LIFT: Last-Mile Fine-Tuning for Table Explicitation

Divij Khaitan
Microsoft Corporation
Bangalore, India
{t-dkhaitan}@microsoft.com

Ashish Tiwari
Microsoft Corporation
Redmond, WA 98052 USA
{ashish.tiwari}@microsoft.com

Abstract

We propose last-mile fine-tuning, or LIFT, a pipeline in which a pre-trained large language model extracts an initial table from unstructured clipboard text, and a fine-tuned small language model (1B–24B parameters SLM) repairs errors in the extracted table. On a benchmark of 2,596 tables from three datasets, LIFT matches or exceeds end-to-end SLM fine-tuning on tree-edit-distance-based similarity (TEDS) metric while requiring as little as 1,000 training examples — where it outperforms end-to-end fine-tuning by up to 0.144 TEDS points. We term this approach last-mile fine-tuning and show it also more robust to input format variability. Comparisons with self-debug and end-to-end fine-tuning approaches show that last-mile fine-tuning provides an attractive option when training data is limited or when robustness to input variation is sought without compromising on accuracy.

1 Introduction

Tables are a convenient means for structuring large amounts of data to make it interpretable to humans. Having data in tabular format enables analyses of the data using table-processing software, such as Microsoft Excel. However, very often tables are shared using PDF documents, such as, in academic papers or financial reports. We envision a copy-paste paradigm for extracting tables from PDF documents; that is, a user selects the table (or a part of a table) of interest, copies it, and then pastes it *as a table*. Apart from its simplicity and ease of use, the copy-paste paradigm also has other benefits: it gives control to the user on what they want to extract as a table. Furthermore, it also eliminates any requirement to share or upload the full PDF document to extract tables.

When a user copies a table from a PDF document, the clipboard is populated with some unstructured text containing the data in the table and almost negligible information about the table structure. Extracting a table from this unstructured text is a problem that has not been extensively studied, but there is some very recent work (Mehrotra et al., 2025), which serves as our inspiration.

Extracting tables from PDF documents is challenging. A PDF document may be image-based (scanned PDFs) or text-based. Image-based PDFs require techniques such as optical character recognition (OCR) and are out-of-scope for us since it is not even possible to *select* regions from such a document to copy. When PDFs are text-based, it is possible to select regions and copy a table, or parts of a table, from it. What shows up in clipboard depends on the PDF reader being used by the user. For example, Acrobat readers can put content in rich-text format (RTF) in the clipboard in some cases. We do not make any assumptions on the reader the user is using, and hence work with clipboard contents dumped by the most basic PDF readers, where the clipboard just contains the textual data in the selected region without much information about the table topology, as in Mehrotra et al. (2025).

When considering table extraction from PDFs, a common paradigm is where the user uploads the PDF to a server and the software extracts tables while having access to the full PDF document. This is typically done using symbolic heuristics (Aristarán et al., 2012; Adobe, 2026). The quality of output is better when the software performs native PDF

(a) Original PDF

(b) Clipboard Text

(c) Predicted Table

(d) Repaired Table

Figure 1: Illustrating last-mile repair on table explicitation problem. The user selects a table from a PDF document (Figure (a)) and copies it. The text copied into the clipboard is shown in Figure (b). A baseline method (symbolic or using an LLM) is used to extract table from this text, but it generates a wrong table (Figure (c)). The table is repaired using an SLM fine-tuned for last-mile repair to give us back the original table (Figure (d)).

structural analysis, which is possible for certain text-based PDFs and when one has access to code that implements the PDF specification (Adobe, 2026). When that is not the case, then table extraction is not accurate and almost always require the human to repair the generated table (Aristarán et al., 2012).

Large Language Models (LLM) have shown proficiency on several text-based tasks, such as document summarization and sentiment analysis, even when they are not explicitly trained for those tasks (Chowdhery et al., 2023; Zhang et al., 2022; 2024a). Such emergent behaviors in LLMs through in-context learning have turned LLMs into general-purpose transformers, though they are noisy and probabilistic in nature. It is, thus, natural to apply a LLM to extract tables from PDF documents. Again, several LLM-based agents provide the option of uploading a PDF document and then they can extract tables when prompted to do so.

Our interest is in extracting tables from the clipboard text generated when a user copies a selected region from a text-based PDF document. This is more difficult problem than extracting tables from PDF since we do not have access to the document here. Additionally, this is a more general problem and potentially more widely useful because it makes fewer assumptions about what we start with. In fact it can also potentially apply when the starting text is not necessarily from a PDF document, but maybe from a text email or an application log. We shall follow the lead of Mehrotra et al. (2025) and call this the *table explicitation problem*.

A good baseline is to use pre-trained LLMs to perform table explicitation using in-context learning. This is also the first step of our last-mile fine-tuning approach. After this first step, there are two options. One option would be to do an iterative loop, in self-debug style, to fix any errors in the table generated in the previous iteration (Mehrotra et al., 2025). In this paper, we explore the second option where we repair the generated table using a small language model (SLM) specifically fine-tuned for the repair task. Small Language Models (SLM) have shown the capacity to display comparable performance to LLMs through fine-tuning on specific tasks, while offering significant benefits in terms of performance and cost of inference (Ayala et al., 2025; Gu et al., 2024). Apart from the self-debug and last-mile fine-tuning approaches, there is another option of fine-tuning an SLM for the end-to-end table explicitation task. We compare these approaches on a table explicitation benchmark that we built for this purpose.

1.1 Overview

Let us consider a concrete example of the table explicitation problem. Figure 1 illustrates the full workflow of last-mile fine-tuning approach. Imagine here a user working on a PDF

document that contains a table shown in Figure 1(a). Say the user wants to copy this table to a table processing application. The user selects the table and copies it. Our task is now to paste the copied content as a tabular structure. Unfortunately, the clipboard often only contains the raw text containing the contents of the table, but no structural information. This text is shown in Figure 1(b). Turning this text into a structured table is the *table explicitation* problem. There are several approaches here. The last-mile fine-tuning approach, or LIFT in short, has two steps. In the first step, we attempt to solve the problem using some baseline method, which can be a heuristic rule-based method or an LLM-based approach. Since the source text has very little structural information, what we get after the first step is a "broken" table, such as the one shown in Figure 1(c). In the second step, this table is repaired. We fine-tune a small language model (SLM) for just this last-mile repair task. The fine-tuned SLM is used in the second step to repair the broken table and finally yield the correct table, shown in Figure 1(d).

1.2 Related Work

Related work can be divided into two parts: work related to last-mile fine-tuning and work related to table explicitation.

