Characterizing Failure Points in Rule-Based Spreadsheet Data Transformation: A Stage-Oriented Taxonomy and Empirical Failure Matrix

Fakhrul Adli Mohd Zaki, Mustafa Man, Mohamad Nor Hassan
Faculty of Computer Science and Mathematics, Universiti Malaysia Terengganu, Terengganu, Malaysia

***Abstract*—Rule-based spreadsheet data transformation remains widely used for converting human-centred spreadsheet tables into structured formats for reporting, analytics, and data integration because it is transparent, auditable, and reproducible. However, rule-based approaches often fail when spreadsheets encode structural meaning through visual or layout cues, such as multi-row headers, merged cells, interleaved subtotals, cross-tabulated values, and multiple logical tables within a worksheet. This study characterizes such failures using a stage-oriented taxonomy aligned with a six-stage transformation pipeline: ingestion, region detection, structural interpretation, semantic labeling, canonicalization, and validation. A deterministic baseline pipeline was evaluated against six controlled spreadsheet cases representing common semi-structured layouts. The resulting empirical failure matrix identified 15 severity-2 failures across the controlled cases. The most critical failures were concentrated in merged cell and hierarchy handling, multi-region and boundary interpretation, role mislabeling, wrong granularity, and silent semantic errors. These findings show that many transformation failures are not isolated rule defects but arise from missing pipeline capabilities, particularly explicit hierarchy representation, block-level segmentation, granularity control, and semantic invariant checking. The study contributes a reproducible failure-analysis framework that supports stage-localized diagnosis, comparison of rule-based spreadsheet transformation systems, and targeted improvement of transformation pipelines.**

Keywords*—Rule-based transformation; spreadsheet table understanding; data wrangling; failure taxonomy; failure matrix; canonicalization

I. Introduction

This study examines a necessary first step toward improving rule-based spreadsheet transformation: identifying how and where such transformers fail. Rather than proposing a new model or algorithm, the study provides a structured failure analysis. Specifically, it proposes 1) a failure taxonomy that maps failures to stages in a common rule-based transformation process and 2) a failure matrix derived from controlled test cases (S1-S6) that represent common spreadsheet layout patterns. This analysis supports more comparable failure reporting, improves rule-writing practices, and helps future designers create more robust validation and debugging methods.

The study makes three main contributions. First, it introduces a stage-oriented taxonomy that maps failures to components of a typical rule-based pipeline, including interpretation of the input spreadsheet, rule selection, rule execution, and output validation. Instead of presenting only a long list of error symptoms, the taxonomy supports failure diagnosis, system comparison, and structured reporting of transformation problems. Second, it presents an empirical failure matrix with measured severity scores. The matrix represents each identified failure type against six controlled spreadsheet test-case patterns (S1-S6), documenting which failures are triggered by particular structural or semantic issues and which case-code pairs produce the highest-severity failures. This representation is easier to interpret than isolated error lists because it highlights recurring and under-reported failure categories while grounding each failure in a concrete example. Third, it derives practical implications for improving rule writing and pipeline design. The results suggest the need for additional checkpoints to detect silent errors, more structure-aware interpretation of spreadsheet semantics, and explicit handling of multiline headers, grouped headers, merged cells, and subtotal rows.

A. Practical Motivation and Scope

Spreadsheet tables are designed for human users and often rely on a variety of non-schema-based cues for layout and data semantics. Such cues include consistent indentation, spacing, cell merging, bold text, borders, and footnotes. Automating the interpretation of spreadsheet tables using only the content of cells and cells' properties proves challenging because they lack formal semantics defined externally. Many researchers attempt to overcome these limitations by inferring table semantics from visual cues and their context. [6], [7]

In practice, rule-based scripts are commonly utilized to extract recurring table formats, such as monthly financial reports, departmental budgets, customer purchase orders, and survey responses. The strengths of rule-based conversion scripts are their transparency, auditability, maintainability with version control, reusability across different reports, and suitability for batch processing. One approach to productionizing rule-based conversion is through rule-based platforms, which apply a set of predefined transformation rules to the structured table data. These platforms ensure consistent and repeatable processing, but first, the rules for a particular table must be correctly defined to reflect its structure. [7]

However, many rule-based conversion approaches implicitly assume that tables have a uniform structure throughout their scope, which is a fragile assumption. First, it generally requires that a table consist of a single continuous

block. Second, it may require that the header depth be constant throughout. Third, if the cells in the table were merged to make reading more convenient for users, then the merging should be consistently applied and handled in the script. Fourth, it generally requires that subtotals and totals are clearly marked (perhaps using bold font or a common text string such as "Total") so they can be identified easily by filtering rows based on that specific visual cue. When tables are not consistent in these respects, rule-based conversion either will fail to process the spreadsheet table altogether, resulting in no output, or it will correctly process the input rows but generate incorrect table outputs, even though the outputs may be syntactically valid. These are insidious errors because the bad output might be used in downstream data analysis without producing any visible errors. Work in previous rule-based techniques shows that minor changes to the layout (e.g., adding a bold font) are sufficient to change the semantic meaning of the table, leading to a functionally different table from the one that the user is looking at in the spreadsheet. [6], [7]

