GridBook: Natural Language Formulas for the Spreadsheet Grid

Sruti Srinivasa Ragavan*
Microsoft Research, Cambridge, United Kingdom
a-srutis@microsoft.com

Zhitao Hou*
Microsoft Research, Beijing, China
zhith@microsoft.com

Yun Wang
Microsoft Research, Beijing, China
wangyun@microsoft.com

Andrew D Gordon
Microsoft Research, Cambridge, United Kingdom
adg@microsoft.com

Haidong Zhang
Microsoft Research, Beijing, China
haidong.zhang@microsoft.com

Dongmei Zhang
Microsoft Research, Beijing, China
dongmeiz@microsoft.com

ABSTRACT

Writing formulas on the spreadsheet grid is arguably the most widely practiced form of programming. Still, studies highlight the difficulties experienced by end-user programmers when learning and using traditional formulas, especially for slightly complex tasks. The purpose of GridBook is to ease these difficulties by supporting formulas expressed in natural language within the grid; it is the first system to do so.

GridBook builds on a parser utilizing deep learning to understand analysis intents from the natural language input within a spreadsheet cell. GridBook also leverages the spatial context between cells to infer the analysis parameters underspecified in the natural language input. Natural language enables users to analyze data easily and flexibly, to build queries on the results of previous analyses, and to view results intelligibly within the grid—thus taking spreadsheets one step closer to computational notebooks.

We evaluated GridBook via two comparative lab studies, with 20 data analysts new *only* to GridBook. In our studies, there were no significant differences, in terms of time and cognitive load, in participants' data analysis using GridBook and spreadsheets; however, data analysis with GridBook was significantly faster than with computational notebooks. Our study uncovers insights into the application of natural language as a special purpose programming language for end-user programming in spreadsheets.

CCS CONCEPTS

• **Human-centered computing** → Human computer interaction (HCI); Interaction paradigms; Natural language interfaces.

KEYWORDS

Spreadsheets, End-User Programming, Programming by Natural Language, Data Analysis

*Both authors contributed equally to the paper.

IUI '22, March 22–25, 2022, Helsinki, Finland

ACM ISBN 978-1-4503-9144-3/22/03…$15.00
https://doi.org/10.1145/3490099.3511161

ACM Reference Format:
Sruti Srinivasa Ragavan, Zhitao Hou, Yun Wang, Andrew D Gordon, Haidong Zhang, and Dongmei Zhang. 2022. GridBook: Natural Language Formulas for the Spreadsheet Grid. In *27th International Conference on Intelligent User Interfaces (IUI '22), March 22–25, 2022, Helsinki, Finland*. ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/3490099.3511161

1 INTRODUCTION

Data literacy is gaining increasing importance in several fields, and professionals from diverse fields (sales, finance, administration, scientific research, journalism, etc.) are required to analyze data to complete their job tasks (e.g., write a research paper, make business decisions). To do this, they engage in some form of programming [70], despite having no computing background, and having little to no formal programming training. Such data analysts are end-user programmers: they program with an end in mind [40, 50, 53].

Learning to complete data analysis tasks is not easy. Even users of WYSIWYG tools such as spreadsheets, Microsoft PowerBI[1] and Tableau[2], generally considered easy to use for end-user programmers, must spend a lot of time learning how to accomplish tasks in that environment, learning about the correct APIs, or undergoing training and seeking help from colleagues [53], or the Internet. This is even worse for analysts who use traditional programming tools such as Python or MATLAB, where they must explicitly write code for every single action – from loading the data, through analyzing it, to presenting it.

Paradoxically, although such end-user programmers might benefit from investing the time to master a data analysis tool, or a programming language, they do not do so [8, 9]. Instead, they learn the bare minimum required to get the job done, rather than learning to do it in the best possible way. They often seek help from the Internet to accomplish even simple tasks, reuse code from the Internet, constantly look up API help and "*debug their code to existence*" [40, 45]. Simply put, an analyst not only has to worry about their real task at hand, namely what analysis to conduct, but also must bear the additional cognitive burden of expressing their analysis intents in the tool or the programming language of their choice.

Can we lower this burden of translating analysis intents to programming constructs? Specifically, does the use of natural language for programming improve the time and cognitive requirements for data analysts working on data analysis tasks? In this paper, we explore the possibility of this direction via GridBook.

[1]https://powerbi.microsoft.com/
[2]https://www.tableau.com/

GridBook draws inspiration from two ideas in programming literature. The first is natural programming. GridBook allows spreadsheet users to write declarative queries in Natural Language (NL). The user can write, manipulate, and view these NL queries on the grid, and they behave just like regular spreadsheet formulas (e.g., recalculate on changes). The results of these NL formulas are rendered on the grid, right below the NL utterance. GridBook also intelligently infers the context of the NL formula, thereby supporting underspecified NL utterances that is commonly observed in human-human communication and in other NL programming interfaces (e.g., [19]).

Second, GridBook draws inspiration from Donald Knuth's idea of literate programming, where a user must be able to read a program just like an essay, top to bottom. In GridBook, self-documenting NL formulas are interleaved with results of the calculations on the grid. Moreover, progressive analysis where a user can build analysis on builds on top of previous ones. Together, these capabilities take GridBook one step closer to literate programming environments, exemplified in computational notebooks.

GridBook is not the first system to facilitate end-user programming using natural language or bring natural language programming to spreadsheets (e.g., [22]). However, it is the first tool that allows for NL formulas to live and be manipulated in the spreadsheet grid, and to be re-calculated when inputs change, just like regular spreadsheet formulas. In contrast, most prior work at the intersection of NL and spreadsheets has focused on synthesizing programs from NL input, after which a programmer must work with the synthesized code, rather than the original natural language.

We evaluated GridBook via two in-lab comparative studies, comparing it against conventional spreadsheets and computational notebooks, respectively:

- In Study A, we compared GridBook against traditional spreadsheet software with experienced spreadsheet users (N=10). There were no significant differences between GridBook and Microsoft Excel, both in task completion times and cognitive load. Still, when participants reflected on their experience learning advanced Excel features, they expected GridBook to be simpler for beginners to learn and use.
- In Study B, we compared GridBook with Jupyter notebooks (and Python). Participants (N=10) were significantly quicker in completing data analysis tasks in GridBook, than in Jupyter notebooks, even though they were using GridBook for the first time but had prior experience using Jupyter and Python.

These results highlight the promise of natural language as an alternative to traditional programming languages for end-user data analysis. Our study also reveals various requirements for building natural-language programming systems, as well as for bringing computational notebook capabilities in spreadsheets.

2 RELATED WORK

GridBook takes spreadsheets one step closer to literate programming tools, namely computational notebooks. It also aims to make data analysis more natural for analysts. Thus, related work for GridBook is in natural language programming (especially, for data analysis), computational notebooks, and general spreadsheet literature.

2.1 Natural language programming

The idea of using natural language to make programming simpler for novices goes back to at least the 1960s [34]. Proponents of "natural" programming—that programming should be as close as possible to the way programmers think about their intent—also consider natural language as one of the ways of making programming natural, thereby reducing the overhead of translating thoughts into the constructs of a programming tool or language [57]. Following these arguments, several researchers (e.g., [7, 49, 51, 58]) have conducted human studies analyzing the utterances of people to NL systems and other people, to understand what affordances an NL-based programming environments should support. Others have engineered programming systems that accept natural language input for various domains (e.g., [4, 62]). Unfortunately, these earlier systems have not met with great commercial success, largely due to limited natural language understanding capabilities.

In recent years, developments in machine learning have led to renewed interest in natural-language programming. There is a lot of recent work on conversational interfaces to synthesize code from natural language inputs, as well as on the design of the dialog between programmers and the programming. For example, fuSE [89] can synthesize descriptions, methods, and API signatures from NL utterances with a high degree of accuracy. SmartSynth translates NL voice input to generate workflows for mobile phones [47]. Tools like HISyn and Open AI's Codex generate general-purpose code from NL inputs [52, 56]. Research also exists on using NL for programming robots, and targets both professional and end-user programmers. (e.g., [35, 46, 72, 78]).

In addition to generating code from NL utterances, researchers have also explored the use of NL to help developers query their code and other software-development artifacts for tasks like debugging (e.g., [14]), or to seek help (e.g., [79]), or to explain a program's working (e.g., [17]). The difference between GridBook and this body of work is that it is targeted towards end-user programmers and for data analysis tasks.

A well-studied area in NL-based data analysis is visual analytics; several tools and systems have attempted to solve various challenges arising from issues of interpretation, ambiguity, explanations, and interface design in NL systems for visualizations. For example, Orko, a multi-modal interactive visualization system uses NL as one of the different modes of input [76]. Setlur and colleagues have built a set of NL-based data visualization tools, offering various affordances for building queries on top of each other, or for disambiguation [29, 74]; some of these are implemented in the commercial tool, Tableau. DataTone is an interactive NL-based visualization environment, that includes "disambiguation widgets" to help disambiguate ambiguities in NL inputs [19].

Another widely explored area is Natural Language Interfaces to Data Bases (NLIDB) that synthesize SQL queries from NL input for end-user programmers to analyze data. Earlier work in the area involved NL interfaces for specific databases (e.g., [88]), followed by more generic tools to be used across multiple databases (e.g., [77]). Recently, with advancements in machine intelligence, and

the release of the large datasets such as WikiSQL [96] and Spider [94], there are a lot of efforts on generating SQL from NL. These tools use a variety of underlying NL processing and code synthesis technologies; some also include human-in-the-loop feedback (e.g., [16, 30, 31, 33, 48, 87]]). See [86] for a broad overview of prior work in the area.

GridBook differs from these tools in the following ways. 1) GridBook is geared towards the spreadsheet domain, with the constraints of the two-dimensional grid interface. 2) It preserves the user's NL utterance for the user to revisit and manipulate it later. 3) The fact that GridBook's results are automatically recomputed with changes to the original data (as with formulas) imposes a unique set of design and implementation challenges, unlike in prior tools where the analysis is manually run to fetch results at a given snapshot in time.

In the area of spreadsheets, one of the first works to employ NL formulas is Gulwani et al.'s NLyze [22]. NLyze lets spreadsheet users create a single formula from a natural language query. It provides multiple formula options to the user when there is ambiguity. NLyze also supports AI interpretability via highlighting which parts of the query are used (e.g., strikethrough parts of the query not included in the final formula). Gulwani and colleagues have also combined NL inputs with programming by example approaches to ease behavior specification for the user [64]. Unfortunately, evaluation of NLyze with users is limited.

Wachtel and colleagues have recently employed deep learning to synthesize spreadsheet formulas from natural language. Their system JustLingo allows the user to specify behavior using NL in a chat-like dialog [83]. The dialog interface also asks the user for missing information, and to help disambiguate their intent. Their user evaluation shows promising results for NL interface for end-user programming in spreadsheets. Encouraged by the initial success, they have refined their machine learning models with table detection to let users specify context easily [82] and to support analysis of heterogenous data from multiple sources (e.g., databases) [81].

But, unlike GridBook, these prior works do not preserve the NL utterance originally used to arrive at the formula; as a result, future edits by the user must happen via the formula, or the user must come up with the right natural language utterance again. Another distinguishing aspect of GridBook and prior NL-based spreadsheet formula works is in automatically inferring context. Wachtel et al.'s JustLingo partially supports this by allowing users to specify the context of the analysis, but it happens as part of the dialog between the user and JustLingo [82]. The third difference is that GridBook allows progressive analysis, where a user can analyze data from prior results. Finally, the results of the analysis are presented interleaved with the NL formula used for the calculation.

Going beyond research tools, commercial spreadsheet tools have also taken advantage of machine intelligence, including natural language, to make data analysis easier for their users. Specifically, the Explore feature in Google Sheets [13], and the Analyze Data feature in Microsoft Excel allow a user to ask questions in natural language about the data in the spreadsheet. The output of the question is usually a pivot table, or a chart, or formulas that a user can then place in their spreadsheet. As with GridBook, these tools automatically infer the context of the NL query (e.g., user doesn't have to specify cell ranges for analyzing data). However, unlike GridBook, the NL queries are not always preserved, and the results are not automatically recalculated on modifications (as happens with regular and GridBook formulas).

2.2 Literate programming and computational notebooks

The idea of literate programming dates back to a 1984 paper [38], where Donald Knuth argues for code to be as readable as a good literary essay. Ideally, a literate programming tool lets a programmer "arrange the parts of a program in any order and extract documentation and code from the same source file", to make it comprehensible.

Computational notebooks are considered to best exemplify this idea. The cells of code and output are essentially chunks, and a programmer can independently run, move, copy, add or delete chunks. Cells can contain code, its output, and documentation – thus a programmer can read the code, interleaved with data, results and visualizations, building on top of each other, and top to bottom, as they would other literature. These affordances are considered particularly valuable for exploratory data analysis, for which these tools are widely adopted [44].

Over the last decade, data has become central to organizations and governments, and data literacy has gained importance. Therefore, there is a lot of recent interest in the HCI research community around computational notebooks and supporting exploratory data analysis. Research largely falls into the following three categories. First, there are *new computational notebooks* exploring new paradigms in various design aspects; over 60 such commercial and research tools are summarized in detail in [44]. Second, there are user studies investigating how analysts use computational notebooks and what challenges they face (e.g., [6, 11, 84]). Finally, researchers have built tools *on top of* computational notebooks, to support activities such as visualizations, versioning, and documentation generation, to better support user needs (e.g., [24, 37, 66, 67, 85, 90]).

A recent paper by Lau et al. [45] brings together computational notebooks and spreadsheets. Via interviews, they found that data scientists often use computational notebooks together with other tools, including spreadsheets. For example, they do data cleaning in spreadsheets, then export the cleaned data to computational notebooks for analysis, and then export that data back to spreadsheets for visualization and presentation. To reduce back and forth between tools, they built TweakIt that integrates spreadsheets with Jupyter notebooks, along with affordances to support reuse of python code snippets. Whereas in TweakIt spreadsheets live alongside and are linked to traditional computational notebooks, in GridBook we aim to turn the spreadsheet itself into a computational notebook. The advantage of the latter approach is that an analyst does not have to learn a traditional programming language like Python, but instead use familiar NL alongside familiar spreadsheet formulas [57].

Since GridBook aims to bring to spreadsheets the advantages of computational notebooks (or rather a computational board, because the grid is two-dimensional), Figure 1 illustrates, using the classification framework proposed by [44], where GridBook's affordances lie in the design space of computational notebooks. In Section 3, we will discuss in detail the design of GridBook.