Related to last-mile fine-tuning, a similar two-stage approach has been proposed for code generation in low-resource languages. Specifically, Bavishi et al. (2022) proposed fixing an LLM-generated broken program using a neurosymbolic approach (but without any fine-tuning). In subsequent work, the repair phase was enhanced with program repair techniques implemented using LLMs (Joshi et al., 2023). Fine-tuning for the repair task, which is central to LIFT, appears in some recent work (Fu et al., 2025), but again it was for code repair. Based on similar intuitions as LIFT, the work Plug-in and Fine-tune (Kim et al., 2025) proposes an architecture where a frozen LLM is embedded in a tunable SLM and the combination is fine-tuned for end-to-end Natural Language understanding and generation tasks. Although it is end-to-end fine-tuning, the work (Kim et al., 2025) has flavors of LIFT and shows good generalization capabilities. We apply LIFT to table explicitation and, unlike earlier works, we identify trade-offs between the three approaches, self-debug, LIFT, and end-to-end fine-tuning. We observe that all three approaches are viable options depending on the application constraints. Furthermore, in conjunction with the work on LIFT for code generation, our work also establishes that LIFT is a general principle that applies to multiple tasks.

Related to table explicitation, recent work (Mehrotra et al., 2025) uses a self-debug loop, supported with a symbolic checker for this problem. A different approach based on end-to-end fine-tuning has also been used for several table-related tasks (not table explicitation) (Xing et al., 2025). Table extraction has been extensively explored from various different sources. Extracting tables from images is one popular field of study, where the table extraction task is further subdivided into two subtasks:

- Table Structure Recognition (TSR), which refers to the problem of extracting the structural layout of a table from an image, usually in the form of a markup language or some simplified version of it, without extracting the content of individual cells. TableNet Paliwal et al. (2019) uses an encoder backbone from VGG-19, with two separate decoders to extract the main tex segments and the different column segments respectively. Zhang et al. (2024b) introduces a dataset of real-world images for the TSR task, and proposes a method for detection based on instance segmentation using CNNs and spliting-then-merging. GNNs (Li et al., 2021), transformers (Yang et al., 2022) and VLMS(Chen et al., 2023) have also been used for this task.
- Table Detection, which refers to the problem of identifying a bounding box around a table from an image. This is often the first step in a larger pipeline for structural recognition or extraction, such as Prasad et al. (2020).

Tabularis Revilio (Singh et al., 2024) puts forth a three step pipeline for the table explicitation task, starting with a dedicated header detection module, then using an LLM to generate a sketch of the table given the header and raw text and finally using a symbolic extractor on the raw text, guided by the generated sketch to rank candidates. Our work could be applied

as a post-processor to this method as well. Wu et al. (2022) introduced the *text-to-table* task, which is the extraction of data from a long piece of text into a tabular format. It is expressed as the inverse problem of the table-to-text, where a language model is asked to generate descriptions of information presented in the table, usually in response to a query. Deng et al. (2024) proposes a multi step approach, that extracts structured tuples from a dataset and postprocesses those tuples into a table. Our work differs from these in that rather than being a given a long piece of natural language text, the clipboard text we start with just contains contents of the table. The starting text in text-to-table has more context and also more irrelevant information. Table explicitation is fundamentally different from the text-to-table task.

1.3 Contributions

We make three contributions. First, we show that replacing the iterative LLM self-debug loop of Mehrotra et al. (2025) with a single pass through a fine-tuned SLM achieves comparable or superior accuracy at significantly lower inference cost (one LLM call + one SLM call vs. two LLM calls). Second, we demonstrate that LIFT requires less training data than end-to-end fine-tuning: with only 1,000 examples, LIFT with Mistral-7B achieves 0.875 tree similarity score versus 0.731 for end-to-end fine-tuning. Third, we show LIFT is more robust to out-of-distribution input formats than end-to-end fine-tuning, inheriting the LLM's ability to generalize to unseen formats. While our experiments focus on table explicitation, the LIFT framework is task-agnostic: any task where an LLM produces a noisy initial output that can be systematically repaired is a candidate. Recent work on code repair using small models supports this hypothesis, though direct empirical validation on other tasks remains future work. For such tasks, our work shows that LIFT is a promising approach to use when training data is scarce and when resiliency is desired.

2 Last-Mile Fine-Tuning

Consider the task of transforming an input $x \in X$ to an output $y \in Y$ such that some requirement $\Phi(x, y)$ holds between the input-output pair (x, y). Due to their ability for in-context learning, a pre-trained LLM can be prompted to solve this task to give us a function f that predicts an output $\hat{y} = f(x)$ for input x. If $\Phi(x, \hat{y})$ evaluates to `True`, then we can return the prediction $\hat{y}$. However, if $\Phi(x, \hat{y})$ is `False`, then the output $\hat{y}$ needs to be *repaired*. The *last-mile repair task* is the problem of transforming the predicted output $\hat{y}$ to the output y such that $\Phi(x, y)$ holds.

The last-mile repair task can again be solved by suitably prompting a pre-trained LLM. This approach is usually called the *self-debug* approach. Let f' denote the repair function obtained by prompting an LLM to generate y given x and $\hat{y}$. The self-debug approach returns $\hat{y} = f'(x, f(x))$ given input x. An alternate to self-debug is the *last-mile fine-tuning*, or LIFT, approach where a SLM is fine-tuned on the task of predicting y given x and $\hat{y}$ task. Let f_s denote the function computed by the fine-tuned SLM. Thus, the last-mile fine-tuning approach returns $f_s(x, f(x))$ given the input x. We assume only one iteration of repair here, but the repair function can be applied multiple times guided by the check Φ in both approaches.

The probability, $Pr_{\text{LIFT}}(y \mid x)$, of generating y given x in the LIFT-approach can be computed as follows:

$$Pr_{\text{LIFT}}(y \mid x) = \sum_{\hat{y}} Pr_{LLM}(\hat{y} \mid x) Pr_{SLM}(y \mid x, \hat{y})$$

This probability will be higher than $Pr_{LLM}(y \mid x)$ if the repair module has high probability of correctly repairing some incorrect $\hat{y}$ answers.

A third approach here is the end-to-end fine-tuning approach, or EEFT, where an SLM is fine-tuned for the task of predicting y given x. If f_{sf} is the function computed by the end-to-end fine-tuned SLM, then the EEFT approach return $f_{sf}(x)$ given the input x.