This study is about failure modes of rule-based spreadsheet data extraction. Instead of introducing a new extraction technology or a novel machine learning technique for such a purpose, this study is an attempt to provide structure to the discussion around different failure modes of rule-based systems in spreadsheet data extraction. Designers of rule pipelines for such purposes should use this characterization to understand where failures come from, how they propagate through the pipeline, and how they might be fixed. This study first provides a stage-specific taxonomy of failures in rule-based extraction systems and then presents an empirically measured failure matrix based on a set of carefully chosen case spreadsheets exhibiting common layout types. This allows for a concise summary of the coverage of empirical experiments and recurring weaknesses that have been identified in such systems. [5]

B. Study Organization

The remainder of this study is divided into nine sections. Section II provides a survey of prior research, focusing on interactive data wrangling, programming by example, spreadsheet table understanding, and rule-based transformations. Section III details the formal setup for transformations and introduces the concept of a transformation failure point. Section IV outlines the architecture of the baseline data transformation pipeline, defines the associated set of basic rules, describes the controlled cases created for the experiment, and explains the metrics and procedure used for evaluating outcomes. Section V elaborates on the comprehensive taxonomy of failure types encountered during transformation. Section VI presents the constructed failure matrix, along with calculated severity scores, and discusses the observed trends and prominent types of failures. Section VII analyzes the identified failure patterns and proposes strategies for more robust rule creation. Section VIII addresses potential limitations and threats to the study's validity. Finally, Section IX summarizes the key findings and suggests avenues for future research.

II. Related Work

This section covers previous research on the manipulation and preprocessing of data stored in spreadsheets. The literature review is divided into four categories: data wrangling and interactive transformation, programming-by-example, table understanding and rule-based transformations, and data preparation validation/error detection. The categories are related due to the inherent nature of spreadsheet transformations; a transformation is often not simply changing data format but also inferring spreadsheet layout, interpreting tabular structure, maintaining semantic relations, and creating an output that is accurate and useful. The review explores how different techniques empower users to clean, transform, extract, and validate data in spreadsheets and highlights challenges that remain with current methods.

A. Data Wrangling and Interactive Transformation

Research findings demonstrate that data wrangling is a repeated activity during which users perform certain transformations and then observe the outputs. This interactive cycle involves letting users visualize transformation outputs and allowing them to reverse actions if outputs are invalid. Petricek et al. built the notion of AI assistants for data wrangling upon this idea; users supply constraints to the assistant to obtain a system-generated transformation satisfying constraints and minimizing wasted effort through trial and error to carry out practical cleaning and reshaping operations [1]. In practice, Wrangler is perhaps the best example of this kind of tool, being another mixed-initiative approach, which blends direct manipulation of data with an automatically inferred sequence of transformation steps, allowing analysts to incrementally craft the transformations for operations like splitting a field, removing noise, and transforming a column’s appearance, all while keeping the transformation transparent [2]. Despite the gains in usability these tools provide, they encounter challenges with implicit structural cues such as the use of merged headers for defining sections, separate groups of cells to define regions or sections of data, and lines/ separators to make report layouts clearer, which reinforces the need for more robust structure treatment and automated validation in ETL (Extract, Transform, Load) systems [5].

B. Programming by Example in Spreadsheets

Learning transformations from examples, known as programming-by-example methods, greatly reduces the burden of program specification. They perform best at cell-level and string-level operations common in typical spreadsheet manipulation. In [3], Gulwani et al. provide a survey on how to do data manipulations in spreadsheets using examples. The study demonstrates the effectiveness of program synthesis with examples to learn both simple string and complex cell transformation tasks that would otherwise be inefficient to write with manually defined formulas or scripts. However, the vast majority of spreadsheet data manipulations deal not just with value transformation but with transformation of spreadsheet structure itself, like reorganizing data from a crosstab format into record-based rows, maintaining hierarchical headers spanning multiple rows, or separating

subtotals from data points. To support such transformations, one must often deduce implicit relationships among values and recover latent table structure and semantic roles, which are underspecified by only a limited number of input-output examples and depend on a wide range of cues, such as layout, visual appearance, and semantics associated with values or context [5], [6].

C. Spreadsheet Table Understanding, Extraction, and Rule-Based Transformation

The research into spreadsheet table understanding focuses on identifying specific regions within a spreadsheet table, recognizing its headers, and assigning roles to each cell in order to represent the grid as a meaningful logical structure that can be transformed. Building upon this, rule-based transformations apply rules that explicitly encode human interpretation steps in analysis and processing. Shigarov and Mikhailov present a rule-based technique coupled with a table object model that creates two conceptual layers: one to store physical cell information and another to contain logical semantic items. This layering facilitates the transformation of varied tables into a standardized relational format [4]. Numerous rule-engine implementations exist that demonstrate how to use production rules to analyze table formatting, content in natural language, and spatial structure to derive meaningful relations between labels and their associated entries, then use such derived relationships to guide the generation of standard tables [6]. Recently, TabbyXL commercialized this vision by incorporating logical items, such as entries, labels, and categories, into a table object model, allowing user-written rules to be executed either by an engine that conforms to the JSR-94 specification (e.g., Drools) or by transforming a tailored rule language known as CRL into actual code for execution as Java programs [7]. While these works focus primarily on parsing and analysis, other related framework work points out the full requirements for a comprehensive extraction system. A robust extraction system must also handle data cleanup, information provenance/lineage tracking, and generation of relational and potentially linked data. This suggests the necessity for a uniform workflow that accommodates the diverse array of table structures [5].

