeLogic", {"num":1, "path":"要素赋能_入口.js"}, {"执行函数":"初始化弹窗系统"}]
["样式注入",      {"num":1, "path":"动态样式注入函数.js"}, {"执行函数":"动态样式注入函数"}]
// registry entry, 基础结构/脚本/脚本登记表.js:1463-1467 (abridged)
"要素赋能_入口.js": { 地址: "https://anon.example/脚本动态入口/要素赋能_入口.js",
                      版本: "2.0.0", 导出函数: [..., "初始化弹窗系统", ...] }
```

Listing 2. An SL of the student scenario (t1): registration, entry function, and one invocation with its returned record.

```
// bridge-table row: keyword 考勤筛选 ("attendance filter") -> entry function 阈值筛选 ("threshold filter")
["考勤筛选", {"num":1, "path":"../../sl/筛选.js"}, {"执行函数":"阈值筛选"}]

// sl/筛选.js
export function 阈值筛选(mdu, params) {
  const { op = "<", threshold } = params;
  const output = op === "<" ? mdu.value < threshold : mdu.value > threshold;
  return { sl: "阈值筛选", mdu: { id: mdu.id, indicator: mdu.indicator, value: mdu.value },
           params: { op, threshold }, output };
}

// invocation through the unmodified dispatcher
await 动态加载函数("考勤筛选", { id: "S-017", indicator: "attendance", value: 68 },
                   { op: "<", threshold: 75 });
// -> { sl: "阈值筛选", mdu: { id: "S-017", indicator: "attendance", value: 68 },
//      params: { op: "<", threshold: 75 }, output: true }
```

Listing 3. Core of the dispatcher, abridged from 基础结构/脚本/脚本动态管理－本地版.js:170–219.

```
export async function 动态加载函数(keyword, ...args) {          // "dynamic load function"
  const table = JSON.parse(localStorage.getItem("脚本桥接字符串") || "[]");
  const row = table.find((r) => Array.isArray(r) && r[0] === keyword);
  if (!row) throw new Error(`关键字「${keyword}」未配置`);        // unregistered keyword
  const execFn = row.find((o) => o?.["执行函数"])?.["执行函数"];
  let mods = loadedKeywords.get(keyword)?.modules;              // invocation cache
  if (!mods) {
    const scripts = row.filter((o) => o && o.path).sort((a, b) => (a.num ?? 0) - (b.num ?? 0));
    mods = await loadMods(scripts);                             // importOnce(url): module cache
    loadedKeywords.set(keyword, { modules: mods });
  }
  if (execFn) {
    const fn = mods.at(-1)?.[execFn];
    if (typeof fn !== "function") throw new Error(`模块未导出函数「${execFn}」`); // missing export
    return await Promise.resolve(fn(...adaptArgs(execFn, args)));
  }
  return { status: "success", modules: mods };
}
```

The structure supports the SL contract of Section 3.1, but registration alone does not guarantee analytical correctness or atomicity; those are established by the operation's design and the criteria in Section 3.2. The complete construct-to-code mapping, with files, functions, and line numbers, is provided as Supplementary Material S1, and the cited source files as S2. The scenario SLs in Listing 2 and the benchmark harness of Section 6.2, including the verbatim copy of the dispatcher and its checksum, are provided as S3.

4.6. Data Management and Performance

Fine-grained interaction creates technical demands. Treating each cell, row, feature, time point, or prediction case as a selectable MDU increases the burden on indexing, memory, rendering, navigation, and history storage. A scalable design requires caching, indexing, and aggregation so that users can inspect relevant units without the system executing or exposing every unit at once. Structured dataflow can reduce manual repetition [6], and progressive visualization can provide useful intermediate results [25]. The implementation provides module and invocation caching and on-demand loading. Section 6.2 quantifies their effect and identifies the dominant remaining cost: the dispatcher re-reads and re-parses the bridge table on every call.

4.7. Multi-User Use, Concurrency, and Access Control

The data layer of the current implementation is not suitable for sensitive, multi-user data, and we state this technically rather than only ethically. The user identifier is a value stored in browser-local storage and used as a path prefix for backend files; it is a namespace, not an authenticated identity. Synchronization uploads each of the seven configuration keys as a whole file, so concurrent edits from two sessions resolve as last-writer-wins without conflict detection. Browser-local storage is readable by any script of the same origin and is not encrypted, and there is no authorization check when an operation reads data. The implementation is therefore single-tenant, and the student scenario in Section 5.3 must not be deployed on it.

SL-MDU nevertheless provides natural enforcement points, and a deployment handling student data would require the following controls. 1) Authenticated identity, for example institutional single sign-on, replaces the client-side identifier. 2) Authorization is enforced server-side at the level of the MDU and the SL: every read of an MDU and every SL invocation is checked against a role policy, because the client-side dispatcher can be modified by its user. 3) Access follows the granularity levels of **Table 2**: tutors may reach L2-L3 only for their assigned students, program managers are limited to L0-L1 with suppression of segments smaller than a minimum size, and L4 tutor notes form a separately permissioned class that is never cached in browser-local storage. 4) Provenance is append-only and attributes every event, including overrides, to an authenticated agent, which is what makes an override accountable. 5) Concurrent edits to shared configuration use optimistic versioning per record instead of whole-file replacement. 6) Data minimization and retention limits apply to indicators and notes. These controls are design requirements (Section 1.5); they complement, rather than replace, the ethical boundary in Section 5.3.6.

5. Applications and Illustrative Scenarios

This section presents one implemented application area, a short account of extensions that the architecture supports, and a detailed illustrative scenario.

5.1. Visual Analytics Interface Design

Visual analytics is represented in the current implementation by its table and chart subsystems. A dashboard may show a summary without exposing how its data were selected, transformed, and rendered. SL-MDU makes these steps identifiable: a user selects a variable, applies a comparison operation, and inspects the output. This is consistent with visual analytics as interactive model building [14] and with a knowledge-generation cycle connecting data, visualization, interpretation, and action [15]. Implemented mechanisms include cell, row, and column selection, sorting, filtering, range summation, chart-type switching, and conditional data retrieval.