3 Table Explicitation Problem

The goal of table explicitation is to transform an unstructured or semi-structured input string x, which is likely lacking explicit row/column delimiters, into a string y that is a structured representation of a table. In our experiments, the output y will be a string representing a table in HTML. There are several possible sources for x, but our immediate interest is in the text string x that is stored in the clipboard when a user copies a table from any document, typically a PDF document. The clipboard data has no structural information of the table and this makes the task of reliably retrieving the correct table quite challenging.

For example, as shown in Figure 1(b), a copied table text loses all information about structure of the source table (in Figure 1(a). The goal of the table explicitation problem is to start with the text in Figure 1(b) and regenerate a representation of the original table in Figure 1(a).

We solve the table explicitation problem using all three approaches:
(a) self-debug,
(b) end-to-end fine-tuning, and
(c) last-mile fine-tuning.
We compare these approaches and understand the trade-offs.

Let us say we have some training data (X_{train}, Y_{train}) for the end-to-end table explicitation task. We use a suitably-prompted pre-trained LLM to solve the table explicitation task, and let f be the function represented by such an LLM. Then, the training data for the last-mile repair task consists of $((X_{train}, f(X_{train})), Y_{train})$. As our pre-trained LLM we use Gpt-4o. We perform our experiments using several different SLMs; namely, (1) the 1B and 3B models from the Llama 3.2 family (Grattafiori et al., 2024), (2) the 4B model from the Qwen3 family (Yang et al., 2025), (3) version 3 of the 7B model and the 24B mistral small from Mistral AI (Team, 2025; Jiang et al., 2023), and (4) the base model from the phi-4 family (Abdin et al., 2024b;a).

4 Dataset and Metrics

We make use of three different datasets in our experiments, those being PubTabNet, FinTabNet and SciTSR.

1. PubTabNet (Zhong et al., 2020): A collection of Images and HTML-encoded tables, totaling to about 568,000 tables. The Images are extracted from the PubMed Central Open Access Subset (PMCOA), with the HTML representations coming from the XML representations of the tables present in the PMCOA. Each individual table was converted into a standalone HTML document, which was in turn converted to a PDF from which the raw clipboard text was extracted.
2. FinTabNet (Lysak et al., 2023): FinTabNet OTSL is a collection of about 70k tables taken from annual reports of companies tracked by the S&P 500, adapated from FinTabNet by creating a new markup language for table recognition. The HTML annotations served as the groundtruth, with the raw text extracted in the same manner as PubTabNet.
3. SciTSR (Chi et al., 2019): A collection of 15000 PDFs and corresponding structural annotations in JSON. The JSON tables contain content as well and cell spanning information, which can procedurally be converted into an HTML table. Since the PDFs themselves are already standalone tables here, the raw text can be extracted directly from them.

We took a random sample of 15515 table explicitation tasks from the above three datasets. We tried to extract a table for each task using GPT-4o. We used a simple programmatic checker to determine if the extracted table was malformed, or a very poor quality table that was not close to the ground truth. If so, then we removed that task from our collection because it is not a candidate for last-mile repair. The remaining tasks were used to get a train-validation-test split. Specifically, we obtained a train/val/test split of 8967/1133/2596 samples this way; see Table 12 for contribution of each dataset to this final set.

4.1 Metrics

We need to compare the predicted table $\hat{y}$ with the ground-truth table y, where both tables are represented in HTML. A naive metric would be exact string equality, or *exact match* (EM), but there are more refined metrics that provide a more fine-grained view on the performance of an approach.

4.1.1 Tree Edit Distance (TED)

Since the ground-truth and the generated table are both HTML strings, they can both be represented as trees using the HTML tags as node labels. We can use tree edit distance, $\texttt{Ted}(t_1, t_2)$, to compute the distance between two such trees, say t_1 and t_2, where $\textsc{Ted}(t_1, t_2)$ is the least number of replacements, deletions and additions needed to transform t_1 to t_2, and can be found using the Zhang-Shasha algorithm Zhang & Shasha (1989). This can be converted to a similarity score between 0 and 1, called *Tree Edit Distance Similarity*, or TEDS, using the formula $\textsc{Teds}(t_1, t_2) = 1 - \texttt{Ted}(t_1, t_2)/(|t_1| + |t_2|)$, where $|t_i|$ denotes the size (number of nodes) of tree t_i. Note that TEDS counts replacement of string s by string s' as 1 edit. Since not all string edits require equal effort from the user, we can modify the TED distance between two labels from being a 0-1 value to to being the Levenshtein or String edit distance. This gives us the $\texttt{Lev} - \texttt{Ted}$ distance metric. While the $\texttt{Lev} - \texttt{Ted}$ metric may seem more reflective of how much effort the user has to make (to transform tree t_1 to t_2), it is worth noting that it can be heavily skewed by outliers (benchmarks with large content strings). Hence, we report both numbers. Higher is better for TEDS similarity and lower is better for the Lev-Ted metric since that would indicate that the predicted tree t_1 is closer to ground-truth tree t_2.

4.1.2 Grid Table Similarity (GriTS)

GriTS is a metric introduced by Smock et al. (2023). This processes the matrix representations of both tables and finds a lower and upper bound on the 2D most similar substructure between both matrices, which is a relaxation of the 2D most common substructure problem. Solving this problem using different similarity functions for topology and content leads to two different metrics, GriTS_{top} and GriTS_{con} measuring the similarity of table topology and content respetively. These metrics are computed as a pair of values, a lower bound and an upper bound, and the true value is in between those values. Since the two bounds were very close in our experiments, we report only the mean of the bounds. Since these are similarity measures, higher is better for these metrics.

5 Evaluation

We now evaluate the LIFT approach by focusing on three key questions:
(1) How does LIFT compare to self-debug and EEFT approaches on the table explicitation task? We clearly understand the trade-off in terms of the compute cost – because we know exactly how many LLM calls and how many SLM calls each technique uses – so, we focus here purely on how well each approach does on the metrics outlined above.
(2) How does the limited availability of training data impact LIFT and EEFT approaches?
(3) Are LIFT and EEFT both equally robust to changes in the input text?