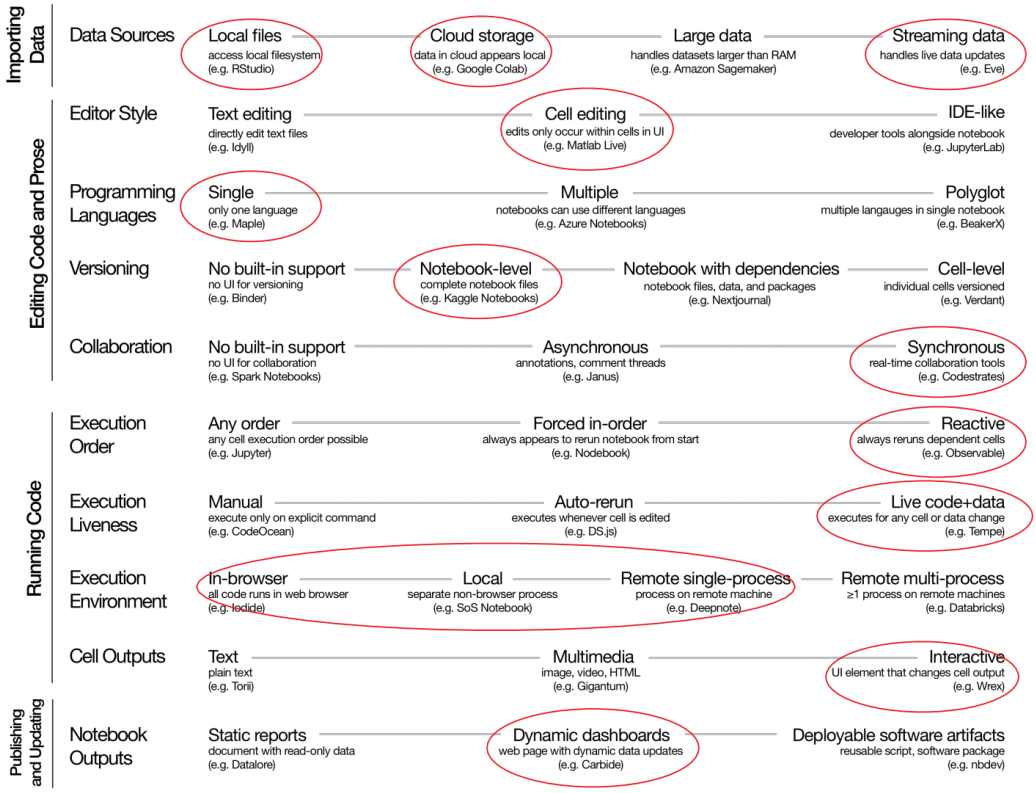

Figure 1: GridBook in computational design space. The red highlights (circles) indicate the design choices available in GridBook, in the computational notebook design space as illustrated by Lau et al. [44].

2.3 Other spreadsheet literature

Going beyond research on data analysis and NL-to-formula synthesis in spreadsheets, a vast body of spreadsheet literature has been built over three decades.

Some of the earliest works are on spreadsheet use in organizations [53]. Spreadsheet errors are a major area of research, and researchers have studied their occurrence, detection, prevention, and correction. [59] and [61] offer an in-depth review of the spreadsheet error literature. There is also a sizeable body of literature on end-user software engineering calling for rigorous software engineering practices (e.g., design, type checking, testing, refactoring) to be adopted for critical spreadsheets. Notable works include those of Margaret Burnett (e.g., [7]), Gregg Rothermel [65], Martin Erwig (e.g., [1]) and Felienne Hermans (e.g., [26]); see [5] for a detailed review of this area. There is also literature on user activities such as spreadsheet authoring [25], debugging [21], comprehension [43, 75], and reuse [36], and various tools to support these activities (e.g., [28, 36, 68]).

Pertinent to this paper, there is prior research on better supporting data analysis in spreadsheets, using WYSIWYG rather than NL; examples are [3] and [69]. There are also papers that apply machine learning to better understand the semantics of data in spreadsheets (e.g., [15, 41, 42]), or to build intelligent experiences on top of spreadsheet data (e.g., formula prediction [12], clone detection [95], error detection [91]).

The above summary is only a bird's eye view of the broader spreadsheet literature and is intended for the sake of completeness. The overall spreadsheet literature is mostly unrelated to the main ideas in this paper, and in any case is too vast to discuss in detail.

3 GRIDBOOK: DESIGN AND IMPLEMENTATION

In this section, we describe: 1) GridBook's design goals, 2) the interactions and user experiences that realize these goals, and 3) the details of the underlying implementation. A web implementation of GridBook is available at [97].

3.1 Design goals

Our primary design goal with GridBook is to make data analysis for end-user programmers simpler and as natural as possible. Based on recent work on challenges with computational notebooks [11] and an interview study with data analysts on their workflows [45], we formulated four design goals (summarized in Table 1).

Table 1: Four design goals for GridBook.

	Design Goal	Evidence	Description
D1	Minimal learning	[11, 44]	Lower barriers to learning the syntax, semantics and APIs of a programming language.
D2	Forgiving syntax	[11, 44]	Lower barriers to programming, reduce errors, and improve productivity by syntax forgiving of misspellings, cases, or the use of synonyms (e.g., "total" or "add" in place of "sum").
D3	Minimize re-analysis efforts on change	[45]	Do not require fixing code, or re-running analysis every time the data is changed (e.g., recurring data), or an intermediate computation is changed.
D4	Literateness	[44]	Make the analysis code literate, with automated documentation and interleaved results, and to build one analysis on top of other top-down

Design goal #1: Minimal learning. A characteristic attitude of end-user programmers is to focus on task completion, learning only the bare minimum required to get the job done. We wanted GridBook to be inclusive of such task orientation in end-user programmers [40] and therefore *lower* the barrier to learning the syntax, semantics, and APIs of a programming language. We also wanted to reduce the need for users to switch between two modes, namely "what analysis to perform" and "figuring out how to do something in a language" (e.g., via Google search or look up documentation or ask a colleague). Data analysts from Lau's study [45] described such switching as "inefficient, annoying and time-consuming".

Design goal #2: Forgiving syntax. Traditional programming languages are not very tolerant of simple errors such as misspellings, indentations, spaces, parameter ordering or use of alternative vocabulary (e.g., "total" instead of "sum"). As a result, users must spend additional time debugging and recovering from such simple errors [11]. Our second design goal, therefore, is to lower the barriers imposed by rigid syntax and semantics raised by a traditional programming language and make programming more forgiving of user errors.

Design goal #3: Minimize re-analysis efforts on change. Analysts often engage in repetitive tasks; for example, they must generate reports every week, or run the same analysis on the data from various trials of the same experiment. With state-of-the-art computational notebooks, they must manually re-run the analysis even with small changes to an intermediate step in the analysis (e.g., sort some data ascending instead of descending). We want to eliminate the effort and possible errors from manually re-running the analysis code, especially when they are under time pressure [45].

Design goal #4. Literateness. Finally, the popularity of computational notebooks demonstrates the practical value of literate programming systems where code, documentation, results are all interleaved, and make for easier reading and comprehension of programs. They also allow for analyses to be built on top of previous ones, a common phenomenon in data analysis (e.g., [74]). Such affordances let an analyst easily tell a story with their analysis. We want to preserve this "literateness" of computational notebooks.

3.2 GridBook design

To realize our design goals, we first looked at what end-user data analysts already know. Spreadsheets are a widely popular tool that various people across domains use to organize and analyze data [70]. Even users of traditional computational notebooks, or other scripting languages, use Excel as part of their data analysis workflow [45]. Spreadsheets are also hailed as an exemplar for easy-to-use programming environments, and this is one of the reasons for their wide popularity [71]. Therefore, we chose to retain spreadsheets as the primary data analysis environment to realize the design goals listed above.

However, conventional spreadsheets present challenges for users to master advanced features [10]. For example, using advanced formulas poses the same challenges of API discovery, and using pivot tables requires users to form a mental model of how to think about, and frame, their analysis. To eliminate this need, we chose natural language to help users directly express their intent in their spoken language.

We, however, did not want natural language as a substitute for regular spreadsheet programs (via formulas). Instead, we wanted the two to seamlessly coexist and work together. That way, users will be able to leverage the ease of use of NL formulas, while also being able to fall back to familiar regular formulas for very specific analytical needs. We also wanted users to pick the best tool—natural language or formulas—for their task.

In the rest of the section, we discuss the finer details of GridBook's design. For this, we will walk through how a fictional user Anna performs exploratory data analysis using GridBook. Anna manages the sales in a car dealership. She has a spreadsheet containing the sales for various car models (Figure 2; section 1). She wants to understand the sales of SUV, and their contribution to the overall sales for each manufacturer.

To accomplish this task, Anna must first filter the data to obtain rows for the SUV category, and then add the sales for each manufacturer. She must then remove the filter and compute the totals for each manufacturer. Then, she must compute the fraction of sales value to which SUV sales contribute. Thus, this is a relatively complex task and can be hard to perform using Excel formula functions, pivot tables or even SQL (because it requires joins and nested queries). However, in GridBook, Anna can use plain English to specify the task in a declarative way.

Anna first selects an empty cell nearby the car sales table and enters a natural language formula ':list all brands' to gather the manufacturers from the data table (Figure 2 (a)). All that Anna must do to see the result is to type the colon prefix followed by a brief specification of her intent in familiar natural language. She is not required to perform any additional setup to import the data or run the query; nor does she have to learn the way to express the intent

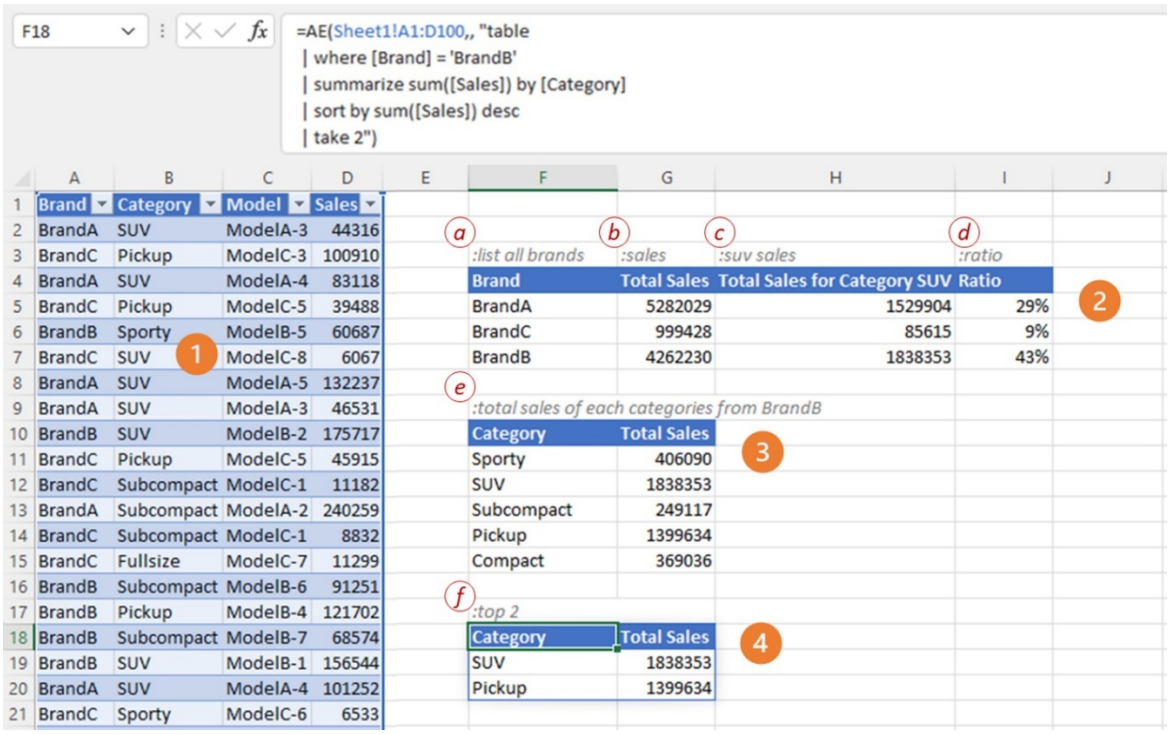

Figure 2: The GridBook Interface. Section (1) shows the data table. Section (2) shows progressive exploration of data, column wise. Section (3) shows a complex query returning multiple columns of results. Section (4) demonstrates progressive exploration.

in an artificial language, such as the one shown in the formula bar. All Anna needs to know at this stage are the colon prefix and the NL querying affordance it offers. Just as formulas in spreadsheets have an equals (=) prefix (e.g., =A1+B2), NL queries in GridBook are prefixed with the colon sign.

Realization of design goal #1 (Minimal learning): GridBook allows for natural language to analyze data; lowering the effort required for users to learn complex semantics and API of an artificial programming language (e.g., Python).

At this point, the following question arises: how does Anna know that she needs to use the exact words ':list all brands' to get a list of unique brands? Our second design goal, namely forgiving syntax, minimizes exactly this need. GridBook seeks to allow a user to arrive at the same result using literally different but semantically equivalent utterances for the same intent; some examples are ':all brands', ':distinct brands', ':brand names' or even just ':brands' or ':Brand'.

> *Realization of design goal #2 (Forgiving syntax): GridBook's NL syntax is more tolerant of errors such as misspelling, casing, or the use of synonyms (instead of exact keywords), than traditional programming environments which stop on such errors.*

Also notice that, in the natural language utterance ':list all brands', Anna does not need to specify the exact ranges of cells denoting the desired data table and data column. GridBook interprets the natural language utterance in the context of the spreadsheet grid and infers that 'brands' refers to the 'Brand' column in the data table ranged from cells A1 to D100. In effect, this context-inference reduces the need for Anna to learn and deal with the syntactical specifics of table ranges, or to scroll to look up the table's boundaries, or explicitly assign the input to a named variable and then refer to it. Such contextual understanding allows interpretation of underspecified NL utterances, a phenomenon in how humans communicate with each other using NL. The idea has also been explored and found helpful for users in prior work in NL-based programming (e.g., [74]).

When Anna types the NL query, the following happens under the hood. Based on natural language understanding and the context it has inferred, GridBook generates from the NL utterance a corresponding formula and inserts it into the cell below the NL formula. The result of this formula is a table that is returned as a dynamic array.

A dynamic array is a two-dimensional array of values; if a formula in a cell evaluates to a dynamic array, items from the array may populate adjacent cells by spilling below and to the right. Dynamic arrays are a recent addition to spreadsheet implementations [92]. In the example in Figure 2, GridBook interprets the phrase ':list all brands' in cell F3 as a dynamic array formula that computes the unique brands in the column Brand, and inserts the formula below (F4). The formula evaluates to a 4x1 dynamic array, which spills from cell F4 into cells F5, F6 and F7. The top row in the dynamic array comprises of the headers, as seen row F4. Anna can click on

one of the cells in the spilled array to inspect the formula in the Excel formula language within the cell, or in the formula bar, just like they can do with regular formulas.

Continuing with Anna's task, now that she has fetched the list of manufacturers, Anna wants to juxtapose the results with the total sales for that manufacturer. For this, Anna simply types ':sales' in G3, right next to the previous query for brands (Figure 2 (b)). Since Anna has typed in ':sales' right next to the brands, GridBook infers that Anna's intent is to summarize the sales by each value in the list of manufacturers. She does not specify the scope for aggregating sales numbers in her natural language utterance explicitly, nor the criteria for the grouping. Similarly, Anna uses ':suv sales' in cell H3 to get the total sales of SUVs for all manufacturers (Figure 2 (c)).