5.2. Architecturally Supported Extensions

Three further application areas are supported by the architecture but not implemented. In modular model construction, feature selection, normalization, retraining, and comparison of feature importance would each be registered as SLs, making the modeling sequence explicit without removing the need for statistical expertise [3] [8] [23]. In interactive feature engineering, a feature would be an MDU and a transformation such as binning or encoding an SL, so that model behavior can be compared before and after each change [7] [19]. In decision support, a recommendation would be traced back to the indicators and rules that produced it, so that decision-makers can inspect and contest it [1] [9]. Each extension requires writing and registering domain SLs that satisfy the criteria of Section 3.2; the student scenario below works one such extension through in detail.

5.3. Illustrative Student Risk-Warning Dashboard

This illustrative scenario shows how an academic-support team could use SL-MDU to identify students who may need support and inspect the basis for that assessment. It is neither a shipped module nor an empirical study, and no real student data were used. **Table 5** maps the dataset, MDUs, operations, workflow, outputs, and evaluation questions to mechanisms in the reference implementation, and **Figure 4** illustrates the intended dashboard and a complete trace for student S-017.

5.3.1. Scenario Setting

Consider a course with 142 students. After the midterm examination, a team of tutors and advisors needs to identify students who may benefit from intervention. The illustrative data comprise attendance records, examination scores, assignment submissions and login records from the learning-management system (LMS), and free-text tutor notes. Four numerical indicators are considered alongside the notes: attendance rate, midterm score, assignment completion, and weekly logins.

At level L3, an MDU is one indicator for one student, such as S-017's attendance rate, rather than the entire grade sheet; it carries the identifiers of the student, the indicator, and the observation time. An SL is one operation applied to that unit,

such as a threshold test, a comparison with the cohort median, or a rule-based classification. Derived outputs become MDUs in subsequent steps, provided their relationship to the source inputs remains inspectable.

Three indicators enter the classification: attendance, midterm score, and assignment completion. The illustrative rule assigns HIGH when at least two of these indicators are flagged, MEDIUM when exactly one is flagged, and LOW when none is flagged. Login frequency is displayed as context and is not part of the rule. A contextual override is a separate, explicit tutor action that preserves the rule-generated result and records its reason. These thresholds illustrate a traceable interaction; they are not validated criteria for predicting student outcomes.

Table 5. SL-MDU application in an illustrative student risk-warning dashboard.

Component	Illustrative student scenario	Implementation correspondence and gap
Dataset	142 students; attendance, examination, assignment/LMS records, and tutor notes. Four numerical indicators plus contextual text.	Tabular rendering and per-user storage exist. A synthetic 142 × 4 cohort is used for measurement (Section 6.2); no real student dataset or deployed dashboard.
MDU	One student indicator (L3): attendance 68%; midterm 52/100; assignments 6/10; weekly logins 3; or one tutor note (L4).	Cells, rows, and columns can be selected. Levels L0 - L4 and scenario identifiers must be configured.
SL operations	Filter attendance < 75%; compare score with the cohort median; test assignments < 8/10; classify ≥ 2 flags as HIGH; explain; review and override.	Filter, comparison, threshold, and classification SLs are registered and measured in the unmodified dispatcher (Listing 2). Explanation and override handling are not implemented.
Workflow	Select units → filter/compare/threshold → classify → inspect → revise. Reordering must preserve input dependencies.	Ordered module chains and configurable operation pages exist. They do not implement a validated dependency graph.
Output	Student/cohort comparison, provisional risk table, and an explicit data-operation trace; rule result distinguished from tutor assessment.	Workspace, tables, and charts exist; rendering the 142-row table is measured. A chronological trace view and historical comparison are not implemented.
Evaluation	Identify why S-017 is flagged; explain the rule and override; revise a valid workflow; measure workload and latency.	Latency, scaling, and trace size are measured (Table 7). User-facing questions are addressed by the hypothesis in Section 6.4.

5.3.2. Workflow Walkthrough for Student S-017

The following steps instantiate the cycle of Section 3.5 and correspond to the explanation/history panel in Figure 4. The median of 71 is the illustrative value used in Figure 4.

t1 Filter. The tutor selects the attendance-rate MDU and applies attendance < 75%. S-017's attendance is 68%, so flag 1 is TRUE. The trace associates the input value, the parameters, and the result: (attendance 68%, filter < 75%) → TRUE, which is more specific than recording only that attendance is low.

t2 Compare. The tutor selects the midterm-score MDU and compares it with the cohort median. S-017 scored 52/100 against a median of 71/100, a difference of −19 points, so flag 2 is TRUE. The baseline and the flagging rule are explicit

parameters that can be inspected and changed.

t3 Threshold. The tutor selects the assignment-completion MDU and applies completed assignments < 8/10. S-017 completed 6/10 assignments, so flag 3 is TRUE.

t4 Classify. The three flags form a derived MDU. Applying at least two flags → HIGH gives a provisional label of HIGH, and the output panel associates the label with the three contributing indicators and their operations. The label is an inspectable composition of prior steps rather than an unexplained score.

t5 Review and revise. The tutor selects a note documenting illness and marks the attendance flag as excused. Re-running the unchanged rule leaves two active flags, so its output remains HIGH. If the tutor judges MEDIUM to be the more appropriate support priority, that change is recorded as an explicit human override with the note and rationale attached. The intended provenance view retains the initial result, the recomputed rule result, and the tutor's assessment. It must not imply that removing one of three flags automatically produces MEDIUM under the same rule.

5.3.3. Boundary Conditions of the Classification Rule

The rule in t4 is only traceable if its boundary cases are as explicit as the override in t5. The following conditions are therefore part of the rule's parameters rather than implementation details. 1) All comparisons are strict: attendance of exactly 75% and exactly 8/10 assignments are not flagged. 2) A score equal to the median is not flagged; with an even cohort of 142, the median is the mean of the two middle scores and may be fractional—in the synthetic benchmark cohort it is 72.5, so a score of 72 is flagged. 3) Because the midterm flag is defined relative to the median, roughly half of any cohort receives it by construction; in the synthetic cohort the rule labels 23 students HIGH, 70 MEDIUM, and 49 LOW. Label prevalence is thus determined by the rule's parameters, and a tutor inspecting the distribution at L0 should see this before interpreting MEDIUM as a warning. 4) A missing indicator yields an undefined flag, not FALSE; the rule must either report the label as undetermined or state that it was computed over fewer indicators. 5) An excused flag is excluded from the count but remains visible in the trace. 6) Only the three named indicators enter the rule; logins never change the label.

