Learning to Detect Table Clones in Spreadsheets

Yakun Zhang
State Key Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
zhangyakun18@otcaix.iscas.ac.cn

Wensheng Dou
Jiaxin Zhu
State Key Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
{wsdou,zhujiaxin}@otcaix.iscas.ac.cn

Liang Xu
Jinling Institute of Technology
Nanjing, China
xuliang@jit.edu.cn

Zhiyong Zhou
National Engineering Research Center of Fundamental Software, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
zhiyong@nfs.iscas.ac.cn

Jun Wei
Dan Ye
State Key Lab of Computer Sciences, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
{wj,yedan}@otcaix.iscas.ac.cn

Bo Yang
North China University of Technology
Beijing, China
yangbo090313@163.com

ABSTRACT

In order to speed up spreadsheet development productivity, end users can create a spreadsheet table by copying and modifying an existing one. These two tables share the similar computational semantics, and form a table clone. End users may modify the tables in a table clone, e.g., adding new rows and deleting columns, thus introducing structure changes into the table clone. Our empirical study on real-world spreadsheets shows that about 58.5% of table clones involve structure changes. However, existing table clone detection approaches in spreadsheets can only detect table clones with the same structures. Therefore, many table clones with structure changes cannot be detected.

We observe that, although the tables in a table clone may be modified, they usually share the similar structures and formats, e.g., headers, formulas and background colors. Based on this observation, we propose *LTC* (*L*earning to detect *T*able *C*lones), to automatically detect table clones with or without structure changes. LTC utilizes the structure and format information from labeled table clones and non table clones to train a binary classifier. LTC first identifies tables in spreadsheets, and then uses the trained binary classifier to judge whether every two tables can form a table clone. Our experiments on real-world spreadsheets from the EUSES and Enron corpora show that, LTC can achieve a precision of 97.8% and recall of 92.1% in table clone detection, significantly outperforming the state-of-the-art technique (a precision of 37.5% and recall of 11.1%).

ISSTA '20, July 18–22, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8008-9/20/07…$15.00
https://doi.org/10.1145/3395363.3397384

CCS CONCEPTS

• **Applied computing** → **Spreadsheets**; • **Software and its engineering** → *Software testing and debugging*.

KEYWORDS

Spreadsheet, table clone, structure, format

ACM Reference Format:
Yakun Zhang, Wensheng Dou, Jiaxin Zhu, Liang Xu, Zhiyong Zhou, Jun Wei, Dan Ye, and Bo Yang. 2020. Learning to Detect Table Clones in Spreadsheets. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '20), July 18–22, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3395363.3397384

1 INTRODUCTION

Spreadsheet systems are one of the most successful end-user programming platforms, and have been widely used in various business tasks, including data storage, data analysis, financial reporting and so on [44]. Scaffidi [48] estimated that, in 2012, over 55 million users in the United States worked with spreadsheets. There must be many more users working with spreadsheets nowadays.

Spreadsheets are code, too. Spreadsheets usually play a similar role to source code in conventional programming languages [25]. Similar to code reuse, end users often reuse existing spreadsheets to speed up their development productivity. For example, a user can prepare a new financial report by copying-pasting-modifying an existing one, thus saving amount of time. For two tables created by copy-paste-modify, they share the same or similar computational semantics. We refer to them as a table clone. For example, the two tables in Figure 1a and Figure 1b form a table clone.

Similar to code clones in conventional programs [13, 32, 36, 43], table clones can be applied on some important spreadsheet analysis scenarios. First, table clones can be used to find the same errors scattered in many spreadsheets. Second, inconsistent modifications may introduce errors into table clones [18, 30], thus causing financial losses. Table clones can facilitate to detect and fix spreadsheet errors by analyzing their inconsistencies [18]. Third, table clones are usually used to perform the same business task. Thus, data analysis tools, e.g., PowerBI [4], can extract and analyze all table clones together. Fourth, table clones usually share common computational semantics and structures. Extracting table templates from table clones can facilitate end users to create new table instances.

However, there are no records in the spreadsheet systems (e.g., Microsoft Excel) documenting that two tables were created by copy-paste-modify [31]. Therefore, it is important to effectively detect table clones in spreadsheets. Existing clone detection approaches in spreadsheets [18, 30] mainly focus on the clones with the same data or structures. For example, Hermans et al. [30] consider two blocks of numerical cells with (almost) the same values as a data clone. TableCheck [18] considers two blocks of numerical cells with the same headers, i.e., structures, as a table clone. However, our empirical study on real-world spreadsheets from the EUSES [24] and Enron [26] corpora (Section 4.1) shows that 58.5% of table clones involve structure changes, which cannot be detected by existing approaches. Moreover, code clone detection approaches in conventional programs, e.g., CCFinder [36], Deckard [32], CloneDetective [35], and CCLearner [42], cannot be applied on spreadsheets, because the programming model in spreadsheets is totally different from those in conventional programs.

In this paper, we propose *LTC* (*L*earning to detect *T*able *C*lones), to detect table clones with or without structure changes in spreadsheets. We observe that the structures and formats, e.g., headers, formulas, and cell fonts, can effectively characterize the computational semantics of a table. If two tables share the same or similar structures and formats, they are likely to be a table clone, and share the similar computational semantics. Therefore, we can treat table clone detection as classifying two tables as a clone or non clone based on their structure and format similarities. Specifically, given two tables, we extract 12 features about structures and formats from each table, and compute the similarity score for each feature. Then, we apply supervised classification algorithms on labeled table clones and non table clones to train a binary classifier. With the trained classifier, LTC first identifies tables in spreadsheets, and then compares every two tables to detect table clones.

We evaluate LTC on the real-world spreadsheets from the EUSES [24] and Enron [26] corpora, which are two of the most widely-used corpora for spreadsheet research [7, 19, 22]. The experimental results show that LTC can detect table clones effectively, with a precision of 97.8% and recall of 92.1%. As a comparison, TableCheck [18] can only detect table clones with a precision of 37.5% and recall of 11.1%. This result shows that LTC can significantly outperform existing approaches.

In summary, we make the following contributions in this paper.

- We propose a commonly-used notation in spreadsheets, table clone, in which two tables share the similar computational semantics.
- We propose a learning-based approach, LTC, to detect table clones with or without structure changes, by exploiting the structure and format similarities among tables.
- We implement LTC and evaluate it on real-world spreadsheets from the EUSES and Enron corpora. The experimental results show that LTC can detect table clones effectively.

	A	B	C	D	E	F
1	January	Week 1	Week 2	Week 3	Total Hours	Overtime Hours
2	Green	10	10	5	=SUM(B2:D2)	=MAX(E2-120,0)
3	Jones	15	18	20	=SUM(B3:D3)	=MAX(E3-120,0)
4	Smith	13	21	16	=SUM(B4:D4)	=MAX(E4-120,0)
5	Johnson	42	38	43	=SUM(B5:D5)	=MAX(E5-120,0)
6	White	44	40	42	=SUM(B6:D6)	=MAX(E6-120,0)
7	Edwards	23	35	38	=SUM(B7:D7)	=MAX(E7-120,0)

(a) January

	A	B	C	D	E	F
1	February	Week 1	Week 2	Week 3	Total Hours	Overtime Hours
2	Green	10	10	5	=SUM(B2:D2)	=MAX(E2-120,0)
3	Jones	15	18	20	=SUM(B3:D3)	=MAX(E3-120,0)
4	Smith	13	21	16	=SUM(B4:D4)	=MAX(E4-120,0)
5	Johnson	35	39	30	=SUM(B5:D5)	=MAX(E5-120,0)
6	White	36	43	32	=SUM(B6:D6)	=MAX(E6-120,0)
7	Edwards	30	40		=SUM(B7:C7)	=MAX(E7-80,0)

(b) February

	A	B	C	D	E	F
1	March	Week 1	Week 2	Week 3	Total Hours	Overtime Hours
2	Green	20	30	22	=SUM(B2:D2)	=MAX(E2-120,0)
3	Jones	17	42	24	=SUM(B3:D3)	=MAX(E3-120,0)
4	Smith	35	25	28	=SUM(B4:D4)	=MAX(E4-120,0)
5	Johnson	30	23	35	=SUM(B5:D5)	=MAX(E5-120,0)
6	Edwards	21	43	30	=SUM(B6:C6)	=MAX(E6-80,0)
7	Amy	30	30	30	90	0

(c) March

	A	B	C	D	E
1	April	Week 1	Week 2	Total Hours	Overtime Hours
2	Green	10	30	=SUM(B2:C2)	=MAX(D2-80,0)
3	Jones	22	13	=SUM(B3:C3)	=MAX(D3-80,0)
4	Smith	12	20	=SUM(B4:C4)	=MAX(D4-80,0)
5	Johnson	43	38	=SUM(B5:C5)	=MAX(D5-80,0)
6	Amy	44	40	=SUM(B6:C6)	=MAX(D6-80,0)

(d) April

Figure 1: Four worksheet excerpts extracted from the EUSES corpus [24]. The tables in every two worksheets form a table clone. The cells in the rectangles show the key differences between the current and previous worksheets. The cells marked by a red right-cornered triangle contain errors.