Finally, Anna enters a natural language formula ':ratio' in cell I3 for calculating the percentages of contribution of SUV sales for each manufacturer (Figure 2 (d)). Based on the context of neighboring columns, GridBook can resolve the parameters of the ratio calculation, which are underspecified in the utterance.

The above walkthrough illustrates how Anna may use successive NL formulas to progressively build a complex analysis table. It demonstrates the ease of performing exploratory data analysis in GridBook.

Figure 2 also shows more complex queries, where multiple columns are returned from a single utterance, as against building the analysis table one column at a time. For example, ':show me sales of each category from Brand B' returns a table with two columns (Figure 2 (e)). In the following step, when the user types the utterance ':top 2', GridBook infers the context as the result from the previous analysis, based on spatial proximity (Figure 2 (f)).

The next question is: what happens when the data changes, or when an intermediate calculation changes, or if Anna simply wants to change a part of the analysis (e.g., when Anna changes ':suv sales' to ':pickup sales')? In GridBook, NL formulas are at par with regular formulas. Therefore, just as changes to data or formulas are automatically recomputed in spreadsheets with regular formulas, changes are also propagated and instantly recomputed for NL formulas. Thus, the What You See Is What You Get (WYSIWYG) quality of spreadsheets' automatic recalculation is preserved in GridBook's NL formulas. The fact that changing NL changes the calculations also allows for easier explorations for users, who do not have any overhead of re-running a calculation to propagate a change (as they must do in conventional computational notebooks). By implementing NL formulas using compilation to regular formulas, we obtain automatic re-calculation, and hence achieve Design Goal #3, namely minimal re-analysis effort on changes.

> *Realization of design goal #3 (Minimize reanalysis efforts on change):* GridBook's NL formulas are like regular formulas; change to either NL utterance or to data leads to automatic recalculation of the analysis.

Finally, as Figure 2 shows, the NL utterance remains on the grid, and does not disappear after the formula is synthesized. The result of the computation also appears right below the NL query. Thus, code and results of the code are interleaved in GridBook just as in conventional spreadsheets. Moreover, NL utterances are self-documenting in that they tell a user which query was used to generate a result, as well as what the code is doing. Hence, GridBook takes us a step closer to the vision of literate programming in spreadsheets, and of spreadsheets as computational notebooks (Design goal #4).

> *Realization of design goal #4 (Literateness):* The NL utterance remains on the grid, and the results of the analysis show up right below each utterance. Since self-documenting NL formulas and their results are interleaved, GridBook takes spreadsheets closer to computational notebooks.

3.3 System overview

GridBook is implemented as an Excel Web add-in to integrate NL formulas into Excel. GridBook monitors the cell editing events that occur in the spreadsheet grid. When a user enters a string starting with ':' in a cell, GridBook identifies it as an NL formula and sends the corresponding utterance to a backend service, which attempts to translate it to a regular spreadsheet formula. The corresponding spreadsheet formula is returned, and the results of the calculation are rendered on the grid, right below the NL formula.

Figure 3 shows the processing pipeline of GridBook. It consists of three core components:

- **Context Provider:** GridBook identifies various contextual objects from the grid to provide the grid context for understanding the NL queries and synthesizing formulas. We describe it in Section 3.4.
- **NL Interpreter:** The NL Interpreter parses the input NL query to identify analysis terms to represent the analysis semantics conveyed in the utterance. We describe it in Section 3.5.
- **Formula Composer:** Formula Composer generates a corresponding regular spreadsheet formula to fulfill the analysis task specified in NL. Specially, we implement a custom formula function based on Excel formula language as the target formula. We describe it in Section 3.6.

3.4 Context provider

To interpret an NL utterance in the context of the spreadsheet grid, GridBook considers three types of contextual objects on the grid: data tables, analysis tables, and neighboring columns.

Data tables. In regular Excel formulas, users need to use cell or range addresses to specify what data to manipulate, which is tedious and error prone. By leveraging data table objects as context, GridBook enables specifying data entity names in NL formulas; we believe this is a more "natural" way for users to express their analysis intents. For example, when processing the NL formula in Figure 2 (a), GridBook looks up the context of data tables in the spreadsheet and infers that '*brands*' refers to the '*Brand*' column of the data table in Figure 2 (1).

To facilitate this, GridBook capitalizes the fact that spreadsheet users commonly organize their data in a column-major tabular format on the grid. These data tables often serve as the data source against which users conduct data analysis. Common spreadsheet packages allow two kinds of data tables. An *explicit data table* arises when the user formats a range of data as a table (e.g., using the Ctrl+T keyboard shortcut in Excel), and the table boundary and metadata (e.g., column names, column data types) are maintained

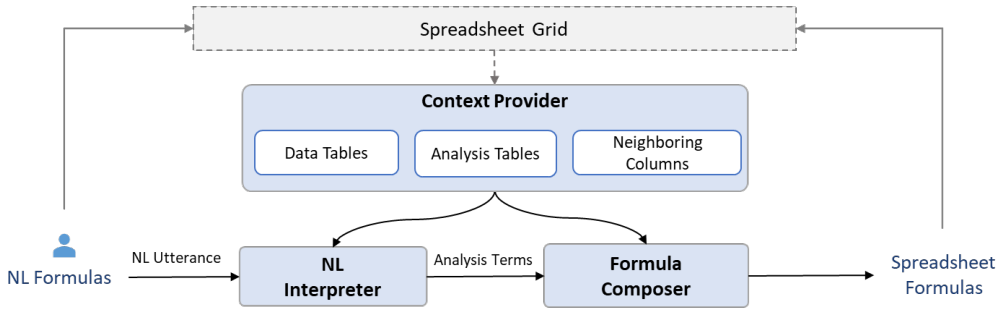

Figure 3: The GridBook Workflow.

Table 2: The NL interpretation results and generated Excel formulas for NL formulas (a) – (f) in Figure 2. The NL interpretation output includes: (a) Abstract Terms, (b) Analysis Role Terms, and (c) Analysis Intent Terms.

NL Formula	NL Interpretation								Excel Formula
:list all brands	(a)	*List*	*all*	*<column>*					=AE(A1:D100, F4#, "table | summarize by [Brand]")
	(b)	O	O	B-Topic.Field					
	(c)	query;							
:sales	(a)	*<measure>*							=AE(A1:D100, F4#, "table | summarize sum([Sales])")
	(b)	B-Topic.Field							
	(c)	query;							
:suv sales	(a)	*<value>*	*<measure>*						=AE(A1:D100, F4#, "table | where [Category] = 'SUV' | summarize sum([Sales])")
	(b)	B-Value.Filter	B-Topic.Field						
	(c)	query;							
:ratio	(a)	*ratio*							=H5:H7/G5:G7
	(b)	O							
	(c)	ratio/percentage;							
:total sales of each category from BrandB	(a)	*Total*	*<measure>*	*of*	*each*	*<column>*	*from*	*<value>*	=AE(A1:D100, , "table | where [Brand] = 'BrandB' | summarize sum([Sales]) by [Category]")
	(b)	O	B-Topic_sum.Field	O	O	B-Topic.Field	O	B-Value.Filter	
	(c)	query;							
:top 2	(a)	*Top*	*<number>*						=AE(A1:D100, , "table | where [Brand] = 'BrandB' | summarize sum([Sales]) by [Category] | sort by sum([Sales]) desc | take 2")
	(b)	O	B-TopN.Number						
	(c)	filter; sort							

explicitly in the spreadsheet package and can be retrieved via an API. Figure 2 (1) shows an example of an explicit data table. There may also be implicit data tables on the grid. An *implicit data table* occupies a rectangular range of cells with data stored in table structure where the user has not explicitly signaled that it forms a table. In the current version, GridBook recognizes only explicit data tables. (Excel has a proprietary feature for repeated calculations also known as a "data table" – instead, we use the term to mean a range of data organized as a column-major table.)

Analysis tables. An *analysis table* (or analysis result) is a range of the grid computed by the formula obtained from an NL query. An analysis table appears just like a data table, but instead of being input data it is the output of an analysis. Analysis tables are computed from one (or potentially more) data tables via one or more formulas in the grid. They are generated from data tables through analysis queries or formulas. In Figure 2, tables (2) - (4) are all analysis tables generated via NL formulas based on the data table (1). When the source data table changes, the dependent analysis tables are updated accordingly.

GridBook manages the metadata of each analysis table in the grid: this includes table boundaries, table structures, and corresponding formulas. It utilizes these analysis tables as important context for synthesizing formulas from NL, to allow progressive data analysis (i.e., queries issued on top of the previous analysis results). As illustrated in Figure 2, based on the context of analysis tables on the grid, GridBook automatically infers that the NL formula (f) is a query to follow up the nearby analysis table (1). Thus, GridBook incorporates the analysis specification of Figure 2 (1) accordingly when generating the Excel formula for Figure 2 (f).

Neighboring columns. As Figure 2 illustrates, in GridBook, users can divide a whole analysis into steps and build the analysis results incrementally by using NL formulas to append new columns to an existing analysis table. The neighboring columns provide important context which GridBook can leverage to resolve underspecified

arguments and infer the analysis intents of a new NL formula. For example, Figure 2 (table 2), ':sales' and ':suv sales' depend on the first column 'Brand' as context for conducting aggregations for each manufacturer, while ':ratio' uses neighboring columns "sales" and "suv sales" for calculation.

3.5 NL interpreter

Given an input utterance, the NL Interpreter aims to translate it into a logical form as a formal representation of its meaning. Specifically, it uses a set of analysis terms extracted from the NL utterance and turns it into a logical form, to capture its semantic meaning.

There are two types of analysis terms: Analysis Role terms and Analysis Intent terms.

Analysis Role terms represent specific elements of an analysis query. To identify a set of Analysis Role terms that cover a variety of aspects commonly involved in data analysis, we examined the specification of data query languages (e.g., SQL), surveyed data analysis related features in spreadsheet and BI applications (e.g., Excel, Power BI, and Tableau), and reviewed research that summarized analysis tasks and intents (e.g., [2, 73]), and initially defined 78 types of Analysis Role terms in our implementation. For example, a *B-TopN.Number* term marks the number operand of selecting top N items (e.g., 'top **2** brand by total sales'); a *B-Filter.Subject* term identifies the subject upon which filtering is operated (e.g., 'filter by **Sales** > 10M'); a *B-Topic.Field* term annotates data columns to be queried (e.g., '**sales** by **category**'); a *B-Topic_sum.Field* indicates a data column to be aggregated with SUM function (e.g., 'total **sales** in USA'); etc (Table 2). Our set of Analysis Role terms is not exhaustive, but extensible.

Analysis Intent terms capture the high-level analysis intents of an NL utterance. In our implementation, we categorize several high-level analysis intents including *query* (e.g., 'sales by brand'), *filter* (e.g., 'exclude category SUV'), *sort* (e.g., 'sort by sales in descending order'), and *ratio/percentage* (e.g., 'ratio of SUV sales'). A single NL utterance may express multiple analysis intents.

The GridBook interpreter consists of two steps: *token recognition* and *analysis term extraction*. At the token recognition step, it recognizes special terms from the input utterance and replaces them with abstract terms to obtain an abstracted utterance. At the analysis term extraction step, it takes the abstracted utterance as input and uses a deep learning-based model to extract analysis intent terms and analysis role terms as the output of the interpreter. Table 2 summarizes the outputs of each stage in NL Interpreter for Anna's NL queries (Figure 2).

Token Recognition. To reduce the vocabulary and uncertainty of analysis term extraction, we first use a token recognizer to identify and abstract data table related entities in the utterance. Based on the contextual data tables on the grid, the recognizer enumerates through the n-grams in the utterance to find entities (words or phrases) that can be matched with the column names and cell values of a data table. We obtain the abstracted utterance by replacing these entities with abstract terms like *<column>*, *<value>*, etc. In addition to entities related to data tables, we also identify numbers, dates, and quoted strings in the utterance by regular expression matching and then replace them with corresponding abstract terms such as *<number>*, *<date>*, and *<string>*.

Analysis Term Extraction. Central to the interpreter is the ability to predict the analysis terms in the abstracted utterance. We use two sets of analysis terms, namely intent terms and role terms, to capture both high-level analysis intention and low-level analysis elements. Therefore, we treat this step as two NLP sub-tasks:

- *intent classification*, a multi-label classification task to detect potentially multiple intents from an utterance,
- *role tagging*, a sequence labeling task using BIO tagging [63], to recognize token chunks that represent various analysis roles from one sentence.

The implementation uses models based on deep learning. While various prior work used rule-based methods (e.g., both FlowSense [93] and NL4DV [54] use lexical and dependency parsing structures) for precise recognition, we argue that heuristic rules usually only handle limited forms of utterances. In comparison, deep learning models are more capable of flexibly interpreting diverse utterances. Moreover, one can easily extend the capability of the NL interpreter by adding more training data.

We designed a neural model based on the Transformer model architecture [80]. The model simultaneously performs the two tasks mentioned above, outputting separate lists for analysis intent terms and analysis role terms, respectively. Specifically, it stacks multiple Transformer encoder blocks, which connects to a Dense layer for predicting Analysis Role terms and connects to a pooling layer for predicting Analysis Intent terms.

We train our model on a dataset constructed from two sources. We first converted 28K utterances from a proprietary NL2SQL corpus labeled by human annotators with format as a tuple of input utterance and expected SQL statements. To convert the dataset, we built a tool to automatically annotate the analysis terms by aligning the SQL slots in the ground truth queries. To further increase the coverage of the NL formula query, we used Amazon Mechanical Turk to collect 1.2K utterances specifically for short queries, such as "Sort" and "Ratio", which do not have sufficient occurrences in the NL2SQL corpus. In total, our dataset contains 29.2K English utterances. For reasons of data privacy and confidentiality, we are unable to make the NL2SQL corpus public.

We split the dataset into two pieces, a training set including 2/3 of the 29.2K data, and a testing set including the rest of 1/3 of the data. Our Transformer is configured as follows: the number of transformer encoder block layers is 2, the embedding dimension is 128, the feed forward dimension is 256, and the number of heads is 4. The Analysis Role term accuracy on the test set is 0.9845 with F1 score of 0.9555. The Analysis Intent term accuracy on the test set is 0.9987. These results show that our model can interpret NL queries reasonably well.

3.6 Formula composer

The Formula Composer takes analysis terms extracted by the NL Interpreter as input, resolves ambiguity using the grid context, and produces an Excel formula as output. Table 2 lists the generated Excel formulas for various NL queries, based on Anna's example from Figure 2

Some analysis intents, such as calculating ratios or percentages, can be resolved as simple calculations. For them, the composer

directly generates regular Excel formulas using corresponding calculation functions. For example, as shown in Table 2, '*:ratio*' query is translated to a divide calculation among two numerical columns. The composer identifies the two involved columns from the neighboring columns and generates a corresponding Excel divide function as result.