5.3.4. Granularity in the Scenario

Applying the policy of Section 3.4, the tutor does not begin with 568 indicator values. The dashboard opens at L0, showing the distribution of each indicator and the counts per label. Because the classification SL fires for 23 students (T1), the HIGH segment is offered at L1; with a budget of B = 20, it is ranked by the number of active flags and then by the margin to the nearest threshold, and the three lowest-ranked students are shown as one aggregate row. S-017, with three flags, is near the top, and selecting the student opens L2. Only when the tutor questions the label (T2) are the three contributing indicator MDUs expanded to L3 with their rules and parameters (t1 - t3), and only when the tutor considers an override

is the L4 tutor note opened. A program manager reviewing the same cohort remains at L0-L1. The tutor thus inspects at most a few dozen units on the path to one decision, and every change of level is itself recorded. Section 6.2 shows that the policy is also computationally motivated: executing the rules per unit costs about 30 ms for 142 students but grows linearly with cohort size, whereas cohort-level execution remains below 4 ms at 14,200 students.

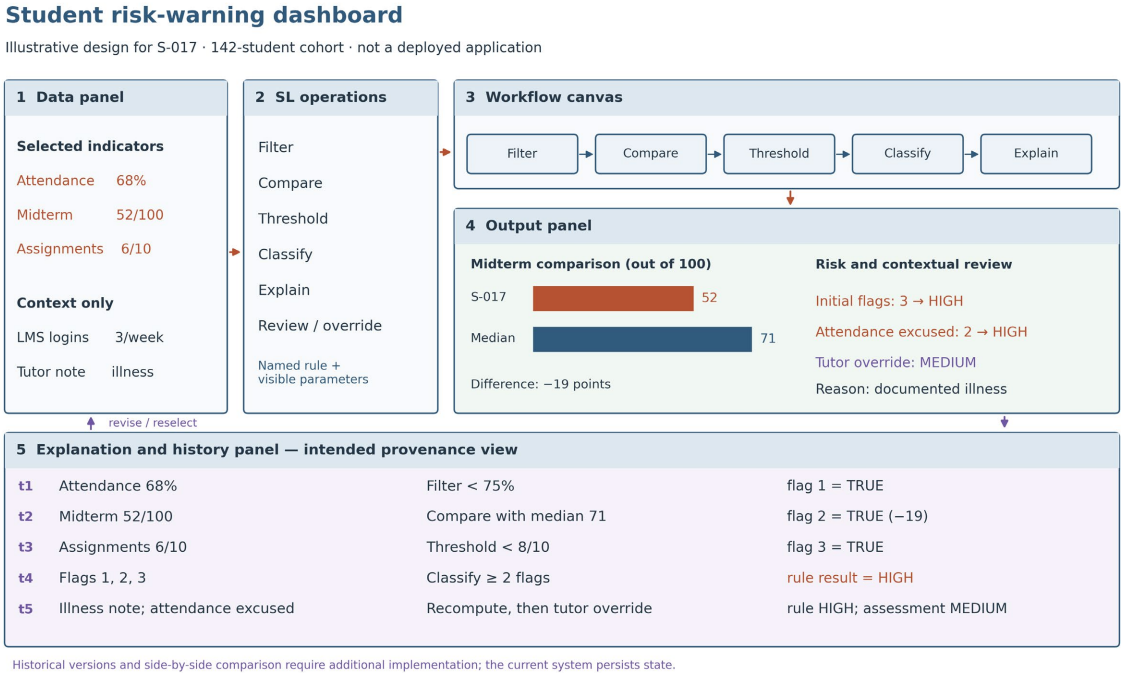


Figure 4. Illustrative student risk-warning dashboard. Panels 1-5 show the intended data, operation, workflow, output, and history views. The t5 tutor override is explicitly distinguished from the recomputed HIGH rule result.

5.3.5. Dashboard Interface and Relationship to the Implementation

Figure 4 organizes the scenario around the five functional elements of Section 4.3. The data panel lists selectable indicators and contextual notes; the operation panel makes filtering, comparison, thresholding, classification, and explanation available as named actions; the workflow canvas shows the ordered sequence and a route back to selection; the output panel presents the indicators and provisional label while distinguishing rule output from contextual assessment; and the explanation/history panel shows the t1 - t5 trace.

The reference implementation supplies the mechanisms needed to construct the dashboard—keyword-addressed operations and a common dispatcher; selection at cell, row, or column level; table operations; timestamped window state; and per-user configuration storage—and the filter, comparison, threshold, and classification SLs of the scenario have now been registered and executed through the unmodified dispatcher on synthetic data (Listing 2; Section 6.2).

Completing the dashboard would still require explanation and override operations, the granularity policy, an append-only event log with preserved versions of rule outputs and human overrides, historical comparison, and the access controls

of Section 4.7. Workflow editing must also respect data dependencies: downstream classification cannot consume flags that have not been produced, so the user-control principle permits replacing or reordering operations only when their input requirements remain satisfied.

5.3.6. Evaluation Focus and Ethical Boundary

The scenario maps to the dimensions in Table 6. Transparency concerns whether a user can identify the data units, operations, and parameters responsible for the provisional HIGH label; interpretability, whether the user can reconstruct t1 - t4 and distinguish the unchanged rule output from the override in t5; flexibility, whether a valid operation or parameter can be replaced and the workflow rerun; and cognitive load, the effort of moving between the cohort overview and an individual indicator. Section 6.4 turns the first of these into a falsifiable hypothesis.

A risk label is a prompt for contextual assessment and possible support, not a verdict about a student. Making the sequence inspectable does not remove bias in the data or guarantee that an intervention is appropriate; login counts, for example, may not mean the same for students with different study habits. The tutor remains responsible for contextual judgment, and any override must be distinguishable from the rule output. The scenario therefore illustrates explainable and contestable decision support in which human participation assists rather than replaces judgment [1] [5] [9], subject to the technical controls of Section 4.7.

6. Evaluation

6.1. Evaluation Design

Using FEDS [13], evaluation proceeds in three episodes of increasing naturalism. Episode 1, reported here, is artificial and formative: a technical performance evaluation of the implemented mechanism on the illustrative workload (Section 6.2) and a heuristic evaluation of the implemented element-empowerment cycle by three evaluators (Section 6.3). Episode 2 is a controlled laboratory study and Episode 3 a domain study with intended users; both are specified in Section 6.4. Table 6 lists the evaluation dimensions, their measures, and the evidence available after Episode 1.