5.1 Comparing the three approaches

We fine-tuned six SLMs, namely Llama 3.2 1B, Llama 3.2 3B, Qwen 3 4B, Mistral 7B, Phi-4 14B, and Mistral 24B, on two tasks separately: (a) the last-mile repair task and (b) end-to-end table explicitation task. The input for last-mile repair task was generated by using Gpt-4o on the input of the table explicitation task, as described in Section 3. The models were all fine-tuned for 5 epochs, with an effective batch size of 128. A total of 50 warmup steps up to a learning rate of 5×10^{-5} were used. The models were evaluated on the validation set after every epoch, and at the end of the training run the checkpoint with the smallest validation loss was chosen. Training was done at half-precision using the bfloat16 format and an 8-bit

Model	TEDS			LevTED		
	SD(M)	LIFT(M)	EEFT(M)	SD(M)	LIFT(M)	EEFT(M)
Llama 3.2 1B	0.696	0.880	0.887	477	214	192
Llama 3.2 3B	0.743	0.903	0.905	371	148	133
Qwen 3 4B	0.835	0.901	0.838	227	136	132
Mistral 7B	0.761	0.936	0.927	377	98	112
Phi-4 14B	0.855	0.923	0.834	186	104	152
Mistral 24B	0.946	0.951	0.945	74	66	71
Gpt-4o	0.873	–	–	144	–	–

Table 1: Comparing the three approaches, self-debug (SD), last-mile fine-tuning (LIFT) and end-to-end fine-tuning (EEFT) on table explicitation tasks across different choices for underlying repair SLM using the TEDS similarity (higher is better) and LevTed distance (lower is better) metrics.

version of the Adam optimizer. LORA was used with rank 16 and $\alpha = 32$. The prompts we used can be found in Appendix C.

We then compared three approaches:
(1) self-debug, SD(M), where the base (not fine-tuned) model M is used to perform the repair on the table extracted by Gpt-4o,
(2) last-mile fine-tuning, LIFT(M), where the version of M fine-tuned on last-mile repair is used to perform the repair on the table extracted by Gpt-4o, and
(3) end-to-end fine-tuning, EEFT(M), where M that is fine-tuned on the end-to-end table explicitation task is used to extract the table from the input text (no Gpt-4o is used in this scenario).

Table 1 shows data comparing the three approaches on table explicitation tasks across different choices for underlying repair SLM using the TEDS similarity (higher is better) and LevTed distance (lower is better) metrics. We did not perform last-mile fine-tuning or the end-to-end fine-tuning on Gpt-4o and hence the blanks in Table 1. We observe that
(a) Across both metrics and across all models M, LIFT(M) is superior to SD(M), which should be expected since we are using a fine-tuned version of M to repair in LIFT(M), whereas we use the base version of M in SD(M).
(b) Last-mile fine-tuning is a promising alternative to doing full task-specific fine-tuning. Both LIFT(M) and EEFT(M) perform comparably, and on the TEDS metric, in fact, LIFT(M) is either very close or arguably better than EEFT for all models. Moreover, last-mile fine-tuning has other benefits, as we will discuss later.
(c) In general, as the SLMs get bigger, their performance *generally* improves across all approaches – with only Phi-4 breaking the trend (on both metrics) for LIFT(M) and EEFT(M), and only Mistral 7B breaking the trend (on both metrics) for SD(M). We also note that SD with Mistral 24B outperforms SD with Gpt-4o, which is surprising, but digging deeper into it, we observed that Mistral 24B was frequently generating empty outputs, and we ignored those tasks when computing the metrics. Table 4 provides the raw count of how many tasks produced valid tables where we see that SD with Mistrat 24B produced 1625 non-empty and valid tables compared to around 2500 for all other models.

The GriTS metrics were correlated with the TEDS metric and hence we did not show those numbers in Table 1. The results for GriTS metrics can be found in Table 5 in the appendix, along with the consolidated full results in Table 4.

5.2 Limited Training Data

We next consider the scenario when there is limited training data. The hypothesis is that when training data is scarce, the LIFT approach would outperform the EEFT approach since the last-mile repair task is presumably simpler than the end-to-end table explicitation task. To test this hypothesis, we pick two base models, Mistral-7B and Llama 3.2 3B, and fine-tune them each with the same number of training data points – we fine-tune each model once for

Train Set Size	Base Model	TEDS LIFT(M)	TEDS EEFT(M)	Base Model	TEDS LIFT(M)	TEDS EEFT(M)
1000	Mistral-7B	0.875	0.731	Llama3.2-3B	0.710	0.740
2000	Mistral-7B	0.878	0.782	Llama3.2-3B	0.823	0.784
4000	Mistral-7B	0.916	0.832	Llama3.2-3B	0.879	0.841
6000	Mistral-7B	0.920	0.858	Llama3.2-3B	0.891	0.839

Table 2: Comparing the last-mile fine-tuning (LIFT) and end-to-end fine-tuning (EEFT) for a fixed size of training set (higher TEDS score is better).

the last-mile repair task (and use that model in LIFT) and once for the data explicitation task (and use that model for EEFT evaluation).

Table 2 shows the results. We show only the TEDS metric since the other metrics showed the same general trends. The full results are included in the Appendix in Table 6. When using the same number of training data points, the LIFT approach mostly outperforms EEFT, which validates our hypothesis. For Mistral-7B we see a nice trend of the performance gap shrinking as the training set size grows from 1000 to 6000. That trend, however, does not hold for Llama, where there is no clear pattern in the performance gap. In fact, surprisingly, with 1000 training points, EEFT performs slightly better when Llama is the base model. In general, though, the results show that it is beneficial to use LIFT when there is limited training data (and when using a large language model at the first step is an option).

We also note here that there is a cost (and latency) tradeoff between the three approaches SD, LIFT and EEFT. The SD approach has no additional training cost, but requires one LLM call for table explicitation, and another LLM or SLM call for repair at test time. Both LIFT and EEFT require some amount of training, though LIFT requires less of it as we saw above. At test time, LIFT requires one LLM call and one SLM call whereas EEFT requires only one SLM call. So, purely from cost and latency point of view, EEFT is the best, but LIFT provides certain other benefits.

5.3 Robustness to Input Format Variability

A benefit of using a general-purpose LLM is that it can handle inputs that are slightly different from those in the training data without a significant drop in its accuracy. Since the LIFT approach uses an LLM in its first step, a natural question is whether LIFT inherits some of this robustness to input format variability, and how does it compare with EEFT on this aspect.

To answer the above question, we created inputs in two different "formats". We randomly sampled 100 datapoints from the test set. We converted the ground-truth tables into broken CSV and well-formed JSON files. Ground-truth tables with spanning cells were excluded because there is no canonical way of representing spans (adjacent cells merged into one big cell) in either format. Since JSON representation requires column headers, any ground-truth table that did not contain a header (⟨thead⟩) was excluded when creating noisy JSONs. The noisy CSV were generated by starting with a clean CSV and (1) randomly changing some separator to a comma, tab, pipe or caret, and (2) adding data to the header or footer of a CSV. The JSONs were just well-formed JSON representations of ground-truth tables. Such JSONs can be easily converted to HTML tables by a few lines of code, but the noisy CSVs are not so easily translated to HTML tables by code. However, for our experiments, they provide two good tests for robustness to format variability because they both are different from our train and test sets for table explicitation.