2 TABLE CLONES IN SPREADSHEETS

In this section, we explain table clones and our motivation using an illustrative spreadsheet example.

2.1 Motivating Example

Figure 1 shows four worksheet excerpts, which are extracted from the EUSES corpus [24]. These worksheet excerpts are used to analyze the working hours in four months, i.e., from January to April.

In January, Alice created worksheet *January* to analyze the working hours of her team members in January. In February, Alice copied worksheet *January*, and created a new worksheet *February*. She updated the data and formulas of Johnson, White and Edwards. Because Edwards did not work in "Week 3", Alice left cell D7 in blank, and fixed the formulas in E7 and F7 accordingly. We can see that the computational semantics, structures and formats in worksheet *January* and *February* are almost the same. Similarly, based on worksheet *February*, Alice created worksheet *March*. She removed White (row 6 in Figure 1b) and added a new member Amy (row 7 in Figure 1c). Note that, Alice updated the value in D6, and forgot to update the formulas in E7 and F7, introducing two formula errors. Further, Alice directly filled all values for Amy without using formulas, introducing two missing formula errors in E7 and F7. For worksheet *April*, Alice deleted column "Week 3" and removed Edwards (row 6 in Figure 1c).

2.2 Table Clones

Although the four worksheet excerpts in Figure 1 have different data (e.g., January![B5:D7][1] and February![B5:D7]), and structures (e.g., column B-F in Figure 1c and column B-E in Figure 1d), they have the similar computational semantics, i.e., they are all used to analyze the working hours in the same way. Thus, the tables in every two worksheets in Figure 1 form a table clone. For example, January![A1:F7] and February![A1:F7] form a table clone.

Definition 1: A *table* is a rectangular block of cells, prescribing certain business task. For example, January![A1:F7] in Figure 1 form a table for storing and analyzing the working hours in January.

In spreadsheets, table is the key structure for data processing and information presentation [15, 17]. A table typically contains the following four elements. (1) *Header cell*: Header cells describe other cells in a table. For example, January![B1:F1] and January![A1:A7] in Figure 1a are the headers of table January![A1:F7]. Through header cell B1 and A2, we can know that 10 in cell B2 means Green worked for 10 hours in Week 1. (2) *Data cell*: Data cells store the business data, e.g., the numerical cells in January![B2:D7] in Figure 1a. (3) *Formula cell*: Formula cells analyze data in a table, e.g., January![E2:F7] in Figure 1a. (4) *Cell format*: The cell formats can facilitate user inspection. For example, header cells January![B1:F1] are bold and italic, and have bottom borders.

Definition 2: A *table clone* is a table pair (t_1, t_2), in which table t_1 and t_2 share the same or similar computational semantics, and prescribe the same or similar business task. In Figure 1, the tables in every two worksheets can form a table clone, e.g., table January![A1:F7] and February![A1:F7].

Note that if a table contains only one row/column, it has limited information, e.g., no headers or data. Thus, we require that tables in a table clone have at least two rows and columns. We have several observations on table clones in spreadsheets. First, headers describe the semantics of a table, and tables in a table clone usually share the same or similar headers. For example, table January![A1:F7] and February![A1:F7] share the same headers. Second, formulas present the computations, and table clones usually share the same or similar formulas. Although the formulas in Figure 1a and Figure 1d are different, they have similar computations, in which they are used to summarize the working hours for each worker. Third, table clones usually share similar cell formats, e.g., fonts, borders. In addition, tables in a table clone usually store different data for similar tasks. For example, the working hours in Figure 1a and Figure 1c are very different. These observations motivate us to detect table clones by learning the similarities of structures and formats in tables.

[1]This denotes cells [B5:D7] of worksheet *January* in Figure 1a. We use similar form to reference cells throughout this paper.

	A	B	C	D	E	F	G
1		***Month***	***Week 1***	***Week 2***	***Week 3***	***Total Hours***	***Overtime Hours***
2	***Green***	January	10	10	5	=SUM(C2:E2)	=MAX(F2-120,0)
3	***Green***	Februrary	10	10	5	=SUM(C3:E3)	=MAX(F3-120,0)
4	***Green***	March	20	30	22	=SUM(C4:E4)	=MAX(F4-120,0)
5	***Green***	April	10	30		=SUM(C5:D5)	=MAX(F5-80,0)
6	***Jones***	January	15	18	20	=SUM(C6:E6)	=MAX(F6-120,0)
7	…	…	…	…	…	…	…
8	***Amy***	April	44	40		=SUM(C8:D8)	=MAX(F8-80,0)

Figure 2: The summary of four worksheets in Figure 1.

2.3 Potential Applications of Table Clones

Table clones can be further applied on some important spreadsheet analysis scenarios, e.g., error detection, data analysis, and spreadsheet reuse. This motivates us to effectively detect table clones in spreadsheets. We explain these potential applications as follows.

Detecting errors related to table clones. In our motivating example, cell E6 and F6 in Figure 1c suffer from formula errors, in which a cell's formula is wrong. Cell E7 and F7 in Figure 1c suffer from missing formulas, in which a cell is supposed to contain a formula, but it does not. In a table clone, its corresponding cells usually share the same / similar computational semantics. The inconsistencies among the corresponding cells usually indicate errors. For example, cell E7 in Figure 1a and cell E6 in Figure 1c should have the same computational semantics. However, they contain inconsistent formulas. Based on this observation, we can use table clones to detect spreadsheet errors by analyzing their inconsistencies [18, 33]. Further, given a spreadsheet error, e.g., the formula error in cell E6 of Figure 1c, we can extract all related table clones and check whether they suffer from the same error. This also highlights the importance of table clones.

Data integration and analysis among table clones. Table clones are usually used to perform the same / similar business task. However, these related tables usually scatter in many worksheets or spreadsheets. For examples, the four tables in Figure 1 scatter in four worksheets. For data integration, one key question is where to find related tables. The detected table clones can be a good source for table integration. Table clones can help manage all related tables, and facilitate integrating all related tables into a consistent form [15]. For example, Figure 2 is the summary of the four tables in Figure 1. This summary table can be easily used by other data analysis tools, e.g., Insights in Excel [3] and PowerBI [4].

Table template extraction applications. When end users reuse an existing table, they usually need to delete the original data, and revise the table according to new requirements [51]. These tasks are usually daunting and error-prone. For example, end users need to delete old data, fix formulas and fill new data when creating Figure 1c based on Figure 1b. We cannot extract table templates from only one table. Table clones provide a group of instances about

Table 1: Table Clone Types

Variance	Type-1	Type-2	Type-3	Type-4
format	○	○	○	○
data	✗	✓	○	○
formula	✗	✓	○	○
header	✗	✗	✓	○
Row/col insertion	✗	✗	✗	✓
Row/col deletion	✗	✗	✗	✓

Note: ○ = non-exclusion, ✗ = exclusion, ✓ = inclusion

how a table template is used. Given a group of table clones, we can infer how end users reuse and revise these tables, e.g., what semantics and structures should be kept and which cells will be changed. Table clones can be used to extract table templates [8], which can facilitate end users to create new table instances and avoid errors.

2.4 Table Clone Types

In conventional programs, code clones are categorized into four different types [13, 42] according to how developers modify code clones. Since spreadsheets have completely different programming model from conventional programs, the tables in a table clone can be modified in different ways. According to how users modify tables in a table clone, we completely redesign the clone categorization, and further categorize table clones into four types. For a spreadsheet table, headers, formulas and data mainly reflect its computational semantics. Therefore, our table clone type categorization reflects how many computational semantics are changed. Table 1 shows the comparison among the four types of table clones. Note that, table clones are exclusive in their types. That's said, a table clone cannot belong to two types in the same time. We describe four types of table clones as follows.