6.2. Technical Performance Evaluation

Setup. The dispatcher (脚本动态管理－本地版.js) and the export registry (脚本登记表.js) were copied verbatim from the reference implementation; the copy is byte-identical to the submitted source (checksum in S3). The scenario SLs—threshold filter, baseline comparison, rule classification, cohort median, indicator retrieval, and their cohort-level variants—were written as ES modules and registered in the bridge table alongside the implementation's own seed rows. A synthetic cohort of 142 students with four indicators was generated from a fixed random seed, with S-017 set to the values of Section 5.3. Pages were served from a local HTTP server and run in headless Google Chrome 154 on a desktop computer

(Intel Core i9-10900K, 3.7 GHz; 128 GB RAM; Windows 11). Timings used performance.now() in a cross-origin-isolated page (5 μs resolution). Each configuration was measured in 20 independent browser sessions (5 for the scaling runs); Table 7 reports the median across sessions of the per-session median, with the range of the per-session values.

Table 6. Evaluation dimensions, measures, and status after Episode 1.

Dimension	Measures	Status after Episode 1
Performance	Dispatch latency (cold, cached), workload response time, scaling, rendering delay, trace size.	Measured (Section 6.2, Table 7).
Usability	Heuristic violations and severity; later SUS, task completion time, error rate.	Heuristic evaluation completed: 41 problems, 18 strengths (Section 6.3, Table 8, Table 9); SUS and task measures in Episode 2.
Transparency	Correct identification of the MDU, SL, and parameters that produced an output (attribution accuracy).	Primary outcome of hypothesis H1 (Section 6.4).
Interpretability	Correctness of users' explanations of inputs, rules, and outputs; distinction between rule result and human override.	Secondary outcome (H2); requires the trace view and an explanation rubric.
Flexibility	Successful valid replacement or reordering of SLs; effort to revise a task.	Bridge table editable; to be measured in Episode 2 with dependency checks.
Cognitive load	NASA-TLX; navigation and operation counts; granularity level changes.	Non-inferiority outcome (H3); requires task-level event logging and the granularity policy.

Measures. 1) First dispatch in a page, including the registry import; 2) cold dispatch of a not-yet-loaded keyword, *i.e.*, module fetch, import, and execution, and the cached second call; 3) cached dispatch compared with a direct function call, timed over batches of 200 calls; 4) sensitivity to the size of the bridge table; 5) the per-unit workload—for each student, t1 - t3, retrieval of the login indicator, and t4, *i.e.*, 710 dispatches—on its first and subsequent passes; 6) the same rules applied to indicator columns at cohort level (five dispatches); 7) rendering the 142-row result table in the workspace; and 8) the size and write time of the explicit (MDU, SL, parameters, output) trace of one pass.

Table 7. Performance of the unmodified dispatcher on the illustrative workload (synthetic data; headless Chrome 154).

Measurement	Median	Range across sessions
First dispatch in page (registry import + module load + execution)	6.28 ms	5.41 - 7.23 ms
Cold dispatch, new keyword (module fetch + import + execution)	1.57 ms	1.40 - 1.67 ms
Cached second call of the same keyword	0.070 ms	0.065 - 0.075 ms
Cached dispatch, per call (bridge table of 84 rows)	43.5 μs	40.2 - 43.8 μs
Direct call of the same entry function, per call	0.20 μs	0.20 - 0.23 μs
Cached dispatch, per call (bridge table of 500 rows)	189.7 μs	170.2 - 190.6 μs
Per-unit workload, 142 × 4, first pass (710 dispatches)	46.4 ms	43.5 - 48.3 ms
Per-unit workload, 142 × 4, cached passes (710 dispatches)	30.5 ms	28.4 - 31.0 ms

Continued

Cohort-level workload, same rules (5 dispatches), cached	0.27 ms	0.25 - 0.28 ms
Per-unit workload, 1420 students (7100 dispatches) †	276 ms	276 - 277 ms
Per-unit workload, 14,200 students (71,000 dispatches) †	2748 ms	2742 - 2762 ms
Cohort-level workload, 14,200 students †	3.49 ms	3.45 - 3.72 ms
Render 142-row result table incl. layout	4.16 ms	4.10 - 4.41 ms
Trace of one pass: 710 records, 85.5 KiB; write to local storage	0.095 ms	0.085 - 0.110 ms

† Separate run of five sessions with the same bridge-table size. In all 20 sessions of the main run, per-unit and cohort-level execution produced identical labels for all 142 students.

Findings. First, caching works as intended: loading a new operation costs about 1.6 ms, whereas calling it again costs 0.07 ms. Second, the cached dispatcher adds about 40 µs per call over a direct function call. A follow-up run that replaced console.log by a no-op without modifying the dispatcher located this overhead: of 38.5 µs per call, 28.2 µs is spent re-reading and re-parsing the 7.1 KB bridge table from local storage on every call, and about 10 µs on per-call logging; the remaining dispatch logic costs under 1 µs. The overhead therefore grows with the size of the operation library (190 µs at 500 rows) and could be removed by caching the parsed table and invalidating it on change. Third, the full illustrative workload at unit granularity completes in about 30 ms, well within the 100 ms associated with perceived immediacy [39] and far below the 500 ms at which latency degrades exploratory analysis [40]; rendering adds about 4 ms. Fourth, per-unit execution scales linearly—276 ms at 1420 students and 2.7 s at 14,200—whereas cohort-level execution of the same rules stays below 4 ms. This is the computational counterpart of the granularity policy: compute at the coarsest level, and execute and expose unit-level SLs only for units that a descent trigger selects. Finally, an explicit per-unit trace is small at this scale (85.5 KiB per pass, written in 0.1 ms), but its size grows at the same rate; at 14,200 students it would be about 8.8 MB per pass, beyond the typical 5 MB per-origin quota of browser local storage, so the append-only history of Section 4.4 needs server-side storage.

Limitations. These are laboratory measurements on a single high-end desktop with a local server; network latency, slower devices, and the full application's resident modules would increase absolute times. The scenario SLs are simple rules, so the measurements characterize the interaction mechanism rather than analytical computation. The harness, raw results, and instructions to reproduce them are provided in S3.