D. Validation and Error Detection in Data Preparation

One of the largest problems in data preparation is that we can never confirm the correctness of output; countless variations are acceptable on the surface, each a plausible candidate depending on specific implicit or explicit constraints. In light of this, data preparation solutions generally suggest employing validation checks, error identification, and interactive assistance. Validations might simply involve executing schema type and constraint checks, but could extend to sophisticated domain rules and checks that verify reconciliation and logical consistencies. This challenge plays itself out in the humble spreadsheet, which has implicit rules obvious to a user but unstated: We should not export a row labelled "total" as an observation, column headers should properly identify the contents of each group in a group-and-sort operation, a column of names should not have duplicates. This is addressed in rule engines with explicit constraints to reject or filter records, for example, rejecting records whose first field contains "total" [6]. Another strategy is used in platform-based solutions: the comparison of transformed data against source truth, which checks whether required "Functional Items" have been extracted, whether the relationships extracted reflect source truth, etc. [7]. A problem with this is that it differentiates a failed operation that stops the process from producing output and an operation that, in effect, creates data that looks valid on the surface but is semantically garbage. Failure detection would ideally highlight errors in a repeatable manner, identifying specifically repeatable tests that can be built into a repeatable data transformation [5].

Existing studies have laid important foundations for spreadsheet data wrangling, programming-by-example, table understanding, rule-based interpretation, and validation [1]–[8], [10], [11]. These studies mainly focus on transformation support, table extraction, semantic interpretation, or output generation, rather than systematic failure localization across different stages of a spreadsheet transformation pipeline. This limitation is particularly important for spreadsheet data because spreadsheet meaning is often embedded in structural and visual cues, such as merged headers, section labels, subtotals, and multiple regions of related data [4]–[7], [10], [11]. Interactive wrangling and programming-by-example approaches reduce the burden of specifying spreadsheet transformations, but they provide limited explanation of where structural or semantic failures originate during the transformation process [1]–[3]. Rule-based table understanding approaches offer transparent and auditable mechanisms for extracting and transforming spreadsheet information, but their evaluations are commonly organized around specific interpretation or extraction tasks, such as header mapping, entity relationship extraction, or table-to-relational conversion, rather than a systematic stage-wise failure taxonomy [4]–[8]. Recent machine learning and transformer-based approaches demonstrate the importance of learning internal representations for table structure and semantics, including relational table representation, spatial structure, hierarchical structure, and tabular representation learning [12]–[16]. Nevertheless, rule-based table understanding remains relevant for task-specific transformation scenarios that require deterministic, auditable, reproducible, and domain-specific solutions [4], [6], [7]. Most existing approaches either address transformation for general data sources, focus on spreadsheet extraction, or emphasize validation after transformation. To date, limited attention has been given to stage-oriented failure characterization that systematically maps spreadsheet layout patterns, transformation pipeline stages, failure types, empirical severity scores, and corresponding examples. Therefore, this study addresses the gap in systematically characterizing stage-wise failures and their relationship with spreadsheet layout patterns, pipeline stages, failure types, severity levels, and concrete failure examples.

III. Problem Definition

A worksheet is defined as the grid C of cells, where each cell is an entity with a specific coordinate (r,c), value v, optional formula f, and associated visual features like borders and colorization g. A function T, defined as the rule-based pipeline, converts this worksheet S into a structured dataset D, the schema of which can be varied. Two critical processes in understanding table information are identifying which cells

play which functional role (i.e., metadata, headers, labels, data cells, or derived cells) and determining the table layout. This process precedes and makes possible the transformation step [10], [11].

A transformation pipeline failure point is the stage at which the applied rule set produces one of the following conditions:

- No output is produced when output is expected, for example, when the pipeline fails to detect any table region.
- The output is structurally incorrect, for example, when the detected region boundaries are wrong or when the predicted headers and data cells do not correspond to the intended table structure.
- The output is semantically incorrect, for example, when the extracted field names are wrong, the inferred data structure does not reflect the table structure, or the record granularity does not correspond to the perceived entities in the source table.

We can distinguish between overt pipeline failures (e.g., pipeline crash or empty return) and silent failures (e.g., the pipeline returns syntactically correct but conceptually incorrect results). Silent failures are the more problematic type for the automatic pipeline process.

The pipeline transformation *T* function described can, in practice, be understood as a set of stages that process and refine the intermediate table structure. Earlier stages aim to segment cells belonging to a potential table. Middle stages deduce header depth and structure and demarcate data regions. Finally, later stages arrange cells in canonical records and enforce data validity rules. This staged pipeline view is significant for diagnosing failure sources because failures often lead to subsequent failures: incorrect boundaries lead to incorrect headers, which lead to wrong field names, leading to wrong record keys and incorrect granularity. Accordingly, the failure diagnostic should record the specific stage at which a problem is introduced into the internal table representation, in addition to the stage at which it first becomes visible in the generated output.

IV. Methodology

Following Section III, the transformation *T* is defined as a deterministic sequential operation that takes a worksheet *S* and produces either a dataset D_{out} or an empty output accompanied by an error message. The multistage pipeline isolates where errors arise and when they become visible in the output.

A. Pipeline Stages

The proposed model consists of six stages, labelled A to F. At each stage, the pipeline processes an intermediate representation of the worksheet, produces an output, and records diagnostics. The staged view supports debugging because an early structural issue, such as failure to recognize a table header at Stage B, may later appear as a semantic error in Stage E. Table parsing also benefits from staged decomposition because visual layout cues can convey semantics that cannot be captured by schema interpretation alone [10]. Rule-based table extraction systems also estimate semantic links between cells from spatial, typographic, and textual cues, making staged diagnostics a natural output of such systems [6]. Existing rule engines expose actions for analysis, semantic interpretation, canonical output generation, and evaluation, which makes staged processing useful for both development and reproducible debugging [7], [5].

