

1 **Spreadsheet Data Extractor (SDE): A Performance-Optimized, User-Centric Tool**
2 **for Transforming Semi-Structured Excel Spreadsheets into Relational Data**

6 ANONYMOUS AUTHOR(S)

7 SUBMISSION ID:

10 Spreadsheets are widely used across domains, yet their human-oriented layouts (merged cells, hierarchical headers, multiple tables)
11 hinder automated extraction. We present the Spreadsheet Data Extractor (SDE), a user-in-the-loop system that converts semi-structured
12 sheets into structured outputs without programming. Users declare the structure they perceive; the engine broadcasts selections
13 deterministically and renders results immediately. Under the hood, SDE employs incremental loading, byte-level XML streaming
14 (avoiding DOM materialisation), and viewport-bounded rendering.

15 Optimised for time-to-first-visual, SDE delivers about **70 \times** speedup over Microsoft Excel when opening a selected sheet in a
16 large real-world workbook; under workload-equivalent conditions (single worksheet, full parse) it remains about **10 \times** faster, while
17 preserving layout fidelity. These results indicate SDE's potential for reliable, scalable extraction across diverse spreadsheet formats.

19 CCS Concepts: • Applied computing → Spreadsheets; • Information systems → Data cleaning; • Software and its engineering
20 → Extensible Markup Language (XML); • Human-centered computing → Graphical user interfaces; • Theory of computation
21 → Data compression.

23 Additional Key Words and Phrases: Spreadsheets, Data cleaning, Relational Data, Excel, XML, Graphical user interfaces

25 ACM Reference Format:

27 Anonymous Author(s). 2026. Spreadsheet Data Extractor (SDE): A Performance-Optimized, User-Centric Tool for Transforming
28 Semi-Structured Excel Spreadsheets into Relational Data. In *Proceedings of Conference on Human Factors in Computing Systems (ACM*
29 *CHI '26)*. ACM, New York, NY, USA, 25 pages. <https://doi.org/10.1145/XXXXXXX.XXXXXXX>

31 **1 Introduction**

33 Spreadsheets underpin workflows across healthcare [5], nonprofit organizations [16, 21], finance, commerce, academia,
34 and government [10]. Despite their ubiquity, reliably analyzing and reusing the data they contain remains difficult
35 for automated systems. In practice, many real-world spreadsheets are organized for human consumption [20] and
36 therefore exhibit layouts with empty cells, merged regions, hierarchical headers, and multiple stacked tables. While
37 such conventions aid human reading [15], they hinder machine readability and complicate extraction, integration, and
38 downstream analytics.

40 The reliance on spreadsheets as ad-hoc solutions poses several limitations:

- 42 • **Data Integrity and Consistency:** Spreadsheets are prone to errors, such as duplicate entries, inconsistent
43 data formats, and inadvertent modifications, which can compromise data integrity. [1, 7]

45 Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not
46 made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components
47 of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on
48 servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

49 © 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

50 Manuscript submitted to ACM

52 Manuscript submitted to ACM

- 53 • **Scalability Issues:** As datasets grow in size and complexity, spreadsheets become less efficient for data storage
54 and retrieval, leading to performance bottlenecks. [14, 19]
- 55 • **Limited Query Capabilities:** Unlike databases, spreadsheets lack advanced querying and indexing features,
56 restricting users from performing complex data analyses.
- 57
- 58

59 Transitioning from these ad-hoc spreadsheet solutions to standardized database systems offers numerous benefits:
60

- 61 • **Enhanced Data Integrity:** Databases enforce data validation rules and constraints, ensuring higher data
62 quality and consistency.
- 63 • **Improved Scalability:** Databases are designed to handle large volumes of data efficiently, supporting complex
64 queries and transactions without significant performance degradation.
- 65 • **Advanced Querying and Reporting:** Databases provide powerful querying languages like SQL, enabling
66 sophisticated data analysis and reporting capabilities.
- 67 • **Seamless Integration:** Databases facilitate easier integration with various applications and services, promoting
68 interoperability and data sharing across platforms.
- 69
- 70
- 71

72 Given the abundance of spreadsheet data and the clear advantages of database systems, there is a pressing need for
73 tools that bridge these formats. Automated, accurate extraction from spreadsheets into relational representations is
74 essential.

75 Prior work introduced a tool for extracting data from Excel files [3]. While effective, that system exhibited performance
76 issues on large workbooks and imprecise rendering of cell geometry, which limited usability for complex, real-world
77 files. The user interface was also fragmented, requiring context switches between hierarchy definition and output
78 preview.

79 This paper presents the *Spreadsheet Data Extractor (SDE)*, which transforms data from semi-structured spreadsheets
80 into machine-readable form. Users declare structure directly via cell selections—no programming is required. The design
81 addresses the above limitations and supports large, irregular spreadsheets with predictable, interactive performance.

82 1.1 Contributions

- 83 (1) **Unified interaction.** SDE integrates the selection hierarchy, worksheet view, and output preview into a single
84 interface, reducing context switching and interaction count.
- 85 (2) **DOM-free, byte-level worksheet parsing.** SDE implements a custom parser that operates directly on
86 XLSX worksheet bytes, avoiding DOM construction and regular-expression passes over decoded strings; this
87 substantially reduces memory footprint, parsing time, and improves robustness on large (“bloated”) sheets.
- 88 (3) **Incremental loading.** Worksheets are loaded and parsed on demand from the XLSX archive, enabling near-
89 instant open times and interactive latency on large files (quantified in 7. Evaluation).
- 90 (4) **Excel-faithful rendering.** Row heights, column widths, and merged regions are recovered from the XML to
91 render worksheets faithfully to Excel, improving user orientation.
- 92 (5) **Viewport-bounded rendering.** The renderer draws only cells intersecting the viewport; selection queries are
93 pruned with cached axis-aligned bounding boxes. Together these keep per-frame work proportional to what is
94 visible rather than to sheet size.
- 95
- 96
- 97
- 98
- 99
- 100
- 101
- 102
- 103

Name	Technologies	Output	Accessibility	Frequency
DeExcelerator	heuristics	relational data	partially open source no access to GUI code	last publication 2015
FlashRelate	AI programming-by-example	relational data	proprietary, no access	last publication 2015
Senbazuru	AI	relational data	partially open source no access to GUI code	last commit 2015
XLIndy	AI	relational data	no access	discontinued
TableSense	AI	diagrams	proprietary, no access	last commit 2021
Aue et al. (converter)	User-centric Selection-Workflow	relational data	no public repository	last publication 2024

Table 1: Spreadsheet Data Extractor counterparts

2 Related Work

The extraction of relational data from semi-structured documents, particularly spreadsheets, has garnered significant attention due to their ubiquitous use across domains such as business, government, and scientific research. Several frameworks and tools have been developed to address the challenges of converting flexible spreadsheet formats into normalized relational forms suitable for data analysis and integration as summarized in Table 1.

2.1 Aue et al.'s Converter

Aue et al. [3] developed a tool to facilitate data extraction from Excel spreadsheets by leveraging the Dart *excel* package [8] to process .xlsx files. This tool allows users to define data hierarchies by selecting relevant cells containing data and metadata. However, the approach faced significant performance bottlenecks due to the *excel* package's requirement to load the entire .xlsx file into memory, resulting in slow response times, particularly for large files.

In addition to memory issues, the tool calculated row heights and column widths based solely on cell content, ignoring the dimensions specified in the original Excel file. This led to rendering discrepancies between the tool and the original spreadsheet. Furthermore, the tool rendered all cells, regardless of their visibility within the viewport, significantly degrading performance when handling worksheets with large numbers of cells.

2.2 DeExcelerator

Eberius et al. [11] introduced **DeExcelerator**, a framework that transforms partially structured spreadsheets into first normal form relational tables using heuristic-based extraction phases. It addresses challenges such as table detection, metadata extraction, and layout normalization. While effective in automating normalization, its reliance on predefined heuristics limits adaptability to heterogeneous or unconventional spreadsheet formats, highlighting the need for more flexible approaches.

2.3 XLIndy

Koci et al. [12] developed **XLIndy**, an interactive Excel add-in with a Python-based machine learning backend. Unlike DeExcelerator's fully automated heuristic approach, XLIndy integrates machine learning techniques for layout inference and table recognition, enabling a more adaptable and accurate extraction process. XLIndy's interactive interface allows users to visually inspect extraction results, adjust configurations, and compare different extraction runs, facilitating

157 iterative fine-tuning. Additionally, users can manually revise predicted layouts and tables, saving these revisions as
 158 annotations to improve classifier performance through (re-)training. This user-centric approach enhances the tool's
 159 flexibility, allowing it to accommodate diverse spreadsheet formats and user-specific requirements more effectively
 160 than purely heuristic-based systems.
 161

163 2.4 FLASHRELATE

165 Barowy et al. [4] presented **FLASHRELATE**, an approach that empowers users to extract structured relational data from
 166 semi-structured spreadsheets without requiring programming expertise. FLASHRELATE introduces a domain-specific
 167 language, **FLARE**, which extends traditional regular expressions with spatial constraints to capture the geometric
 168 relationships inherent in spreadsheet layouts. Additionally, FLASHRELATE employs an algorithm that synthesizes
 169 FLARE programs from a small number of user-provided positive and negative examples, significantly simplifying the
 170 automated data extraction process.
 171

172 FLASHRELATE distinguishes itself from both DeExcelerator and XLIndy by leveraging programming-by-example
 173 (PBE) techniques. While DeExcelerator relies on predefined heuristic rules and XLIndy incorporates machine learning
 174 models requiring user interaction for fine-tuning, FLASHRELATE allows non-expert users to define extraction patterns
 175 through intuitive examples. This approach lowers the barrier to entry for extracting relational data from complex
 176 spreadsheet encodings, making the tool accessible to a broader range of users.
 177

179 2.5 Senbazuru

180 Chen et al. [6] introduced **Senbazuru**, a prototype Spreadsheet Database Management System (SSDBMS) designed to
 181 extract relational information from a large corpus of spreadsheets. Senbazuru addresses the critical issue of integrating
 182 data across multiple spreadsheets, which often lack explicit relational metadata, thereby hindering the use of traditional
 183 relational tools for data integration and analysis.
 184