- **Type-1**: Type-1 table clones are identical tables allowing variations in formats, e.g., cell formats (including height, width, border, color, font) and hidden rows / columns.
- **Type-2**: Type-2 table clones have variations on data and formulas, and also allow variations in Type-1.
- **Type-3**: Type-3 table clones have variations on table headers, and also allow variations in Type-2.
- **Type-4**: Type-4 table clones allow extra modifications, e.g., inserting or deleting rows / columns, and also allow variations in Type-3.

We further illustrate the four types of table clones using Figure 1. (1) If we copy table January![A1:F7] into cells [A11:F17] in Figure 1a, we can form a Type-1 table clone, January![A1:F7] and January![A11:F17]. (2) Table January![A1:F7] and February![A1:F7] only have variations in data and formulas, and form a Type-2 table clone. Note that, headers are not changed in these two tables. (3) Table January![A1:F7] and March![A1:F7] have different headers (i.e., row 7), and form a Type-3 table clone. Note that, in Type-3 table clones, two tables have the same size, i.e., row and column numbers. (4) Table January![A1:F7] and April![A1:E6] have different row and column numbers, and form a Type-4 table clone.

2.5 Other Clones in Spreadsheets

Hermans et al. [30] consider two blocks of numerical cells with (almost) the same values as a data clone, e.g., January![B2:D4] and February![B2:D4]. Data clones are very different from table clones. TableCheck [18] considers two blocks of numerical cells with the same headers as a table clone, e.g., January![B2:F7] and February![B2:F7]. Ideally, TableCheck can detect Type-1 and Type-2 table clones. Therefore, neither data clone detection nor TableCheck can detect Type-3 and Type-4 table clones. However, our empirical study on real-world spreadsheets (Section 4.1) shows that, 58.5% of table clones belong to Type-3 and Type-4. This motivates us to detect all types of table clones.

3 LEARNING-BASED TABLE CLONE DETECTION

The key insight of our approach is that table clones share the similar structures and formats, even though they contain various variations. By using structures and formats, we are able to predict whether two tables can form a table clone.

Figure 3 presents the overview of our approach, LTC. Given some spreadsheets, LTC first identifies tables (Section 3.1), and then extracts 12 features about structures and formats in each table. For every table pair from the identified tables, LTC computes their feature similarities (Section 3.2). Finally, table clones are predicted by leveraging a binary classification on feature similarities. To train a binary classifier, LTC extracts 12 features from labeled table clones and non table clones, and uses Random Forest [14] to obtain a binary classifier (Section 3.3).

3.1 Table Identification

A spreadsheet usually contains multiple worksheets. End users may put multiple tables into the same worksheet. For example, Figure 4 shows a worksheet excerpt, which contains two tables, i.e., [A1:G8] and [A10:C14]. It is improper to treat the whole worksheet as a table, since different parts of a worksheet implement different functions, e.g., [A1:G8] and [A10:C14] in Figure 4 have totally different functions. In spreadsheets, we observe that tables are usually circumscribed by empty cells, e.g., empty cells [A9:G9] in Figure 4. We first classify spreadsheet cells into four types, and then identify tables in each worksheet based on cell types.

3.1.1 Cell Types. We utilize the approaches proposed by Hermans et al. [29], and classify cells into four types. (a) *Data*: A cell filled with data, e.g., cell B2 in Figure 4. (b) *Formula*: A cell containing a calculation on other cells, e.g., cell E2 in Figure 1a. (c) *Label*: A cell containing text and expressing the meanings of other cells, e.g., cell B1 in Figure 4. (d) *Empty*: A blank cell.

We classify cells in a worksheet as follows. First, we mark all numerical cells without formulas as *data* cells. Second, we mark all numerical cells with formulas as *formula* cells. Third, we mark the remaining cells contain strings as *label* cells. Fourth, we mark all blank cells as *empty* cells.

3.1.2 Table Identification. According to Definition 1, a table is a rectangular block of cells, in which related information prescribing certain business task is put together. We observe that tables are usually divided by empty cells and boundaries. For example, in

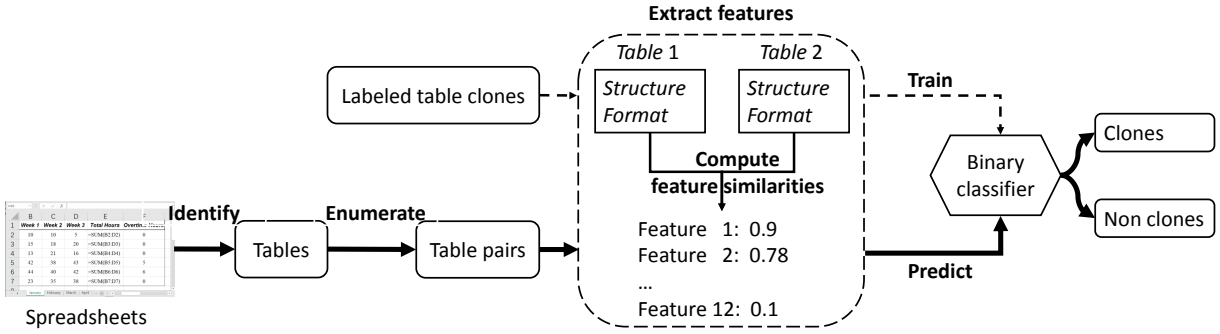

Figure 3: Overview of LTC.

	A	B	C	D	E	F	G
1		*Week 1*	*Week 2*	*Week 3*	*Week 4*	*Total Hours*	*Overtime Hours*
2	*Green*	10	10.5	5	9	34	0
3	*Smith*	15	18	20	18	71	0
4	*Jones*	13	21	16	17	67	0
5	*Adams*	42	38	43	41	164	4
6							
7	*Total hours*	80	87.5	84	84	335	4
8	*Max hours*	42	38	43	41	164	4
9							
10		*Pay rate*	*Gross pay*				
11	*Green*	$7.50	$255.00				
12	*Smith*	$7.25	$512.33				
13	*Jones*	$8.50	$566.67				
14	*Adams*	$8.25	$1,353.00				

Figure 4: A worksheet excerpt extracted from the EUSES corpus [24], which contains two tables: [A1:G8] and [A10:C14].

Algorithm 1: Identifying column headers in a table.

Input: *table*
Output: *headerRows*
1 **for** $curRow \leftarrow 1; curRow \leq 4; curRow++$ **do**
2 **if** *table.getRow(curRow).isHeader()* **then**
3 *headerRows.add(curRow)*;
4 **else**
5 **break**;
6 **end**

Figure 4, table [A1:G8] and [A10:C14] are divided by empty cells [A9:G9]. Thus, we can use empty cells and boundaries to identify tables in a worksheet.

Given a worksheet, our basic table identification algorithm works as follows. (a) We treat the whole worksheet as a cell block. (b) For a cell block, we identify all empty rows and columns in the cell block. If we find some empty rows or columns, we divide the cell block into multiple cell blocks according to the identified empty rows and columns. (c) For each new identified cell block, we repeat the second step, until we cannot find any new cell blocks.

Take the worksheet in Figure 4 as an example. We first identify two empty rows, i.e., row 6 and 9, and thus divide the worksheet into three cell blocks, i.e., [A1:G5], [A7:G8] and [A10:G14]. Further, column D-G in cell block [A10:G14] are empty, thus, we obtain a new cell block [A10:C14]. So far, we obtain three cell blocks, i.e., [A1:G5], [A7:G8] and [A10:C14].

However, we cannot treat these three cell blocks as three tables. A table can be separated into multiple cell blocks by empty rows and columns. For example, cell block [A1:G5] and [A7:G8] in Figure 4 should belong to the same table [A1:G8], because cells [A7:G8] are used to compute the total and max hours for cells [B2:G5], and empty row 6 is used to separated data and computation. Inspired by ExpCheck [20], we further merge related cell blocks into a big one by using table header information (Section 3.1.3). We observe that, although a table can be separated into multiple cell blocks by empty rows / columns, they usually share the same headers. For example, in Figure 4, cell block [A1:G5] and [A7:G8] share the same column headers (i.e., cells [B1:G1]). Based on this observation, we merge two neighboring cell blocks that share the same row headers or column headers. For the worksheet excerpt in Figure 4, we merge cell block [A1:G5] and [A7:G8] into a new cell block [A1:G8]. Finally, we obtain two tables, i.e., [A1:G8] and [A10:C14].