Stage A: Ingestion and Normalization. Cleans the cells: removes blank spaces at the start and end of a cell text, converts numerical values to a common format when possible, replaces spreadsheet formulas with their results when possible, records merged cells as objects that retain the anchor and span coordinates, and documents merge operations.

Stage B: Region Detection. Scans the worksheet and identifies rectangular regions, then filters and ranks them based on heuristics related to the density of text and contiguity. Output is a ranked list of candidate regions; the best region R, is prioritized for subsequent processing.

Stage C: Structural Interpretation. Estimates the depth k of the header, partitioning the region R into a header and a body, and infers structure patterns such as a cross-tabulated or a multi-header table section or other organizational structures within the table.

Stage D: Semantic Labelling. Identifies a row or column or a cell as a specific kind of label or data, or an annotation, and associates header labels with fields that'll be produced in the output. Uses blank value labelling rules and normalization for strings.

Stage E: Canonicalization. Converts the internal table structure into a flat, relational table structure, flattening hierarchical headers, defining keys for records, and potentially dropping non-observation rows such as subtotals.

Stage F: Validation. Validates the output against schema constraints, datatype constraints, and key constraints, and adds plausibility checks, such as an acceptable range for the number of records. Rejects clearly malformed results and highlights potential problems.

B. Baseline Rule Set

For this study, a representative rule set comprising six deterministic rules, denoted R1-R6, was developed to demonstrate the failure characterization technique. The rules were not designed to optimize table accuracy metrics; instead, they serve as diagnostic heuristics that reveal frequent failures when spreadsheet semantics are communicated through layout patterns. These rules resemble generic heuristic rules commonly found in industrial rule-based pipelines and can be improved after specific failures are identified [6], [8]. The same baseline rules are applied to all controlled cases to avoid confounding the results with rule variations and to ensure that the observed failures are attributable to layout patterns.

R1 (Region Selection): This rule detects and selects the densest rectangular region of non-empty cells to be treated as a candidate table.

R2 (Header Depth Inference): This rule estimates the depth of the table header (k) by heuristically inferring the depth of numerical columns, but it is capped at a maximum value.

R3 (Merged-Cell Propagation): This rule propagates the content of merged cells among the cells that they span.

R4 (Body Extraction): This rule extracts the table body, defined as the region of the spreadsheet directly below the estimated header.

R5 (Header Flattening): This rule "flattens" hierarchical table headers, such as when nested cells are used to represent broader categories. It accomplishes this by concatenating the cell content along each header path, delimited by forward slashes.

R6 (Subtotal Filtering): This rule eliminates rows that are identified as containing subtotal data (e.g., rows containing terms such as "Total" or "Subtotal") based on a simple keyword-matching pattern.

In our empirical study, we run these rules against each case in the controlled experiments and log all failures that arise. These failures are categorized based on our classification system of failure codes. We populate the failure matrix by noting the severity of the highest failure observed for each case-code pair. This matrix thus summarizes the failure pattern for the baseline rules on all of our carefully crafted test cases [7].

C. Controlled Spreadsheet Cases

Six controlled spreadsheet cases (S1-S6) were used to represent common semi-structured layout patterns that challenge rule-based approaches. The cases differ in three ways: 1) each contains a single worksheet, 2) each includes layout artifacts that must be handled, such as merged headers, interleaved subtotals, crosstab layouts, or multiple table regions, and 3) each has a ground-truth specification identifying the primary data region, header depth, fields, and observation formation rules, including the exclusion of subtotal rows (Table I).

TABLE I. CONTROLLED SPREADSHEET CASES

Case	Pattern	Key Characteristic	Expected Output Type
S1	Simple tidy table	Single header row, contiguous region, no merges	Direct row-to-record mapping
S2	Multi-row header	2-3 header rows, hierarchical column labels	Composite field construction from the header hierarchy
S3	Crosstab	Row categories × column categories with values	Reshape to relational (long-form records)
S4	Merged-cell semantics	Merged headers/group labels encode hierarchy	Hierarchy-preserving header interpretation
S5	Interleaved subtotals	Subtotal rows within body, mixed numeric and text	Granularity control (filter non-observations)
S6	Multi-region sheet	Multiple tables plus notes or separators	Region segmentation and correct table selection

The expected output for a single observation record is the corresponding record within a relational database table, stripped of decorative titles, any spacing lines, notes associated with the data, and subtotals or totals.

For tables that span multiple rows within the header region, the concatenated header fields for each data point become the singular header label for the data, and this concatenation follows a specific, consistent ordering (matching R5). Cross-tabulated data are transformed into a long-form configuration in which the reported values each represent a corresponding cell identified by their respective row and column values.

D. Evaluation and Failure Scoring Procedure

Checks were implemented aligned with the three outcome categories from Section III to operationalize failure reporting using taxonomy codes (Section III). For each check, we record a binary pass/fail status and a descriptive diagnostic message that indicates which check was performed, when it became apparent in the pipeline (stage), and which baseline rule identifiers are involved. For a particular table, the failure matrix (Table III) sums the severity score across all failed checks for each taxonomy code. An occurrence is thus characterized by the tuple (table ID, taxonomy code) for which the failure matrix reports the highest severity score.