185 Senbazuru comprises three primary functional components:
 186

- 187 (1) **Search:** Utilizing a textual search-and-rank interface, Senbazuru enables users to quickly locate relevant
 188 spreadsheets within a vast corpus. The search component indexes spreadsheets using Apache Lucene, allowing
 189 for efficient retrieval based on relevance to user queries.
 190
- 191 (2) **Extract:** The extraction pipeline in Senbazuru consists of several stages:
 - 192 • **Frame Finder:** Identifies data frame structures within spreadsheets using Conditional Random Fields
 193 (CRFs) to assign semantic labels to non-empty rows, effectively detecting rectangular value regions and
 194 associated attribute regions.
 195
 - 196 • **Hierarchy Extractor:** Recovers attribute hierarchies for both left and top attribute regions. This stage
 197 also incorporates a user-interactive repair interface, allowing users to manually correct extraction errors,
 198 which the system then generalizes to similar instances using probabilistic methods.
 199
 - 200 • **Tuple Builder and Relation Constructor:** Generates relational tuples from the extracted data frames
 201 and assembles these tuples into coherent relational tables by clustering attributes and recovering column
 202 labels using external schema repositories like Freebase and YAGO.
 203
 204 (3) **Query:** Supports basic relational operations such as selection and join on the extracted relational tables, enabling
 205 users to perform complex data analysis tasks without needing to write SQL queries.
 206

209 Senbazuru’s ability to handle hierarchical spreadsheets, where attributes may span multiple rows or columns without
210 explicit labeling, sets it apart from earlier systems like DeExcelerator and XLIIndy. By combining learning methods
211 with an interactive repair workflow, Senbazuru improves extraction accuracy and consistency and produces relations
212 suitable for integration into databases.
213

214 2.6 TableSense

215 Dong et al. [9] developed **TableSense**, an end-to-end framework for spreadsheet table detection using Convolutional
216 Neural Networks (CNNs). TableSense addresses the diversity of table structures and layouts by introducing a com-
217 prehensive cell featurization scheme, a Precise Bounding Box Regression (PBR) module for accurate boundary detection,
218 and an active learning framework to efficiently build a robust training dataset.
219

220 While **DeExcelerator**, **XLIIndy**, **FLASHRELATE**, and **Senbazuru** focus primarily on transforming spreadsheet
221 data into relational forms through heuristic, machine learning, and programming-by-example approaches, **TableSense**
222 specifically targets the accurate detection of table boundaries within spreadsheets using deep learning techniques.
223 Unlike region-growth-based methods employed in commodity spreadsheet tools, which often fail on complex table
224 layouts, TableSense achieves superior precision and recall by leveraging CNNs tailored for the unique characteristics of
225 spreadsheet data. However, TableSense focuses on table detection and visualization, allowing users to generate diagrams
226 from the detected tables but does not provide functionality for exporting the extracted data for further analysis.
227
228

229 2.7 Comparison and Positioning

230 A direct head-to-head comparison was not possible due to the lack of artifacts, because for the UI-oriented systems
231 **FLASHRELATE**, **Senbazuru**, **XLIIndy**, and **DeExcelerator**, we contacted the authors mentioned in the publications
232 to obtain research artifacts (source code, UI prototypes). As of the submission deadline, we had either not received any
233 responses or that the project was discontinued; we could not find publicly accessible UI artifacts or runnable packages.
234

235 Moreover, unlike the aforementioned tools that rely on heuristics, machine learning, or AI techniques—which
236 can introduce errors requiring users to identify and correct—we adopt a user-centric approach that gives users full
237 control over data selection and metadata hierarchy definition. While this requires more manual input, it eliminates the
238 uncertainty and potential inaccuracies associated with automated methods. To streamline the process and enhance
239 efficiency, our tool includes user-friendly features such as the ability to duplicate hierarchies of columns and tables, and
240 to move them over similar structures for reuse, reducing the need for repetitive configurations.
241

242 By combining the strengths of manual control with enhanced user interface features and performance optimizations,
243 our tool offers a robust and accessible solution for extracting relational data from complex and visually intricate
244 spreadsheets. These enhancements not only improve performance and accuracy but also elevate the overall user
245 experience, making our tool a valuable asset for efficient and reliable data extraction from diverse spreadsheet formats.
246

247 3 Design Philosophy

248 Spreadsheet tables are heterogeneous, noisy, and locally structured in ways that are hard to infer reliably with fully
249 automatic extraction. Our goal is not to replace the analyst with an opaque model, but to *amplify* their judgment: the
250 user points to the structure they already perceive (regions, labels, value columns), and the system guarantees a faithful,
251 auditable transformation into a relational view. Instead of automatically extracting and then searching for mistakes, we
252 invert the workflow: *select first, broadcast deterministically, render immediately*. This keeps discrepancies visible at the
253 point of selection, where they can be corrected with minimal context switches.
254

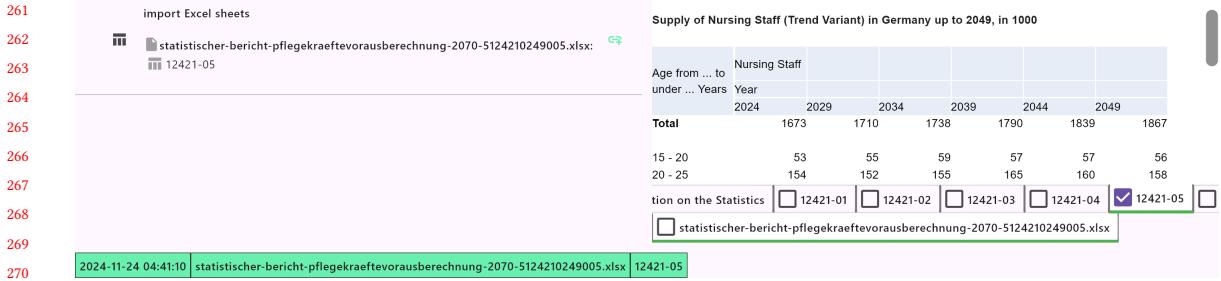


Fig. 1: The SDE Interface Overview.

We enforce three invariants: (i) **Provenance**: every emitted tuple is traceable to a set of visible source cells via an explicit mapping; (ii) **Stability**: small edits to selections induce bounded, predictable changes in the output (no global re-writes); (iii) **Viewport-bounded cost**: interactive operations run in time proportional to the number of cells intersecting the viewport, not the worksheet size. The parsing, indexing, and rendering subsystems are organized to uphold these invariants at scale.

User-Centric Data Extraction. The core interaction in the *SDE* lets users declare structure directly on the spreadsheet canvas and organize it into a hierarchy that drives a deterministic transformation into a relational view.

Hierarchy Definition. Users select individual cells or ranges (click, Shift-click for multi-selection). Each selection denotes either data (values) or metadata (labels/headers). Selections are arranged into a hierarchical tree: each node represents a data element; child nodes represent nested data or metadata. This hierarchy specifies how the *SDE* broadcasts selections into rows/columns in the output.

The interface supports flexible management: users drag-and-drop nodes to reparent or reorder the hierarchy when a different organization is more appropriate. Users can also introduce *custom nodes* with user-defined text to encode metadata that is implicit in the spreadsheet but absent from cell contents.

Reusability and Efficiency. To reduce repetitive work on sheets with repeated layouts, users can duplicate an existing selection hierarchy and apply it to similar regions. When moving a hierarchy, some cells may need to remain fixed (e.g., vertically stacked tables where only the topmost table repeats header rows). While in “move and duplicate” mode, the *SDE* provides a *lock* function: users freeze specific cells while relocating the rest of the hierarchy. Locks can be toggled via the lock icon at the top-left corner of a cell or next to the corresponding selection in the hierarchy panel; locked cells remain stationary while other selections shift accordingly (see Figure 4 in 4. Example Workflow). Already-locked selections are visually indicated and can be unlocked at any time.

4 Example Workflow

Consider a spreadsheet containing statistical forecasts of future nursing staff availability in Germany [17]. Figure 1 shows the *SDE* interface, which consists of three main components:

Hierarchy Panel (Top Left): Displays the hierarchy of cell selections, initially empty.

Spreadsheet View (Top Right): Shows the currently opened Excel file and worksheet for cell selection.

Output Preview (Bottom): Provides immediate feedback on the data extraction based on current selections.

313 12421-05

314 Supply of Nursing Staff (Trend Variant) in Germany up to 2049, in 1000

315 ✓ B3 Nursing Staff,

316 ✓ A6 Total,

317 ✓ A6 : A17 Total, 15 - 20, 20 - 25,

318 ✓ B5 2024,

319 ✓ B6 : B17 1673, 53, 154,

320 2024-11-24 04:32:22 statistischer-bericht-pflegekraeftevorausberechnung-2070-5124210249005.xlsx 12421-05 Nursing Staff Total Total 2024 B6 1673

321 2024-11-24 04:32:22 statistischer-bericht-pflegekraeftevorausberechnung-2070-5124210249005.xlsx 12421-05 Nursing Staff Total 15 - 20 2024 B7 53

322 2024-11-24 04:32:22 statistischer-bericht-pflegekraeftevorausberechnung-2070-5124210249005.xlsx 12421-05 Nursing Staff Total 20 - 25 2024 B8 154

Fig. 2: Selection of the First Column Metadata

4.1 Selection of the First Column

The user adds a node to the hierarchy and selects the cell containing the metadata "Nursing Staff" (Figure 2). This cell represents metadata that is common to all cells in this worksheet. Therefore, it should be selected first and should appear at the beginning of each row in the output CSV file.

Within this node, the user adds a child node and selects the cell "*Total*", which serves as both a table header and a row label. This selection represents the table header of the first subtable. The user adds another child node and selects the range of cells containing row labels (e.g., "*Total*", "15-20", "20-25" and so forth) by clicking the first cell and shift-clicking the last cell.