Table 3 contains the results of evaluating LIFT and EEFT on noisy CSV and clean JSON inputs. We used the same model checkpoints that were used in earlier experiments reported in Section 5.1. Compared to how they performed in Table 1, both approaches performed worse here. On the one hand, this is expected since the inputs here are "out-of-distribution" compared to what the models were trained on. On the other hand, even though the inputs

Model M	CSV		JSON	
	LIFT(M)	EEFT(M)	LIFT(M)	EEFT(M)
Llama 3.2 1B	0.844	0.643	0.785	0.504
Llama 3.2 3B	0.774	0.769	0.503	0.508
Qwen 3 4B	0.869	0.751	0.630	0.695
Mistral 7B	0.762	0.751	0.673	0.751
Phi-4 14B	0.904	0.831	0.750	0.652
Mistral 24B	0.844	0.810	0.820	0.809

Table 3: Comparing the last-mile fine-tuning (LIFT) and end-to-end fine-tuning (EEFT) on differently-formatted inputs - broken CSV and well-formed JSON using TEDS (higher score is better).

are different here, they actually have more information than the text clipboard dumps from PDF selections since there is some structural information in these inputs, but that does not seem to be helping either approach as much.

The main take-away from Table 3 is that LIFT performs consistently better than EEFT on the broken CSVs across the whole set of SLMs. This serves as evidence for the hypothesis that last-mile fine-tuning is more robust than end-to-end fine-tuning. If we look at the performance on JSON, there is no clear winner across models. This is probably because JSON is a standard format and we didn't inject any noise or errors in the JSONs. Different SLMs may have different expertise with manipulating JSON format, which may be reflected in the results of Table 3.

6 Conclusion

In this paper, we explored the table explicitation task and experimentally evaluated three possible approaches for it; namely, (a) self-debug, where a pre-trained LLM is used to solve the task and then (the same LLM or a different SLM) is used to repair the output, (b) last-mile fine-tuning, where an LLM is used to solve the task and then a fine-tuned SLM is used to repair the output, and (c) end-to-end fine-tuning, where an SLM is fine-tuned for the entire task. A few clear trade-offs emerged. When a large amount of training data is available, it is beneficial to fine-tune an SLM for the end-to-end task because this is the most cost-effective solution as it involves no LLM calls. If data is limited and a certain amount of robustness to input variability is desired, the last-mile fine-tuning approach is most effective – it does no worse than or better than end-to-end fine-tuning. It also does better than self-debug and is also more cost-effective than self-debug with LLMs. If the option to fine-tune and deploy fine-tuned SLMs is not available, then using an LLM to solve the task and then using the same LLM or a powerful SLM to repair (without any fine-tuning) is a good option.

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024a.

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Hassan Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Singh Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Caio C'esar Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Parul Chopra, Allison Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Victor Fragoso, Dan Iter, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Young Jin Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Chen Liang, Weishung Liu, Eric Lin, Zeqi Lin, Piyush Madan, Arindam Mitra, Hardik Modi, Anh Hong Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker,

Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Corby Rosset, Sambudha Roy, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Xianmin Song, Olatunji Ruwase, Praneetha Vaddamanu, Xin Wang, Rachel Ward, Guanhua Wang, Philipp Andre Witte, Michael Wyatt, Can Xu, Jiahang Xu, Sonal Yadav, Fan Yang, Ziyi Yang, Donghan Yu, Cheng yuan Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yunan Zhang, Xiren Zhou, and Yifan Yang. Phi-3 technical report: A highly capable language model locally on your phone. *ArXiv*, abs/2404.14219, 2024b. URL https://api.semanticscholar.org/CorpusID:269293048.

Adobe. Adobe acrobat PDF to Excel, 2026. URL https://www.adobe.com/acrobat/online/pdf-to-excel.html. Accessed: March 2026.

Manuel Aristarán, Mike Tigas, and Jeremy B. Merrill. Tabula: Tool for liberating data table locked inside pdf files. https://tabula.technology/, 2012. Accessed: 2026-03-22.

Orlando Marquez Ayala, Patrice Bechard, Emily Chen, Maggie Baird, and Jingfei Chen. Fine-tune an SLM or prompt an LLM? the case of generating low-code workflows. In *The First Structured Knowledge for Large Language Models Workshop*, 2025. URL https://openreview.net/forum?id=39ATVtO5H1.

Rohan Bavishi, Harshit Joshi, José Cambronero, Anna Fariha, Sumit Gulwani, Vu Le, Ivan Radiček, and Ashish Tiwari. Neurosymbolic repair for low-code formula languages. *Proc. ACM Program. Lang.*, 6(OOPSLA2), October 2022. doi: 10.1145/3563327. URL https://doi.org/10.1145/3563327.

Leiyuan Chen, Chengsong Huang, Xiaoqing Zheng, Jinshu Lin, and Xuanjing Huang. TableVLM: Multi-modal pre-training for table structure recognition. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2437–2449, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.137. URL https://aclanthology.org/2023.acl-long.137/.

Zewen Chi, Heyan Huang, Heng-Da Xu, Houjin Yu, Wanxuan Yin, and Xian-Ling Mao. Complicated table structure recognition, 2019. URL https://arxiv.org/abs/1908.04729.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023. URL http://jmlr.org/papers/v24/22-1144.html.

Zheye Deng, Chunkit Chan, Weiqi Wang, Yuxi Sun, Wei Fan, Tianshi Zheng, Yauwai Yim, and Yangqiu Song. Text-tuple-table: Towards information integration in text-to-table generation via global tuple extraction. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9300–9322, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.523. URL https://aclanthology.org/2024.emnlp-main.523/.

David Jiahao Fu, Aryan Gupta, Aaron Councilman, David Grove, Yu-Xiong Wang, and Vikram Adve. Slmfix: Leveraging small language models for error fixing with reinforcement learning, 2025. URL https://arxiv.org/abs/2511.19422.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/8ac015d409635f196f9e3e9dcfb9a94e-Paper-Conference.pdf.

Albert Qiaochu Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *ArXiv*, abs/2310.06825, 2023. URL https://api.semanticscholar.org/CorpusID:263830494.

Harshit Joshi, José Cambronero Sanchez, Sumit Gulwani, Vu Le, Ivan Radiček, and Gust Verbruggen. Repair is nearly generation: multilingual program repair with llms. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. ISBN 978-1-57735-880-0. doi: 10.1609/aaai.v37i4.25642. URL https://doi.org/10.1609/aaai.v37i4.25642.