3.1.3 Header Identification. Header cells are used to describe other cells in a table. For example, cells [B1:G1] and [A2:A8] are used to describe data cells [B2:G8] in Figure 4. We use *row header* to denote the header for a row, and *column header* to denote the header for a column. Take table [A1:G8] in Figure 4 as an example. Cell A2 is the row header of row 2, and indicates that cells [B2:G2] belong to "Green". Cell B1 is the column header of column B, and indicates that cells [B2:B8] belong to "Week 1".

We have three observations to identify row headers and column headers in a table. (a) Row headers are usually located in the first few (e.g., 4) columns, and column headers are usually located in the first (e.g., 4) few rows. (b) Row headers in a table usually occupy the whole columns, e.g., column A in table [A1:G8] in Figure 4. Similarly, column headers in a table usually occupy the whole row. (c) Row / column headers are usually *label* cells, e.g., cells [A2:A5] in Figure 4. Note that, the only difference between row headers and column headers is their directions. Thus, for clarity, we only use column headers to explain our header identification algorithm.

Algorithm 1 shows how to identify column headers in a table. It starts from the first row of the table, and then checks whether the row contains at least one *label* cells, and all others cells are *empty* (i.e., *isHeader()*). If yes, the *label* cells in the row can be considered as column headers, and it further checks next row of the table, until

we have checked the first four rows in the table. If no, the row can not be considered headers, and the algorithm returns.

Note that, for a cell block discussed in the previous section, we can identify its column headers, too. If we do not find the column headers for a cell block, we further check its nearest cell block *upBlock* on the top, and identify column headers of cell block *upBlock* as its column headers.

Take the worksheet excerpt in Figure 4 as an example. For cell block [A1:G5], Algorithm 1 identifies cells [B1:G1] as its column headers. For cell block [A7:G8], Algorithm 1 does not find its column headers. So, we further identify the column headers of its nearest cell block on the top, i.e., cell block [A1:G5], and consider cells [B1:G1] as its column headers.

3.2 Feature Extraction

We observe that structures and formats can effectively characterize a table, and the clones always have similar structures and formats.

Given a table, LTC extracts 12 features of structures. Among these 12 features, some (e.g., font color) are also used by existing work [39], and some of them are unique to table clone detection, e.g., row headers, column headers and formulas. For each table pair, LTC computes a similarity score for each feature, and then constructs a 12-value similarity vector to characterize whether this table pair is a table clone. This similarity vector is further used in training and prediction for table clone detection in Section 3.3.

3.2.1 Structure Features. Headers, formulas and cell types can represent the key structures of a table.

As described earlier, headers are used to describe the contents in other cells in a table. For example in Figure 1a, header cell E1 means that cells [E2:E7] are used to compute the total working hours for each team member. Since column headers and row headers describe different aspects of a table, we separate them into two features.

Column header (*Feature* #1): We use the header identification approach in Section 3.1.3 to identify column headers. We denote the column headers of table t as a set H_t, then the similarity score sim_{F1} for column headers in two tables $t1$ and $t2$ is calculated as follows:

$$sim_{F1} = \frac{|H_{t1} \cap H_{t2}|}{|H_{t1} \cup H_{t2}|} \tag{1}$$

According to the formula, when two tables are identical and have the same column headers, the similarity score sim_{F1} is 1. When two tables are totally different and have no column headers in common, the similarity score sim_{F1} is 0. Thus, the range of similarity score for column headers is [0, 1].

Row header (*Feature* #2): Similar to column headers, we use the header identification approach in Section 3.1.3 to identify row headers, and then compute the similarity of row headers in two tables in the same way as column headers.

Formula (*Feature* #3): Formulas in a table usually represent the analyses of the data. Formulas in the corresponding cells among table clones often have the different expressions in the A1 format[2], but the same expression in the R1C1 format. For example in Figure 1a, cells [E2:E7] have different formulas in the A1 format, but have the same formula SUM(RC[−3]:RC[−1]) in the R1C1 format. Therefore, we use formulas in the R1C1 format to represent the computations in a table.

A R1C1 formula can be divided into two parts: operators and input variables. Take the R1C1 formula SUM(RC[−3]:RC[−1]) in Figure 1a as an example. Its operators are "SUM", and its input variables are RC[−3], RC[−2] and RC[−1]. We denote the operators used by all formulas in a table t as a set OP_t, and input variables as a set Var_t. The similarity score sim_{F3} is calculated as follows:

$$sim_{F3} = \left(\frac{|OP_{t1} \cap OP_{t2}|}{|OP_{t1} \cup OP_{t2}|} + \frac{|Var_{t1} \cap Var_{t2}|}{|Var_{t1} \cup Var_{t2}|}\right) \times \frac{1}{2} \tag{2}$$

Our similarity computation for formulas can obtain high similarity scores for similar formulas. Take formulas in Figure 1a and Figure 1d as an example. The R1C1 formula of table in Figure 1a is SUM(RC[−3]:RC[−1]). The R1C1 formula of table in Figure 1d is SUM(RC[−2]:RC[−1]). We can see that they share the similar computation. First, they have the same operator "SUM". Second, formula SUM(RC[−3]:RC[−1]) has three input variables (i.e., RC[−3], RC[−2] and RC[−1]), and SUM(RC[−2]:RC[−1]) has two input variables (i.e., RC[−2] and RC[−1]). Their input variables are similar. According to the above similarity computation formulas, their similarity score is 5/6. We can see that, although they have different formulas in Figure 1a and Figure 1d, their similarity score for formulas is high.

Cell type (*Feature* #4): A table can contain *label*, *formula*, *data* and *empty* cells. These cell types can reflect the content structure of a table. By counting the proportion of the four cell types, we can understand the organizational structure of a table. For table clones, they usually share the similar cell type distributions. We use a key-frequency list of $< key, count >$ to represent the cell type distribution in a table, where key is a cell type, and $count$ shows the occurrence count of type key. We use L_t to denote the key-frequency list for cell types in a table t. The similarity score of cell types sim_{F4} is calculated as follows.

$$sim_{F4} = 1 - \frac{\sum_x |freq(L_{t1}, x) - freq(L_{t2}, x)|}{\sum_x |freq(L_{t1}, x) + freq(L_{t2}, x)|} \tag{3}$$

In this formula, $|freq(L_{t1}, x) - freq(L_{t2}, x)|$ denotes the frequency difference for a cell type x, and $freq(L_{t1}, x) + freq(L_{t2}, x)$ denotes the frequency summarization for a cell type x. Intuitively, more cells in two tables share with the same cell types, the smaller frequency difference each cell type has, the higher similarity score we can obtain.

3.2.2 Format Features. Format information is also the important feature to help table clone detection. We observe that two tables that share the similar formats are likely to be a table clone, for example, bold borders, background colors and font-styles. Inspired by this observation, we leverage various format features in our table clone prediction model. We extract the following eight format features for each cell: **font color (*Feature* #5)**, **font type (*Feature* #6)**, **font style (*Feature* #7)**, e.g., bold and italic, **bottom border (*Feature* #8)**, **top border (*Feature* #9)**, **left border (*Feature* #10)**, **right border (*Feature* #11)**, **background color (*Feature* #12)**. Take cell A2 in Figure 1a as an example, its font color is Black, its font

[2]Spreadsheet systems usually have two built-in formats to represent a cell reference: A1 and R1C1 formats. In the A1 format, a cell at the x-th column and y-th row is denoted as xy, e.g., D2. In the R1C1 format, a cell at m rows below and n columns right to the current cell is denoted as R[m]C[n].

type is Geneva, its font style is bold and italic, and it has right border and top border, etc.

For each format feature, we analyze all cells in a table, and build a key-frequency list of $< key, count >$, where key is a value for a feature, and *count* shows the occurrence count of key. Then, we use the similarity score calculation of the cell type feature to compute the similarity score of each format feature in two tables, as shown in Formula (3).

Note that, we do not consider a data cell's value as a feature. In spreadsheets, the values in a table have the similar role as the inputs of a function in programming languages. Thus, the values cannot usually reflect the computational semantics of a table. For two tables in a table clone, they usually store different values, e.g., Figure 1a and Figure 1d. Identical values are only associated with Type-1 clones, which can be effectively detected based on our current features. Including the feature of values may be helpful for identifying Type-1 clones. However, it may impact the accuracy for identifying other three types of clones. Therefore, we do not consider a data cell's value as a feature.