6.3. Heuristic Evaluation

Method. Three evaluators inspected the implemented element-empowerment cycle [41] [42]: two developers of the reference implementation and one professional software tester who was not involved in its development. Each evaluator worked through the same seven-step walkthrough script (Supplementary Material S4):

open the environment; open the element-empowerment window; select a target element; apply a style and an attribute operation; close and reopen the window; invoke an unregistered keyword; and inspect the operation library. Evaluators assessed the interface against Nielsen's ten heuristics [42] and three SL-MDU heuristics derived from Section 3: S1 unit-operation visibility (the selected MDU and the applied SL with its parameters are visible together), S2 rule-judgment separation (a rule output and a human revision are distinguishable and both recoverable), and S3 granularity fit (the number of units presented matches the task level). Each problem was recorded with the violated heuristic, location, and severity on Nielsen's 0 - 4 scale. The three problem lists were then combined; the counts reported below are the sum over the three evaluators, and the mean severity is taken over all reported problems. Evaluators also recorded positive findings; because two evaluators had access to the source code, each positive finding is linked to the code that implements it, so that it can be checked in S2.

Results. The three evaluators reported 41 problems in total, with a mean severity of 2.3 (Table 8). The most severe concerned user control and error prevention: there is no undo, and whole-element replacement deletes the original element irreversibly (H3, mean 3.0); the target list contains every element with an identifier, including the controls of the element-empowerment window itself, so an operation can be applied to the panel (H5, 2.8); and a selected target container is bound only when focus leaves the panel, so parameters changed before then have no effect and nothing indicates this (H1, 2.5). The evaluators also recorded 18 positive findings (Table 9), notably the real-time preview that back-fills the form from the target's actual state, the highlighting of the selected container, per-property enable switches, and the prevention of duplicate identifiers.

Table 8. Heuristic evaluation of the element-empowerment cycle (three evaluators; counts summed over evaluators).

Heuristic	Problems	Mean severity (0 - 4)	Most severe problem
H1 Visibility of system status	4	2.5	A selected target container is bound only when focus leaves the panel; parameters changed before then have no effect, and nothing indicates this.
H2 Match with the real world	3	2.3	The target drop-down lists raw DOM ids (e.g., id_1719…_x8k2) that users cannot map to regions of the page.
H3 User control and freedom	4	3.0	No undo; "whole replacement" deletes the original element and cannot be reverted.
H4 Consistency and standards	4	2.3	The three tabs commit changes differently: auto-save (track panel), a button (style injection), on input (element assignment).
H5 Error prevention	4	2.8	The target list contains every element with an id, including the panel's own controls, so an operation can be applied to the panel itself.
H6 Recognition rather than recall	3	2.0	Style injection requires a fixed sequence (selector → two-level menu → inject) that the interface does not show.

Continued

H7 Flexibility and efficiency of use	3	2.3	The target drop-down cannot be searched; all ids on the page are listed in one flat list.
H8 Aesthetic and minimalist design	2	2.0	Ten parameter groups are laid out flat; the read-only "dynamic columns/cells" fields look like editable inputs.
H9 Recognize, diagnose, recover from errors	3	2.0	Injection errors are written only to the browser console; the user sees nothing.
H10 Help and documentation	2	2.0	No in-window help or hints; documentation exists only as developer markdown.
S1 Unit-operation visibility	3	2.7	If any checkbox exists on the page, style injection enters multi-select mode even when none is ticked, reports "select at least one element", and ignores the target already shown in the preview.
S2 Rule-judgment separation	3	2.0	When the target has several classes, the user must choose which to replace, although a rule could decide.
S3 Granularity fit	3	1.7	The tabs operate on different units: container (track panel); class, whole, or child element (style injection); container or leaf (element assignment).
Total	41	2.3	

Table 9. Positive findings of the heuristic evaluation, with the code that implements each.

Heuristic	Positive finding	Code basis
H1 Visibility of system status	Element assignment previews in real time: input is applied to the target immediately and the form is back-filled from the target's current values, so the panel shows the element's actual state.	动态修改属性函数.js: requestAnimationFrame(sync) → 按元素填充表单(nodes[0])
H1/S1	Choosing a target container in the drop-down highlights the corresponding element on the page, linking the id to the visible unit.	div 元素信息_提取.js: classList.add ("当前选中容器")
H3 User control and freedom	Track-panel parameters are saved automatically after a 300 ms debounce, so configuration is not lost by forgetting to save.	地轨面板_逻辑.js: debouncedSave
H3/S3	Each property in element assignment has its own enable switch; only ticked properties are changed, leaving other styles untouched.	动态修改属性函数.js: pick(chk, ctl)
H4 Consistency and standards	The track-panel form and its data are bound in both directions, so the two always agree.	地轨面板_逻辑.js: Proxy set → updateFormElement
H4/H8	Switching tabs clears leftover checkboxes and preview content, so each tab starts from a clean state.	要素赋能_入口.js: pageLoadHandler (clean-up)
H5 Error prevention	Id changes are excluded in multi-select mode, and a new id that already exists on the page is not written, preventing duplicate ids.	动态修改属性函数.js: delete props.id; !document.getElementById(p.id)
H5	Whole replacement carries the original class, id, and style into the new template, so the element keeps its grid position (grid-area).	动态样式注入函数.js: 动态模板()
H5	A blacklist keeps ids that should not be targets out of the target list.	div 元素信息_提取.js: localStorage "黑名单"
H5	Each style class is injected into a <style> element only once, so repeated operations do not accumulate duplicate rules.	动态样式注入函数.js: injectedStyleSet
H6 Recognition rather than recall	Track-panel settings are stored per container and restored when the user returns to it, so they need not be remembered.	地轨面板_逻辑.js: getOrCreateProxy + initFormValues

Continued

H6	Style boxes are chosen from a two-level menu (operation type, then style), by recognition rather than by name.	<dropdown-menu data=“要素赋能”>
H7 Flexibility and efficiency	With dynamic application on, column count and cell width follow the browser width automatically.	可绘表列_行数.js: browser-width listener
H7	Styles and attributes can be changed for several ticked targets at once.	处理多选模式()/getTargets()
H7	The window can be dragged, resized, and minimized; its position and size are remembered, so it can be used alongside the page.	弹窗_3.1: popup_state_<id>
H9 Error recovery	Most messages about missing preconditions say what is missing, e.g. both style-box menus must be chosen, or no element with the given id exists.	动态样式注入函数.js: alert messages
S2 Rule-judgment separation	The rule lives in the template: the style box’s mode (replace-class, whole, child) is defined there, so the user decides only which effect is wanted.	获取模板数据(currentKey).mode