No-output failures. We identify a no-output failure when the pipeline outputs an empty table (while there is a valid input table). This can occur either if Stage B fails to find at least one candidate exceeding a given threshold or if Stage F rejects all produced candidate regions. A no-output failure likely indicates issues arising during the region discovery stage (Stage B) and enforced by the selection process (Stage F).

Structural incorrectness. We consider the output to be structurally incorrect when the extracted region differs from the specified or intended region. Additionally, we note a structural error when the automatically derived header depth lies outside the range of [0, max header depth] or when the output structure violates basic requirements, such as the absence of essential columns or inconsistencies between headers and bodies. Failures of this nature are predominantly associated with problems in stages B and C.

Semantic incorrectness. We deem the output semantically incorrect when it breaks assumptions about its correct interpretation and meaning, despite possessing a valid structure. This encompasses various issues: null values or composite key incoherence, duplicated keys resulting from improper propagation of label interpretations, omission of row-level summary figures (subtotals), and errors in flattening cross-tabulations, which may misassign values to cell-group combinations. This classification explicitly encompasses subtle semantic problems that do not trigger schema checks but still violate semantic constraints. Semantic errors frequently emerge from issues in stages D and E and become observable during Stage F.

Severity scoring. We quantify the severity of each recorded failure: 0 (no failure), 1 (partial degradation, fixable with a localized rule modification), and 2 (critical failure requiring new capability, e.g., reasoning across multiple regions, explicitly representing a hierarchy, etc.). These scores form the entries in the failure matrix in Section VI.

E. Pseudo Algorithm (Algorithm 1)

Algorithm 1: Stage-wise transformation and failure logging

Input: Worksheet S; baseline rules R1..R6; failure taxonomy FA2-FF1.

Output: Extracted dataset Dout (or empty) and failure log L

Normalize S (clean cell content; represent merged spans).

Detect candidate regions; select R using R1.

If no valid region is selected, log a no-output failure (taxonomy code in Stage B) and return (emptyset, L).

Infer header depth k using R2; partition R into header/body.

Propagate merged header labels using R3; assign roles/labels and construct field labels.

Flatten and canonicalize using R4–R6 (including hierarchical header flattening and subtotal filtering).

Validate output using schema/key/plausibility checks.

For each failed check, append to L a record Failure logging:

For each failed check, append to L a record (case, taxonomy code, outcome type, stage visible, severity, diagnostic), where outcome type is one of {no-output, structural, semantic} and severity is one of {0,1,2}.

Return: Output (Dout, L). The failure matrix (Table III) is populated by taking the maximum severity for each case (case, taxonomy code) pair.

V. Failure Taxonomy

Existing approaches for spreadsheet table understanding or rule-based transformation largely focus on region detection, role extraction, table semantics, or simply generate relational outputs, while this taxonomy aims to localise failure and describe failure propagation during the pipeline. It not only labels the visible symptom in an incorrect output but also helps to determine at which stage of processing the error occurred because failure might originate during region detection or interpretation stage, but become visible later, only during normalisation or validation, such as invalid boundaries show up as error during semantics labelling or remain unseen as silent failure during validation. The taxonomy is therefore a useful addition to the previous work on table understanding, which serves a diagnostic purpose, linking the types of failure, the stages of the pipeline where they originate, the patterns that trigger them in the spreadsheet, and the effects they cause in the output result. This stage-oriented view also provides for a fairer or more reasonable comparison between the rule-based pipelines because failures can be reported in terms of origin, type, and severity, rather than random output error alone.

The taxonomy supports two primary uses. First, it enables stage localization by identifying which component introduced a fault into the pipeline. Second, it supports targeted mitigation by distinguishing faults that can be repaired through rule adjustment from faults that require more advanced capabilities, such as explicit hierarchy-aware header representation or support for multiple regions and region selection.

TABLE II. Failure Taxonomy

Code	Stage	Failure Type	Symptom
FA2	Ingestion	Merge mis-resolution	Merged spans expanded incorrectly, and hierarchy spans were lost
FB1	Region detection	Under-selection	Table region clipped and misses rows or columns
FB2	Region detection	Over-selection	Notes or adjacent blocks included as part of the table
FC1	Structural interpretation	Header depth error	Wrong number of header rows or columns inferred
FC2	Structural interpretation	Boundary ambiguity	Header-body boundary misidentified due to mixed types
FC3	Structural interpretation	Multi-region confusion	The wrong table was selected when multiple tables exist
FD1	Semantic labeling	Role mislabeling	Headers treated as data or notes treated as fields
FD2	Semantic labeling	Hierarchy collapse	Parent-child header relations lost during flattening
FE1	Canonicalization	Wrong granularity	Subtotals or group headers are emitted as observations
FE2	Canonicalization	Key construction failure	Composite keys inconsistent or missing
FF1	Validation	Silent semantic error	Output passes schema checks but is semantically wrong

VI. Failure Matrix and Analysis

The pipeline (R1-R6) was developed in Java using Apache POI for XLSX file handling, including access to merged-cell metadata. Each controlled case (S1-S6) was processed using the same rule-execution settings. Therefore, differences in behaviour are attributable to layout variation rather than rule configuration or implementation changes. During rule execution, the pipeline logs stage-visible failure events. Each event is assigned a taxonomy code (FA2-FF1) and a severity score from 0 to 2, as described in Section IV-D. Table III is constructed by collecting logs from all controlled cases and reporting the maximum severity for each pair of controlled case and taxonomy code. The raw logs were exported in CSV format to support reproducibility and future extension of the rule-based execution process [7], [5].