A further child node is then placed under the row labels node, and the user selects the year "2024". Subsequently, an additional child node is created beneath the year node, and the user selects the corresponding data cells (e.g., "1673", "53", "154", etc.).

At this point, the hierarchy consists of five nodes, each—except the last one—containing an embedded child node. In the upper-right portion of the interface, the chosen cells are displayed in distinct colors corresponding to each node. The lower area shows a preview of the extracted output. For each child node, an additional column is appended to the

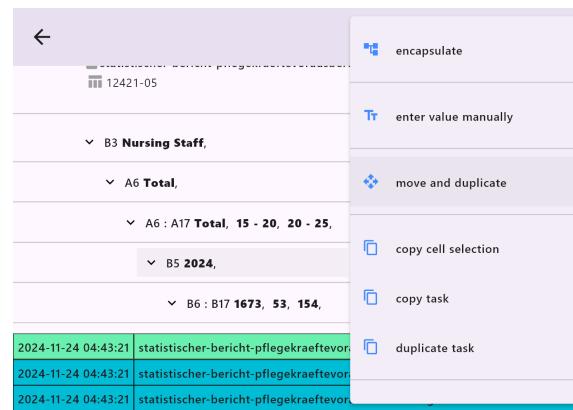


Fig. 3: Invoking the "move and duplicate" feature on the 2024 column node.

365
366
367
368
369 Supply of Nursing Staff (Trend Variant) in Germany up to 2049, in 1000
370
371 Age from ... to
372 under ... Years Year
373 2024 2029 2034 2039 2044 2049
374 Total 1673 1710 1738 1790 1839 1867
375 15 - 20 53 55 59 57 57 56
20 - 25 154 152 155 165 160 158
376
377 Fig. 4: Moving the hierarchy one cell to the right while adjusting the number of repetitions to duplicate the column
378 selection.
379
380 output. When multiple cells are selected for a given node, their values appear as entries in new rows of the output,
381 reflecting the defined hierarchical structure.
382

		Nursing Staff						
Age from ... to under ... Years		Year	2024	2029	2034	2039	2044	2049
Total			1673	1710	1738	1790	1839	1867
15 - 20			53	55	59	57	57	56
20 - 25			154	152	155	165	160	158

Fig. 4: Moving the hierarchy one cell to the right while adjusting the number of repetitions to duplicate the column selection.

4.2 Duplicating the Column Hierarchy

To avoid repetitive manual entry for additional years, the user duplicates the hierarchy for "2024" and adjusts the cell selections to include data for subsequent years (e.g., "2025," "2026") using the "Move and Duplicate" feature.

To do this, the user selects the node of the first column "2024" and right-clicks on it. A popup opens in which the action "move and duplicate" appears, which should then be clicked, as shown in Figure 3.

4.3 Duplicating the Table Hierarchy

Subsequently, a series of buttons opens in the app bar at the top right, allowing the user to move the cell selections of the node as well as all child nodes, as seen in Figure 4. By pressing the button to move the selection by one unit to the right, the next column is selected. However, this would also deselect the first column since the selection was moved. To preserve the first column, the "move and duplicate" checkbox can be activated. This creates the shifted selection in addition to the original selection. The changes are only applied when the "accept" button is clicked. The next columns could also be selected in the same way. But this can be done faster, because instead of moving the selection and

401
402 12421-05
403
404 Supply of Nursing Staff (Trend Variant) in Germany up to 2049, in 1000
405
406
407
408
409
410
411
412
413
414
415
416

▼ B3 Nursing Staff,	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Age from ... to under ... Years	Nursing Staff						
▼ A6 Total,	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2024	2029	2034	2039	2044	2049		
▼ A6 : A17 Total, 15 - 20, 20 - 25,	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Total	1673	1710	1738	1790	1839 1867		
▼ B5 2024,	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	15 - 20	53	55	59	57	57 56		
▼ B6 : B17 1673, 53, 154,	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20 - 25	154	152	155	165	160 158		
				tion on the Statistics		<input type="checkbox"/> 12421-01	<input type="checkbox"/> 12421-02	<input type="checkbox"/> 12421-03	<input type="checkbox"/> 12421-04	<input checked="" type="checkbox"/> 12421-05	<input type="checkbox"/>
				<input type="checkbox"/> statistischer-bericht-pflegekraeftevorausberechnung-2070-5124210249005.xlsx							
2024-11-24 04:47:51	statistischer-bericht-pflegekraeftevorausberechnung-2070-5124210249005.xlsx	12421-05	Nursing Staff	Total	Total	2024	B6	1673			
2024-11-24 04:47:51	statistischer-bericht-pflegekraeftevorausberechnung-2070-5124210249005.xlsx	12421-05	Nursing Staff	Total	15 - 20	2024	87	53			
2024-11-24 04:47:51	statistischer-bericht-pflegekraeftevorausberechnung-2070-5124210249005.xlsx	12421-05	Nursing Staff	Total	20 - 25	2024	88	154			

Fig. 5: Resulting Hierarchy using the "Move and Duplicate" Function to Replicate the Column Selection Across Years

417 duplicating it only once, the "repeat" input field can be filled with as many repetitions as there are columns. By entering
 418 the number 5, the selection of the first column is shifted 5 times by one unit to the right and duplicated at each step.
 419

420 The user reviews the selections in the spreadsheet view, where each selection is highlighted in a different color
 421 corresponding to its node in the hierarchy. Only after the user has reviewed the shifted and duplicated selections in
 422 the worksheet and clicked the "accept" button are the nodes in the hierarchy created as desired. Figure 5 shows the
 423 resulting selection after the user approved the proposed changes.
 424

425 The same method that worked effectively for duplicating the columns can now be applied to the subtables, as shown
 426 in Figure 6.

427 By selecting the node with the value "Total" and clicking the "Move and Duplicate" button, we can apply the selection
 428 of the "Total" subtable to the other subtables. This involves shifting the table downward by as many rows as necessary
 429 to overlap with the subtable below.
 430

431 However, there is a minor issue: the child nodes of the "Total" node also include the column headers. If these column
 432 headers were repeated in the subtables below, shifting the selections downward would work without modification.
 433 Since these cells are not repeated in the subtables, we need to prevent the column headers cells from moving during the
 434 duplication process.
 435

Supply of Nursing Staff (Trend Variant) in Germany up to 2049, in 1000						
Age from ... to under ... Years	Year	15 - 20	20 - 25	25 - 30	30 - 35	35 - 40
Total	2024	1673	1710	1738	1790	1839
15 - 20	53	55	59	57	57	56
20 - 25	154	152	155	165	160	158
25 - 30	173	174	168	170	181	175
30 - 35	178	186	184	178	180	191
35 - 40	187	192	193	196	190	192
40 - 45	175	201	204	209	207	201
45 - 50	176	190	218	220	226	222
50 - 55	177	177	193	225	228	233
55 - 60	215	177	177	197	225	227
60 - 65	166	170	139	140	156	178
65 - 70	26	36	37	31	31	34
Male		284	304	321	339	356
15 - 20	12	13	14	13	13	13
20 - 25	32	32	32	34	33	33
25 - 30	39	40	39	39	41	40
30 - 35	39	43	43	42	42	45
35 - 40	38	42	46	46	44	45
40 - 45	27	35	38	42	42	41
45 - 50	24	26	35	38	41	41
50 - 55	26	25	27	36	39	42
55 - 60	25	25	25	27	35	39
60 - 65	18	18	18	18	19	25
65 - 70						
Female		1390	1406	1416	1451	1484

466 Fig. 6: Selection of All Cells in the Subtables by Duplicating the Hierarchy of the First Table
 467

To achieve this, we can exclude individual nodes from being moved by locking their selection. This is done by clicking the padlock icon on the corresponding nodes, which freezes their cell selection and keeps them fixed at their original position, regardless of other cells being moved.

Therefore, we identify and select the nodes containing the column headers—specifically, the years 2024 to 2049—and lock their selection using the padlock button. By shifting the selection downward and duplicating it, we can easily move and duplicate the cell selections for the subtables below. By setting the number of repetitions to 2, all subtables are completely selected.

4.4 Path-wise Broadcasting

Figure 7 shows the selection hierarchy that users create on the spreadsheet. For readability, the example restricts itself to the first three columns and rows of the leftmost sub-table.

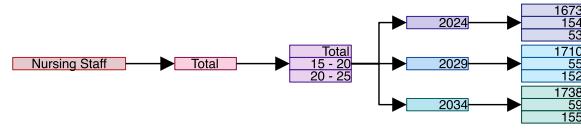


Fig. 7: Selection hierarchy before path-wise broadcasting.

To produce a flat, relational table, the SDE performs *path-wise broadcasting*: for each root-to-leaf path $P = (v_0, \dots, v_k)$ that ends in a list of N value cells (x_1, \dots, x_N) , the SDE materializes N output rows by repeating or aligning the labels found along the path.

Concretely:

- (1) **Broadcast singletons.** If a path node carries a single label (e.g., *Nursing Staff*, *Total*, or a single year), that label is replicated N times so each value cell inherits it.
- (2) **Align equal-length lists.** If a path node provides a list of N labels, those labels are paired index-wise with the N value cells.
- (3) **Emit one tuple per value cell.** For each $j \in \{1, \dots, N\}$, emit a row that contains the labels gathered from the path at position j (broadcast or aligned) together with the value x_j .

The resulting hierarchy after broadcasting is illustrated in Figure 8: upstream labels are repeated or aligned so that each numeric cell is paired with its full context, yielding one relational row per cell.

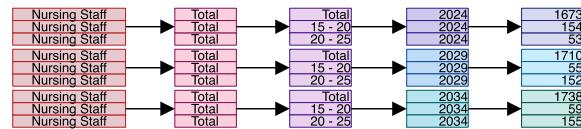


Fig. 8: Selection hierarchy after path-wise broadcasting. Each value cell is paired with its repeated/aligned upstream context, producing one row per cell.