3.2.3 Default Settings for Similarity Scores. For all 12 features, their ranges of similarity scores are [0,1]. We observe that not all tables have these 12 features. We need to set default similarity scores. If both tables do not have certain feature, we set the default similarity score of this feature as θ_{both}. If one table does not have certain feature but the other does, we set the default similarity core of this feature as θ_{one}. According to our experiments in Section 4.2, when θ_{both} = 0.5 and θ_{one} = 0, LTC can obtain the best performance.

3.3 Training and Prediction

3.3.1 Training. To train a binary classifier for table clone detection, we need a group of table clones and non table clones as our training data. Since there is not such training data for table clone detection, we sample a group of spreadsheets from the EUSES [24] and Enron [26] corpora, which are the most widely used spreadsheet corpora. We further manually identify all table clones and non table clones in these spreadsheets. More details about training data can be found in Section 4.

For each table clone or non clone, we analyze its tables by using the Apache POI library [2], and extract features of each table. These features reflect the semantic characteristics of tables. For each feature, we compute the similarity score of two tables in a clone or non clone. Thus, we obtain a similarity vector of 12 similarity scores. Our training data can be denoted as a list of $< similarity\_vector, label >$, where $label$ is 1 for table clones, and 0 for non table clones.

There are many popular classifiers. We set the chosen classifier as parameter θ_{model}. According to our experiments (Section 4.2), when we use Random Forest algorithm [14], LTC can obtain the best performance. We use the open source machine learning library scikit-learn [1] to train Random Forest classifier, under its default setting.

3.3.2 Prediction. Given some spreadsheets, LTC first identifies all the tables in it. It further removes tables with only one row / column, since they contain very limited information and are hardly used for table clones. LTC then enumerates all possible table pairs, and uses the trained binary classifier to determine whether a table pair is a table clone or not.

4 EXPERIMENTAL DESIGN

In this study, we address the following research questions.

RQ1: *How effective is LTC in detecting table clones in spreadsheets?*

RQ2: *How is LTC compared with existing techniques, e.g., TableCheck?*

RQ3: *Is LTC robust on different datasets?*

To answer RQ1, we evaluate LTC with the built ground truth and analyze its performance. To answer RQ2, we evaluate LTC and TableCheck with the built ground truth to compare their performance. We further analyze the root causes of their performance difference. To answer RQ3, we evaluate LTC on some larger datasets, e.g., FUSE [10], EUSES [24] and Enron [26] and manually validate the detected table clones.

4.1 Dataset Construction

4.1.1 Experimental Subjects. We select the EUSES [24] and Enron [26] corpora as our experimental subjects. The EUSES corpus was collected from Internet in 2005, and consists of more than 4,000 real-life spreadsheets of 11 categories, e.g., financial, grades and modeling. Since its creation, it has been used by many spreadsheet researches [7, 18, 19, 30, 52]. The Enron corpus was collected from the Enron email archive within the Enron Corporation [38], and contains more than 15,000 spreadsheets. Note that, the spreadsheets in the Enron corpus were not categorized. These two corpora are the most commonly used public industrial spreadsheet datasets.

Table clones are not documented during spreadsheet development. To extract all table clones in spreadsheets, we need to inspect all table pairs in them. Because we are not the authors of these spreadsheets in EUSES and Enron, it is challenging to label table clones for all their spreadsheets. Therefore, we randomly sample a fixed number of spreadsheets from EUSES and Enron to manually build the ground truth. In our experiments, we finally obtain 100 spreadsheets, in which, 50 spreadsheets are from EUSES and 50 spreadsheets are from Enron.

4.1.2 Sampling Process. To facilitate the labelling process, we built an Excel plugin, in which participants can mark a region as a table, and two tables as a clone pair in a visual manner. Participants can further view and edit these labelled tables and clone pairs in the plugin. All labelled results were stored in a spreadsheet, which can be used in further analysis.

We build our ground truth through the following steps. (a) We randomly chose a spreadsheet from the corresponding corpus, i.e., EUSES and Enron. We inspected all the worksheets in the sampled spreadsheet for table clones. (b) EUSES and Enron contain multiple versions of a spreadsheet [22, 50], which have very similar structures and contents. To guarantee the diversity of spreadsheets in the ground truth, we excluded this spreadsheet, if it had almost the same structures and contents with any previously selected one. (c) A spreadsheet may contain multiple worksheets that have the same or similar structures. To guarantee the diversity of table clones, for the worksheets with the same or similar structures, we only randomly kept two of them. We also removed worksheets, which were empty or only contain figures. If no worksheet was left in a

Table 2: Statistics of the Ground Truth

Corpus	Category	Spreadsheet	Worksheet	Table clone					Non clone
				Type-1	Type-2	Type-3	Type-4	Total	
EUSES	cs101	1	1	0	0	0	1	1	0
	database	9	18	0	3	2	7	12	6
	financial	12	27	2	84	4	44	134	1,349
	forms3	1	2	0	6	4	4	14	239
	grades	4	7	0	7	0	2	9	9
	homework	4	6	7	11	1	8	27	397
	inventory	3	5	0	5	1	10	16	72
	jackson	3	9	6	9	4	12	31	596
	modeling	12	32	1	21	13	32	67	456
	personal	1	2	0	1	0	0	1	5
	Total	**50**	**109**	**16**	**147**	**29**	**120**	**312**	**3,129**
Enron		50	107	12	95	53	178	338	7,086
Total		**100**	**216**	**28**	**242**	**82**	**298**	**650**	**10,215**

Table 3: Experimental Design

	Corpus	Spreadsheet	Table clone					Non clone
			Type-1	Type-2	Type-3	Type-4	Total	
Training	EUSES	35	10	104	8	99	**221**	2635
	Enron	35	10	57	19	153	**239**	5487
	Total	**70**	**20**	**161**	**27**	**252**	**460**	**8122**
Testing	EUSES	15	6	43	21	21	**91**	
	Enron	15	2	38	34	25	**99**	
	Total	**30**	**8**	**81**	**55**	**46**	**190**	

spreadsheet, we removed it from consideration. (d) For all remaining worksheets in a spreadsheet, we manually identified all tables in them, and further identified table clones among them. Note that, we not only identify table clones in a worksheet but also among worksheets. For each table clone, we further identified its clone type according to the type categorization in Section 2.4. If a spreadsheet did not contain any table clone, we removed it from consideration. (e) We repeated the above sampling process until 50 spreadsheets for EUSES and Enron were selected, respectively.

Note that it is challenging to manually compare all tables among spreadsheets. Thus, we do not identify table clones among spreadsheets. Meanwhile, having too many similar spreadsheets and worksheets can degrade the performance of our classification model. For example, if a spreadsheet contains 10 similar worksheets and each worksheet contains one table, we can identify 10*9/2=45 table clones of the same family. These table clones can make our model bias to the large table clone families. To address the problem, we removed some spreadsheets and worksheets with the same or similar structures.

To make our sampled spreadsheets and table clones accurate, the first author and two master students carefully inspected the sampled spreadsheets, and tried their best to understand the structures and contents, and further identified tables and table clones. Note that, we only explained the concepts of table and table clones to the two master students, and did not tell them how LTC works. Finally, they carefully cross-validated all identified tables and table clones.

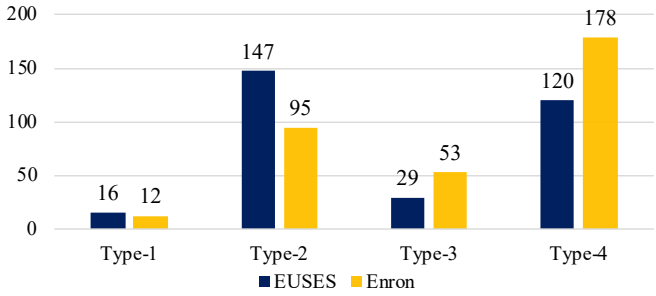

Figure 5: Table clone type distribution in the ground truth.

4.1.3 Statistics of Ground Truth. Through the above sampling process, we obtain 100 spreadsheets and identify 650 table clones.

Table clones: As shown in Table 2, the spreadsheets in our ground truth are diverse: 50 spreadsheets cover ten categories in EUSES, and 50 spreadsheets come from Enron. We can see that 82

Table 4: LTC Results on the Ground Truth

Corpus	Spreadsheet	Table clone	Detected	TP					FP	FN
				Type-1	Type-2	Type-3	Type-4	Total		
EUSES	15	91	86	5	41	19	19	**84**	2	7
Enron	15	99	93	2	36	32	21	**91**	2	8
Total	**30**	**190**	**179**	**7**	**77**	**51**	**40**	**175 (97.8%)**	**4 (2.2%)**	**15 (7.9%)**

(12.6%) and 298 (45.8%) table clones belong to Type-3 and Type-4, respectively. This indicates that table clones with structure changes are common (58.5%). Figure 5 further shows the distribution of table clone types in the ground truth.