For complex tabular queries, we provided an Excel custom function *AE* implemented using Excel LAMBDA functions [20] (AE stands for Analysis Expression). The syntax of the AE function is as below:

= AE(Table Range, Context Column,Query String)

- **Table Range** is the address of the source table against which the query is conducted. It could be a data table or an analysis table.
- **Context Column** (Optional) is the address of a column region containing a list of distinct categorical values. When provided, each query will be calculated using the row column context filter for querying the specified Query String.
- **Query String** is an Analysis Expression, a query statement using piped data-flow syntax to specify data queries in a declarative way. The query language is similar to other data analysis languages such as SQL and Kusto[3].

To compose AE formulas from the NL interpreter's output, the first step is to determine the source table and fill its address into the first parameter of the AE custom function. The current implementation only supports analysis queries on a single table. We select the table whose columns or values are mentioned in the utterance as the source table of the query.

Then, the composer synthesizes the *Query String*. Given the analysis intent terms, analysis role terms and entities extracted by the NL interpreter, we take a bottom-up approach to synthesize the terms into a target query language expression (AE). The target query language AE is a pipeline of expression trees. Each piped query statement takes a table as input and outputs a new table. For example, the NL utterance ':show me models for BrandA with total sales greater than 2000' can be turned into an AE query string:

= table
| where [Brand] = 'BrandB'
| summarize sum([Sales]) by [Model]
| where sum([Sales]) > 2000.

To achieve the query string in the example, the '2000' token labeled as '*B-GreaterThan.Target*' generates an expression sketch "[?] > 2000" while the 'Sales' term labeled '*B-sum.Field*' generates another expression sketch "sum([Sales])". We traverse the analysis terms to find corresponding analysis operations on corresponding entities, such as aggregation, group by, filtering, etc. During the process, we use heuristic rules to link between entities and to fill smart defaults when slots are underspecified. For example, by applying a heuristic rule that "sum(Sales) is numerical value" and "? > 2000" require a numerical subject, we can combine the two expression sketches to generate a filter expression "sum([Sales]) > 2000".

During the construction, the composer also fills the underspecified slots by inferring them from the grid context with similar strategies as [73]. For example, the query ':sales after Oct 2019' does not specify which column to filter the date, so the composer looks up the data table to identify the first column with date time values. In the query ':SUV sales', the utterance does not specify which aggregation type to apply. Therefore, the composer uses the default aggregation type, namely SUM, for calculating sales. After the AST of the AE is composed, the composer traverses the expression tree to convert the AST into a query string using piped syntax. The QueryString is then filled into the 3rd parameter of the AE custom function.

Next, the composer checks neighboring columns to determine potential context columns. Specifically, it checks from the first left cell of the NL formula input cell to the leftmost range boundary, to collect continuous analysis columns. For a column specified using an AE table formula, the composer also parses the Query String from the table formula to extract the analysis semantics, and to check whether it is a dimension group by column, or aggregation column or which filter predicate to apply. For a dimension group by column, the composer fills it into the second parameter of the AE function, to indicate that the analysis will be conducted using each dimension value as a row filter context. For example, for "sales" query in Table 2, its AE formula takes the "Brand" column as context column and the query specified in Query String will perform scoping for each value in the context column.

Finally, when there is no neighboring column context, the composer will scan for vertically neighboring analysis tables above the current query, to check if the new query is a *follow up query* that refines a previous analysis, which we call the *context query*. If there is an analysis table nearby above, the distance to the table is less than a threshold (currently, 3 rows), and vertically placed in the same columns, the composer treats the new query as a candidate follow up query. Using a set of heuristic scoring rules it further checks the compatibility of the follow up query intent with previous ones. For example, ':sort by sales' is not compatible with ':rating by product' because there is no sales column specified. Compatible context queries with underspecified arguments are scored higher, and when the score exceeds a threshold, the composer treats the new query as a follow up query. It then combines the existing context query with the new query by chaining the analysis expressions together. For example, in Figure 2 (f), the follow up query ':top 2' is interpreted by chaining *sort* and *take* clauses at the end of the previous ':total sales by category for BrandB' query, generating the following AE formula:

=AE(A1:D100, , "table
| where [Brand] = 'BrandB'
| summarize sum([Sales]) by [Category]
| sort by sum([Sales]) desc
| take 2")

The formula thus composed is placed in the cell below the NL utterance. Then Excel's regular formula computation engages to evaluate the formula. The result of the formula is a dynamic array that is rendered on the grid, along with preset formatting (e.g., for column labels) that GridBook automatically applies for presentability.

[3]https://web.kusto.windows.net/

Table 3: Participant demographics.

Participant	Gender	Age	Functional area of job	Professional spreadsheet experience (yrs.)	Python experience (yrs.)
Study A: GridBook vs. Excel					
A01	Man	26-40	CS/IT	5	-
A02	Man	26-40	Accounts/finance	10	-
A03	Woman	18-25	Science/Engg. (non CS/IT)	3	-
A04	Man	18-25	Accounts/finance	3	-
A05	Man	26-40	Accounts/finance	7	-
A06	Man	>60	Accounts/finance	30	-
A07	Man	41-60	CS / IT	26	-
A08	Man	26-40	Operations/sales/marketing	6	-
A09	Man	26-40	CS / IT	4	-
A10	Man	41-60	Accounts/finance	34	-
Study B: GridBook vs. Jupyter					
B01	Man	18-25	CS / IT	2	5-6
B02	Man	26-40	CS / IT	5	3
B03	Man	18-25	CS / IT	1	5
B04	Man	26-40	CS / IT	3	3.5
B05	Man	18-25	CS / IT	1	3
B06	Man	26-40	CS / IT	4	4-5
B07	Woman	26-40	CS / IT	5	2+
B08	Man	26-40	CS / IT	3	6
B09	Man	26-40	CS / IT	0.5	5
B10	Man	26-40	Science/Engg. (non CS/IT)	12	10

3.7 Limitations

This paper presents an initial implementation of GridBook, and it contains several design and NL interpretation limitations that future versions need to address.

First, GridBook is limited in the format and nature of data it supports. It only works on tabular data, specifically a single explicit data table. Recent research [94] has suggested methods of dealing with multiple data tables, as well as automatic table detection work (e.g., TableSense [15]) to recognize implicit data tables (e.g., contiguous group of tabular cells). Continued research should improve the ability of handling multiple tables and implicit data tables in future versions of GridBook.

Second, even though GridBook supports and can interpret over 70 analysis terms, our set of Analysis Role terms is not exhaustive. For example, GridBook does not support arithmetic operators. However, our interpreter is extensible to support new analysis role terms. We also plan to support Analysis Role terms related to advanced analysis semantics such as outlier detection, clustering, and correlation in the future.

Third, our training data is limited. Specifically, we have a sparsity of natural language utterances describing complex queries. As a result, we do not support parsing nested queries with more than two layers of nesting. We plan to collect a larger training dataset on an ongoing basis to improve the capability of our model in solving more complex natural language queries. Moreover, currently we only support English as input. We wish to support multiple languages in the future versions, to facilitate users with different backgrounds.

Finally, as our user studies revealed, the user interface for GridBook suffers several limitations. Specifically, future research should consider ways to make GridBook interpretable to the user, and the NL formulas composable with other spreadsheet features (e.g., sorting). We discuss these design limitations in Section 8.

4 EVALUATION: LAB STUDIES

To evaluate GridBook, we compared its effectiveness and usefulness for data analysis, versus both conventional spreadsheets (namely, Microsoft Excel) and computational notebooks (namely, Jupyter notebooks). Specifically, we conducted two remote comparative lab studies: 1) Study A for GridBook versus Microsoft Excel, and 2) Study B for GridBook versus Jupyter notebooks.

4.1 Participants

We recruited a total of 20 participants (10 for Study A, 10 for Study B), via email, using purposive sampling. Our expectation was that even experienced users of spreadsheets and computational notebooks will be able to accomplish tasks faster with GridBook, with minimal training. Therefore, we recruited participants who reported themselves as very familiar with spreadsheets, performed data analysis at work using spreadsheets, and were familiar with English language. Additionally, the 10 Study B participants also identified themselves as performing data analysis using Jupyter notebooks. Table 3 summarizes participant demographics.

Table 4: Tasks. Participants performed two sets of three data analysis tasks, one with GridBook and one with Excel/Jupyter.

	Task Set 1: Crimes against women	Task Set 2: Juvenile crimes
1.	How many cases of crimes against women have been reported in each state in India?	How many juveniles have been apprehended under each crime category?
2.	Which are the three most and least safe states in India?	What crimes are juveniles the most and least apprehended for?
3.	In the three worst states, how promptly do police investigate crimes against women?	For the three most common crimes, what fraction of apprehended juveniles are boys and girls?

4.2 Tasks

We did not find prior studies with real-world spreadsheet data analysis tasks. Therefore, we obtained the tasks from a recent study involving the use of spreadsheets in the real world [75]. One of the participants in that previous study had conducted data analysis on a public data set [32], to support an article. We used the same data set, and similar tasks and scenarios for our study.

Participants in our study were given the scenario of a journalist Sarah, who wants to write two articles on crimes in India. Sarah requires answers to the questions in Table 4, for her two articles. Participants were asked to help Sarah with the data analysis. They were provided with the right spreadsheets containing the relevant data, instead of the entire dataset.

To ensure the authenticity of our tasks, we asked participants in their post-task retrospective interviews whether they performed such tasks as part of their job; 17/20 participants replied in the affirmative.

4.3 Protocol

We conducted a remote lab study. Each study session involved a single participant and lasted 1-1.5 hours. At the beginning, participants signed an informed consent form and filled in a demographic questionnaire. After that, they received a short training about GridBook lasting 5-10 minutes using a sample data set different from the study data sets. The training involved a demonstration of the colon notation, progressive analysis column wise, and writing longer queries. The training covered only "happy paths" and did *not* cover edge cases, model limitations, the availability of regular Excel formulas, or debugging.

Participants were then introduced to the two task sets. Participants worked remotely on the researcher's computer set up with GridBook, Jupyter, and Excel. They were asked to perform one set of tasks as they would normally do, using the features, libraries, and resources of their choice (including the Internet), and the other set of tasks using GridBook. The assignment of task sets to control (Jupyter/Excel) or treatment (GridBook) were balanced. The order in which the treatments were assigned to participants (GridBook first vs. Excel/Jupyter first) was also balanced. However, the task orders were not balanced (Task 1 always came before Task 2); in retrospect, this was an oversight in our study design.

During the study, participants were encouraged to think aloud as they performed their task. At the end of each Task Set participants answered the NASA Task Load Index (NASA-TLX) survey [23]. At the end of both tasks, we conducted a semi-structured interview to elicit their experiences about their tasks, and the tools they used. Participants' audio, video, and screen activity during their tasks and the retrospective interviews were recorded. Their queries on GridBook were also logged. At the end of the study, participants were compensated via electronic vouchers worth USD 60 or equivalent.

4.4 Study limitations

Every study has limitations, and the following are the key limitations of our study.

First, we compared GridBook against the Microsoft Excel spreadsheet package and Jupyter notebooks. While these are two of the most popular tools, there are other spreadsheet tools (e.g., Google Sheets) and programming languages (e.g., R, MATLAB) that analysts use for data analysis. There are also other data analysis tools that implement a different paradigm altogether (e.g., SQL, Tableau). Our results may not generalize against such tools.

Second, we used a limited sample size, with only 10 participants in each study. Moreover, participants possessed prior experience in Excel and Jupyter notebooks, and were new only to GridBook. As a result, our study does not offer a comparison of GridBook's NL formulas against new users of spreadsheet formulas or pivot tables, or Python. Future studies are required to compare learnability of NL programming in GridBook against learnability of programming using spreadsheets or traditional programming languages.

Third, ours is a lab study, and participants only performed a small set of tasks. We tried to keep our tasks grounded in real world, by designing our tasks based on real-world user tasks observed in prior studies. Given that our participants were end-user programmers, we built a realistic scenario of helping a journalist with data analysis, based on real-world observations. We also explicitly confirmed with participants whether our tasks were representative of their data analysis tasks; 17/20 participants agreed. However, a wide variety of data analysis tasks are possible (e.g., correlations, regression) and it is possible that our results might not apply to other kinds of data analysis tasks that data analysts might perform. Our results might also not generalize to more complex forms of data analysis that are possible with GridBook.

Finally, GridBook itself is a limited system. It currently only supports English language NL queries and that is one of the reasons we recruited participants with English familiarity. There are also many tasks outside its scope (e.g., hypotheses testing, regression analysis) that data analysts routinely perform. Future versions of

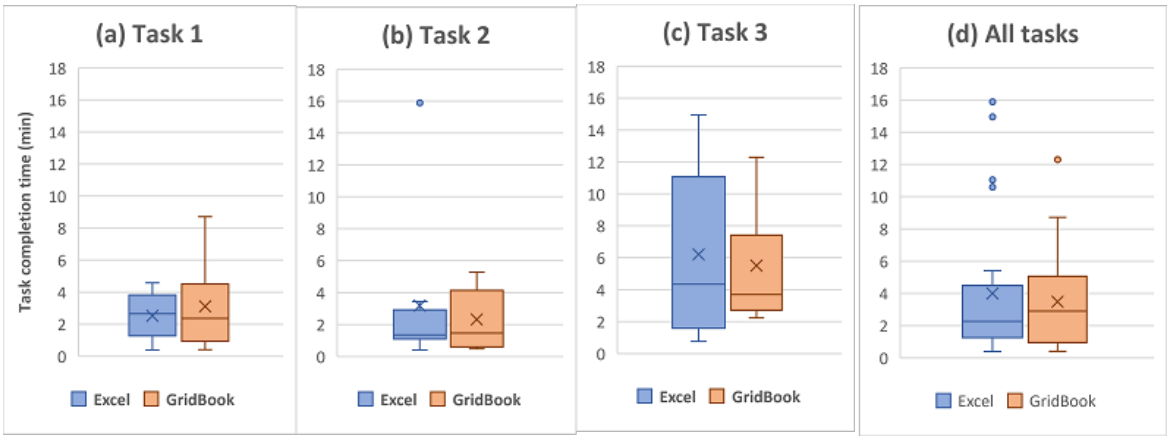

Figure 4: Task completion times: Excel vs. GridBook. We did not find significant differences in the task completion times between GridBook and Excel.

GridBook supporting a wider range of functions should also evaluate its suitability for such tasks and train its models for non-English languages.

5 QUANTITATIVE EVALUATION

We evaluated GridBook via the following hypotheses:

- H1: Data analysis with GridBook is faster and less cognitively intensive than with conventional spreadsheets, even for users experienced with spreadsheets.
- H2: Data analysis with GridBook is faster and less cognitively intensive than with computational notebooks, even for users experienced with computational notebooks.

Study A compared GridBook against conventional spreadsheets (Microsoft Excel) and Study B compared GridBook with computational notebooks (Jupyter). These studies are evidence towards hypotheses H1 and H2.

5.1 Study A: GridBook vs. spreadsheets

We compared GridBook and spreadsheets (and computational notebooks) using three measures: task completion rates, task completion times and cognitive load.