Kyeonghyun Kim, Jinhee Jang, Juhwan Choi, Yoonji Lee, Kyohoon Jin, and YoungBin Kim. Plug-in and fine-tuning: Bridging the gap between small language models and large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5434–5452, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.271. URL https://aclanthology.org/2025.acl-long.271/.

Yiren Li, Zheng Huang, Junchi Yan, Yi Zhou, Fan Ye, and Xianhui Liu. Gfte: graph-based financial table extraction. In *International conference on pattern recognition*, pp. 644–658. Springer, 2021.

Maksym Lysak, Ahmed Nassar, Nikolaos Livathinos, Christoph Auer, and Peter Staar. Optimized table tokenization for table structure recognition. In Gernot A. Fink, Rajiv Jain, Koichi Kise, and Richard Zanibbi (eds.), *Document Analysis and Recognition - ICDAR 2023*, pp. 37–50, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-41679-8.

Nikita Mehrotra, Aayush Kumar, Sumit Gulwani, Arjun Radhakrishna, and Ashish Tiwari. Ten: Table explicitization, neurosymbolically, 2025. URL https://arxiv.org/abs/2508.09324.

Shubham Singh Paliwal, Vishwanath D, Rohit Rahul, Monika Sharma, and Lovekesh Vig. Tablenet: Deep learning model for end-to-end table detection and tabular data extraction from scanned document images. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pp. 128–133, 2019. doi: 10.1109/ICDAR.2019.00029.

Devashish Prasad, Ayan Gadpal, Kshitij Kapadni, Manish Visave, and Kavita Sultanpure. Cascadetabnet: An approach for end to end table detection and structure recognition from image-based documents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

Mukul Singh, Gust Verbruggen, Vu Le, and Sumit Gulwani. Tabularis revilio: Converting text to tables. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, pp. 4056–4060, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704369. doi: 10.1145/3627673.3680000. URL https://doi.org/10.1145/3627673.3680000.

Brandon Smock, Rohith Pesala, and Robin Abraham. Grits: Grid table similarity metric for table structure recognition. In Gernot A. Fink, Rajiv Jain, Koichi Kise, and Richard Zanibbi (eds.), *Document Analysis and Recognition - ICDAR 2023*, pp. 535–549, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-41734-4.

Mistral AI Team. Mistral small 3, Jan 2025. URL https://mistral.ai/news/mistral-small-3/.

Xueqing Wu, Jiacheng Zhang, and Hang Li. Text-to-table: A new way of information extraction. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2518–2533, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.180. URL https://aclanthology.org/2022.acl-long.180/.

Junjie Xing, Yeye He, Mengyu Zhou, Haoyu Dong, Shi Han, Dongmei Zhang, and Surajit Chaudhuri. Table-LLM-specialist: Language model specialists for tables using iterative fine-tuning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35443–35460, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1795. URL https://aclanthology.org/2025.emnlp-main.1795/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. TableFormer: Robust Transformer Modeling for Table-Text Encoding. In *Proc. of ACL*, 2022.

K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022. URL https://arxiv.org/abs/2205.01068.

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B. Hashimoto. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57, 2024a. doi: 10.1162/tacl_a_00632. URL https://aclanthology.org/2024.tacl-1.3/.

Zhenrong Zhang, Pengfei Hu, Jiefeng Ma, Jun Du, Jianshu Zhang, Baocai Yin, Bing Yin, and Cong Liu. Semv2: Table separation line detection based on instance segmentation. *Pattern Recognition*, 149:110279, 2024b. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2024.110279. URL https://www.sciencedirect.com/science/article/pii/S003132032400030X.

Xu Zhong, Elaheh ShafieiBavani, and Antonio Jimeno Yepes. Image-based table recognition: Data, model, and evaluation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision – ECCV 2020*, pp. 564–580, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58589-1.

A Detailed Experimental Results

A.1 Comparing the three approaches

Table 4 shows the full results for the experiment comparing self-debug, last-mile file-tuning, and end-to-end fine-tuning. It includes the following baselines:

1. Using GPT 4o to extract the table from the raw text
2. Using GPT 4o to extract the table, and a second call to GPT 4o to repair the table, which we have called SD($Gpt4o$).

The results are shown in Table 4, using the 1B and 3B models from the Llama 3.2 family (Grattafiori et al., 2024), the 4B model from the Qwen3 family (Yang et al., 2025), version 3 of the 7B model and the 24B mistral small from Mistral AI (Team, 2025; Jiang et al., 2023), and the base model from the phi-4 family (Abdin et al., 2024b;a). In this table, "SD" denotes self-debug where the table explicitation task is first solved by Gpt-4o, and its result is repaired using either Gpt-4o (Row 2) or an SLM *with no fine-tuning*. The term "LIFT" denotes last-mile fine-tuning, where Gpt-4o again solves the table explicitation task first and then its output is repaired using a fine-tuned SLM. The term "EEFT" denotes the end-to-end fine-tuning approach, where the end-to-end task is performed using a fine-tuned model. We only fine-tuned SLMs, so there are no results for LIFT (Gpt-4o) or EEFT (Gpt-4o).

Model	Exact Match / Valid Tables	GriTS$_{top}$	GriTS$_{con}$	TEDS	Lev-TED
GPT 4o (extraction)	166/2596	0.678 - 0.678	0.561 - 0.569	0.686	491.435
GPT 4o (SD)	710/2571	0.899 - 0.899	0.842 - 0.846	0.873	143.877
Llama 3.2 1B (SD)	66/2485	0.679 - 0.679	0.539 - 0.548	0.696	477.304
Llama 3.2 3B (SD)	182/2425	0.732 - 0.732	0.602 - 0.610	0.743	370.878
Qwen 3 4B (SD)	424/2470	0.825 - 0.825	0.747 - 0.752	0.835	226.900
Mistral-7B (SD)	262/2494	0.760 - 0.761	0.629 - 0.638	0.761	377.253
Phi-4 (SD)	500/2586	0.862 - 0.862	0.787 - 0.792	0.855	186.273
Mistral-24B (SD)	319/1625	0.852 - 0.852	0.771 - 0.776	0.881	184.952
Llama 3.2 1B (LIFT)	552/2581	0.841 - 0.841	0.745 - 0.752	0.880	214.420
Llama 3.2 3B (LIFT)	782/2588	0.878 - 0.878	0.811 - 0.816	0.903	148.014
Qwen 3 4B (LIFT)	696/2585	0.882 - 0.883	0.816 - 0.821	0.901	136.280
Mistral-7B (LIFT)	1232/2577	0.926 - 0.926	0.879 - 0.881	0.936	97.610
Phi-4 (LIFT)	1006/2596	0.915 - 0.915	0.870 - 0.873	0.923	103.786
Mistral-24B (LIFT)	1488/2579	0.951-0.951	0.924 - 0.925	0.951	65.580
Llama 3.2 1B (EEFT)	552/2568	0.855 - 0.855	0.763 - 0.769	0.887	192.132
Llama 3.2 3B (EEFT)	812/2587	0.888 - 0.888	0.821 - 0.826	0.905	132.714
Qwen 3 4B (EEFT)	729/2586	0.884 - 0.885	0.820 - 0.825	0.838	131.920
Mistral-7B (EEFT)	1408/2591	0.917-0.917	0.866-0.867	0.927	112.272
Phi-4 (EEFT)	912/2594	0.887 - 0.887	0.832 - 0.832	0.834	152.23
Mistral-24B (EEFT)	1408/2591	0.942-0.943	0.911-0.912	0.945	70.71