521 **5 Interface Fidelity for Navigation**

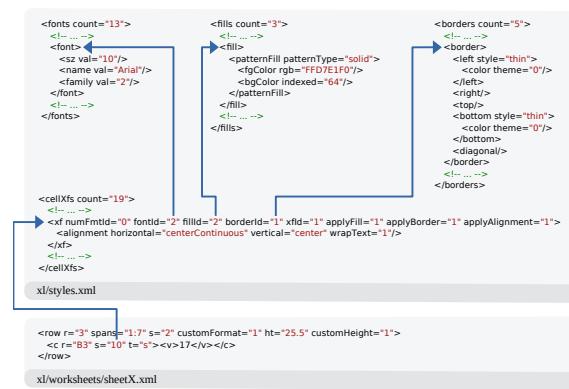
522
 523 To ensure that users can navigate worksheets without difficulty, we prioritize displaying the worksheets in a manner
 524 that closely resembles their appearance in Excel. This involves accurately rendering cell dimensions, formatting, and
 525 text behaviors.

526
 527 *5.0.1 Displaying Row Heights and Column Widths.* Our solution extracts information about column widths and row
 528 heights directly from the Excel file's XML structure. Specifically, we retrieve the column widths from the *width* attribute
 529 of the *<col>* elements and the row heights from the *ht* attribute of the *<row>* elements in the *sheetX.xml* files.

530
 531 In Excel, column widths and row heights are defined in units that do not directly correspond to pixels, requiring
 532 conversion for precise on-screen rendering. Moreover, different scaling factors are applied for columns and rows.
 533 Despite extensive research, we were unable to find official documentation that explains the rationale behind these
 534 specific scaling factors. Based on empirical testing, we derived the following scaling factors:

- 535
 536 • **Column Widths:** Multiply the *width* attribute by 7.
 537 • **Row Heights:** Multiply the *ht* attribute by $\frac{4}{3}$.

538
 539 *5.0.2 Cell Formatting.* Cell formatting plays a crucial role in accurately representing the appearance of worksheets.
 540 Formatting information is stored in the *styles.xml* file, where styles are defined and later referenced in the *sheetX.xml*
 541 files as shown in Figure 9.



559 Fig. 9: Illustration of the relationship between style definitions in *xl/styles.xml* (fonts, fills, borders, and *cellXfs*) and
 560 their application in a worksheet file (*xl/worksheets/sheetX.xml*).

561
 562
 563 Each cell in the worksheet references a style index through the *s* attribute, which points to the corresponding *<xf>*
 564 element within the *cellXfs* collection. These *<xf>* elements contain attributes such as *fontId*, *fillId*, and *borderId*, which
 565 reference specific font, fill (background), and border definitions located in the *fonts*, *fills*, and *borders* collections,
 566 respectively. By parsing these references, we can accurately apply the appropriate fonts, background colors, and border
 567 styles to each cell.

568
 569 Through meticulous parsing and application of these formatting details, we ensure that the rendered worksheet
 570 closely mirrors the original Excel file, preserving the visual cues and aesthetics that users expect.

573 *5.0.3 Handling Text Overflow.* In Excel, when the content of a cell exceeds its width, the text may overflow into adjacent
574 empty cells, provided those cells do not contain any data. If adjacent cells are occupied, Excel truncates the overflowing
575 text at the cell boundary. Replicating this behavior is essential for accurate rendering and user familiarity.
576

577 We implemented text overflow handling by checking if the adjacent cell to the right is empty before allowing text to
578 overflow. If the adjacent cell is empty, we extend the text rendering. If the adjacent cell contains data, we truncate the
579 text at the boundary of the original cell.
580

581 Figure 1 illustrates this behavior. The text "Supply of Nursing Staff ..." extends into the neighboring cell because it is
582 empty. If not for this handling, the text would be truncated at the cell boundary, leading to incomplete data display as
583 shown in Figure 10.
584

585

586 [To the table of c](#)

587

588 Supply of Nurs

589

Age from ... to under ... Years	Nursing Staff						
Total	Year	2024	2029	2034	2039	2044	2049
		1673	1710	1738	1790	1839	1867

593 **594** Fig. 10: Incorrect rendering without overflow for cells adjacent to empty cells.
595

596 By accurately handling text overflow, we improve readability and maintain consistency with Excel's user interface,
597 which is crucial for users transitioning between Excel and our tool.
598

600 6 Scalable Parsing, Indexing, and Rendering

601 This section describes how the SDE achieves interactive performance on large or bloated worksheets: (i) *incremental*
602 loading of XLSX assets, (ii) a *byte-level worksheet parser* that avoids DOMs and regex, (iii) compact *indexes* for merged
603 regions and column geometry, (iv) *on-demand* streaming of rows and cells, and (v) *viewport-bounded* rendering.
604

606 6.1 Incremental Loading of Worksheets

608 Opening large Excel files traditionally involves loading the entire file and all its worksheets into memory before
609 displaying any content. In files containing very large worksheets, this process can take several seconds to minutes,
610 causing significant delays for users who need to access data quickly.
611

612 To facilitate efficient data extraction from multiple Excel files, we implemented a mechanism for incremental loading
613 of worksheets within the SDE. Excel files (.xlsx format) are ZIP archives containing a collection of XML files that
614 describe the worksheets, styles, and shared strings. Key components include:
615

- 616** • **x1/sharedStrings.xml**: Contains all the unique strings used across worksheets, reducing redundancy.
- 617** • **x1/styles.xml**: Defines the formatting styles for cells, including fonts, colors, and borders.
- 618** • **x1/worksheets/sheetX.xml**: Represents individual worksheets (sheet1.xml, sheet2.xml, etc.).
619

620 Our solution opens the Excel file as a ZIP archive and initially extracts only the essential metadata and shared
621 resources required for the application to function. This initial extraction includes:
622

623 (1) Metadata Extraction:

Algorithm 1: Backward search for </sheetData>

```

625 Input:  $b$ : bytes;  $[lo, hi]$ : search window
626 Output:  $sheetDataCloseByte$  or  $-1$ 
627
628 Function  $FindSheetDataEndBackward(b, lo, hi)$ :
629      $pat \leftarrow$  bytes of </sheetData>
630      $m \leftarrow |pat|$ 
631     for  $i \leftarrow hi - m$  downto  $lo$  do
632         if  $b[i] = pat[0]$  and  $b[i + m - 1] = pat[m - 1]$  and  $\text{EqualAt}(b, i, pat)$  then
633             return  $i$ 
634
635 return  $-1$ 

```

636 We read the archive’s directory to identify the contained files without decompressing them fully. This step is
 637 quick, taking only a few milliseconds, and provides information about the available worksheets and shared
 638 resources.
 639

640 (2) **Selective Extraction:**

641 We immediately extract the `sharedStrings.xml` and the `styles.xml` files because these files are small and
 642 contain information necessary for rendering cell content and styles across all worksheets. These files are parsed
 643 and stored in memory for quick access during rendering.

644 (3) **Deferred Worksheet Loading:** The individual worksheet files (`sheetX.xml`) remain compressed and are
 645 loaded into memory in their binary unextracted form. They are not decompressed or parsed at this stage.

646 (4) **On-Demand Parsing:**

647 When a user accesses a specific worksheet—either by selecting it in the interface or when a unit test requires
 648 data from it—the corresponding `sheetX.xml` file is then decompressed and parsed. This parsing occurs in the
 649 background and is triggered only by direct user action or programmatic access to the worksheet’s data.

650 (5) **Memory Release:**

651 After a worksheet has been decompressed and its XML parsed, we release the memory resources associated
 652 with the parsed data. This approach prevents excessive memory usage and ensures that the application remains
 653 responsive even when working with multiple large worksheets.

654 By adopting this incremental loading approach, users experience minimal wait times when opening an Excel file.
 655 The initial loading is nearly instantaneous, allowing users to begin interacting with the application without delay. This
 656 contrasts with traditional methods that require loading all worksheets upfront, leading to significant wait times for
 657 large files.
 658

6.2 Parsing Worksheet XML with Byte-Level Pattern Matching

659 DOM-based parsers and regex-on-strings do not scale for very large worksheets: they require full decoding to UTF-
 660 16/UTF-8 and materialize enormous trees. In Dart, regular expressions cannot operate on byte arrays, so converting
 661 a gigabyte-scale `Uint8List` to a string alone can cost seconds. The SDE therefore parses *directly on bytes*, matching
 662 ASCII tag sentinels and reading attributes in place, decoding strings only on demand.

663 *Parsing roadmap.* Excel worksheets expose a stable top-level order: `<sheetFormatPr>`, `<cols>`, `<sheetData>`,
 664 `<mergeCells>`. We first anchor the end of `<sheetData>` by a backward byte search; then we parse metadata around it:

- 665 • `<mergeCells>` (after `</sheetData>`),
- 666 • `<sheetFormatPr>` (before `<sheetData>`) and

- 677 • <cols> (before <sheetData>)

678 and enter *sheet-data mode* only when rows or cells are actually needed.

680 *Anchoring & metadata.* We find the closing </sheetData> sentinel by scanning backward and validating the 12-byte
 681 pattern (Alg. 1). This yields byte indices that bound all subsequent searches and lets us enumerate <mergeCell ...
 682 ref="A1:B3"> elements linearly, converting each A1 pair to (r, c) and inserting the span into a compact index with
 683 binary-search probes and a prefix-maximum (used later by spanAt, Alg. 4).

686 *Defaults and column bands.* We record the default row height H_d (attribute defaultRowHeight) and the default
 687 column width W_d (attribute defaultColWidth) in the <sheetFormatPr ...> node. From <cols> we parse each element
 688 of the form <col min="i" max="j" width="w">, which defines a *column band* $[i:j]$ with width w . Bands are stored
 689 in ascending order of min; queries such as retrieving the width at column c or the column at a given horizontal offset
 690 are answered via a linear or $O(\log B)$ search over these bands.