Task completion rate: Out of the 30 total tasks assigned per condition (10 participants x 3 tasks per condition = 30 tasks), participants completed 29/30 tasks using Excel and 26/30 tasks using GridBook. Fisher's exact test indicated no significant differences in the task completion rate between GridBook and Excel (p=0.3533).

Task completion times: Figure 4 compares the time participants took to complete tasks using GridBook and Excel. Although median task times were lower for Excel (median=1.95 minutes) than for GridBook (median=2.9 minutes), the differences were not statistically significant (Wilcoxon signed rank test taken across all tasks; Z=-0.0127, p=0.51). In other words, task completion with GridBook was not any slower than with spreadsheets, even though participants had prior experience with spreadsheets, and were encountering GridBook for the first time. For individual tasks also, task completion times were not significantly different between GridBook and Excel (Wilcoxon signed rank test for Task 1: Z=0.153, p=0.8785; Task 2: Z=0.53311, p=0.594; Task 3: Z=0, p=1).

Cognitive load: To measure cognitive load, we administered the NASA Task Load Index (TLX) test at the end of each treatment. Figure 5 compares the test scores from using Excel and GridBook. In each graph, lower scores are better. As the figure shows, participants generally found that GridBook imposed a higher cognitive load than conventional Excel; however, these differences were not significant, both for overall scores as well as for the individual parameters. (Wilcoxon signed rank test; p-value > 0.1 in all cases).

> **H1:** Data analysis with GridBook is faster and less cognitively intensive than with conventional spreadsheets, even for experienced spreadsheet users.
>
> **Result:** We did not find significant differences in task completion rates, task completion times, or cognitive load between Excel and GridBook, in experienced spreadsheet users.

One reason data analysis with Excel was somewhat quicker and less cognitively intensive than in GridBook is the fact that participants were very familiar with Excel. Participants had a median experience of 6.5 years using Excel. As a result, they were able to complete their tasks using a combination of tools (e.g., pivot tables, power query, formulas) without spending time figuring out how those tools worked, or on debugging. In contrast, participants were using GridBook for the first time, and needed time and cognitive effort to understand the way it inferred natural queries, and then to frame their queries. We found that 7 out of 10 participants explicitly mentioned their familiarity with Excel, when comparing their experiences using GridBook and Excel; for example:

> B04: "Currently, I would use pivot tables but just because I know it. I guess that the first times I used pivot tables, I was a little frustrated as well. With a few more tries with GridBook, it could be easier than pivot tables?"

Similarly,

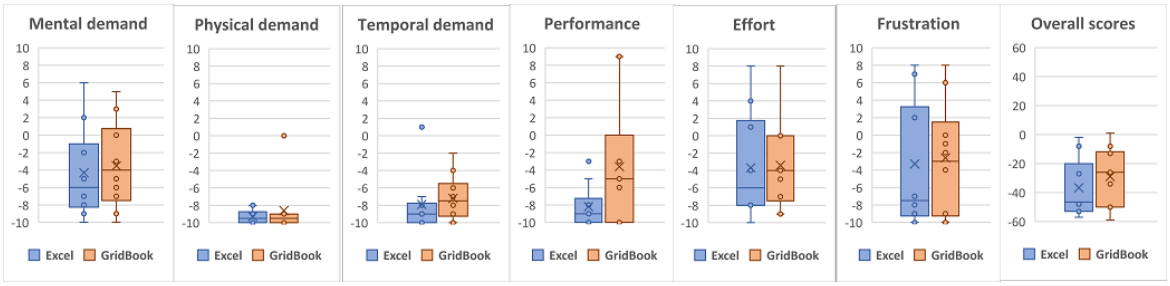

Figure 5: NASA-TLX Scores: Excel vs. GridBook. (Lower is better). Participants, namely experienced spreadsheet users, generally found data analysis using the unfamiliar GridBook more cognitively intensive than the unfamiliar spreadsheets. These differences were not significant.

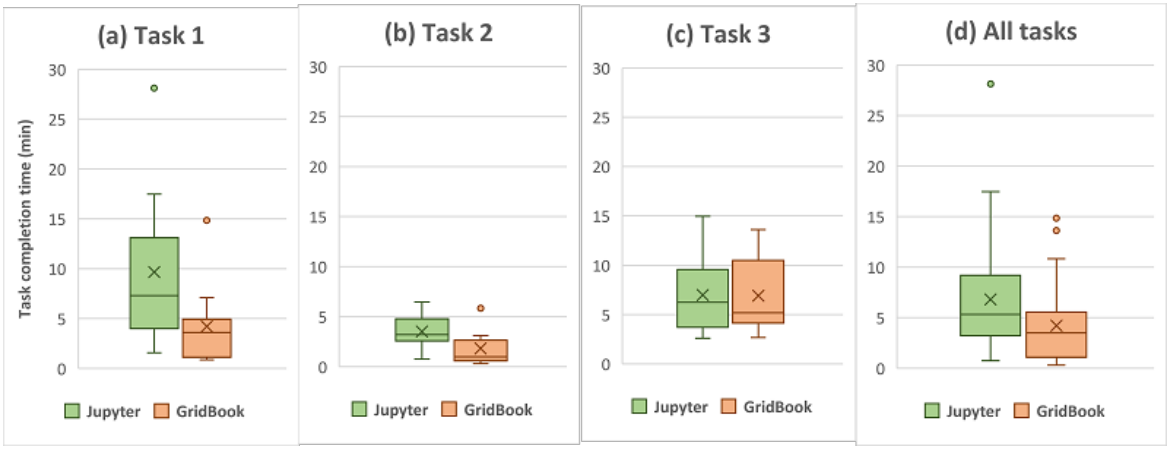

Figure 6: Task completion times: Jupyter vs. GridBook. We did not find significant differences in the task completion times between GridBook and Excel.

> A06: "I've been using pivot tables since about 1995 . . . I could just use one of these [pivot tables] and get the thing I want very quickly and that is faster and easier for *me* because of my experience. . . as opposed to this [GridBook] which is learning something brand new. I think had I been using GridBook for a year or more and I got used to its nuances, I might find that as easy to carry out the task."

5.2 Study B: GridBook vs. computational notebooks

Task completion rate: In Study B, participants completed the same number of tasks (27/30) using GridBook and Jupyter notebooks.

Task completion times: Participants were significantly quicker in completing their tasks using GridBook (median = 3.5minutes), than with Jupyter notebooks (median = 5.32 minutes) (Wilcoxon signed rank test taken across all tasks; Z=-2.35, p=0.019). As Figure 6 shows, this gain primarily came from Tasks 1 and 2. For Task 1, we found significant differences in median completion times (Wilcoxon signed rank test; Task 1: Z=-2.19, p=0.028), while for Task 2, the differences were significant with a 90% significance level (Wilcoxon signed rank test; Task 2: Z=-1.72, p=0.086). There were no significant differences in Task 3 (Wilcoxon signed rank test; Z=-0.14, p=0.889).

Notice in Figure 6 the large and significant time differences between Jupyter and GridBook for completing Task 1. This is because, in Jupyter notebooks, participants had to write code to get started; they had to write code to import necessary libraries, read data from a CSV, and then load that data into the notebook, before they could write the first line of code specific to that task. This provides evidence that GridBook meets our Design Goal #1 (minimal setup).

Moreover, along the way of analyzing data to complete their tasks, participants also had to go to the Internet for API help (e.g., "how to read a csv in pandas") and to debug their code (e.g., incorrect file paths), effectively increasing their task times. This is consistent with the findings of [11] on the pain points in using computational notebooks. Recall also that these significant improvements are even though participants had prior experience with data analysis with Jupyter, but they had none with GridBook.

Cognitive load: In general, data analysis tasks with GridBook imposed a lower cognitive load on participants than did Jupyter notebooks (Figure 7). However, none of these differences were significant (Wilcoxon signed rank test; p>0.1 for overall scores as well as the score for individual components).

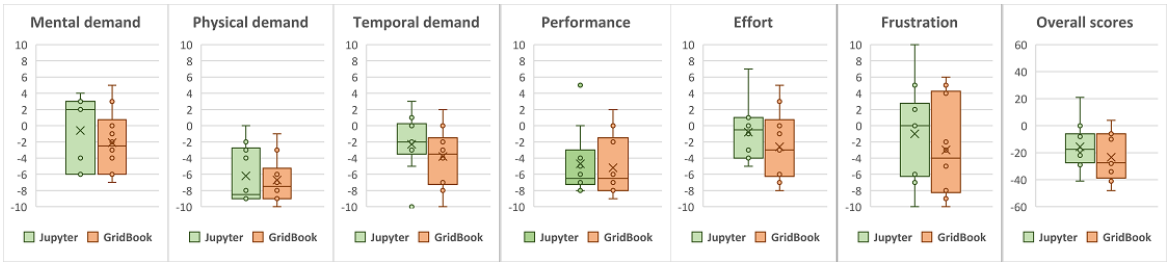

Figure 7: NASA-TLX Scores: Excel vs. GridBook. (Lower is better). Participants, namely experienced spreadsheet users, generally found data analysis using the unfamiliar GridBook more cognitively intensive than the unfamiliar GridBook. These differences were not significant.

> **H2:** Data analysis with GridBook is faster and less cognitively intensive than with computational notebooks.
>
> **Result**: Participants took significantly lesser time to complete their data analysis with GridBook than with Jupyter notebooks. We found no significant differences in the cognitive load from the use of both these tools.

Retrospective interviews with participants found two key contributors to the higher cognitive load using Jupyter notebooks. First, 9/10 participants (except B01) had to seek help from external resources, seeking out the Internet several times, to find the right APIs to accomplish their tasks (e.g., "group by in pandas"). Participants reported that such need increased their mental demand, effort and frustration with Jupyter notebooks. For example, B04 said:

> B04: "This task was higher mentally required because it took more time, and it required google search, and all this."

Second, participants found the need to debug minor issues as adding to the cognitive demands of the tasks. For example, GridBook was tolerant to errors such as misspelling, pluralization and capitalization, but Jupyter led to errors in such cases. This placed additional demands on participants' cognitive effort to avoid such errors or debug them when things went wrong with Jupyter notebooks. For example, B01 said:

> B01: "With python . . . it needs me to be very careful, about capitalization, about syntax. One little thing will make the whole error."

6 QUALITATIVE EVALUATION

6.1 Data analysis in GridBook was more natural: closer to the way people think about analysis.

The key goal of GridBook is to make data analysis more *natural*, in Pane and Myers' sense of being "closer to the way people think about their task" [57]. In GridBook, participants were able to type in the colon sign, followed by a declarative query in a natural language, and view the results on the grid, without having to spend as much time needed for learning a new programming language and its APIs. Across both studies, 10/20 participants explicitly favored this "naturalness", e.g.,

> B01: "With GridBook, I can do like top 3 and bottom 3, that's nice. It's what is in my head. In Python though, I have to assign to a variable and then get the top3 from that variable."

However, some participants, especially the ones who had already mastered their tools, found more familiar, procedural ways of doing things (using pivot tables, or formulas) more natural than framing declarative queries for GridBook. For example,

> A03: "I like the pivot tables seem more logical to me in terms of how to think through things, rather than translate my thoughts to words and then figure out how that would be understood by GridBook."

6.2 GridBook is easy to learn.

GridBook is a programming environment, and, as we shall discuss in Section 7, participants faced some challenges learning and using it for the first time. However, participants generally found GridBook easy to learn (Design goal #2: Minimal learning). Participants (8/20) explicitly mentioned this as a merit, and discussed its potential usefulness for beginners, or users with non-computing backgrounds.

> B06: "GridBook will be useful for anyone not from CS background. I have a friend in psychology, she cannot do any EDA [exploratory data analysis]. I help her out with R and python. . . people from not CS background. . . psychology or biology. . . it would help them a lot."

Three participants (A01, B05, A08) also considered this learnability as an advantage for GridBook's potential as a communication tool in their work, such as for quick analysis in meetings, communicating with business analysts For example,

> A08: "When I am sitting with the team. . . "hold a minute we can work this out, type it in a minute, da-da-da-da-da. . .."

6.3 GridBook's on-grid queries improved as well as deterred spreadsheet presentability.

GridBook takes inspiration from literate programming and computational notebooks, where code, documentation and the results of

the analysis are all interleaved. GridBook takes spreadsheets one step towards this direction. The data is available in the spreadsheet for the user to see and manipulate. Declarative queries used to perform computations over the data are present and visible in the grid, followed by the result of the computation (Design goal #4: Literateness).

Participants had diverse preferences about the presence of the natural language formulas in the grid. Some participants (8/20) thought that the presence of queries on the grid, alongside the results, allowed for better readability, as well as offered transparency on what the results below represented and how they were created.

> A10: "In GridBook, you have that query. When the auditor asks, "how did you get this?", you have that query. Hopefully that is not there for just debugging, and that will stay in the results."

They also preferred that the original data, NL code and results of computations were always visible on the Grid, unlike with computational notebooks where a person must explicitly display the results in an output cell. For example, B07 said:

> B07: "Nice that it [analysis and results] was right next to the data ... Pandas doesn't have it ... I can keep the data frame and go and look. . .."

However, other participants (3/20) did not prefer the formula appearing on the grid, as it interfered with the presentation of the results, and was not consistent with existing formulas. For example, A05 described why he did not want the NL formulas always visible on the grid:

> A05: "A lot of our thing is about visually making things presentable. That would probably be the biggest thing. . . . Seeing this is helpful for review, but not for someone above me who gets the summarized points."

One participant (A10) wanted the best of both worlds and wanted to be able to format the natural language formulas to make the spreadsheet presentable (e.g., hide colon, change styles for the NL query globally):

> A10: "GridBook comments not enough for manager's manager . . . I'd make it proper case . . . who made it grey?, why is it explanatory text style? ... The fact that it is a style, and it can be edited, if I want to change it globally, I can go and modify this style . . . I want it in title style with the colon hidden. . . . How do you even hide the colon?"

6.4 GridBook's NL formulas are one more tool in the end-user programmer's toolkit.

A distinguishing characteristic of data analyst end-user programmers is that they are task oriented. They pick the tools suitable to achieve their tasks, and if they get the job done, it doesn't necessarily matter which tools they pick and combine.

One of our decisions with GridBook is that natural language formulas exist *beside* existing spreadsheet formulas and augment them, rather than replace them. As a result, when users are unable to make something work with the somewhat limited natural language capabilities, they have the standard formulas to fall back on. All 20 participants using GridBook took advantage of this, and combined standard formulas, with natural language queries to accomplish their tasks. Specifically, GridBook did not recognize arithmetic operations in its language, and when participants could not make it work in the first one or two tries, they quickly fell back to regular formulas for their computation. One participant said:

> A10: The analogy I'm going to give you is I am used to getting across the country on a bicycle and now you have given me a Lamborghini and I start driving across the country from Seattle to NYC and you get me like 98% of the way, and I am going to hop on the bicycle and cross the bridge to get to NYC, I don't mind at all. Especially, it is a one-time or ad hoc analysis, this is enough. If it is something I am going to do regularly, I am going to figure out a way to do it with GridBook."