Non table clones: To train a Random Forest model, we need a set of negative samples, i.e., non table clones. We first manually identify all tables in a spreadsheet in our ground truth. Then, we enumerate table pairs in each spreadsheet. If a table pair is a table clone in our ground truth, we remove it from consideration. All remaining table pairs are non table clones. As shown in Table 2, we obtain 10,215 non table clones.

4.2 Experimental Setting

Training dataset (70 spreadsheets): We randomly select 35 spreadsheets from EUSES and Enron in the ground truth, respectively. We use all 460 table clones in these 70 spreadsheets as positive samples, and all 8,122 non clones in these 70 spreadsheets as negative samples. The first part in Table 3 shows the statistics of spreadsheets used in the training.

Testing dataset (30 spreadsheets): We use the remaining 30 spreadsheets to evaluate LTC. The second part in Table 3 shows the statistics of spreadsheets used in the performance evaluation. We compare LTC with TableCheck on these 30 spreadsheets, too.

Parameter setting: In our approach, two default similarity scores θ_{both} and θ_{one} in Section 3.2.3 should be determined. Their candidate values can be 0, 0.5, and 1. The binary classifier model θ_{model} in Section 3.3 should also be determined. The candidate values for θ_{model} is Random Forest [14], SVM [34], NuSVC [37], Decision Tree [47] and Logistic Regression [41].

We consider all candidate values of the three parameters, and there are 45 (3*3*5) combinations in total. We use 10-fold cross-validation in our training for each combination and calculate the following accuracy indicators: precision, recall and F1-measure. According to the experiment result, we obtain the following best candidates: $\theta_{both} = 0.5$, $\theta_{one} = 0$, and θ_{model} = Random Forest.

5 EXPERIMENTAL RESULTS

5.1 Table Clone Detection Results

We run LTC on the 30 spreadsheets in the testing set of Table 3. Table 4 shows our table clone detection results. It gives the numbers of detected table clones (Detected), and true table clones of each type (TP/Type-1, 2, 3, 4) for the EUSES and Enron spreadsheets.

LTC reports 179 table clones in total, and 175 (97.8%) table clones are true. LTC misses 15 table clones in the ground truth, i.e., the recall for table clone detection is 92.1%. Thus, F1-measure of LTC for table clone detection is 94.9%.

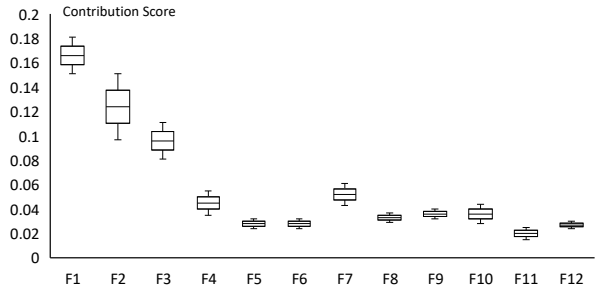

Figure 6: Feature analysis. F1 to F12 correspond to the features in Section 3.2.

False positives of table clone detection: LTC wrongly detects 4 table clones (FP in Table 4). We further investigate the causes for these false positives. There are two reasons for these false positives. (1) Our table identification algorithm in Section 3.1 is imprecise. First, end users may put some unrelated data and comments near a table, and our table identification algorithm wrongly considers these data and comments as a part of the table. Second, end users may put multiple tables together, without being separated by empty rows / columns. Thus, our table identification algorithm wrongly considers them as a big table. Imprecise table identification causes three false positives. A more precise table identification approach can help improve LTC's precision. (2) The learned classifier for table clone detection may make wrong predication. In some cases, the table pairs may have some similar features in cell fonts and cell types, and the learned classifier wrongly considers them are table clones. One false positive belongs to this case.

False negatives of table clone detection: LTC misses 15 table clones (FN in Table 4). The reasons for the missed table clones are the same as the false positives of table clone detection. First, four false negatives are caused by imprecise table identification algorithm, which does not identify the tables in the ground truth. Second, the learned classifier wrongly considers table clones as non table clones. Eleven false negatives belong to this case.

Feature analysis: We use *ReleifFAttributeEval* and *Ranker* evaluator provided by Weka [5], to evaluate the importance of 12 features in table clone detection. We adopt 10-fold cross validation approach to do feature analysis in Weka, and the result is shown in Figure 6. We can see that our 12 features all contribute to the table clone detection. According to the contribution of each feature, the rank of

Table 5: TableCheck Results on the Ground Truth

Corpus	Spreadsheet	Table clone	Detected	TP					FP	FN
				Type-1	Type-2	Type-3	Type-4	Total		
EUSES	15	91	30	5	9	0	0	**14**	16	77
Enron	15	99	26	2	5	0	0	**7**	19	92
Total	**30**	**190**	**56**	**7**	**14**	**0**	**0**	**21 (37.5%)**	**35 (62.5%)**	**169 (88.9%)**

contributions are as follows (from high to low): column header, row header, formula, font bold, cell type, border top, font color, border bottom, border left, background color, border right and font type.

Based on the above analyses, we can draw the following conclusion to RQ1: *LTC is effective in detecting table clones in spreadsheets. The precision and recall for LTC is 97.8% and 92.1%, respectively.*

5.2 Comparison with TableCheck

We compare LTC with TableCheck [18] on their table clone detection capability. Note that, we do not compare LTC with data clone detection [30], since a data clone is a pair of two numerical cell blocks with (almost) the same values, which differs from table clones.

TableCheck can only detect table clones with the same headers, e.g., Type-1 and Type-2 table clones, and cannot detect Type-3 and Type-4 table clones. TableCheck extracts table clones by grouping cells with the same headers. First, TableCheck extracts row headers and column headers for each numerical cell. Different from LTC, TableCheck uses the nearest *label* cells as a cell's row or column headers. If a cell does not have row or column headers, TableCheck excludes it from consideration. Second, TableCheck searches for two cell blocks, in which all corresponding cells share the same row and column headers. The *table* definition in TableCheck is different from the common *table* concept used in existing studies [15, 17]. For example, cells January![A1:F5] in Figure 1a are considered as a table. However, they are only a partition of table January![A1:F7].

We run TableCheck on the same 30 spreadsheets in the testing set of Table 3, and compare LTC and TableCheck. Since the tables detected by TableCheck do not contain headers, we append their headers to the corresponding tables for fair comparison.

Table 5 shows the results detected by TableCheck on the 30 spreadsheets. TableCheck detects 56 table clones, and 21 of them are true. Thus, TableCheck achieves a precision of 37.5%, recall of 11.1%, and F1-measure of 17.1%. Figure 7 shows the performance comparison between LTC and TableCheck. As a comparison, LTC achieves a precision of 97.8%, recall of 92.1%, and F1-measure of 94.9%. We further analyze why TableCheck performs worse than LTC on these 30 spreadsheets.

False positives of TableCheck: TableCheck wrongly detects 35 table clones (FP in Table 5) on account of the incorrect identification of tables. For example, in Figure 1, January![A1:F7] and March![A1:F7] form a Type-3 table clone. However, TableCheck considers January![A1:F5] and March![A1:F5] as a table clone. Because the authors of TableCheck view these incomplete table clones as true, the reported precision (92.2%) in that paper is higher [18].

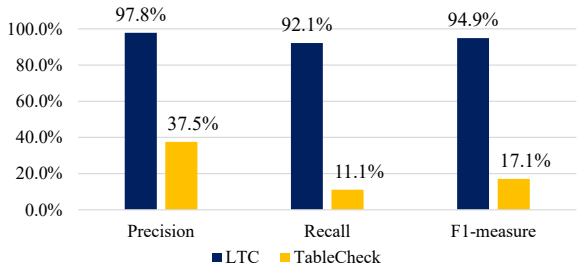

Figure 7: Performance comparison of LTC and TableCheck.

While, LTC first identifies tables in a spreadsheet, and further checks whether a table pair can form a clone. Tables in the reported clones by LTC are usually complete.