693 *Streaming rows and accumulating offsets.* Entering sheet-data mode, we stream <row ...> tags without decoding
 694 payloads. Within each opening tag we read $r = \dots$ (row index) and $ht = \dots$ (height) if present. For each discovered
 695 row we cache its byte interval and compute its top offset incrementally using explicit heights or the default H_d :

$$697 \quad off_1 = (r_1 - 1) H_d, \quad off_i = off_{i-1} + (h_{i-1} \text{ or } H_d) + (r_i - r_{i-1} - 1) H_d.$$

699 See Alg. 2.

701 *Lazy cell parsing.* Given a row byte interval $[s, e)$, cells are parsed on demand. We scan for <c>, read attributes
 702 ($r = "A123"$, $s = \dots$, $t = \dots$), and bound the cell interval by the next <c> or the row end. Values are extracted *within*
 703 this interval from <v>...</v> or (for inline strings) <is>...</is>. See Alg. 3. Because Excel enforces increasing row
 704 indices, we can stop row streaming as soon as we pass the requested row.

706 *Merged regions.* Merged areas are rectangles $[r_1, r_2] \times [c_1, c_2]$. We normalize and sort spans by (r_1, c_1) , store parallel
 707 arrays R_1, C_1, R_2, C_2 , and build a prefix-maximum PM over R_2 . A point query spanAt(r, c) uses (i) a binary search on
 708 R_1 to cap candidates above r , (ii) a binary search over PM to drop candidates ending before r , (iii) a binary search on C_1
 709 to find those with $c_1 \leq c$, then a short local check; an origin map answers “is this cell the origin?” in $O(1)$. See Alg. 4.

712 *Scroll extents (without decoding payloads).* We obtain r_{\max} by scanning backward to the last <row ... r="...">. A
 713 single forward pass accumulates $\sum_{r \in E} ht(r)$ and counts $|E|$; all other rows use H_d . In device units,

$$715 \quad H_{\text{sheet}} = \sum_{r \in E} ht(r) + (r_{\max} - |E|) H_d.$$

718 Horizontally, letting c_{\max} be the largest covered column,

$$719 \quad W_{\text{sheet}} = \text{colOff}(c_{\max}) + w(c_{\max}).$$

721 These extents parameterize the viewport and drive scrollbar sizing in the UI.

723 *Complexity and memory.* All searches are bounded by structural sentinels (>, next <row>, next <c>, or </sheetData>).
 724 Anchoring </sheetData> is $O(N)$ with tiny constants; row streaming is $O(R)$ with $O(1)$ per-row offset updates; a
 725 targeted cell within a row scans only that row’s interval; merge queries run in $O(\log M + \alpha)$ with a short local scan α .
 726 The representation is compact (byte slices + small numeric state), and strings are decoded only when needed.

Algorithm 2: Stream rows & compute offsets

Algorithm 3: Resolve cell by streaming the row

```

781 Input:  $b$ : bytes; row interval  $[S, E)$ ; target column  $c$ 
782 Output: cell interval  $[s, e)$  or  $\perp$ 
783  $i \leftarrow S$ 
784 while  $i \leq E - 2$  do
785   if  $b[i..i + 1] = <c$  then
786      $s \leftarrow i$ 
787      $(c', j) \leftarrow \text{PARSECELLCOL}(b, i, E)$                                 // reads r="A123" and advances to just after >
788     // end of this cell = next <c or  $E$ 
789      $k \leftarrow j$ 
790     while  $k \leq E - 2$  and  $b[k..k + 1] \neq <c$  do
791        $k \leftarrow k + 1$ 
792      $e \leftarrow (k \leq E - 2) ? k : E$ 
793     if  $c' = c$  then
794       return  $[s, e)$ 
795      $i \leftarrow e$ 
796   else
797      $i \leftarrow i + 1$ 
798
799 return  $\perp$ 

```

6.3 Two-Dimensional Grid Viewport Rendering

Given a (potentially very large) worksheet with variable row heights, banded column widths, and merged regions, we lay out only tiles intersecting the current viewport. The renderer works in device pixels but derives all positions from the byte-level parser. Our goal is *frame-local* work proportional to the number of *visible* rows/columns, independent of sheet size. Algorithm 4 summarizes the procedure. Throughout, index intervals are half-open $[a, b)$.

Inputs. Let x_0, y_0 be the horizontal/vertical scroll offsets (device pixels) and $W_{\text{vp}}, H_{\text{vp}}$ the viewport size. We use a small *cache extent* $\Delta > 0$ to pre-build tiles that will imminently enter view, rendering over $[x_0, x_0 + W_{\text{vp}} + \Delta] \times [y_0, y_0 + H_{\text{vp}} + \Delta]$. Row geometry comes from the streaming parser: each row r has a top offset $\text{off}(r)$ and a height $h(r)$ (explicit *ht* if present, otherwise the default). Columns are given as ordered bands; for column c we can query its width $w(c)$ and cumulative offset $\text{colOff}(c)$. Merged regions are indexed by a structure that decides, in logarithmic time, whether a coordinate (r, c) is covered and, if so, by which span.

Visible set. We invert the cumulative-height/width functions:

$$\begin{aligned} r_\ell &= \lfloor \text{rowIndexAt}(y_0) \rfloor, & r_u &= \lceil \text{rowIndexAt}(y_0 + H_{\text{vp}} + \Delta) \rceil, \\ c_\ell &= \lfloor \text{colIndexAt}(x_0) \rfloor, & c_u &= \lceil \text{colIndexAt}(x_0 + W_{\text{vp}} + \Delta) \rceil. \end{aligned}$$

Here $\text{rowIndexAt}(y)$ and $\text{colIndexAt}(x)$ are binary searches over cumulative extents built from parsed row heights and column bands, yielding $O(\log R)$ and $O(\log B)$ lookup time, respectively.

Cell coordinates and merge spans. A merged region is a closed rectangle $[r_1, r_2] \times [c_1, c_2]$ with $r_1 \leq r_2$ and $c_1 \leq c_2$. We sort spans by origin (r_1, c_1) and materialize parallel arrays R_1, C_1, R_2, C_2 plus a prefix-maximum array $\text{PM}[i] = \max_{0 \leq j \leq i} R_2[j]$. An origin map supports $O(1)$ checks that (r, c) is the top-left of a span. Membership “is (r, c) covered?” runs in three bounded steps:

- (1) **Row window (binary search).** $hi = \text{ub}(R_1, r)$; candidates lie in $[0, hi)$. Then $lo = \text{lb}(\text{PM}[0:hi], r)$ discards all indices with $R_2 < r$.
- (2) **Column narrowing (binary search).** $k = \text{ub}(C_1[lo:hi], c) - 1$ is the last origin with $c_1 \leq c$.

Algorithm 4: Viewport layout with merge-aware origin-first placement

Input: $x_0, y_0; W_{vp}, H_{vp}$; cache Δ ; sheet accessors $\text{rowIndexAt}, \text{colIndexAt}, \text{off}, \text{colOff}, h, w$; merge index $\text{spanAt}, \text{isOrigin}$
Output: positioned tiles for current frame

```

833 // Visible indices
834  $r_\ell \leftarrow \lfloor \text{rowIndexAt}(y_0) \rfloor, r_u \leftarrow \lceil \text{rowIndexAt}(y_0 + H_{vp} + \Delta) \rceil$ 
835  $c_\ell \leftarrow \lfloor \text{colIndexAt}(x_0) \rfloor, c_u \leftarrow \lceil \text{colIndexAt}(x_0 + W_{vp} + \Delta) \rceil$ 
836  $B \leftarrow \emptyset // burned merged spans$ 
837
838 // Pass 1: top border probes
839 for  $c \leftarrow c_\ell$  to  $c_u$  do
840    $S \leftarrow \text{spanAt}(r_\ell, c)$ 
841   if  $S \neq \perp$  and  $S \notin B$  and  $r_S < r_\ell$  then
842     place tile for  $S$  at  $(\text{colOff}(c_S) - x_0, \text{off}(r_S) - y_0)$ 
843      $B \leftarrow B \cup \{S\}$ 
844
845 // Pass 2: left border probes
846 for  $r \leftarrow r_\ell$  to  $r_u$  do
847    $S \leftarrow \text{spanAt}(r, c_\ell)$ 
848   if  $S \neq \perp$  and  $S \notin B$  and  $c_S < c_\ell$  then
849     place tile for  $S$  at  $(\text{colOff}(c_S) - x_0, \text{off}(r_S) - y_0)$ 
850      $B \leftarrow B \cup \{S\}$ 
851
852 // Pass 3: interior tiles
853 for  $c \leftarrow c_\ell$  to  $c_u$  do
854    $x \leftarrow \text{colOff}(c) - x_0$ 
855   for  $r \leftarrow r_\ell$  to  $r_u$  do
856      $y \leftarrow \text{off}(r) - y_0$ 
857      $S \leftarrow \text{spanAt}(r, c)$ 
858     if  $S = \perp$  or  $\text{isOrigin}(S, (r, c))$  then
859       place tile at  $(x, y)$ 
860
861 Function  $\text{spanAt}(r, c):$ 
862    $hi \leftarrow \text{ub}(\mathcal{R}_1, r)$ 
863   if  $hi = 0$  then
864      $\perp$ 
865    $lo \leftarrow \text{lb}(\mathcal{P}\mathcal{M}[0:hi], r)$ 
866   if  $lo \geq hi$  then
867      $\perp$ 
868    $k \leftarrow \text{ub}(\mathcal{C}_1[lo:hi], c) - 1$ 
869   if  $k < lo$  then
870      $\perp$ 
871   for  $i \leftarrow k$  downto  $lo$  do
872     if  $\mathcal{C}_1[i] > c$  then
873        $\perp$ 
874     if  $r \leq \mathcal{R}_2[i]$  and  $c \leq \mathcal{C}_2[i]$  then
875        $\text{return } \text{span } i$ 
876   return  $\perp$ 
877
878
879 (3) Local check (constant expected). Scan left from  $k$  while  $\mathcal{C}_1[i] \leq c$ ; accept if  $r \leq \mathcal{R}_2[i]$  and  $c \leq \mathcal{C}_2[i]$ .
880
881 We use  $\text{ub}(A, x) = \min\{i \mid A[i] > x\}$  and  $\text{lb}(A, x) = \min\{i \mid A[i] \geq x\}$ . This yields  $O(\log M + \alpha)$  time, where  $\alpha$  is a
882 short local scan in practice.
883
884
885 Origin-first placement for merged regions. A naïve scan of  $[r_\ell:r_u] \times [c_\ell:c_u]$  would instantiate merged tiles multiple
886 times. Instead, we pre-place only spans whose origins lie outside the leading edges but whose rectangles intersect the
887 viewport: probe the top border  $(r_\ell, c_\ell:c_u)$  and the left border  $(r_\ell:r_u, c_\ell)$ , query  $\text{spanAt}(r, c)$ , and place the tile once at its
888 origin  $(r_S, c_S)$ . We maintain a burned set  $B$  to avoid duplicates in the interior.
889