Similarly, another participant said:

> B08: "It was fun seeing how I could piece together queries, because I felt I was expanding my own toolbox to get the data."

However, GridBook's NL data analysis capabilities were not always composable with other spreadsheet capabilities, and we will discuss these limitations in the next section.

7 ANALYSIS OF FAILURE CASES

Although GridBook uses familiar natural language for programming, GridBook users in our study encountered various roadblocks and failures while programming in GridBook, as with any other programming language and environment. Sometimes, participants were able to recover from these failures, but during other times they had to backtrack and adopt a different strategy to solve their problem or abandon the task entirely.

7.1 The vocabulary problem: using an unfamiliar word for a concept

The first cause of failure for participants was what is referred to as the "vocabulary problem" in human-system communication [18]: when there are multiple words to refer to the same concept, and the system recognizes only one of them, leaving users with errors when they use other correct, equivalent terms.

For example, in our study, A06 was provided data shown in Figure 8(b) and asked to "count how many juveniles have been apprehended under each crime category". A simple GridBook solution would be to type in ':Grand Total for each crime', or to progressively fetch ':Crime' followed by ':Grand Total'. However, A06 began by the utterance ':count juveniles apprehended by category', a phrase he adopted straight from the task description. The query resulted in an error because GridBook was unable to interpret the words "juvenile" and "category" in the context of the given spreadsheet. Figure 9 shows the sequence of queries A06 tried, before he could finally come up with the right query to accomplish his task.

A variant of the vocabulary problem presented when the NL utterance was underspecified and embedded more context than GridBook was able to interpret. For example, to fetch the total number of boys apprehended in juvenile crimes (Figure 8(a)), B09

	A	B	C	D	E	F	G	H
1	Area	Year	Group_ Name	Cases_Reported_this _year	Cases_Pending_Investigation_ from_previous_yr	Cases_Investigated	Cases_dropped_ withdrawn_etc	Cases_Pending_Investigation_ at_Yr_End
2	Andaman & Nicol	2001	Rape	3	3	5	0	1
3	Andhra Pradesh	2001	Rape	871	390	791	77	393

(a) Task set 1: Crimes against women

	A	B	C	D	E	F	G	H	I	J	K	L
1	STATE/UT	Year	CRIME	Boys 7-12 Years	Girls 7-12 Years	Boys 12-16 Years	Girls 12-16 Years	Boys 16-18 Years	Girls 16-18 Years	Total for boys all Age Groups	Total for girls all Age Groups	Grand total
2	Andhra Pra	2001	Murder	3	0	7	0	5	0	15	0	15
3	Andhra Pra	2001	Rape	2	0	15	0	2	1	19	1	20

(b) Task set 2: Juvenile-crimes

Figure 8: Data sets used in the study.

1. A06 typed *':count juveniles apprehended by category'*.
2. GridBook returned an error (Excel error #NA), indicating no result.
3. A06: "*Why am I getting that... (scrolls through data)... let's try something different*".
4. A06 revised the query to *':count crimes by category'*.
5. GridBook returns a single value result with header "distinct count of CRIME" (Word category in query was not processed by GridBook)
6. A06 revised the query to *':Count crimes b crime'* (b is presumably a typo for by).
7. GridBook returned the number of rows in the data for each crime. (The value was 350 for all rows).
8. <recovers from a domain misunderstanding on what juveniles mean>
9. A06 to moderator: *"... the task is done then, is that correct?"*
10. Moderator: *"you might want to check the answer*".
11. <Noticed the bug. Stuck for several minutes and moderator offered hint to "look at the data">
12. A06 *"(scroll and think) ... oh, I can try other columns".*
13. A06 revised the query to *':count state by crime'*.
14. A06: *"Still getting the same result."*
15. A06 revised the query to *':count Grand Total by crime'*.
16. GridBook interpreted A06's intent to likely be the sum of grand total values, and returned the desired result.
17. A06: *"Looks better, that is the answer to that question".*

Figure 9: Sequence of events in A06's task using GridBook.

used queries such as ':sum of boys' and ':sum of boys 7-12 and 16-18' (Figure 10). While such utterances communicated B09's intent well, GridBook had limitations interpreting such queries.

Finally, we also observed several instances where participants used words for concepts that were ambiguous in the context of the given data. For example, Figure 8(a) contains multiple columns starting with the word "cases". When participants wrote queries such as ':total cases' , they got back unexpected results, and they had to recover from them. Figure 10 shows the sequence of actions B09 took to accomplish the task.

Participants almost always recovered from the vocabulary problem for data, and eventually learnt to use specific column names where needed (e.g., if there was ambiguity in column names). However, there are opportunities for GridBook's design to make specifying columns easier for the users, to help users disambiguate underspecified entities, and to provide feedback on words that it is unable to interpret.

7.2 Using an incorrect word for a concept

Another common cause of failure was the use of words that held a different meaning than what the participant had intended. For example, participants used the word "count" to refer to the sum of the values, when it is commonly also used to count the number of values in the data. Take the case of A06, who needed to count the number of juveniles crimes in each state (Figure 8(b)). As the sequence of actions in Figure 9, A06 used queries such as ':count crimes by category', ':count crimes b crime', ':count state by crime' and ':count Grand Total by crime', using the word "count" to refer the sum of number of crimes.

When GridBook interpreted these queries, it exhibited inconsistent behavior. For utterances such as ':count state by crime', where

1. B09 typed *':crime'* to fetch list of crime categories.
2. B09 typed *':sum of boys'* in the next cell to get sum of boys in all age categories.
3. GridBook returned the data for 'total boys 7-12 years', since that was the first match for the word 'boys'.
4. B09 edited the query to *':sum of boys 7-12 and 16-18'*.
5. GridBook did not understand the notion of sum, nor the commonality in the column names 'boys' and so returned ':sum of boys 7-12'.
6. B09 revised the query to *':sum of boys and girls'*, and failed.
7. B09 went back to the data and found the data contained totals for boys and girls.
8. B09 revised the query to *':sum totals'* (presumably to add columns with the word total).
9. GridBook returned no results.
10. B09 revised the query *':sum totals for boys all age groups and total girls for all age groups'*.
11. GridBook returned the data for both columns, instead of their sum.
12. B09 wrote two queries *':sum totals for boys all age groups'* and *':sum totals for girls all age groups'* to fetch data.
13. He used formulas to compute total in the adjacent column, named the column Task 1.
14. ...
15. Typed *':sort Task 1'* to sort the entire data, to fetch top 3 crimes.
16. ...
17. Used Excel's sort feature to sort the data and failed.
18. Used Excel's top N filter to fetch top 3, failed because it did not work on arrays.
19. Used Excel's conditional formatting to highlight top 3, did not find the correct menu.
20. <ran out of time>

Figure 10: Sequence of events in B09's task using GridBook.

the columns 'crimes' and 'state' were textual, GridBook returned the count of the states for each crime. However, for the utterance ':count Grand Total by crime', GridBook returned the sum of the values in Grand Total column for each state.

On the one hand, the power of deep learning helped GridBook to interpret the user's intent accurately, even though it was mis-stated, arguably since users often wanted to sum numerical values than count, or because users often used the term count to refer to sum. However, on the flipside, this could leave users confused in other situations. For example, in this case, A06 might not completely understand that the 'count' operation meant a count of instances, or he might hit roadblocks in situations where he intends to perform a counting operation rather than a summing one.

This fundamental limitation of machine-learning technologies raises questions about the consequences of learning from real-world data, especially where end-user programmers with little programming experience have a higher tendency to mis-specify their intents. It also demands for research into the tradeoffs between getting good quality training examples and tradeoffs against learning common mistakes that happen in the real world.

An alternate interpretation of this situation is to question the common terminology that programming APIs use. For example, if programmers often confuse count for sum (and vice versa), then should we revisit the vocabulary used in APIs? Would it be more appropriate to use COUNT() and SUM() for sum, and use alternative names such as NUMBER_OF_ITEMS for cardinality? Questions such as these can only be answered with further research into the usability of programming APIs.

7.3 Difficulties building on top of previous analyses

Several participants faced challenges when they must build on top of the previous analysis. Specifically, they often wrote NL statements that were incompatible with the previous analysis.

An illustration of this is A05's analysis sequence (Figure 11). For Task 1, A05 was asked to fetch the number of crimes reported against women in each state. For this, he typed ':areas' and then ':cases_reported_this_year' in adjacent cells to complete the task.

His Task 2 was to fetch the most and least safest states, based on the total number of crimes reported. Like many other participants, A05 adopted the strategy of sorting the results from the previous analysis and then picking the top 3 and the bottom 3 items. For this, he used the formulas ':areas' followed by queries such as ':cases_reported_this_year sorted highest to lowest' and ':Cases_reported_this_year sorted by top 3' in the adjacent cell. However, the queries were incorrect. The former query fetched all columns in the original table, for the row with the highest value in the 'cases_reported_this_year' column for each state. The latter query resulted fetching the first three rows from the original table sorted by 'cases_reported_this_year' without actually matching the rows to the areas on the left.

Here, users lack the information needed to understand what was going on under the hood, and why the results were not what participants expected. As a result, users not only struggled to accomplish their tasks, they also spent a lot of time debugging, with almost no debugging support. These failures have implications for the design of interfaces that explain what is going on, and to provide information needed for debugging.

1. A05 typed *':area'* followed by *':cases_reported_this_year'* to complete Task 1
2. A05 edited the second query to *':cases_reported_this_year sorted by highest to lowest'.*
3. GridBook fetched all columns from the original data, picking the row with highest cases_reported_this_year for each area.
4. A05 backtracked and tried to use regular Excel sorting feature to sort the data. It failed because sorting multiple arrays would lead to inconsistencies in the data structure.
5. A05 writes a new query *':Cases_reported_this_year sorted by top 3'*.
6. The query fetches top 3 records with highest *':Cases_reported_this_year'* in the original table.
7. When asked, he said... "wondering about dictations that this functionality use... like a different way of phrasing whatever I already entered in here ... I am thinking of search in google, if you don't get what you want, try some other version of it"
8. ...
9. Adopts a new strategy *':sum of cases_reported_this_year by area'*.
10. Juxtaposes with *':sum of cases_reported_this_year by area'*, and gets a data with two columns and area repeats.
11. A05: "...I was trying to build on top of this, but it re-put the data. May be it was my mistake to put the area again..".
12. A05 revised the query to remove 'by area' to fetch the data he intended.
13. ...
14. <Abandoned task 2 to fetch top and bottom 3 states, to be done manually>
15. ...

Figure 11: Sequence of events in A05's task using GridBook.

1. B02 typed *':cases_investigated / cases_reported_this_year'*
2. The data returned the data for the two columns, not the result of division.
3. B02 revised the query to *':cases_investigated divided by cases_reported_this_year'*
4. GridBook provided the same result, not the result of the division.
5. B02 revised the query to *':cases_investigated as a percentage of cases_reported_this_year'*.
6. GridBook provided the same result, not the result of the division.
7. B02 asks the moderator why they were not working, to be told arithmetic did not work.
8. B02 uses regular formulas to complete the task.

Figure 12: Sequence of events for B02 (Task 3) using GridBook.

7.4 Understanding the bounds of the NL model

A situation most participants faced was in trying to perform arithmetic calculations using GridBook. Participants used various notations such as 'divided by', 'sum of', 'difference of', 'as a percentage of' and operators such as + and / (for division) to mention arithmetic calculations. For example, when asked to calculate the percentage of cases investigated, P06 tried the sequence of queries in Figure 12, to compute percentages. He eventually asked the moderator for help, to learn that GridBook did not support computer arithmetic.

Here, GridBook did not communicate that it did not understand words such as 'divided by' or '/' or 'as a percentage of'. But, more broadly, it did not communicate to the user that it did not understand arithmetic operations. They had to abandon or ask the moderator why something did not work before they were able to make progress using other strategies, namely regular formulas. This raises two questions that need further research:

- what affordances should GridBook's design provide to communicate the bounds of the natural language API / what the tool does not understand, to the user?
- what if users do not possess the skills to perform their calculations using formulas?

7.5 Composition with existing features

One of our design choices with GridBook is that it would combine seamlessly with the results of existing formulas. All 20/20 participants took advantage of this capability and reacted positively to it. However, we did not foresee all the ways in which participants would combine existing Excel features with GridBook's affordances.

For example, when B09 was asked to fetch the total number of juveniles apprehended under each crime category (data shown in Figure 8(B)), he used GridBook to fetch the crime category names, and the data for boys and girls, separately for each crime category. He then used regular formulas to add up the totals. GridBook supported such composition where users could build formulas based on results of GridBook.

However, for task 2, when B09 was asked to pick the top 3 most common juvenile crime categories, he was unable to build on top of the results that he had arrived at using regular Excel formulas. For example, he used the query ':sort Task 1' where Task 1 is the column name containing Task 1 results. Even though the column Task 1 was right next to GridBook-generated table results, GridBook did not recognize "Task 1" because the results were computed using regular formulas.

B09 also attempted to accomplish the sorting using Excel's built-in sorting capabilities. Again, the choice of arrays as the data structures to hold GridBook's results was incompatible with sorting and led to errors.

7.6 Overconfidence in analysis

Prior research in end-user programming has recognized overconfidence among spreadsheet programmers in the accuracy of their spreadsheets; it is considered one of the causes for the high prevalence of spreadsheet errors [60]. In our study too, we found participants mistaking incorrect results for correct ones, because structure of the results resembled their desired output.

For example, A06 assumed that he had computed the total number of juveniles for each crime, when in fact, he had fetched the number of rows in the data for each crime. The structure of the program's output resembled his desired output (e.g., the result was a table with crime category and numbers next to each). Nevertheless the participant overlooked the fact that the numbers were the same in all rows. In P06's case, the moderator had offered a hint before the participant was able to notice the bug and fix it (Figure 9, lines 10-12).

While some participants mentioned that, in real world, they would check the results of the calculation in GridBook to ensure its accuracy, it is possible that the use of familiar language, or the perceived capabilities of the system, might lead to further overconfidence among users. Future research should consider the nature of trust users have on NL programming systems such as GridBook and OpenAI's Codex and provide the information necessary for the user to ensure the accuracy of NL programs.

8 GRIDBOOK LIMITATIONS

Three themes in GridBook's limitations go beyond the specific instances of failure cases described in the previous section.

8.1 Interpretability

In GridBook, when a user types in a user intent, there are several possible outcomes: 1) expected result, 2) unexpected result, 3) no result, or 4) error. In cases of unexpected results, no results and errors, a user needs to understand how the intelligence apparatus has interpreted their utterance, and how they can fix it to get the result they need. Even in the case of expected result, the user must still make sure that the correct result comes from a correct interpretation of their intent, and not a misinterpretation leading to the correct result by chance (logical error).