TABLE III. Failure Matrix

Failure Code	S1	S2	S3	S4	S5	S6
FA2	0	2	0	2	0	2
FB1	0	0	0	0	0	0
FB2	0	0	0	0	1	1
FC1	1	0	1	0	1	1
FC2	0	0	2	0	0	2
FC3	0	0	2	0	2	2
FD1	0	0	2	0	2	0
FD2	1	1	1	1	0	1
FE1	0	0	0	0	2	2
FE2	0	0	0	0	0	0
FF1	0	0	2	0	2	2

For consistent reporting, we generated a matrix that maps taxonomy codes (Table II) to the specific controlled cases (S1-S6). Table III reports measured severity levels ranging from 0 to 2, where a level of 0 represents a successfully covered case. Severity scores were derived by executing the baseline rule pipeline (R1-R6) against each of the test cases, logging each triggered failure according to the procedure discussed in Section IV-D. For every combination of (case, taxonomy-code) that triggered a failure, the matrix stores the highest severity observed among the triggering checks.

Observed patterns. The matrix highlights three predominant failure clusters:

1) Merged header and hierarchical issues (FA2, FD2). A set of sheets exhibits hierarchical or merged header formatting (S2, S4, and S6) that results in consistent FA2 (severity 2) and FD2 (severity 1). Expanding a merged header and subsequent flattening of the header hierarchy preserves the ability to parse structure, but results in incorrect parent-child header relations.

2) Multi-block and boundary problems (FC2, FC3). The collection of mixed-structure worksheets (S3, S5, and S6) consistently yields FC3 (severity 2), representing ambiguity when several blocks, borders, or sections coexist on a single sheet. For S3 and S6, this also comes with FC2 (severity 2) because determining header, body separation breaks down when layout hints, and data types are interwoven (including section title entries, repeated headers, and values appearing at or near the header rows).

3) Semantic corruption and blind errors (FD1, FE1, FF1). When mixed-structure patterns are encountered, errors can propagate down to later analysis stages. The cases S3 and S5 incur FD1 (severity 2), failing to correctly attribute the roles of headers that appear in duplicate sections and data rows that have been identified as part of the header. Cases S5 and S6 receive FE1 (severity 2), showing incorrect cardinality resulting from rows formatted as subtotals and various summarized values or artifacts being included as raw observations. These failures frequently result in FF1 (severity 2) for S3, S5, and S6; although a portion of the table might appear correctly formatted, the extracted semantic structure is flawed.

Cases with less impact. The only case that shows mild consequences is S1 (which has a single tidy table arrangement), generating a minor header depth problem (FC1=1) and an indicator of incomplete hierarchical parsing (FD2=1), which is indicative that such simple contiguous tables are correctly handled by the parser's baseline mode. For cases S2 and S4, the main failures are found in the header structure hierarchy handling (FA2/FD2) and not in downstream semantic extraction.

Code occurrences. There were zero occurrences of FB1 and FE2 codes in the experiments under the baseline configuration of the table extractor. FB1 is rare because the extractor's default bounding behaviour would ensure all relevant non-empty cells are within the defined scope, FE2 only triggers in the context of user-defined, multipart keys being constructed, and stronger tests for key consistency than are required to extract Table III in this study.

VII. Implications and Recommendations

The analysis in Table III indicates that more than half of the observed failures stem from errors that commonly arise in patterns of spreadsheets rather than from a deficiency in a rule covering a rare instance. This situation indicates the necessity to substantially extend the capabilities of the transformation pipeline to handle widespread errors more effectively. The common failure categories were used in our analysis to recommend actions for each stage of the transformation pipeline (stages A through F) according to our taxonomical categories (FA2 through FF1).

A. Explicit Representation of Header Hierarchy (FA2, FD2)

The cases that utilize merged or hierarchical headers (S2, S4, S6) often correlate to misinterpretation of merge resolution (FA2) and collapse of hierarchy (FD2). The method by which we treat merged cells cannot be limited to mere preprocessing. A hierarchical tree or graph that captures the header information should store parent-child relationships between header cells and provide the appropriate content when flattening occurs. The flattening logic should produce consistent, determined outcomes, but this process requires knowledge of the header hierarchy. Merging cells should occur only after establishing a structural hierarchy. For example, "2023" (Level 2) should precede "Revenue" (Level 1), yielding "2023 - Revenue" rather than "Revenue - 2023", thus removing ambiguity and duplicated "Revenue" headers.

B. Multi-Block Reasoning Before Header Inference (FC2, FC3)

Misinterpreting regions (FC3) typically arises in our mixed layout datasets (S3, S5, S6) and is associated with ambiguous region boundaries (FC2). Header and table selection should not be restricted based on only a single bounding box. Often, worksheets contain irrelevant notes or extra separators or consist of multiple independent blocks of information. We add a distinct segmentation phase that identifies dense regions visually by looking for many empty rows, columns, or explicit section separators. For each such region, we assess how likely it is to contain a table before we run depth-based header heuristics. Section labels like "Class A" and "Class B" should not be treated as headers; rather, they should be treated as meta information to the table situated underneath them, to ensure that when headers are flattened, a string such as "Class A" in a data row does not accidentally become a column header.