False negatives of TableCheck: TableCheck misses 169 table clones (FN in Table 5). TableCheck misses 68 Type-1 and Type-2 table clones. There are two reasons for these false negatives. First, some cells in Type-1 and Type-2 table clones do not have row headers or columns headers, and thus the table clones are excluded by TableCheck. Second, the header identification algorithm in TableCheck does not identify headers correctly, thus missing table clones. TableCheck misses all 101 Type-3 and Type-4 table clones, since TableCheck requires that table clones have the same headers and sizes. LTC utilizes 12 features in spreadsheets, and learning-based similarity comparison approach to avoid these issues faced by TableCheck. Thus, LTC is more universal, and has less restrictions on table structures.

Based on the above analyses, we can draw the following conclusion to RQ2: *LTC significantly outperforms TableCheck in table clone detection. The precision and recall for TableCheck is 37.5% and 11.1%. As a comparison, the precision and recall for LTC is 97.8% and 92.1%, respectively.*

5.3 Experiments on Large Datasets

To validate whether LTC is robust on other datasets, we further use our trained model in RQ1 & RQ2 to evaluate LTC on more spreadsheets from FUSE [10], Enron [26] and EUSES [24] corpora.

Experimental subject: FUSE [10] is the biggest spreadsheet corpus so far, and contains about 250,000 spreadsheets. In this experiment, we randomly choose 100 spreadsheets from EUSES, Enron and FUSE, respectively.

We apply LTC to detect intra-spreadsheet table clones and inter-spreadsheet table clones on these spreadsheets. Table 6 shows the

Table 6: Detection Results on Large Datasets

	Spreadsheet		Intra-spreadsheet clone					Inter-spreadsheet clone				
Corpus	**Total**	**Sampled**	**SS**	**Clone**	**Sampled**	**TP**	**Precison**	**SS**	**Clone**	**Sampled**	**TP**	**Precision**
EUSES	4,037	100	65	2,576	100	97	97.0%	43	1,156	100	86	86.0%
Enron	15,926	100	32	289	100	93	93.0%	62	1,332	100	94	94.0%
FUSE	249,376	100	50	1,397	100	98	98.0%	53	2,647	100	97	97.0%
Total	**269,339**	**300**	**147**	**4,262**	**300**	**288**	**96.0%**	**158**	**5,135**	**300**	**277**	**92.3%**

detection result. We detect 4,262 intra-spreadsheet table clones (Intra-spreadsheet clone / Clone), and 5,135 inter-spreadsheet table clones (Inter-spreadsheet clone / Clone) in these spreadsheets. Among these 300 sampled spreadsheets, 147 (49%) spreadsheets contain intra-spreadsheet clones (Intra-spreadsheet clone / SS), and 158 (53%) spreadsheets contain inter-spreadsheet clones (Inter-spreadsheet clone / SS). This also indicates that intra- and inter-spreadsheet table clones are common in practice.

It is challenging to manually validate all detected table clones. Thus, we randomly sample 100 inter-spreadsheet table clones and 100 intra-spreadsheet table clones for each corpus. We follow the same manual inspection procedure described in Section 4.1.2 to check whether they are true table clones. Table 6 shows the validation result. We can see that, LTC can work well on inter- and intra- spreadsheet table clone detection (Precision). LTC can achieve higher (96.0%) precision on intra-spreadsheet table clone detection. The precision of inter-spreadsheet table clone detection is 92.3%, which is also promising. Thus, LTC can also be used to detect inter-spreadsheet table clones.

Based on the above analyses, we can draw the following conclusion to RQ3: *Table clones are common in real-world spreadsheets. LTC can detect inter- and intra-spreadsheet table clones precisely.*

6 THREATS TO VALIDITY

Our experiments are subject to several threats to validity.

Representativeness of experimental subjects. In our experiment, we select EUSES, Enron and FUSE as our experimental subjects, which have been widely used for many spreadsheet-related studies [6, 18, 19, 25, 30]. We further randomly sample a group of spreadsheets from EUSES and Enron. We believe that our sampled spreadsheets are representative.

Ground truth and detection result validation. Since we cannot contact the original authors of the spreadsheets in EUSES and Enron to identify table clones, we have to manually build the ground truth of table clones by ourselves. To alleviate possible mistakes, the first author and two master students carefully cross-validate all table clones in the ground truth and the reported table clones by LTC and TableCheck. In the future, we would like to use crowdsourcing to build larger ground truth to improve LTC further.

7 RELATED WORK

Here, we discuss related work that have not been discussed yet.

Code clone detection. Code clone detection techniques have been widely studied in conventional programs. There are mainly five types of clone detection techniques [13]. (1) Text-based clone detection techniques [23] use line-based string matching to detect clones. (2) Token-based clone detection techniques [9, 36, 43] tokenize program code, and use token comparison to detect clones. (3) Tree-based clone detection techniques [12] parse programs into an abstract syntax tree, and then detect clones by tree matching. (4) Graph-based clone detection techniques [40] parse the programs into program dependence graphs, and detect clones as identifying isomorphic subgraphs. (5) Learning-based clone detection techniques [42, 49] adopt machine learning techniques to detect clones. The above code clone detection techniques cannot apply on spreadsheets, since spreadsheets use a different programming model.

Spreadsheet evolution. Spreadsheets usually have a long life cycle [27]. Based on spreadsheet filenames' similarity, VEnron [22] constructs the first versioned spreadsheet corpus, which can be used for spreadsheet evolution studies. SpreadCluster [50] further adopts machine learning to find versioned spreadsheets. However, these corpora and approaches cannot be used for table clone detection in spreadsheets. Our table clone detection approach can be used to facilitate fine-grained spreadsheet reuse analyses.

Spreadsheet error detection. Spreadsheets contain various errors [45, 46]. Therefore, many spreadsheet error detection approaches have been proposed. UCheck [7] can detect type inconsistency errors in formulas. Hermans et al. proposed to detect inter-worksheet smells [28], data clone related inconsistencies [30]. AmCheck [19], CACheck [21], CUSTODES [16] and ExcelLint [11] detect errors in similar cells. TableCheck [18] can detect inconsistencies in Type-1 and Type-2 table clones. LTC can detect more table clones (e.g., Type-3 and Type-4) precisely, and makes it possible to detect errors in more types of table clones.

8 CONCLUSION

Table clones are widely used to perform similar business tasks in spreadsheets. We observe that the tables in a table clone usually share the similar structures and formats. Based on this observation, we propose a learning-based approach, LTC, to detect table clones with or without structure changes. Our experiments on real-world spreadsheets from the EUSES and Enron corpora show that, LTC can detect table clones effectively, and significantly outperforms existing table clone detection techniques. In the future, we plan to pursue the following research directions. First, we plan to detect inconsistency errors among table clones detected by LTC. Second, we plan to extract table templates based on table clones, and use them in spreadsheet development. Third, we plan to evaluate LTC's practical effectiveness in companies, and explore new application scenarios of table clones.

ACKNOWLEDGEMENTS

We thank Shuo Wang and Jiahong Zhou for their contributions in developing the Excel plugin for labelling. This work was partially supported by National Key National Key Research and Development Program of China (2017YFB1001804), National Natural Science Foundation of China (61702490), Microsoft Research Asia Collaborative Research Program, Frontier Science Project of Chinese Academy of Sciences (QYZDJ-SSW-JSC036), Youth Innovation Promotion Association at Chinese Academy of Sciences, and Beijing College Students' Research Project of High-Level Cross Cultivation of Undergraduate. Wensheng Dou is the corresponding author of this paper.