S2	When the target has several classes, a dialog lets the user choose instead of replacing blindly.	动态样式注入函数.js: 替换样式() multi-class branch

Interpretation. The findings bear directly on the SL-MDU heuristics. S1 is only partly met. In the element-assignment tab the pairing of unit and operation is visible: the chosen container is highlighted and every change is previewed on it. In the style-injection tab, however, the mere presence of checkboxes switches to multi-select mode and ignores the target shown in the preview (S1, 2.7), and the target list shows raw DOM ids rather than recognizable units (H2). Both problems break the pairing the framework requires and argue for a single, explicit “current MDU” state shared by all tabs. S2 produced a mixed judgment that illustrates the rule-judgment boundary itself: asking the user which of several classes to replace was recorded both as a safeguard against blind replacement and as a decision a rule could make. Under criterion C2 (Section 3.2) this choice is an exposed parameter, so the design question is whether to default it by rule while keeping it visible. S3 (1.7) confirms that the tabs operate on different units without declaring them, which is what the level declaration of Section 3.4 would make explicit. The lack of undo (H3) and of any record of earlier states follows from the current-state provenance stated in Section 1.5 and motivates the append-only history of Section 4.4. The findings translate into four design changes for the next iteration: a single current-MDU state shared by all tabs, with the panel’s own controls excluded from the target list and targets shown by recognizable names (S1, H2, H5); undo built on the append-only history (H3); one commit behavior across the three tabs, with a visible binding state (H1, H4); and error messages shown in the interface rather than only in the console (H9). Two limitations follow from the procedure. First, two of the three evaluators developed the system, so the evaluation is not fully independent: developers know the intended sequence of operations and may overlook problems that a first-time user would meet. Second, the problem counts are summed over evaluators rather than merged into unique problems, so a prob-

lem found by several evaluators is counted more than once; the counts indicate where problems concentrate rather than how many distinct problems exist. The controlled study in Section 6.4, with participants who did not build the system, addresses both limitations.

6.4. Hypotheses and Protocol for Episodes 2 and 3

The framework's central claim is that making the MDU-SL pairing explicit helps users account for an output. Episode 2 tests it with a falsifiable primary hypothesis and two secondary hypotheses.

H1 (attribution accuracy). Users who work with an interface exposing explicit MDU-SL traces identify the data unit, operation, and parameter responsible for an output more accurately than users of an equivalent dashboard that shows the same data and outputs without the trace.

H2 (rule-judgment separation). Users with the trace distinguish a rule-generated result from a human override more accurately than users without it.

H3 (non-inferior load). The trace does not increase cognitive load (NASA-TLX) by more than a pre-specified margin of 10 points.

Design. A within-subjects comparison of two interface conditions built on the same implementation, holding data, tasks, and task difficulty constant and counterbalancing condition order and task sets. Participants complete attribution tasks of the form "Which unit and which rule caused this student to be labeled HIGH, and which parameter change would alter the label?" on synthetic cohorts. The primary outcome is the proportion of fully correct (MDU, SL, parameter) attributions, analyzed with a mixed-effects logistic regression with participant and item as random effects. For a paired comparison, a medium effect ($d = 0.5$) at $\alpha = 0.05$ and power 0.80 requires 34 participants [43]; we will recruit at least 36. H1 is falsified if the trace condition does not produce a significantly higher attribution accuracy; in that case the transparency benefit claimed for SL-MDU would not be supported for this task class. Episode 3 then evaluates the dashboard with academic-support staff after the operations, event history, granularity policy, and access controls of Sections 3.4 and 4.7 have been implemented and ethics approval has been obtained. The sampling plan, stopping rule, and analysis will be preregistered before data collection.

7. Discussion

7.1. Potential Benefits of the SL-MDU Framework

The principal anticipated benefit is interaction interpretability: selected inputs, operations, parameters, and outputs can be inspected during analysis. An explicit sequence could support debugging and reuse across datasets, consistent with work on visual model building and modular composition [14] [24]. Object-level controls may also make operations accessible to users with different technical backgrounds, although authoring new SLs still requires expertise. These benefits depend on presentation: explanations must be understandable, not merely available

[1], and excessive detail may obstruct the task. Episode 1 shows that the mechanism is fast enough for the illustrative workload; whether it improves users' performance or understanding is the subject of H1-H3.

7.2. Cognitive Load and Granularity

Cognitive load is the central constraint on the framework, and Section 3.4 therefore specifies a granularity policy rather than deferring the problem: the interface opens at the coarsest level that answers the task, exposes at most B units, and descends only on exception, contestation, or explicit request, following the why-provenance of the output in question. The policy also bounds computation, since Section 6.2 shows that per-unit execution grows linearly with cohort size while cohort-level execution does not. What remains open is empirical and technical: the implementation still offers only manual granularity control, the budget B and relevance scores need calibration with users, and H3 must show that the trace does not add load [7] [9].

7.3. Computational and Performance Trade-Offs

Fine-grained interaction creates additional selections, transformations, intermediate outputs, and history records. The measurements in Section 6.2 show that, for the illustrative workload, module and invocation caching keep dispatch well within interactive limits, and that the dominant remaining cost is avoidable—the per-call re-parsing of the bridge table and logging. At larger scales, the cost of unit-level execution and of the trace both grow linearly, so progressive computation [25], aggregation [38], and server-side history storage become necessary. Coordinating many small components also adds architectural complexity, which the single dispatch point partly contains.

7.4. Applicability to Advanced Analytics

SL-MDU can support interaction with advanced machine learning models, but it does not resolve the opacity of their internal representations. Its contribution is at the interaction level: users can inspect selected data, transformations, feature changes, model outputs, and available explanations. It complements XAI methods rather than replacing them [22] [23]. A trace of the actions taken by a user is not, by itself, an explanation of a model's internal reasoning. SL-MDU should therefore be understood as a framework for inspectable and accountable model use, with benefits that depend on the quality of both the interface and the available model explanations [1] [8].