```

Algorithm 5: AABB-indexed selection lookup over visible grids

```

885 Input: point  $(x, y)$ ; arrays  $L, R, T, B$ ; permutation  $\sigma$  sorting by  $T$ ; prefix max PM over  $B_{\sigma(\cdot)}$ ; original-order  $O$ ; accessor  $\text{findCell}_i(x, y)$ 
886 Output: either  $\langle i, \text{cell} \rangle$  or  $\perp$ 
887 // Binary-search helpers:  $\text{ub}(A, z) = \min\{i \mid A[i] > z\}$ ,  $\text{lb}(A, z) = \min\{i \mid A[i] \geq z\}$ .
888
889 // 1) Y-narrowing: window of candidates whose top  $\leq y$  and bottom  $\geq y$ 
890  $h \leftarrow \text{ub}(T_{\sigma}, y)$   $\perp$  // first index with  $T_{\sigma(h)} > y$ 
891 if  $h = 0$  then
892    $\perp$   $\perp$ 
893 if  $\ell \geq h$  then
894    $\perp$   $\perp$ 
895
896 // 2) X-pruning: keep only boxes covering  $x$ 
897  $C \leftarrow \emptyset$ 
898 for  $j \leftarrow \ell$  to  $h - 1$  do
899    $i \leftarrow \sigma(j)$ 
900   if  $L_i \leq x \leq R_i$  then
901      $\perp$   $\perp$ 
902   if  $C = \emptyset$  then
903      $\perp$   $\perp$ 
904
905 // 3) Stable tie-break and confirmation
906  $C \leftarrow \text{sort } C$   $\perp$  // preserve UI order/colors
907 foreach  $i \in C$  do
908    $ans \leftarrow \text{findCell}_i(x, y)$ 
909   if  $ans \neq \perp$  then
910      $\perp$   $\perp$ 
911
912 return  $\perp$ 

```

909 *Interior tiling.* Traverse the visible grid and place a tile at (r, c) only if (i) no span covers it, or (ii) (r, c) is the origin
910 of its span (checked in $O(1)$). Device positions are $x = \text{colOff}(c) - x_0$, $y = \text{off}(r) - y_0$.

912 *Scroll extents.* We size the vertical scroll domain from row attributes without decoding payloads. Anchored at
913 $</\text{sheetData}>$, we scan backward to the last $<\text{row} \dots>$ to read r_{\max} from $r=" \dots "$. A single forward pass accumulates
914 explicit heights $\sum_{r \in E} \text{ht}(r)$ and counts them as $|E|$; all other rows in $1..r_{\max}$ use the default H_d . With device pixels,
915

$$H_{\text{sheet}} = \sum_{r \in E} \text{ht}(r) + (r_{\max} - |E|) H_d.$$

916 Horizontally, with c_{\max} the highest covered column,

$$W_{\text{sheet}} = \text{colOff}(c_{\max}) + w(c_{\max}).$$

917 These extents parameterize the viewport and drive scrollbar sizing/positioning (*two_dimensional_scrollables* [13]).

918 *Correctness and complexity.* Edge probes ensure any merged region intersecting the viewport but originating above/left
919 is instantiated *exactly once* at its origin. The interior pass either places unmerged cells or the unique origin cell of each
920 merged region. Let $R_{\text{vis}} = r_u - r_l + 1$ and $C_{\text{vis}} = c_u - c_l + 1$. Passes 1-2 cost $O(R_{\text{vis}} + C_{\text{vis}})$; Pass 3 costs $O(R_{\text{vis}} C_{\text{vis}})$ and
921 does no work outside the visible rectangle. The burned set is updated at most once per span per frame.

922 6.4 AABB-Indexed Selection Lookup

923 Each selection source (a task or composed group) maintains a lazily computed, cached *axis-aligned bounding box*
924 (AABB) that is invalidated on updates and recomputed on demand (Algorithm 5). When many such selection sources
925 (grids) intersect the viewport, a linear scan over their bounding boxes can dominate latency. We therefore maintain a
926

Algorithm 6: Lazy row locator: driver and lookup (part A)

```

937 Input: hierarchy files→tasks→sheets→cellTasks; caps  $B_{\max}, R_{\max}$ 
938 Output: on demand: for global row  $r$ , return (RowBlock, local) or  $\perp$ 
939 State: arrays  $blocks, starts$ ; queue  $Q$ ; indices  $fi, sti, si; curFile, curTask; curSheets; (activeWb, activeSheet)$ .
940 Function EnsureBuiltThroughRow( $r$ ):
941   while  $builtRows \leq r$  do
942      $rb \leftarrow \text{NextBlock}()$ 
943     if  $rb = \perp$  then
944        $\perp$ 
945     append  $rb$  to  $blocks$ ; append  $rb.startRow$  to  $starts$ ;
946     PruneHeadIfNeeded()
947
948 Function LocateRow( $r$ ):
949   EnsureBuiltThroughRow( $r$ )
950   if  $blocks = \emptyset$  then
951      $\perp$ 
952    $lo \leftarrow 0; hi \leftarrow |starts|$ 
953   while  $lo < hi$  do
954      $mid \leftarrow \lfloor (lo + hi)/2 \rfloor$ 
955     if  $starts[mid] \leq r$  then
956        $lo \leftarrow mid + 1$ 
957     else
958        $hi \leftarrow mid$ 
959
960      $idx \leftarrow lo - 1$ 
961     if  $idx < 0$  then
962        $\perp$ 
963      $b \leftarrow blocks[idx]; local \leftarrow r - b.startRow$ 
964     if  $0 \leq local < b.len$  then
965        $\perp$ 
966     else
967        $\perp$ 
968
969

```

lightweight index over axis-aligned bounding boxes (AABBs) to answer point queries (x, y) in $O(\log G + \alpha)$ time, where G is the number of visible grids and α is the size of a narrow candidate window. Their bounding boxes can dominate latency. We therefore maintain a lightweight index over axis-aligned bounding boxes (AABBs) to answer point queries (x, y) in $O(\log G + \alpha)$ time, where G is the number of visible grids and α is the size of a narrow candidate window.

Each grid i exposes a bounding rectangle $[L_i, R_i] \times [T_i, B_i]$ in grid coordinates and a point predicate $\text{findCell}_i(x, y)$ that returns the cell at (x, y) or \perp . We build parallel arrays L, R, T, B (left, right, top, bottom) and a stable original-order index O (used to break ties consistently with the UI). Let σ be the permutation that sorts grids by nondecreasing T (top edge). In that order we form a prefix-maximum array $\text{PM}[j] = \max_{0 \leq i \leq j} B_{\sigma(i)}$. For a query row y , the candidate window is the tightest interval $[\ell, h]$ such that all boxes whose top $\leq y$ and bottom $\geq y$ lie in that interval; we obtain $h = \text{ub}(T_\sigma, y)$ and $\ell = \text{lb}(\text{PM}[0:h], y)$ by binary search (ub: strict upper bound; lb: lower bound). We then prune by the x -interval condition $L \leq x \leq R$. Among the surviving candidates we test findCell in increasing O (original) order and return the first hit; if none match, the answer is \perp .

The index builds in $O(G \log G)$ time and consists of six integer arrays plus the permutation π . It is invalidated on any structural change to the set of grids (move, resize, insert/delete) and rebuilt lazily on the next query. For small G (e.g., $G \leq 32$) we fall back to a linear scan; the cutover can be tuned empirically.

Streaming next block. NEXTBLOCK (Alg. 7) emits the next contiguous block. If Q is empty we call **FILLQUEUEIFEMPTY** to advance to the next *source segment* (file, sheet task, sheet pair) with non-empty roots. Otherwise we pop a frame. If the popped task has no children, its **sortedSelectedCells** form a new RowBlock ($\text{start} = \text{builtRows}$, $\text{length} = \text{number}$

of selected cells) and we return it. If the task has children, we enqueue each child with the extended path. This BFS over the selection tree yields a deterministic, top-down order that matches the user's mental model.

Algorithm 7: Row block generation (part B): NextBlock

```

994 Function NEXTBLOCK:
995   if  $\neg$ FillQueueIsEmpty() then
996     ↘ return  $\perp$ 
997   while true do
998     if  $Q = \emptyset$  then
999       ↘ if  $\neg$ FillQueueIsEmpty() then
1000         ↘ return  $\perp$ 
1001       ↘ continue
1002     ( $t, path$ )  $\leftarrow$  pop-front  $Q$ 
1003     if  $t.children = \emptyset$  then
1004       ↘ cells  $\leftarrow t.sortedSelectedCells$ 
1005       if  $|cells| = 0$  then
1006         ↘ continue
1007       ↘ start  $\leftarrow builtRows; len \leftarrow |cells|$ 
1008       ↘ return RowBlock(activeWb, activeSheet, path, cells, start, len)
1009     nextPath  $\leftarrow path \cup \{t\}$ 
1010     foreach  $u \in t.children$  do
1011       ↘ push-back  $(u, nextPath)$  into  $Q$ 

```

6.5 Lazy Output Flattening and Row Location

Large workbooks and selection hierarchies make fully materializing a relational view prohibitively expensive. Empirically, naive flattening leads to seconds or minutes of UI stalls on sheets with up to 10^6 rows (Excel's limit) and many columns. Our goal is to *render only what the user is about to see*, without precomputing the entire output.