C. Enforcement of Role Labeling and Granularity Control (FD1, FE1)

Mislabelling the roles of cells (FD1) and issues in granularity (FE1) tend to happen with recurring headers and subtotal/total-like rows in S3, S5, and S6. If the parser treats these repeating structures as a data row, it mislabels roles and proceeds to incur granularity errors. Two principal controls are proposed for addressing such problems: first, identify repeated headers by detecting matches between the inferred content of a header row and some data within a table region and second, introduce a more stringent granularity filter capable of

distinguishing true data observations from summary data points, such as total, subtotal, average, etc., based on keyword identification as well as structure, formatting and the density of numbers.

D. Semantic Invariant Checks for Silent Failure Detection (FF1)

Silent semantic errors (FF1), where the output is still tabular but meaningless in its content (S3, S5, S6), imply that we must supplement our Stage F validation logic with semantic invariant checks in addition to validating the presence of a schema and data types. Such invariants can be used to: detect incomplete keys (by checking whether key-like columns possess relatively few missing values), define the range of the number of records within each section (e.g., within a 10-20% variation based on prior inspection), verify the match between the inferred section labels and the data within those sections and find any header labels that happen to appear inside the data portion of a table. All these checks can be implemented easily without any ML or deep learning model, and can go a long way toward avoiding incorrect outputs.

E. Summary: Local Rule Tuning Versus Capability Enhancements

Based on the structure of the failure matrix, we have found that a large portion of the errors fall into two distinct types: failures resolvable by fine-tuning existing rules (e.g., expanding keyword sets, modifying numerical thresholds) and failures requiring significant improvements in the pipeline capability (e.g., header representation that handles hierarchy, block-level segmentation). Successfully increasing the robustness of spreadsheet parsers in production systems requires a combined approach of: representation of explicit structure (header spans, block segmentation), better controls for cell role and granularity, and semantic invariant checks to prevent silent misrepresentation of data.

VIII. Threats to Validity

The findings are tied to a specific deterministic baseline rule configuration, R1-R6. The specific events that trigger failures may shift under a different sequence of pipeline stages or under different rule parameters. For example, changing a header-depth heuristic or keyword list could cause a failure to be reported at one stage rather than another. Therefore, to keep results comparable, the rules set must remain consistent and deterministic across all inputs. The matrix in Table III reports the maximum severity score for each case-code pair, ensuring that a triggered failure is not hidden if it appears earlier in the pipeline.

A. Construct Validity

The taxonomy codes (FA2–FF1) and severity scores (0–2) defined in this study provide an operational definition of what is treated as a table-structure error. However, spreadsheet cells may be interpreted differently by different users because their meaning is often shaped by layout, domain context, and the intended use of the worksheet. In the controlled cases, each cell was treated according to an explicit row-and-column interpretation so that the expected output could be evaluated consistently. Nevertheless, some spreadsheet entries may have ambiguous semantic roles. For example, a date-like entry may be interpreted either as a section note or as a data value depending on the user's expectation and domain context. One user may regard such a row as a contextual notice that should be ignored, whereas another user may regard it as part of the data that should be parsed. Therefore, the failures reported in this study are limited to the explicitly checked conditions defined in the evaluation procedure. The severity scores should also be interpreted according to the adopted rubric: a score of 1 indicates a failure that can be repaired through localized adjustment of existing rule parameters, such as threshold tuning or keyword refinement, whereas a score of 2 indicates a missing pipeline capability that requires additional functionality, such as explicit hierarchy representation or multi-block reasoning across related table regions.

B. External Validity

This study considers six common forms of data organization: simple tidy tables, tables with multi-row headers, sectioned blocks, a mixture of structures, report style layout, and a header and then region-level nested lists. However, we do not know the real distribution of types of tables, and this study is only concerned with tables that adhere to these patterns, for instance, more formal types like PivotTable structures, which include formulas that span arrays, multi-sheet workbooks whose information might be pulled from external locations, or other spreadsheets whose formatting goes beyond mere style, are unexamined. Therefore, the pattern of observed failures in Table III should not be inferred as representative statistics of spreadsheet errors in all circumstances; rather, it should be seen as an illustration of fragile components in spreadsheet structures. This future study could further investigate and expand our work through more diverse test data: more domains and actual spreadsheets that include varied structures.

C. Reliability and Reproducibility

The correctness and ability to repeat this process may be affected by nuances of the implementation details, including variations between libraries, different ways of handling, and unified formats of merged cells (which can be consistent throughout various spreadsheet editors but still represent variability between the underlying editors), and different interpretations of spreadsheet data when normalized. As such, the evaluation process is structured to be deterministic with a controlled, baseline system that details and outputs errors that Table III summarizes under an unambiguous reporting of a failure type (maximum per cell code per case). With other rule-based or data parsers, different findings might arise because other techniques or assumptions were applied, especially concerning heuristics used in region segmentation or identification of cell headers of certain types, in addition to possible variation in algorithm implementations of parsing merged cells. Regardless, these findings provide a systematic and reproducible way to categorize, evaluate, and situate failure modes for conversion processes of spreadsheets to tabular form.