REFERENCES

[1] 2007. *scikit-learn: Machine learning in Python.* Retrieved Jan 15, 2020 from https://scikit-learn.org
[2] 2020. *Apache POI - the Java API for Microsoft Documents.* Retrieved Jan 15, 2020 from https://poi.apache.org/
[3] 2020. *Ideas in Excel.* Retrieved January 15,2020 from https://support.office.com/en-ie/article/ideas-in-excel-3223aab8-f543-4fda-85ed-76bb0295ffc4
[4] 2020. *Power BI | Interactive Data Visualization BI Tools.* Retrieved Jan 15, 2020 from https://powerbi.microsoft.com
[5] 2020. *Weka 3: Machine Learning Software in Java.* Retrieved Jan 15, 2020 from http://www.cs.waikato.ac.nz/ml/weka
[6] Robin Abraham and Martin Erwig. 2004. Header and unit inference for spreadsheets through spatial analyses. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC).* 165–172.
[7] Robin Abraham and Martin Erwig. 2007. UCheck: A spreadsheet type checker for end users. *Journal of Visual Languages and Computing* 18, 1 (2007), 71–95.
[8] Robin Abraham, Martin Erwig, Steve Kollmansberger, and Ethan Seifert. 2005. Visual specifications of correct spreadsheets. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC).* 189–196.
[9] Brenda S Baker. 1995. On finding duplication and near-duplication in large software systems. In *Proceedings of Working Conference on Reverse Engineering (WCRE).* 86–95.
[10] Titus Barik, Kevin Lubick, Justin Smith, John Slankas, and Emerson Murphy-Hill. 2015. Fuse: A reproducible, extendable, internet-scale corpus of spreadsheets. In *Proceedings of Working Conference on Mining Software Repositories (MSR).* 486–489.
[11] Daniel W. Barowy, Emery D. Berger, and Benjamin Zorn. 2018. ExceLint: Automatically finding spreadsheet formula errors. In *Proceedings of International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA).* 148:1–148:26.
[12] Ira D Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. 1998. Clone detection using abstract syntax trees. In *Proceedings of International Conference on Software Maintenance (ICSM).* 368–377.
[13] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. 2007. Comparison and evaluation of clone detection tools. *IEEE Transactions on software engineering (TSE)* 33, 9 (2007), 577–591.
[14] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
[15] Zhe Chen and Michael Cafarella. 2014. Integrating spreadsheet data via accurate and low-effort extraction. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).* 1126–1135.
[16] Shing-Chi Cheung, Wanjun Chen, Yepang Liu, and Chang Xu. 2016. CUSTODES: Automatic spreadsheet cell clustering and smell detection using strong and weak features. In *Proceedings of International Conference on Software Engineering (ICSE).* 464–475.
[17] Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. 2019. TableSense: Spreadsheet table detection with convolutional neural networks. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI).* 69–76.
[18] Wensheng Dou, Shing-Chi Cheung, Chushu Gao, Chang Xu, Liang Xu, and Jun Wei. 2016. Detecting table clones and smells in spreadsheets. In *Proceedings of ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE).* 787–798.
[19] Wensheng Dou, Shing-Chi Cheung, and Jun Wei. 2014. Is spreadsheet ambiguity harmful? Detecting and repairing spreadsheet smells due to ambiguous computation. In *Proceedings of International Conference on Software Engineering (ICSE).* 848–858.
[20] Wensheng Dou, Shi Han, Liang Xu, Dongmei Zhang, and Jun Wei. 2018. Expandable group identification in spreadsheets. In *Proceedings of International Conference on Automated Software Engineering (ASE).* 498–508.
[21] Wensheng Dou, Chang Xu, S. C. Cheung, and Jun Wei. 2017. CACheck: Detecting and repairing cell arrays in spreadsheets. *IEEE Transactions on software Engineering (TSE)* 43, 3 (2017), 226–251.
[22] Wensheng Dou, Liang Xu, Shing-Chi Cheung, Chushu Gao, Jun Wei, and Tao Huang. 2016. VEnron: A versioned spreadsheet corpus and related evolution analysis. In *Proceedings of International Conference on Software Engineering (ICSE).* 162–171.
[23] Stéphane Ducasse, Oscar Nierstrasz, and Matthias Rieger. 2004. Lightweight detection of duplicated code - A language-independent approach. *Institute for Applied Mathematics and Computer Science, University of Berne* (2004).
[24] Marc Fisher and Gregg Rothermel. 2005. The EUSES spreadsheet corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. 30, 4 (2005), 1–5.
[25] Felienne Hermans, Bas Jansen, Sohon Roy, Efthimia Aivaloglou, Alaaeddin Swidan, and David Hoepelman. 2016. Spreadsheets are code: An overview of software engineering approaches applied to spreadsheets. In *Proceedings of International Conference on Software Analysis, Evolution, and Reengineering (SANER).* 56–65.
[26] Felienne Hermans and Emerson Murphy-Hill. 2015. Enron's spreadsheets and related emails: A dataset and analysis. In *Proceedings of International Conference on Software Engineering (ICSE)*, Vol. 2. 7–16.
[27] Felienne Hermans, Martin Pinzger, and Arie van Deursen. 2011. Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proceedings of International Conference on Software Engineering (ICSE).* 451–460.
[28] Felienne Hermans, Martin Pinzger, and Arie van Deursen. 2012. Detecting and visualizing inter-worksheet smells in spreadsheets. In *Proceedings of International Conference on Software Engineering (ICSE).* 441–451.
[29] Felienne Hermans, Martin Pinzger, and Arie Van Deursen. 2010. Automatically extracting class diagrams from spreadsheets. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP).* 52–75.
[30] Felienne Hermans, Ben Sedee, Martin Pinzger, and Arie van Deursen. 2013. Data clone detection and visualization in spreadsheets. In *Proceedings of International Conference on Software Engineering (ICSE).* 292–301.
[31] Felienne Hermans and Tijs van der Storm. 2015. Copy-paste tracking: Fixing spreadsheets without breaking them. In *Proceedings of International Conference on Live Coding (ICLC).*
[32] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. 2007. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of International Conference on Software Engineering (ICSE).* 96–105.
[33] Lingxiao Jiang, Zhendong Su, and Edwin Chiu. 2007. Context-based detection of clone-related bugs. In *Proceedings of Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE).* 55–64.
[34] Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of European Conference on Machine Learning (ECML).* 137–142.
[35] Elmar Juergens, Florian Deissenboeck, and Benjamin Hummel. 2009. CloneDetective - A workbench for clone detection research. In *Proceedings of International Conference on Software Engineering (ICSE).* 603–606.
[36] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. 2002. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering (TSE)* 28, 7 (2002), 654–670.
[37] Zaheer Ullah Khan, Maqsood Hayat, and Muazzam Ali Khan. 2015. Discrimination of acidic and alkaline enzyme using Chou's pseudo amino acid composition in conjunction with probabilistic neural network model. *Journal of Theoretical Biology* 365 (2015), 197–203.
[38] Bryan Klimt and Yiming Yang. 2004. The Enron corpus: A new dataset for email classification research. In *Proceedings of European Conference on Machine Learning (ECML).* 217–226.
[39] Elvis Koci, Maik Thiele, Óscar Romero Moral, and Wolfgang Lehner. 2016. A machine learning approach for layout inference in spreadsheets. In *Proceedings of International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management.* 77–88.
[40] Jens Krinke. 2001. Identifying similar code with program dependence graphs. In *Proceedings of Working Conference on Reverse Engineering (WCRE).* 1095–1350.
[41] S Lee. 2005. Application of logistic regression model and its validation for landslide susceptibility mapping using GIS and remote sensing data. *International Journal of Remote Sensing* 26, 7 (2005), 1477–1491.
[42] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. 2017. CCLearner: A deep learning-based clone detection approach. In *Proceedings of International Conference on Software Maintenance and Evolution (ICSME).* 249–260.
[43] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. 2006. CP-Miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on software Engineering (TSE)* 32, 3 (2006), 176–192.
[44] Ephraim R McLean, Leon A Kappelman, and John P Thompson. 1993. Converging end-user and corporate computing. *Commun. ACM* 36, 12 (1993), 78–90.
[45] Raymond R Panko. 2008. Spreadsheet errors: What we know. What we think we can do. *arXiv preprint arXiv:0802.3457* (2008).

[46] Stephen G. Powell, Kenneth R. Baker, and Barry Lawson. 2008. A critical review of the literature on spreadsheet errors. 46, 1 (2008), 128–138.
[47] S Rasoul Safavian and David Landgrebe. 1991. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics* 21, 3 (1991), 660–674.
[48] Christopher Scaffidi, Mary Shaw, and Brad Myers. 2005. Estimating the numbers of end users and end user programmers. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 207–214.
[49] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *Proceedings of International Conference on Automated Software Engineering (ASE)*. 87–98.
[50] Liang Xu, Wensheng Dou, Chushu Gao, Jie Wang, Jun Wei, Hua Zhong, and Tao Huang. 2017. SpreadCluster: Recovering versioned spreadsheets through similarity-based clustering. In *Proceedings of International Conference on Mining Software Repositories (MSR)*. 158–169.
[51] Liang Xu, Wensheng Dou, Jiaxin Zhu, Chushu Gao, Jun Wei, and Tao Huang. 2018. How are spreadsheet templates ssed in practice: A case study on Enron. In *Proceedings of ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 734–738.
[52] Liang Xu, Shuo Wang, Wensheng Dou, Bo Yang, Chushu Gao, Jun Wei, and Tao Huang. 2018. Detecting faulty empty cells in spreadsheets. In *Proceedings of International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 423–433.