However, the current version of GridBook offered no support for such reasoning, that is central to forming the correct mental models needed to learn and use the system effectively, and to recover from errors and debug their programs. In essence, users had no other affordances for reasoning about how GridBook interpreted their NL formula the way it did, and why. This was in fact one key contributor towards the task completion times in GridBook, as well as the frustration component in the NASA-TLX test scores. All 20/20 participants encountered interpretability issues with GridBook.

Specifically, participants were seeking the following classes of information:

- which parts of the NL formula GridBook recognized and accounted for, and which ones it ignored, or did not know how to interpret (e.g., mathematical symbols, cell references),
- which data were linked to names in the formula (e.g., "cases" was linked to cases_investigated instead of cases_reported),
- what it inferred missing clauses as (e.g., what context to run a query on, or what column to sort by when none is specified), and
- where the bounds of the system lie (e.g., cannot do arithmetic, cannot read non-table data).

By way of solving this interpretability problem, two participants (A07, B10) wished that GridBook explained back to them its interpretation of users' inputs. For example,

> A07: "View this as a new language, with syntax and keywords. . . my concern with some keywords is that someone can have a column called Total, and then you have to say total total . . . it would be good to color the text ... what if each keyword is a color. . . With regex it tells what group it is matching; I want to know what it is for NL and what it understood. . . . Tell me back what exactly you understood, make it a conversation."

Similarly,

> B10: "I have used Wolfram Alpha where you could ask "amount of rainfall per country" and the first thing it says is "I assumed you said all countries in the world and the amount of rainfall in each country" or something like that. . . . it tells you what it has assumed which is kinda important I think. "

Prior research on end-user debugging has also revealed the centrality of "why" questions (e.g., "why is this X and not Y", why is this all columns, and not just X) and hypotheses testing in debugging [39]. We found evidence of similar behaviors in GridBook also. See participant utterances in Figure 9 (lines 3, 12) for example.

8.2 Control

Compounding participants' interpretability challenges with GridBook were their difficulties specifying the details of the analysis and hence overriding some of the automatic inference. For example, GridBook always inferred the context in which participants wanted to run an NL query. Our original reasoning here was that users would conduct their analysis top to bottom and keep related analysis next to each other and to the data that it would run on (locality of reference). However, participants did not always do that, and when things went wrong, participants wanted ways to specify what context to run the query on, or override the default sorting order of a query, instead of what it had inferred. Similarly, in cases of commonly confusing vocabulary (e.g., between count and sum), users need a way to go back and provide an alternative phrasing for the task, when the intent is misinterpreted. Research is needed to make the affordances for such control obvious to users.

> B01: "I'm okay with the guessing, it's nice when it guesses. . . but I need a way to go back in and tell it, "Here's what I needed"".

8.3 Composability

One of our design choices with GridBook is that it would combine seamlessly with the results of existing formulas. All 20/20 participants took advantage of this capability and reacted positively to it. However, we did not foresee all the ways in which participants would combine existing Excel features with GridBook's affordances. Common instances of such combinations that GridBook did not support included:

- Composition with traditional formula semantics, e.g., ':add A and B' or ':cases_investigated / (cases_reported + cases_pending)'
- Composition with existing data manipulation tools (e.g., sort and filter results of NL formula using traditional sorting and filtering tools, instead of adding sort/filter clauses to NL formula).
- Tables composed manually were not recognized by GridBook (e.g., Add a column of formulas to GridBook-created results, and then expect the entire data to be picked up in the following GridBook query's context).

Future versions of GridBook must attempt to overcome these limitations, since opportunistic working styles to accomplish tasks *"by hook or by crook"* is a salient aspect in end-user programming, that GridBook aims to support. However, realizing such smooth composability presents opportunities for deep research in various areas such as AI (e.g., table detection), programming languages (e.g., combining NL and formula language) and in implementation technologies for feature-rich software such as Excel. We will discuss some of these opportunities in the Discussion section.

9 DISCUSSION: OPPORTUNITIES FOR FUTURE RESEARCH

Spreadsheets are the most widely used programming systems. Still, learning to program in spreadsheets, programming and debugging formulas, and comprehending them, are all challenging and error prone. This overall challenge with spreadsheets can be viewed from two perspectives – one of technology, and one of design.

From a technological perspective, machine learning capabilities have advanced, with finer language models, or table detection models (e.g., [12, 15]) already working underneath commercial spreadsheets and improving their experiences (e.g., Google Sheets' Explore feature, Excel's Analyze Data feature). It is only reasonable that we take advantage of intelligence to improve spreadsheets, to help users especially novices achieve more with data. GridBook does exactly this and explores the benefits of natural language to make spreadsheet programs easy to learn, write and understand.

From a design perspective, we take inspiration from Don Norman's arguments for narrowing the gulf of execution (how easily a user can tell a system what to do), and the gulf of evaluation (how easily a user can understand the state of the system and manipulate it) [55]. GridBook aims to narrow these gulfs, taking analysis of tabular data as close to the user as possible (rather than forcing the user to master an existing tool to come closer to it). By allowing users to simply type in natural language, GridBook narrows the gulf of execution. Since the user's natural language query is always available on the grid and manipulatable, the gulf of evaluation is also narrowed, in comparison to traditional programming languages or even traditional spreadsheets.

Our empirical evaluation of GridBook suggests learning to perform data analysis using GridBook might be easier and doing so might be quicker than with spreadsheets as well as computational notebooks (one of the most popular classes of data analysis tools). It reduces the need for users to master the semantics of a programming language, or deal with the cognitive burden imposed by syntax, or the need to seek external resources to find the right API methods to use. These results are encouraging, and hint at the suitability of NL and intelligence for spreadsheet programming.

However, our qualitative analysis also revealed opportunities for improvement for future versions of GridBook, or similar NL-based data analysis tools.

9.1 Interpretability of model

First, there is opportunity for further narrowing the gulf of execution by allowing users to fill in subclauses of NL queries (e.g., provide possible table and column names, or common clauses they might use). Second, to narrow the gulf of evaluation and to improve the interpretability of GridBook, there are opportunities to decorate different segments of the query with colors, to indicate which sections have been mapped to which areas of the spreadsheet or have been ignored by the parser and interpreter. Third, there is also a need for GridBook to make explicit how it infers missing parameters; for example, to reframe the NL formula ':sales for each product' with ': sales for each product in ascending order of product'. Finally, there are opportunities also to combine state-of-the-art spreadsheet affordances (e.g., click on column header to select a column as input, or drag fill to select a range on which to run a query) to leverage users' existing mental models of spreadsheet affordances. All these open a wide range of design challenges, given the constraints of the spreadsheet environment as primarily a two-dimensional grid, where computations happen by writing in the grid, or a formula bar (instead of a large open text environment, like a programming editor).

9.2 Debugging explanations for declarative and procedural parts

A second challenge for future versions is to offer debugging support for errors, especially when a GridBook query does not produce a result, and participants have no starting point to begin reasoning about what might be going wrong, and why the output was not what they expected. Tools such as WhyLine [39] answer such why questions for programmers and have shown success in lab studies. However, their adoption in commercial tools has been limited. Moreover, they have been built for more structured programming languages, whereas GridBook operates at a higher level of abstraction, where the procedural ("how") part of programming is automatically generated from the declarative "what" part that the user inputs. Debugging support for systems such as GridBook need to explain problems with both these parts, without burdening the user with too much information, and is an area ripe for research.

9.3 Is natural language the future of formulas?

Today, formulas are at the heart of calculations in spreadsheets. Users wield them to accomplish a wide range of tasks across domains, making the spreadsheet formula language the most widely used programming language [27]. It is also widely agreed that users face various challenges learning, writing, and understanding formulas (especially complex ones) when writing in the formula language, a functional programming language. It is in these circumstances that our study offers empirical evidence on the benefits of using natural language to specify computations in Excel spreadsheets. Then, the question arises: *should NL be the primary way in which formulas are written in spreadsheets?*

Currently, GridBook only lets users write formulas for data analysis on tables (akin to SQL). But prior work, notably NLYze [22], has explored the possibility of using natural language to help users create formulas, even though those natural language inputs are not persisted, and that once the formula is created, the user must work with them like other formulas created from scratch (e.g., drag fill, debug the formula in the formula language, etc). In the future, can NL be the primary way in which formulas are specified in spreadsheets?

This idea begs several questions. Can we provide an alternate representation and interface for authoring regular formulas using NL, combining formulas that do SQL-style analysis, as well as formulas that return a single value and can be drag filled? Is there a certain class of formulas (e.g., LOOKUP functions, INDEX and MATCH, or ones related to regular expressions) that are easier to learn, write and manipulate using NL than with other paradigms (e.g., traditional formula language, programming by example)? Will natural language formulas also liberate users from the comprehension challenges of long formulas [75]? Should natural language formulas also be extended to allow other kinds of content (e.g., graphs, data validations, formatting)?

On the other hand, there are also several challenges that need to be addressed. How do we train language models, and what are the tradeoffs in learning commonly mal-framed NL queries vs. learning the correct queries for the correct intent? Are there ambiguities with NL formulas? Do human readers resolve the ambiguities of other people's NL formulas in the same way that GridBook does? Finally, how do we build holistic interfaces and train users to adopt NL, and how will they fall back when NL is unable to accomplish the tasks they have?

9.4 Implications for intelligence models for spreadsheets

In GridBook, there were several ways to arrive at a table of results as an answer. A user could progressively build an analysis table column by column (Figure 2 (a)-(d)), or by using a single query returning a multi-column spilled array (Figure 2 (e) and (f)), data simply entered into a cell, pivot tables, or formulas computed using existing data, or even the results of a GridBook query. Participants opportunistically combined these tools to make progress in their task and expected them to seamlessly integrate with each other. Notably, they expected GridBook's context inference to intelligently treat contiguous two-dimensional layout of content as a single table abstraction, irrespective of how the rows and columns in that table were derived.

However, GridBook was limited and failed in several instances to pick up such abstractions as part of its context inference. All 20/20 participants made at least one attempt in vain to see whether GridBook was "intelligent enough" to recognize the same abstractions that users visually viewed on their screen. This result calls for developing the models required to recognize such abstractions consistent with what the user sees, rather than the underlying implementations.

The need for recognizing such abstractions also brings with it user interaction as well as implementation questions such as: how do we reference those abstractions without increasing the syntax overhead (e.g., can a user click on a table to automatically recognize the table bounds?). What if the table boundaries change – will the abstractions be preserved, to propagate the changes to dependent computations? There is a rich design space for understanding, recognizing and working with abstractions as close as possible to those of users' mental models.

10 CONCLUSION

GridBook interprets simple English utterances in the spreadsheet grid as queries. Each utterance becomes a query applied to a nearby table, or an augmentation of a previously entered query. The query is built by a predictive algorithm using a deep learning parser and a mechanism to match between items in the utterance and data tables in the grid. GridBook evaluates the query by constructing a spreadsheet formula in the grid: it returns a dynamic array by executing a query interpreter written as a LAMBDA function.

GridBook is the first spreadsheet system to provide natural language formulas in the grid. It brings the merits of computational notebooks to spreadsheets, while offering a forgiving, expressive, and intuitive syntax for queries. In our user study, expert analysts said GridBook was easy to learn, and we found that task completion was no slower than with familiar spreadsheet tools, despite the users being new to GridBook. Moreover, expert analysts performed tasks significantly faster with GridBook than with familiar computational notebooks.

Our findings suggest that natural language formulas could transform spreadsheets, making them as powerful as computational notebooks, yet more usable. In future, we aim to design solutions to the problems of interpreting and debugging natural language formulas uncovered by the user research in this paper.

REFERENCES

[1] Abraham, Robin, and Martin Erwig. "UCheck: A spreadsheet type checker for end users." Journal of Visual Languages & Computing 18, no. 1 (2007): 71-95.
[2] Amar, Robert, James Eagan, and John Stasko. "Low-level components of analytic activity in information visualization." In IEEE Symposium on Information Visualization, 2005. INFOVIS 2005., pp. 111-117. IEEE, 2005.
[3] Amouzgar, Farhad, Amin Beheshti, Samira Ghodratnama, Boualem Benatallah, Jian Yang, and Quan Z. Sheng. "iSheets: a spreadsheet-based machine learning development platform for data-driven process analytics." In International Conference on Service-Oriented Computing, pp. 453-457. Springer, Cham, 2018.
[4] Ballard, Bruce W., and Alan W. Biermann. "Programming in natural language: "NLC" as a prototype." In Proceedings of the 1979 annual conference, pp. 228-237. 1979.
[5] Barricelli, Barbara Rita, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. "End-user development, end-user programming and end-user software engineering: A systematic mapping study." Journal of Systems and Software 149 (2019): 101-137.

[6] Borowski, Marcel, Johannes Zagermann, Clemens N. Klokmose, Harald Reiterer, and Roman Rädle. "Exploring the Benefits and Barriers of Using Computational Notebooks for Collaborative Programming Assignments." In Proceedings of the 51st ACM Technical Symposium on Computer Science Education, pp. 468-474. 2020.

[7] Bruckman, Amy, and Elizabeth Edwards. "Should we leverage natural-language knowledge? An analysis of user errors in a natural-language-style programming language." In Proceedings of the SIGCHI conference on Human Factors in Computing Systems, pp. 207-214. 1999.

[8] Burnett, Margaret. "What is end-user software engineering and why does it matter?" In International symposium on end user development, pp. 15-28. Springer, Berlin, Heidelberg, 2009.

[9] Carroll, John M., and Mary Beth Rosson. "Paradox of the active user." In Interfacing thought: Cognitive aspects of human-computer interaction, pp. 80-111. 1987.

[10] Chambers, Chris, and Chris Scaffidi. "Struggling to excel: A field study of challenges faced by spreadsheet users." In 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 187-194. IEEE, 2010.

[11] Chattopadhyay, Souti, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. "What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities." In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2020.

[12] Chen, Xinyun, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. "SpreadsheetCoder: Formula Prediction from Semi-structured Context." In International Conference on Machine Learning, pp. 1661-1672. PMLR, 2021.

[13] Dhamdhere, Kedar, Kevin S. McCurley, Ralfi Nahmias, Mukund Sundararajan, and Qiqi Yan. "Analyza: Exploring data with conversation." In Proceedings of the 22nd International Conference on Intelligent User Interfaces, pp. 493-504. 2017.

[14] Dogga, Pradeep, Karthik Narasimhan, Anirudh Sivaraman, and Ravi Netravali. "A system-wide debugging assistant powered by natural language processing." In Proceedings of the ACM Symposium on Cloud Computing, pp. 171-177. 2019.

[15] Dong, Haoyu, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. "Tablesense: Spreadsheet table detection with convolutional neural networks." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, pp. 69-76. 2019.

[16] Elgohary, Ahmed, Saghar Hosseini, and Ahmed Hassan Awadallah. "Speak to your parser: Interactive text-to-SQL with natural language feedback." arXiv preprint arXiv:2005.02539 (2020).

[17] Ernst, Michael D. "Natural language is a programming language: Applying natural language processing to software development." In 2nd Summit on Advances in Programming Languages (SNAPL 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[18] Furnas, George W., Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. "The vocabulary problem in human-system communication." *Communications of the ACM* 30, no. 11 (1987): 964-971.