IX. Conclusion and Future Work

This study introduces a failure analysis framework for rule-based spreadsheet-to-relational transformations. The framework uses a stage-based taxonomy spanning ingestion,

region detection, structural interpretation, semantic labeling, canonicalization, and validation. The taxonomy enables failures to be attributed to specific causes and processing stages. By controlling six spreadsheet cases (S1-S6) and using a fixed baseline implementation, the study constructs a failure matrix that records the maximum severity for each case-code pair. The results show that failures are concentrated in three groups: 1) handling merged cells and hierarchical headers (FA2, FD2), 2) handling separate table blocks and ambiguous boundaries (FC2, FC3), and 3) downstream semantic errors and wrong granularities that can lead to silent data corruption (FD1, FE1, FF1). These findings indicate that many errors arise not from trivial rule mistakes but from the inability of extraction systems to represent and enforce explicit structural models.

Future work will extend the controlled spreadsheet cases in two directions: by including more diverse real-world spreadsheets from different domains and by evaluating alternative baseline implementations to determine whether the observed issues can be mitigated through rule adjustment or whether they stem from deeper system constraints. This broader evaluation is consistent with recommendations in [5], which emphasize spreadsheet extraction pipelines that integrate recognition, interpretation, data cleaning, and evaluation across different table formats. Subsequent mitigation work will focus on three areas motivated by the most prevalent failure types in Table III: designing header models that preserve merged-cell and parent-child relationships without reducing them to single-level headers, performing table-block segmentation and ranking before header analysis, and introducing semantic invariant checks such as key completeness, section consistency, and value reconciliation to prevent silent errors. Finally, the taxonomy and logging tools will be released as a benchmark kit so that other rule-based systems can report failures uniformly, compare mitigation strategies, and demonstrate which improvements address which failure codes. In the longer term, the work will move beyond failure analysis toward the semantics of canonical tables, such as linking extracted values to external knowledge resources after the underlying structural and semantic failures have been substantially reduced [9].

REFERENCES

[1] Tomas Petricek, Gerrit J. J. van den Burg, Alfredo Nazabal, Taha Ceritli, Ernesto Jimenez Ruiz, and Christopher K. I. Williams, “AI Assistants A Framework for Semi-automated Data Wrangling,” IEEE Transactions on Knowledge and Data Engineering, Vol. 35, No. 9, 2023, pp. 9295–9306.

[2] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer, “Wrangler Interactive Visual Specification of Data Transformation Scripts,” Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2011, pp. 3363–3372.

[3] Sumit Gulwani, William R. Harris, and Rishabh Singh, “Spreadsheet Data Manipulation Using Examples,” Communications of the ACM, Vol. 55, No. 8, August 2012, pp. 97–105.

[4] Alexey O. Shigarov and Andrey A. Mikhailov, “Rule-Based Spreadsheet Data Transformation from Arbitrary–Relational Tables,” Information Systems, Vol. 71, 2017, pp. 123–136.

[5] Alexey O. Shigarov, Vasiliy V. Khristyuk, Viacheslav V. Paramonov, Alexandr Yu. Yurin, and Nikita O. Dorodnykh, “Toward Framework for Development of Spreadsheet Data Extraction Systems,” CEUR Workshop Proceedings, Vol. 2221, 2018, pp. 1–7.

[6] Alexey O. Shigarov, “Table Understanding Using a Rule Engine,” Expert Systems with Applications, Vol. 42, No. 2, 2015, pp. 929–937.

[7] A. Shigarov, V. Khristyuk, and A. Mikhailov, “TabbyXL Software Platform for Rule-Based Spreadsheet Data Extraction and Transformation,” SoftwareX, Vol. 10, 2019, Article 100270.

[8] A. Shigarov, “Rule-Based Table Analysis and Interpretation,” in Information and Software Technologies, 21–22 October 2015, Kaunas, Lithuania, Communications in Computer and Information Science, Vol. 538, Springer, 2015, pp. 175–186.

[9] Nikita O. Dorodnykh and Aleksandr Yu. Yurin, “Towards a universal Approach for Semantic Interpretation of Spreadsheets Data,” Proceedings of the 24th International Database Engineering and Applications Symposium, 12–14 August 2020, Seoul, Republic of Korea, 2020, 9 pages.

[10] Sara Bonfitto, Elena Casiraghi, and Marco Mesiti, “Table Understanding Approaches for Extracting Knowledge from Heterogeneous Tables,” WIREs Data Mining and Knowledge Discovery, 2021.

[11] Majid Ghasemi Gol, Jay Pujara, and Pedro A. Szekely, “Tabular Cell Classification Using Pre-trained Cell Embeddings,” Proceedings of the 2019 IEEE International Conference on Data Mining, 2019.

[12] P. Yin, G. Neubig, W. Yih, and S. Riedel, “TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data,” Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 8413–8426.

[13] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, “TURL: Table Understanding through Representation Learning,” Proceedings of the VLDB Endowment, Vol. 14, No. 3, 2020, pp. 307–319.

[14] Z. Wang, H. Dong, R. Jia, J. Li, Z. Fu, S. Han, and D. Zhang, “TUTA: Tree-Based Transformers for Generally Structured Table Pre-training,” Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 1780–1790.

[15] H. Iida, D. Thai, V. Manjunatha, and M. Iyyer, “TABBIE: Pretrained Representations of Tabular Data,” Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2021, pp. 3446–3456.

[16] G. Badaro, M. Saeed, and P. Papotti, “Transformers for Tabular Data Representation: A Survey of Models and Applications,” Transactions of the Association for Computational Linguistics, Vol. 11, 2023, pp. 227–249.