Table 4: Consolidated results for table explicitation problem using last-mile repair (labeled "LIFT"), self-debug (labeled "SD"), and end-to-end fine-tuning (labeled "EEFT"). The first row is the Gpt-4o baseline (single LLM call). Not all methods successfully completed all tests and the denominator in 2nd column shows that. Upper and lower bounds for GriTS scores are provided. All models are instruction tuned.

The small models see major improvements in extracting both the topology and the content from supervised finetuning. Mistral 24B appears to have some interesting statistics because even without fine-tuning, its TEDS score 0.946 (for Gpt-4o extraction + Mistral 24B base repair) is better than the TEDS score 0.876 that Gpt-4o gets (when it does both extraction and repair step). In fact, Mistral 24B base outperforms even the fine-tuned version of it's 7B counterpart in terms of Levenshtein TED. However, we also note that Mistral 24B base (without fine-tuning) generates valid HTML much less frequently (1625) than any other model tested (all others have around 2500) and this could have contributed to the skew in numbers. However, the general trends are within expectations, with a loose correlation between model size and performance, and SFT improves the number of valid HTML tables generated, as well as improving the performance on all of the measured metrics.

We had extracted Table 1 from Table 4. Table 5 is identical to Table 1 except that it reports the GriTS metrics, whereas Table 1 reported the TEDS and LevTed metrics.

Model	$\textbf{GriTS}_{top}$			$\textbf{GriTS}_{con}$		
	SD(M)	LIFT(M)	EEFT(M)	SD(M)	LIFT(M)	EEFT(M)
Llama 3.2 1B	0.679	0.841	0.855	0.544	0.749	0.766
Llama 3.2 3B	0.732	0.878	0.888	0.606	0.814	0.824
Qwen 3 4B	0.825	0.882	0.884	0.750	0.819	0.823
Mistral 7B	0.760	0.926	0.917	0.634	0.880	0.867
Phi-4 14B	0.862	0.915	0.887	0.790	0.872	0.832
Mistral 24B	0.852	0.951	0.942	0.774	0.925	0.912

Table 5: Comparing the three approaches, self-debug (SD), last-mile fine-tuning (LIFT) and end-to-end fine-tuning (EEFT) on table explicitation tasks across different choices for underlying repair SLM using the GriTS topology and GriTS content metrics (higher is better).

A.2 Limited Training Data

We find that when data is limited, last-mile fine-tuning-based repair outperforms end-to-end fine-tuning-based generation. To this end, we obtain different model checkpoints for two models, Llama 3.2 3B and Mistral 7B. We train these SLMs on both tasks (generation and repair) with different training set sizes (1000, 2000, 4000, 6000). The full results are shown in Table 6. The repair models, at every size, outperform their generation counterparts on almost every single metric at any amount of data. Llama 3.2 instruct at 1000 is the only exception, doing better at repair than generation in terms of TEDS and both GriTS metrics.

Mistral-7B-Instruct-v0.3	GriTS_{top}	GriTS_{con}	TEDS	Lev-TED
EEFT, 1000 datapoints	0.699 - 0.699	0.558 - 0.566	0.731	409.183
EEFT, 2000 datapoints	0.745 - 0.745	0.636 - 0.642	0.782	330.438
EEFT, 4000 datapoints	0.791 - 0.791	0.713 - 0.718	0.832	253.908
EEFT, 6000 datapoints	0.826 - 0.826	0.744 - 0.749	0.858	222.347
LIFT, 1000 datapoints	0.832 - 0.832	0.730 - 0.739	0.875	211.231
LIFT, 2000 datapoints	0.833 - 0.833	0.741 - 0.749	0.878	196.384
LIFT, 4000 datapoints	0.899 - 0.899	0.836 - 0.841	0.916	128.148
LIFT, 6000 datapoints	0.909 - 0.909	0.856 - 0.859	0.920	122.961
Llama-3.2-3B-Instruct	GriTS_{top}	GriTS_{con}	TEDS	Lev-TED
EEFT, 1000 datapoints	0.688 - 0.688	0.559 - 0.567	0.740	439.855
EEFT, 2000 datapoints	0.722 - 0.723	0.609 - 0.617	0.784	379.255
EEFT, 4000 datapoints	0.789 - 0.789	0.707 - 0.713	0.841	270.541
EEFT, 6000 datapoints	0.790 - 0.791	0.710 - 0.717	0.839	283.733
LIFT, 1000 datapoints	0.650 - 0.650	0.517 - 0.528	0.710	410.683
LIFT, 2000 datapoints	0.759 - 0.760	0.630 - 0.639	0.823	274.262
LIFT, 4000 datapoints	0.838 - 0.839	0.741 - 0.749	0.879	194.125
LIFT, 6000 datapoints	0.863 - 0.864	0.778 - 0.785	0.891	167.004

Table 6: Comparison of LIFT and EEFT approaches at different training dataset sizes. The LIFT approach geenrally performs better than the EEFT approach acroos different train set sizes, with the gap being larger the lesser data there is.

A.3 Robustness to Input Format Variability

Table 7 shows the full results (all four metrics) for both LIFT and EEFT approaches on broken CSVs, which was discussed in Section 5.3. Table 8 shows the same data but for well-formed JSON inputs. Part of the results from these two tables was shown before in Table 3.