8. Conclusion and Future Work

This paper presents SL-MDU as a human-centered framework for data analysis and modeling, grounded through design science in a working reference implementation. An SL is an operation that satisfies four testable atomicity criteria for a specified task, and an MDU is the smallest data entity meaningful for that task

at a declared granularity level. Relative to subset-flow, modular-composition, and provenance systems, the framework contributes a task-level contract that makes each result accountable to an explicit (MDU, SL, parameters, output) record, separates rule outputs from human overrides, and governs which units are exposed through a granularity policy. The implemented dispatcher was documented at code level and measured on the illustrative workload: 710 unit-level operations completed in about 30 ms, the dominant overhead was identified and is removable, and cohort-level execution remained below 4 ms at 14,200 students. A heuristic evaluation by two developers and one professional tester recorded 41 usability problems (mean severity 2.3) and 18 strengths; the most severe problems, missing undo and weak feedback on the targeted unit, point to the append-only history and the explicit current-MDU state as the next design steps. Improvements in users' understanding, cognitive load, and task performance are not yet established; they are stated as hypotheses H1-H3.

Future work combines further implementation with empirical evaluation. Development priorities are caching of the parsed bridge table, an append-only analytical event log stored server-side, version comparison, the granularity policy, domain SLs for explanation and override, and the access controls of Section 4.7. Episode 2 will test H1-H3 in a controlled study, and Episode 3 will evaluate the dashboard with academic-support staff. Public release and any de-anonymization of the implementation will follow the requirements of the submission and publication process.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Ali, S., Abuhmed, T., El-Sappagh, S., Muhammad, K., Alonso-Moral, J.M., Confalonieri, R., *et al.* (2023) Explainable Artificial Intelligence (XAI): What We Know and What Is Left to Attain Trustworthy Artificial Intelligence. *Information Fusion*, **99**, Article 101805. https://doi.org/10.1016/j.inffus.2023.101805

[2] Endert, A., Hossain, M.S., Ramakrishnan, N., North, C., Fiaux, P. and Andrews, C. (2014) The Human Is the Loop: New Directions for Visual Analytics. *Journal of Intelligent Information Systems*, **43**, 411-435. https://doi.org/10.1007/s10844-014-0304-9

[3] Sacha, D., Sedlmair, M., Zhang, L., Lee, J.A., Peltonen, J., Weiskopf, D., *et al.* (2017) What You See Is What You Can Change: Human-Centered Machine Learning by Interactive Visualization. *Neurocomputing*, **268**, 164-175. https://doi.org/10.1016/j.neucom.2017.01.105

[4] Amar, R., Eagan, J. and Stasko, J. (2005) Low-Level Components of Analytic Activity in Information Visualization. IEEE Symposium on Information Visualization, 2005. *INFOVIS* 2005., Minneapolis, 23-25 October 2005, 111-117. https://doi.org/10.1109/infvis.2005.1532136

[5] Rehrl, K. and Gröchenig, S. (2016) A Framework for Data-Centric Analysis of Mapping Activity in the Context of Volunteered Geographic Information. *ISPRS Interna-*

tional Journal of Geo-Information, **5**, Article 37. https://doi.org/10.3390/ijgi5030037

[6] Yu, B. and Silva, C.T. (2017) Visflow—Web-Based Visualization Framework for Tabular Data with a Subset Flow Model. *IEEE Transactions on Visualization and Computer Graphics*, **23**, 251-260. https://doi.org/10.1109/tvcg.2016.2598497

[7] Dudley, J.J. and Kristensson, P.O. (2018) A Review of User Interface Design for Interactive Machine Learning. *ACM Transactions on Interactive Intelligent Systems*, **8**, 1-37. https://doi.org/10.1145/3185517

[8] Mosqueira-Rey, E., Hernández-Pereira, E., Alonso-Ríos, D., Bobes-Bascarán, J. and Fernández-Leal, Á. (2023) Human-in-the-Loop Machine Learning: A State of the Art. *Artificial Intelligence Review*, **56**, 3005-3054. https://doi.org/10.1007/s10462-022-10246-w

[9] Gómez-Carmona, O., Casado-Mansilla, D., López-de-Ipiña, D. and García-Zubia, J. (2024) Human-in-the-Loop Machine Learning: Reconceptualizing the Role of the User in Interactive Approaches. *Internet of Things*, **25**, Article 101048. https://doi.org/10.1016/j.iot.2023.101048

[10] Hevner, A.R., March, S.T., Park, J. and Ram, S. (2004) Design Science in Information Systems Research. *MIS Quarterly*, **28**, 75-106. https://doi.org/10.2307/25148625

[11] Peffers, K., Tuunanen, T., Rothenberger, M.A. and Chatterjee, S. (2007) A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, **24**, 45-77. https://doi.org/10.2753/mis0742-1222240302

[12] Gregor, S. and Hevner, A.R. (2013) Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly*, **37**, 337-355. https://doi.org/10.25300/misq/2013/37.2.01

[13] Venable, J., Pries-Heje, J. and Baskerville, R. (2016) FEDS: A Framework for Evaluation in Design Science Research. *European Journal of Information Systems*, **25**, 77-89. https://doi.org/10.1057/ejis.2014.36

[14] Andrienko, N., Lammarsch, T., Andrienko, G., Fuchs, G., Keim, D., Miksch, S., *et al.* (2018) Viewing Visual Analytics as Model Building. *Computer Graphics Forum*, **37**, 275-299. https://doi.org/10.1111/cgf.13324

[15] Sacha, D., Stoffel, A., Stoffel, F., Kwon, B.C., Ellis, G. and Keim, D.A. (2014) Knowledge Generation Model for Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics*, **20**, 1604-1613. https://doi.org/10.1109/tvcg.2014.2346481

[16] Brehmer, M. and Munzner, T. (2013) A Multi-Level Typology of Abstract Visualization Tasks. *IEEE Transactions on Visualization and Computer Graphics*, **19**, 2376-2385. https://doi.org/10.1109/tvcg.2013.124

[17] Gotz, D. and Zhou, M.X. (2009) Characterizing Users' Visual Analytic Activity for Insight Provenance. *Information Visualization*, **8**, 42-55. https://doi.org/10.1057/ivs.2008.31

[18] Kandel, S., Paepcke, A., Hellerstein, J. and Heer, J. (2011) Wrangler: Interactive Visual Specification of Data Transformation Scripts. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vancouver, 7-12 May 2011, 3363-3372. https://doi.org/10.1145/1978942.1979444

[19] Sarkar, A. (2018) Interactive Analytical Modelling. Technical Report UCAM-CL-TR-920, University of Cambridge, Computer Laboratory.