Design overview. We model the visible table as a single *global row space* obtained by concatenating many small, contiguous *row blocks*. Each block is produced by streaming the hierarchy files → tasks → sheets → cellTasks in a deterministic order. The UI asks for a particular global row index r (e.g., the first row currently in the viewport); we then extend the stream just far enough so that r falls inside a cached block. This *on-demand flattening* avoids touching unrelated parts of the hierarchy.

Data structures. We maintain two parallel arrays: (i) `blocks` contains the realized row blocks (each stores `startRow`, `len`, `path`, `cells`, and the active workbook/sheet), and (ii) `starts` stores the corresponding `startRow` values. By construction, `starts` is strictly increasing. A FIFO queue Q holds pending (`task`, `path`) frames for a breadth-first streaming traversal. Traversal indices over files/sheet-tasks/sheets maintain the current context (`activeWb`, `activeSheet`). Optional caps B_{\max} and R_{\max} bound memory by pruning the oldest blocks.

Driver and lookup. When the UI requests row r , `ENSUREBUILTHOOKHROW` extends the stream until $builtRows > r$ or the source is exhausted (Alg. 6). We then locate the block via a binary search over `starts` (*upper bound* on r and step one back), yielding the block index and the local in-block offset (Alg. 6, `LOCATEROW`). Both steps are $O(k)$ work to extend by k newly discovered rows plus $O(\log |starts|)$ for the lookup.

Source advancement. `FILLQUEUEIFEMPTY` (Alg. 8) advances through the outer hierarchy: it steps files (fi), sheet tasks (sti), then sheet pairs (si). For each sheet pair (wb, sh) it materializes the roots (`importExcelCellsTasks`) into Q and

sets the active context used to form RowBlocks. If a segment has no roots, it is skipped. When the last file is exhausted and Q remains empty, the function returns **false** and the stream is complete.

Algorithm 8: Traversal feeding (part C): FillQueueIfEmpty

```

1044 Function FILLQUEUEIFEMPTY:
1045   while  $Q = \emptyset$  do
1046     if  $curTask = \perp$  then
1047       if  $fi \geq |files|$  then
1048         return false
1049       curFile  $\leftarrow files[fi + 1]$ ;  $sti \leftarrow 0$ 
1050       if  $sti \geq |curFile.sheetTasks|$  then
1051          $curTask \leftarrow \perp$ ; continue
1052        $curTask \leftarrow curFile.sheetTasks[sti + 1]$ ;  $curSheets \leftarrow list(curTask.sheets)$ ;  $si \leftarrow 0$ 
1053       while  $si < |curSheets|$  and  $Q = \emptyset$  do
1054          $(wb, sh) \leftarrow curSheets[si + 1]$ ;  $roots \leftarrow curTask.importExcelCellsTasks$ 
1055         if  $|roots| = 0$  then
1056           continue
1057          $Q \leftarrow$  queue of  $(root, \emptyset)$  for each  $root \in roots$ 
1058          $activeWb \leftarrow wb$ ;  $activeSheet \leftarrow sh$ 
1059       if  $Q \neq \emptyset$  then
1060         return true
1061
1062
1063
1064 Cache pruning. PRUNEHEADIFNEEDED (Alg. 9) trims the front of the cache to respect either (i) a block count cap  $B_{\max}$ ,
1065 or (ii) a total cached row cap  $R_{\max}$ . Trimming removes the oldest blocks and their starts while preserving the invariant
1066 that  $starts$  remains strictly increasing and aligned with blocks. In practice we choose small caps that comfortably
1067 cover one or two viewport heights plus prefetch.
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2380
2381
2382
2383
2384
2385
2386
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2396
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2496
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2597
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2697
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2796
2797
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2887
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2897
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2987
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3028
```

1093 1094 1095 1096 1097 1098 1099 *Complexity.* Let N be the number of leaf tasks that produce non-empty `sortedSelectedCells`, and let $M = \sum \text{len}$ be the total number of output rows. Streaming is linear in the explored work: each internal node is visited once and each leaf contributes $O(1)$ metadata plus $O(\text{len})$ to copy/select references to its cells. A lookup for row r costs $O(k + \log |\text{starts}|)$, where k is how many *new* rows must be streamed to cover r (often $k = 0$ for warm caches). Memory is $O(|\text{blocks}|)$, bounded by B_{\max} or by R_{\max} rows of history.

1100 1101 1102 1103 1104 1105 *UI integration.* On each frame the viewport asks for the *leading* global row (top-left visible), calls `LOCATEROW` to obtain the block and local offset, and renders from there. We prefetch just beyond the viewport by requesting the row at $y_0 + H_{\text{vp}} + \Delta$, which amortizes the streaming cost without over-allocating memory. When the user scrolls back, previously cached blocks are reused; otherwise the binary search still runs in logarithmic time.

1106 1107 1108 1109 *Failure modes.* Segments with empty selections yield no blocks but are skipped efficiently. If the hierarchy ends before covering the requested row, the driver returns \perp ; the UI interprets this as “no more data.” All pruning policies preserve correctness: they only remove *past* blocks and never create gaps inside the realized suffix.

1110 1111 By applying this method, we render only the cells necessary for the current view, thereby optimizing performance and ensuring smooth user interactions even with large and complex worksheets.

1113 7 Evaluation

1114 The Spreadsheet Data Extractor (SDE) builds on the converter by Aue et al. [3], whose extraction effectiveness was evaluated on over 500 Excel files (331 files; 3,093 worksheets; mean 15 min per file, 95 s per worksheet). Our contribution focuses on *user experience* and *latency*: we render worksheets as they appear in Excel, reduce context switching by integrating selection hierarchy, worksheet view, and output preview, and engineer incremental loading, byte-level parsing and plus viewport-bounded rendering.

1121 7.1 Acceleration When Opening Files

1123 We quantify opening latency in two complementary scenarios: (i) a *user-centred* case that measures time-to-first-visual when opening a *selected worksheet* from a large, multi-worksheet workbook; and (ii) a *workload-equivalent* case that uses a single worksheet with 1,048,548 rows to align the amount of work across programmes (first rows vs. last row). Throughout, we report medians over repeated runs and show distributions as box plots on a \log_{10} time axis.

1129 1130 1131 *Setup.* We evaluate three programmes: the *SDE*, Microsoft Excel (automated via PowerShell/COM), and a Dart *excel* package. Each condition was repeated ten times *unless noted otherwise*.

1132 1133 1134 1135 1136 *User-centred latency: opening a selected worksheet.* Opening a selected worksheet from a large real-world workbook [18], SDE initialises only the requested sheet and achieves a median of **0.657 s**, compared to **45.843 s** for Excel—a **69.8×** speedup (Fig. 11a). The Dart *excel* package completed only the first run at 343.568 s; runs 2–10 terminated with out-of-memory on our test machine ($n=1$). Based on that single run, SDE is **522.9×** faster in this scenario.

1137 1138 1139 1140 1141 1142 1143 1144 *Controlling for workload comparability.* The multi-worksheet experiment reflects an important user scenario (time-to-first-visual for a selected sheet), but it is not a like-for-like comparison: SDE initialises only the requested worksheet, whereas the baselines initialise the entire workbook. To control for this confound and establish workload equivalence, we construct a *single-worksheet* benchmark from the example in Section 3 (Design Philosophy) by pruning the workbook to one sheet and duplicating the table vertically until no additional table fits, yielding 1,048,548 rows of real data. The Manuscript submitted to ACM

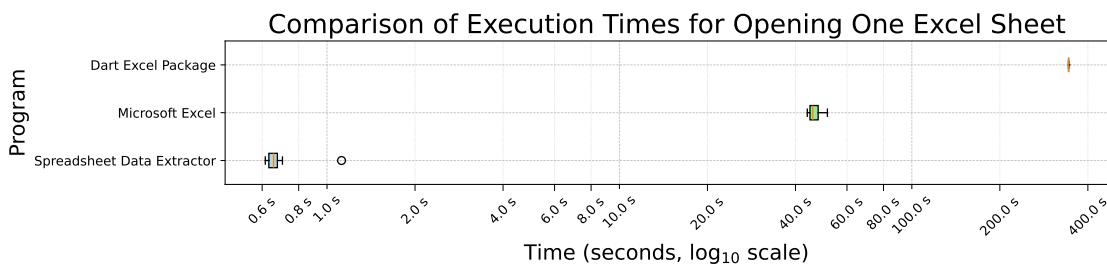
1145 resulting file is ~ 35 MB on disk (XLSX ZIP container) and ~ 282 MB uncompressed. We then report two cases—*first*
 1146 *rows* (time-to-first-visual) and *last row* (forces a full parse)—to align SDE’s measured cost with eager parsers. Under
 1147 these identical I/O and parsing conditions, SDE achieves **1.229 s** vs. **52.756 s** for Excel on first rows (**42.9 \times**), and **5.515 s**
 1148 vs. **52.228 s** on the last row (**9.47 \times**).
 1149

1150 *Results (overview).* Initialising only the selected sheet yields much lower time-to-first-visual (Fig. 11a); under identical
 1151 workload SDE remains $\sim 10\times$ faster even when a full parse is enforced (Fig. 11b).
 1152

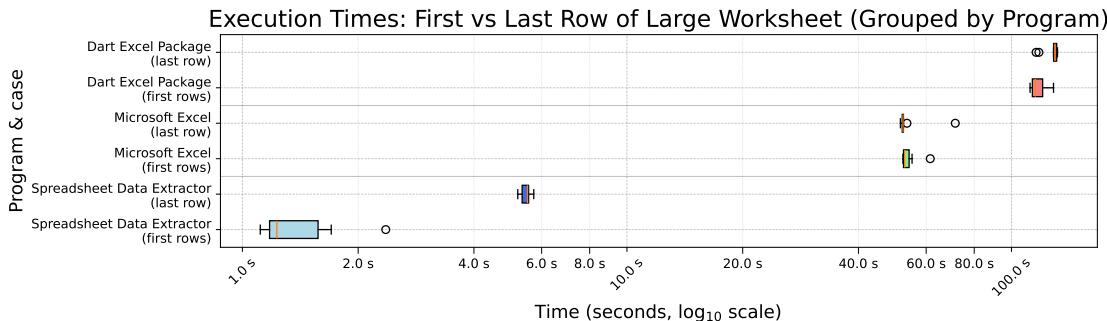
1154 7.2 Interactive scalability

1155 While the underlying extraction model remained effective, the previous prototype did not stay interactive on very large
 1156 sheets. Empirically, on workbooks with hundreds of thousands of rows—and in particular near Excel’s 10^6 -row limit—
 1157 two failure modes dominated:
 1158