[19] Gao, Tong, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. "Datatone: Managing ambiguity in natural language interfaces for data visualization." In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pp. 489-500. 2015.

[20] Gordon, Andy, and Simon Peyton Jones. (January, 2021) "LAMBDA: The ultimate Excel worksheet function." Available at: https://www.microsoft.com/en-us/research/blog/lambda-the-ultimatae-excel-worksheet-function/

[21] Grigoreanu, Valentina, Margaret Burnett, Susan Wiedenbeck, Jill Cao, Kyle Rector, and Irwin Kwan. "End-user debugging strategies: A sensemaking perspective." ACM Transactions on Computer-Human Interaction (TOCHI) 19, no. 1 (2012): 1-28.

[22] Gulwani, Sumit, and Mark Marron. "Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation." In Proceedings of the 2014 ACM SIGMOD international conference on Management of data, pp. 803-814. 2014.

[23] Hart, Sandra G., and Lowell E. Staveland. "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research." In Advances in psychology, vol. 52, pp. 139-183. North-Holland, 1988.

[24] Head, Andrew, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. "Managing messes in computational notebooks." In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2019.

[25] Hendry, David G., and Thomas RG Green. "Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model." International Journal of Human-Computer Studies 40, no. 6 (1994): 1033-1065.

[26] Hermans, F., Pinzger, M., & van Deursen, A. (2015). Detecting and refactoring code smells in spreadsheet formulas. Empirical Software Engineering, 20(2), 549-575.

[27] Hermans, Felienne, Bas Jansen, Sohon Roy, Efthimia Aivaloglou, Alaaeddin Swidan, and David Hoepelman. "Spreadsheets are code: An overview of software engineering approaches applied to spreadsheets." In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol. 5, pp. 56-65. IEEE, 2016.

[28] Hodnigg, Karin, and Roland T. Mittermeir. "Metrics-based spreadsheet visualization: Support for focused maintenance." arXiv preprint arXiv:0809.3009 (2008).

[29] Hoque, Enamul, Vidya Setlur, Melanie Tory, and Isaac Dykeman. "Applying pragmatics principles for interaction with visual analytics." *IEEE transactions on visualization and computer graphics* 24, no. 1 (2017): 309-318.

[30] Huang, Junyang, Yongbo Wang, Yongliang Wang, Yang Dong, and Yanghua Xiao. "Relation Aware Semi-autoregressive Semantic Parsing for NL2SQL." arXiv preprint arXiv:2108.00804 (2021).

[31] Huang, Po-Sen, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. "Natural language to structured query generation via meta-learning." arXiv preprint arXiv:1803.02400 (2018).

[32] India crimes data. https://www.kaggle.com/webaccess/india-crimes-data. Retrieved 1st October, 2020.

[33] Iyer, Srinivasan, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. "Learning a neural semantic parser from user feedback." arXiv preprint arXiv:1704.08760 (2017).

[34] Jean E. Sammet. 1966. The use of English as a programming language. Commun. ACM 9, 3 (March 1966), 228–230. DOI:https://doi.org/10.1145/365230.365274

[35] Jia, Yunyi, Lanbo She, Yu Cheng, Jiatong Bao, Joyce Y. Chai, and Ning Xi. "Program robots manufacturing tasks by natural language instructions." In 2016 IEEE International Conference on Automation Science and Engineering (CASE), pp. 633-638. IEEE, 2016.

[36] Joharizadeh, Nima, Advait Sarkar, Andrew D. Gordon, and Jack Williams. "Gridlets: Reusing spreadsheet grids." In Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-7. 2020.

[37] Kery, Mary Beth, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. "mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks." In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, pp. 140-151. 2020.

[38] Knuth, Donald Ervin. "Literate programming." The computer journal 27, no. 2 (1984): 97-111.

[39] Ko, Amy J., and Brad A. Myers. "Designing the whyline: a debugging interface for asking questions about program behavior." In Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 151-158. 2004.

[40] Ko, Amy J., Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi *et al.* "The state of the art in end-user software engineering." ACM Computing Surveys (CSUR) 43, no. 3 (2011): 1-44.

[41] Koci, Elvis, Dana Kuban, Nico Luettig, Dominik Olwig, Maik Thiele, Julius Gonsior, Wolfgang Lehner, and Oscar Romero. "Xlindy: Interactive recognition and information extraction in spreadsheets." In Proceedings of the ACM Symposium on Document Engineering 2019, pp. 1-4. 2019.

[42] Koci, Elvis, Maik Thiele, Óscar Romero Moral, and Wolfgang Lehner. "A machine learning approach for layout inference in spreadsheets." In IC3K 2016: Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management: volume 1: KDIR, pp. 77-88. SciTePress, 2016.

[43] Kohlhase, A., Kohlhase, M., & Guseva, A. (2015). Context in Spreadsheet Comprehension. In SEMS@ ICSE (pp. 21-27).

[44] Lau, Sam, Ian Drosos, Julia M. Markel, and Philip J. Guo. "The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry." In 2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 1-11. IEEE, 2020.

[45] Lau, Sam, Sruti Srinivasa Ragavan, Ken Milne, Titus Barik, and Advait Sarkar. "TweakIt: Supporting End-User Programmers Who Transmogrify Code." In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2021.

[46] Lauria, Stanislao, Guido Bugmann, Theocharis Kyriacou, Johan Bos, and A. Klein. "Training personal robots using natural language instruction." IEEE Intelligent systems 16, no. 5 (2001): 38-45.

[47] Le, Vu, Sumit Gulwani, and Zhendong Su. "Smartsynth: Synthesizing smartphone automation scripts from natural language." In Proceeding of the 11th annual international conference on Mobile systems, applications, and services, pp. 193-206. 2013.

[48] Li, Yuntao, Bei Chen, Qian Liu, Yan Gao, Jian-Guang Lou, Yan Zhang, and Dongmei Zhang. "" What Do You Mean by That?" A Parser-Independent Interactive Approach for Enhancing Text-to-SQL." arXiv preprint arXiv:2011.04151 (2020).

[49] Lieberman, Henry, and Hugo Liu. "Feasibility studies for programming in natural language." In End User Development, pp. 459-473. Springer, Dordrecht, 2006.

[50] Lieberman, Henry, Fabio Paternò, and Volker Wulf. (eds). "End-user development". Kluwer/ Springer. 2006.

[51] Miller, Lance A. "Natural language programming: Styles, strategies, and contrasts." IBM Systems Journal 20, no. 2 (1981): 184-215.

[52] Nan, Zifan, Hui Guan, and Xipeng Shen. "HISyn: human learning-inspired natural language programming." In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 75-86. 2020.

[53] Nardi, Bonnie A. A small matter of programming: perspectives on end user computing. MIT press, 1993.

[54] Narechania, Arpit, Arjun Srinivasan, and John Stasko. "NL4DV: A toolkit for generating analytic specifications for data visualization from natural language

queries." IEEE Transactions on Visualization and Computer Graphics 27, no. 2 (2020): 369-379.
[55] Norman, Don. The design of everyday things: Revised and expanded edition. Basic books, 2013.
[56] OpenAI Codex. https://openai.com/blog/openai-codex/. Retrieved 8th October, 2021.
[57] Pane, John F., and Brad A. Myers. "More natural programming languages and environments." In End user development, pp. 31-50. Springer, Dordrecht, 2006.
[58] Pane, John F., and Brad A. Myers. "Studying the language and structure in non-programmers' solutions to programming problems." International Journal of Human-Computer Studies 54, no. 2 (2001): 237-264.
[59] Panko, Raymond R. "Spreadsheet errors: What we know. what we think we can do." arXiv preprint arXiv:0802.3457 (2008).
[60] Panko, Raymond R. "Two experiments in reducing overconfidence in spreadsheet development." *Journal of Organizational and End User Computing (JOEUC)* 19, no. 1 (2007): 1-23.
[61] Powell, Stephen G., Kenneth R. Baker, and Barry Lawson. "A critical review of the literature on spreadsheet errors." Decision Support Systems 46, no. 1 (2008): 128-138.
[62] Price, David, Ellen Rilofff, Joseph Zachary, and Brandon Harvey. "NaturalJava: A natural language interface for programming in Java." In Proceedings of the 5th international conference on Intelligent user interfaces, pp. 207-211. 2000.
[63] Ramshaw, Lance A., and Mitchell P. Marcus. "Text chunking using transformation-based learning." In Natural language processing using very large corpora, pp. 157-176. Springer, Dordrecht, 1999.
[64] Raza, Mohammad, Sumit Gulwani, and Natasa Milic-Frayling. "Compositional program synthesis from natural language and examples." In Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015.
[65] Rothermel, K. J., Cook, C. R., Burnett, M. M., Schonfeld, J., Green, T. R., & Rothermel, G. (2000, June). WYSIWYT testing in the spreadsheet paradigm: An empirical evaluation. In Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium (pp. 230-239). IEEE.
[66] Rule, Adam, Aurélien Tabard, and James D. Hollan. "Exploration and explanation in computational notebooks." In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2018.
[67] Rule, Adam, Ian Drosos, Aurélien Tabard, and James D. Hollan. "Aiding collaborative reuse of computational notebooks with annotated cell folding." Proceedings of the ACM on Human-Computer Interaction 2, no. CSCW (2018): 1-12.
[68] Sarkar, Advait, Andrew D. Gordon, Simon Peyton Jones, and Neil Toronto. "Calculation view: multiple-representation editing in spreadsheets." In 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 85-93. IEEE, 2018.
[69] Sarkar, Advait, Mateja Jamnik, Alan F. Blackwell, and Martin Spott. "Interactive visual machine learning in spreadsheets." In 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 159-163. IEEE, 2015.
[70] Scaffidi, Christopher, Mary Shaw, and Brad Myers. "Estimating the numbers of end users and end user programmers." In 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), pp. 207-214. IEEE, 2005.
[71] Scaffidi, Christopher. "The impact of human-centric design on the adoption of information systems: A case study of the spreadsheet." In 2016 11th Iberian Conference on Information Systems and Technologies (CISTI), pp. 1-7. IEEE, 2016.
[72] Schlegel, Viktor, Benedikt Lang, Siegfried Handschuh, and André Freitas. "Vajra: step-by-step programming with natural language." In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pp. 30-39. 2019.
[73] Setlur, Vidya, Melanie Tory, and Alex Djalali. "Inferencing underspecified natural language utterances in visual analysis." Proceedings of the 24th International Conference on Intelligent User Interfaces. 2019.
[74] Setlur, Vidya, Sarah E. Battersby, Melanie Tory, Rich Gossweiler, and Angel X. Chang. "Eviza: A natural language interface for visual analysis." In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pp. 365-377. 2016.
[75] Srinivasa Ragavan, Sruti, Advait Sarkar, and Andrew D. Gordon. "Spreadsheet Comprehension: Guesswork, Giving Up and Going Back to the Author." In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1-21. 2021.
[76] Srinivasan, Arjun, and John Stasko. "Orko: Facilitating multimodal interaction for visual exploration and analysis of networks." *IEEE transactions on visualization and computer graphics* 24, no. 1 (2017): 511-521.
[77] Tang, Lappoon R., and Raymond Mooney. "Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing." In 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, pp. 133-141. 2000.
[78] Tellex, Stefanie, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. "Robots that use language." Annual Review of Control, Robotics, and Autonomous Systems 3 (2020): 25-55.
[79] Tichy, Walter F., Rolf L. Adams, and Lars Holter. "NLH/E: A natural language help system." In Proceedings of the 11th international conference on Software engineering, pp. 364-374. 1989.
[80] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In Advances in neural information processing systems, pp. 5998-6008. 2017.
[81] Wachtel, Alexander, Dominik Fuchß, Matthias Przybylla, and Walter F. Tichy. "Natural Language Data Queries on Multiple Heterogenous Data Sources." In International Symposium on End User Development, pp. 174-182. Springer, Cham, 2019.
[82] Wachtel, Alexander, Michael T. Franzen, and Walter F. Tichy. "Context Detection in Spreadsheets Based on Automatically Inferred Table Schema." International Journal of Computer and Information Engineering 10, no. 10 (2016): 1892-1899.
[83] Wachtel, Alexander, Sebastian Weigelt, and Walter F. Tichy. "Initial implementation of natural language turn-based dialog system." Procedia Computer Science 84 (2016): 49-56.
[84] Wang, April Yi, Anant Mittal, Christopher Brooks, and Steve Oney. "How data scientists use computational notebooks for real-time collaboration." Proceedings of the ACM on Human-Computer Interaction 3, no. CSCW (2019): 1-30.
[85] Wang, April Yi, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D. Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. "Themisto: Towards Automated Documentation Generation in Computational Notebooks." arXiv preprint arXiv: 2102.12592 (2021).
[86] Wang, Chenglong, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. "Robust text-to-sql generation with execution-guided decoding." arXiv preprint arXiv:1807.03100 (2018).
[87] Wang, Chenglong, Marc Brockschmidt, and Rishabh Singh. "Pointing out SQL queries from text." (2018).
[88] Warren, D. H. D. and Pereira, F. C. N. (1982). An efficient easily adaptable system for interpreting natural language queries. Comput. Linguist., 8(3-4):110–122.
[89] Weigelt, Sebastian, Vanessa Steurer, Tobias Hey, and Walter F. Tichy. "Programming in natural language with fuse: Synthesizing methods from spoken utterances using deep natural language understanding." In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 4280-4295. 2020.
[90] Weinman, Nathaniel, Steven M. Drucker, Titus Barik, and Robert DeLine. "Fork It: Supporting Stateful Alternatives in Computational Notebooks." In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1-12. 2021.
[91] Williams, Jack, Carina Negreanu, Andrew D. Gordon, and Advait Sarkar. "Understanding and Inferring Units in Spreadsheets." In 2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 1-9. IEEE, 2020.
[92] Williams, Jack, Nima Joharizadeh, Andrew D. Gordon, and Advait Sarkar. "Higher-Order Spreadsheets with Spilled Arrays." In ESOP, pp. 743-769. 2020.
[93] Yu, Bowen, and Cláudio T. Silva. "FlowSense: A natural language interface for visual data exploration within a dataflow system." IEEE transactions on visualization and computer graphics 26, no. 1 (2019): 1-11.
[94] Yu, Tao, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma *et al.* "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task." In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 3911-3921. 2018.
[95] Zhang, Yakun, Wensheng Dou, Jiaxin Zhu, Liang Xu, Zhiyong Zhou, Jun Wei, Dan Ye, and Bo Yang. "Learning to detect table clones in spreadsheets." In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 528-540. 2020.
[96] Zhong, Victor, Caiming Xiong, and Richard Socher. "Seq2sql: Generating structured queries from natural language using reinforcement learning." arXiv preprint arXiv:1709.00103 (2017)
[97] GridBook. Microsoft Research. https://www.microsoft.com/en-us/research/project/gridbook/. Retrieved 10th February, 2022.