[20] Kulesza, T., Burnett, M., Wong, W. and Stumpf, S. (2015) Principles of Explanatory Debugging to Personalize Interactive Machine Learning. In: Brdiczka, O. and Chau,

P., Eds., *Proceedings of the 20th International Conference on Intelligent User Interfaces*, ACM, 126-137. https://doi.org/10.1145/2678025.2701399

[21] Amershi, S., Cakmak, M., Knox, W.B. and Kulesza, T. (2014) Power to the People: The Role of Humans in Interactive Machine Learning. *AI Magazine*, **35**, 105-120. https://doi.org/10.1609/aimag.v35i4.2513

[22] Alicioglu, G. and Sun, B. (2022) A Survey of Visual Analytics for Explainable Artificial Intelligence Methods. *Computers & Graphics*, **102**, 502-520. https://doi.org/10.1016/j.cag.2021.09.002

[23] Yuan, J., Chen, C., Yang, W., Liu, M., Xia, J. and Liu, S. (2021) A Survey of Visual Analytics Techniques for Machine Learning. *Computational Visual Media*, **7**, 3-36. https://doi.org/10.1007/s41095-020-0191-7

[24] Mei, H., Chen, W., Ma, Y., Guan, H. and Hu, W. (2018) Viscomposer: A Visual Programmable Composition Environment for Information Visualization. *Visual Informatics*, **2**, 71-81. https://doi.org/10.1016/j.visinf.2018.04.008

[25] Ulmer, A., Angelini, M., Fekete, J., Kohlhammer, J. and May, T. (2024) A Survey on Progressive Visualization. *IEEE Transactions on Visualization and Computer Graphics*, **30**, 6447-6467. https://doi.org/10.1109/tvcg.2023.3346641

[26] Callahan, S.P., Freire, J., Santos, E., Scheidegger, C.E., Silva, C.T. and Vo, H.T. (2006) VisTrails: Visualization Meets Data Management. *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, Chicago, 27-29 June 2006, 745-747. https://doi.org/10.1145/1142473.1142574

[27] Freire, J., Koop, D., Santos, E. and Silva, C. (2008) Provenance for Computational Tasks: A Survey. *Computing in Science & Engineering*, **10**, 11-21. https://doi.org/10.1109/mcse.2008.79

[28] Heer, J., Mackinlay, J., Stolte, C. and Agrawala, M. (2008) Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. *IEEE Transactions on Visualization and Computer Graphics*, **14**, 1189-1196. https://doi.org/10.1109/tvcg.2008.137

[29] Ragan, E.D., Endert, A., Sanyal, J. and Chen, J. (2016) Characterizing Provenance in Visualization and Data Analysis: An Organizational Framework of Provenance Types and Purposes. *IEEE Transactions on Visualization and Computer Graphics*, **22**, 31-40. https://doi.org/10.1109/tvcg.2015.2467551

[30] Xu, K., Ottley, A., Walchshofer, C., Streit, M., Chang, R. and Wenskovitch, J. (2020) Survey on the Analysis of User Interactions and Visualization Provenance. *Computer Graphics Forum*, **39**, 757-783. https://doi.org/10.1111/cgf.14035

[31] Buneman, P., Khanna, S. and Wang-Chiew, T. (2001) Why and Where: A Characterization of Data Provenance. In: Van den Bussche, J. and Vianu, V., Eds., *Lecture Notes in Computer Science*, Springer, 316-330. https://doi.org/10.1007/3-540-44503-x_20

[32] Cheney, J., Chiticariu, L. and Tan, W. (2009) Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, **1**, 379-474. https://doi.org/10.1561/1900000006

[33] Moreau, L. and Missier, P. (2013) PROV-DM: The PROV Data Model. W3C Recommendation, World Wide Web Consortium. https://www.w3.org/TR/prov-dm/

[34] March, S.T. and Smith, G.F. (1995) Design and Natural Science Research on Information Technology. *Decision Support Systems*, **15**, 251-266. https://doi.org/10.1016/0167-9236(94)00041-2

[35] Hevner, A.R. (2007) A Three Cycle View of Design Science Research. *Scandinavian*

Journal of Information Systems, **19**, 87-92. https://aisel.aisnet.org/sjis/vol19/iss2/4/

[36] Sweller, J. (1988) Cognitive Load during Problem Solving: Effects on Learning. *Cognitive Science*, **12**, 257-285. https://doi.org/10.1207/s15516709cog1202_4

[37] Shneiderman, B. (1996) The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. *Proceedings* 1996 *IEEE Symposium on Visual Languages*, Boulder, 3-6 September 1996, 336-343. https://doi.org/10.1109/vl.1996.545307

[38] Elmqvist, N. and Fekete, J. (2010) Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. *IEEE Transactions on Visualization and Computer Graphics*, **16**, 439-454. https://doi.org/10.1109/tvcg.2009.84

[39] Card, S.K., Robertson, G.G. and Mackinlay, J.D. (1991) The Information Visualizer, an Information Workspace. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Reaching through Technology*, New Orleans, 27 April-2 May 1991, 181-186. https://doi.org/10.1145/108844.108874

[40] Liu, Z. and Heer, J. (2014) The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics*, **20**, 2122-2131. https://doi.org/10.1109/tvcg.2014.2346452

[41] Nielsen, J. and Molich, R. (1990) Heuristic Evaluation of User Interfaces. *Proceedings of the Sigchi Conference on Human Factors in Computing Systems Empowering People*, Seattle, March 1990, 249-256. https://doi.org/10.1145/97243.97281

[42] Nielsen, J. (1994) Heuristic Evaluation. In: Nielsen, J. and Mack, R.L., Eds., *Usability Inspection Methods*, John Wiley & Sons, 25-62.

[43] Cohen, J. (1988) Statistical Power Analysis for the Behavioral Sciences. 2nd Edition, Lawrence Erlbaum Associates.

Supplementary Material

S1 SL-MDU Implementation Evidence: construct-, principle-, and layer-to-code mapping with files, functions, and line numbers. **S2** Redacted source files cited in S1 and in Listings 1-3. **S3** Performance benchmark: verbatim dispatcher and registry with checksum, scenario SL modules, harness, raw results of all sessions, and reproduction instructions. **S4** Heuristic evaluation materials and results: walkthrough script, heuristic definitions, severity scale, recording form, and the consolidated problem and positive-finding lists.