- 1160 (1) **Eager materialisation in views.** When the selection view or the output view attempted to realise large regions
 eagerly, per-frame times rose above 1 s, making panning and selection unusable.
- 1162 (2) **Large selection hierarchies.** With many nested selections, highlight updates and duplicate/move previews
 performed work proportional to the total number of selected cells, causing multi-second pauses per interaction
 and making operations on large datasets impractical.
 1166



1169 (a) Large multi-worksheet workbook: opening one selected sheet (user-centred latency).
 1170



1173 (b) Single-worksheet benchmark (1,048,548 rows): first vs. last row per programme.
 1174

1175 Fig. 11: Time-to-first-visual vs. full-parse costs across tools. (a) shows the practical benefit of initialising only the selected
 1176 sheet in a multi-worksheet file; (b) isolates algorithmic differences under identical workload. Both plots aggregate 10
 1177 runs on a logarithmic time axis.
 1178



Fig. 12: SDE performance profiling during scrolling. Timeline shows per-frame rendering costs (blue); average 18ms/frame and peak 54ms/frame.

These observations motivated the engineering in 6.2 Parsing Worksheet XML with Byte-Level Pattern Matching: (i) byte-level, on-demand parsing of worksheet XML; (ii) viewport-bounded rendering that lays out only $[r_t:r_u] \times [c_t:c_u]$ (Algorithm Two-Dimensional Grid Viewport Rendering); (iii) AABB-based filtering of selection sources with cache invalidation (6.4 AABB-Indexed Selection Lookup); and (iv) on-demand flattening of the output table (6.5 Lazy Output Flattening and Row Location). Together these changes bound per-interaction work to the visible area rather than the sheet size.

On a sheet with 10^6 rows and a selection covering all cells, the UI remained smooth during scrolling, sustaining ≈ 2 ms per frame (Fig. 12), which corresponds to ≈ 500 FPS. When dragging the scrollbar—filling the entire viewport at once with new content—frame times rose to about 60-80 ms ($\approx 12\text{-}17$ FPS), which remained acceptable for rapid navigation at that scale.

Benchmarking environment. All tests were conducted on a machine with an AMD Ryzen 5 PRO 7535U with Radeon Graphics (6-core CPU at 2.9 GHz), 31.3 GB RAM, and a hard disk drive (HDD), running Microsoft Windows 11 Enterprise (build 26100, 64-bit). The version of Microsoft Excel used was version 16.0.

8 Conclusion

In this paper, we introduced the Spreadsheet Data Extractor [2], an enhanced tool that builds upon the foundational work of Aue et al. [3]. By addressing key limitations of the existing solution, we implemented significant performance optimizations and usability enhancements. Specifically, SDE employs incremental loading of worksheets and optimizes rendering by processing only the visible cells, resulting in performance improvements that enable the tool to open large Excel files.

We also integrated the selection hierarchy, worksheet view, and output preview into a unified interface, streamlining the data extraction process. By adopting a user-centric approach that gives users full control over data selection and metadata hierarchy definition without requiring programming knowledge, we provide a robust and accessible solution for data extraction. Our tool offers user-friendly features such as the ability to duplicate hierarchies of columns and tables and to move them over similar structures for reuse, reducing the need for repetitive configurations.

By combining the strengths of the original approach with our enhancements in user interface and performance optimizations, our tool significantly improves the efficiency and reliability of data extraction from diverse and complex spreadsheet formats.

References

- [1] Rui Abreu, Jácome Cunha, Joao Paulo Fernandes, Pedro Martins, Alexandre Perez, and Joao Saraiva. 2014. Faulty sheet detective: When smells meet fault localization. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 625–628.
- [2] Anonymous. [n. d.]. Spreadsheet Data Extractor (SDE). https://anonymous.4open.science/r/spreadsheet_data_extractor-13BD/README.md. Accessed: 2024-12-08.

- 1249 [3] Alexander Aue, Andrea Ackermann, and Norbert Röder. 2024. Converting data organised for visual perception into machine-readable formats. In *44.*
 1250 *GIL-Jahrestagung, Biodiversität fördern durch digitale Landwirtschaft*. Gesellschaft für Informatik eV, 179–184.
 1251 [4] Daniel W Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. 2015. FlashRelate: extracting relational data from semi-structured spreadsheets
 1252 using examples. *ACM SIGPLAN Notices* 50, 6 (2015), 218–228.
 1253 [5] D.J. Berndt, J.W. Fisher, A.R. Hevner, and J. Studnicki. 2001. Healthcare data warehousing and quality assurance. *Computer* 34, 12 (2001), 56–65.
 doi:10.1109/2.970578
 1254 [6] Zhe Chen, Michael Cafarella, Jun Chen, Daniel Prevo, and Junfeng Zhuang. 2013. Senbazuru: A prototype spreadsheet database management
 1255 system. *Proceedings of the VLDB Endowment* 6, 12, 1202–1205.
 1256 [7] Jácome Cunha, Joao Paulo Fernandes, Pedro Martins, Jorge Mendes, and Joao Saraiva. 2012. Smellsheet detective: A tool for detecting bad smells in
 1257 spreadsheets. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 243–244.
 1258 [8] Kaval Desai. 2024. Excel Dart Package. <https://pub.dev/packages/excel>. Accessed: 2024-12-03.
 1259 [9] Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. 2019. Tablesense: Spreadsheet table detection with convolutional neural
 1260 networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 69–76.
 1261 [10] Angus Dunn. 2010. Spreadsheets-the Good, the Bad and the Downright Ugly. *arXiv preprint arXiv:1009.5705* (2010).
 1262 [11] Julian Eberius, Christopher Werner, Maik Thiele, Katrin Braunschweig, Lars Dannecker, and Wolfgang Lehner. 2013. DeExcelerator: a framework for
 1263 extracting relational data from partially structured documents. In *Proceedings of the 22nd ACM international conference on Information & Knowledge
 1264 Management*. 2477–2480.
 1265 [12] Elvis Koci, Dana Kuban, Nico Luettig, Dominik Olwig, Maik Thiele, Julius Gonsior, Wolfgang Lehner, and Oscar Romero. 2019. Xlindy: Interactive
 1266 recognition and information extraction in spreadsheets. In *Proceedings of the ACM Symposium on Document Engineering 2019*. 1–4.
 1267 [13] Kate Lovett. 2023. two_dimensional_scrollables package - Commit 4c16f3e. <https://github.com/flutter/packages/commit/4c16f3ef40333aa0aebe8a1e46ef7b9fef9a1c1f> Accessed: 2023-08-17.
 1268 [14] Kelly Mack, John Lee, Kevin Chang, Karrie Karahalios, and Aditya Parameswaran. 2018. Characterizing scalability issues in spreadsheet software
 1269 using online forums. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–9.
 1270 [15] Sajjadur Rahman, Mangesh Bendre, Yuyang Liu, Shichu Zhu, Zhaoyuan Su, Karrie Karahalios, and Aditya G Parameswaran. 2021. NOAH: interactive
 1271 spreadsheet exploration with dynamic hierarchical overviews. *Proceedings of the VLDB Endowment* 14, 6 (2021), 970–983.
 1272 [16] Gursharan Singh, Leah Findlater, Kentaro Toyama, Scott Helmer, Rikin Gandhi, and Ravin Balakrishnan. 2009. Numeric paper forms for NGOs. In
 1273 *2009 International Conference on Information and Communication Technologies and Development (ICTD)*. IEEE, 406–416.
 1274 [17] Statistisches Bundesamt (Destatis). 2024. *Statistischer Bericht - Pflegekräftevorausberechnung - 2024 bis 2070*. Technical Report. Statistisches
 1275 Bundesamt, Wiesbaden, Germany. <https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Bevoelkerung/Bevoelkerungsvorausberechnung/Publikationen/Downloads-Vorausberechnung/statistischer-bericht-pflegekraeftevorausberechnung-2070-5124210249005.html> Statistical Report -
 1276 Projection of Nursing Staff - 2024 to 2070.
 1277 [18] Statistisches Bundesamt (Destatis). 2024. Statistischer Bericht: Rechnungsergebnis der Kernhaushalte der Gemeinden. <https://www.destatis.de/DE/Themen/Staat/Oeffentliche-Finanzen/Ausgaben-Einnahmen/Publikationen/Downloads-Ausgaben-und-Einnahmen/statistischer-bericht-rechnungsergebnis-kernhaushalt-gemeinden-2140331217005.html> Accessed: 2024-11-29.
 1278 [19] Alaaeddin Swidan, Felienne Hermans, and Ruben Koesoemowidjojo. 2016. Improving the performance of a large scale spreadsheet: a case study. In
 1279 *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 673–677.
 1280 [20] Dixin Tang, Fanchao Chen, Christopher De Leon, Tana Wattanawaroon, Jeaseok Yun, Srinivasan Seshadri, and Aditya G Parameswaran. 2023.
 1281 Efficient and Compact Spreadsheet Formula Graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2634–2646.
 1282 [21] Hannah West and Gina Green. 2008. Because excel will mind me! the state of constituent data management in small nonprofit organizations. In
 1283 *Proceedings of the Fourteenth Americas Conference on Information Systems*. Association for Information Systems, AIS Electronic Library (AISeL).
 1284 <https://aisel.aisnet.org/amcis2008/336>
- 1285
 1286
 1287
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1299
 1300