Model	GriTS$_{top}$	GriTS$_{con}$	TEDS	Lev-TED
LLM	0.966 - 0.966	0.923 - 0.924	0.925	53.76
Mistral-7B (EEFT)	0.822 - 0.822	0.663 - 0.667	0.751	248.386
Mistral-24B (EEFT)	0.883 - 0.884	0.802 - 0.803	0.810	165.75
Llama 3.2 1B (EEFT)	0.641 - 0.641	0.414 - 0.420	0.643	466.286
Llama 3.2 3B (EEFT)	0.824 - 0.885	0.672 - 0.676	0.769	262.220
Qwen 3 4B (EEFT)	0.879 - 0.879	0.790 - 0.793	0.751	160.46
Phi-4 (EEFT)	0.904 - 0.904	0.804 - 0.805	0.831	143.231
Phi-4 (LIFT)	0.945 - 0.945	0.882 - 0.883	0.904	88.505
Qwen 3 4B (LIFT)	0.876 - 0.876	0.798 - 0.802	0.869	121.260
Llama 3.2 1B (LIFT)	0.878 - 0.881	0.767 - 0.776	0.844	192.950
Llama 3.2 3B (LIFT)	0.822 - 0.822	0.677 - 0.681	0.774	216.460
Mistral 7B (LIFT)	0.821 - 0.821	0.695 - 0.697	0.762	226.277
Mistral 24B (LIFT)	0.934 - 0.934	0.843 - 0.844	0.844	108.283

Table 7: Comparing LIFT and EEFT on extracting tables from broken CSV files. Note that all models were still fine-tuned on the original training data coming from the clipboard when copying from PDF documents.

Model	GriTS$_{top}$	GriTS$_{con}$	TEDS	Lev-TED
LLM	0.880 - 0.880	0.783 - 0.784	0.826	256.142
Mistral-7B (EEFT)	0.503 - 0.503	0.243 - 0.263	0.751	793.021
Mistral-24B (EEFT)	0.803 - 0.803	0.662 - 0.674	0.809	293.244
Llama 3.2 1B (EEFT)	0.407 - 0.407	0.123 - 0.143	0.504	1027.071
Llama 3.2 3B (EEFT)	0.411 - 0.412	0.152 - 0.172	0.508	1044.437
Qwen 3 4B (EEFT)	0.617 - 0.617	0.443 - 0.460	0.695	499.795
Phi-4 (EEFT)	0.590 - 0.590	0.370 - 0.394	0.652	599.279
Phi-4 (LIFT)	0.708 - 0.708	0.573 - 0.587	0.750	413.111
Qwen 3 4B (LIFT)	0.511 - 0.511	0.330 - 0.348	0.630	630.068
Llama 3.2 1B (LIFT)	0.746 - 0.747	0.629 - 0.639	0.785	361.231
Llama 3.2 3B (LIFT)	0.459 - 0.460	0.222 - 0.236	0.503	976.220
Mistral 7B (LIFT)	0.679 - 0.679	0.455 - 0.467	0.673	606.344
Mistral 24B (LIFT)	0.811 - 0.813	0.686 - 0.699	0.820	226.495

Table 8: Comparing LIFT and EEFT on extracting tables from well-formed JSON files. Note that all models were still fine-tuned on the original training data coming from the clipboard when copying from PDF documents.

B More Workflow Examples

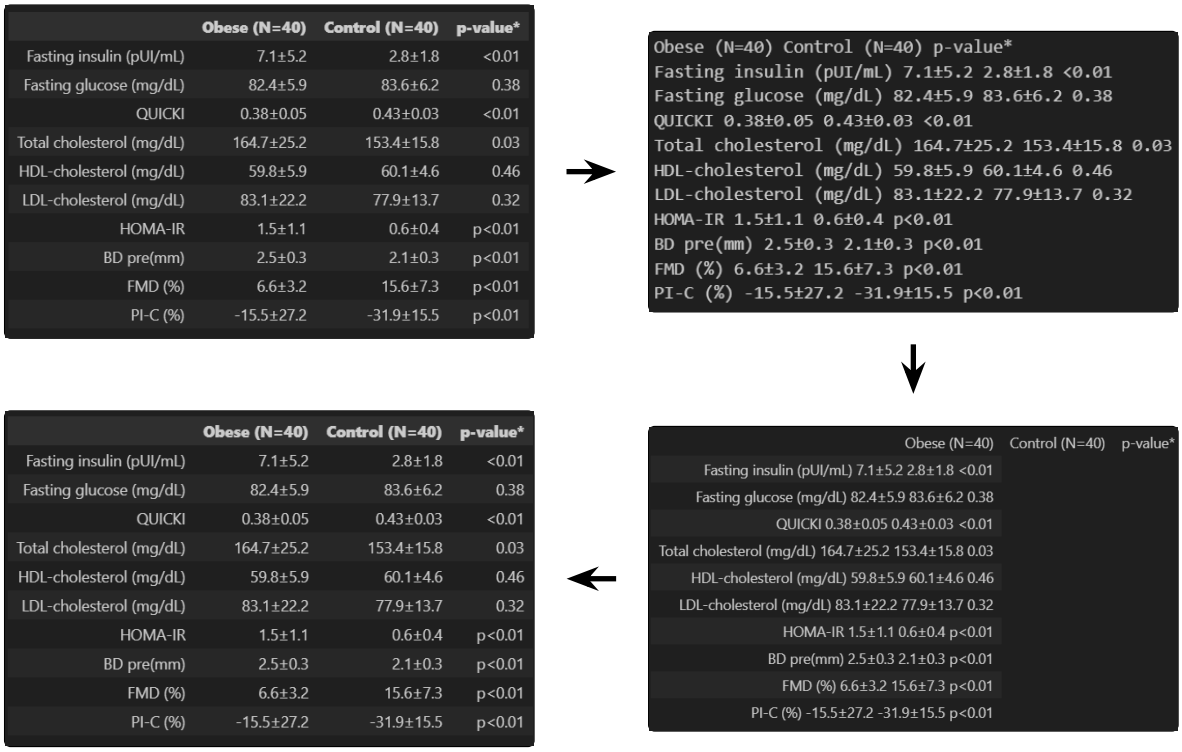
Following the style of the illustration presented in Figure 1, we present a few other illustrations of last-mile repair using fine-tuned SLMs on the table explicitation problem in Figure 2, Figure 3, Figure 4, Figure 5, and Figure 6.

C Prompts, Hyperparameters, Dataset Splits

The models were all fine-tuned for 5 epochs, with an effective batch size of 128. 50 warmup steps up to a learning rate of 5×10^{-5} were used. The model was evaluated on the validation set after every epoch, and at the end of the training run the checkpoint with the smallest validation loss was chosen. Training was done at half-precision using the bfloat16 format and an 8-bit version of the Adam optimizer. LORA was used with rank 16 and $\alpha = 32$.

We have two prompts, one for table explicitation and one for table repair.

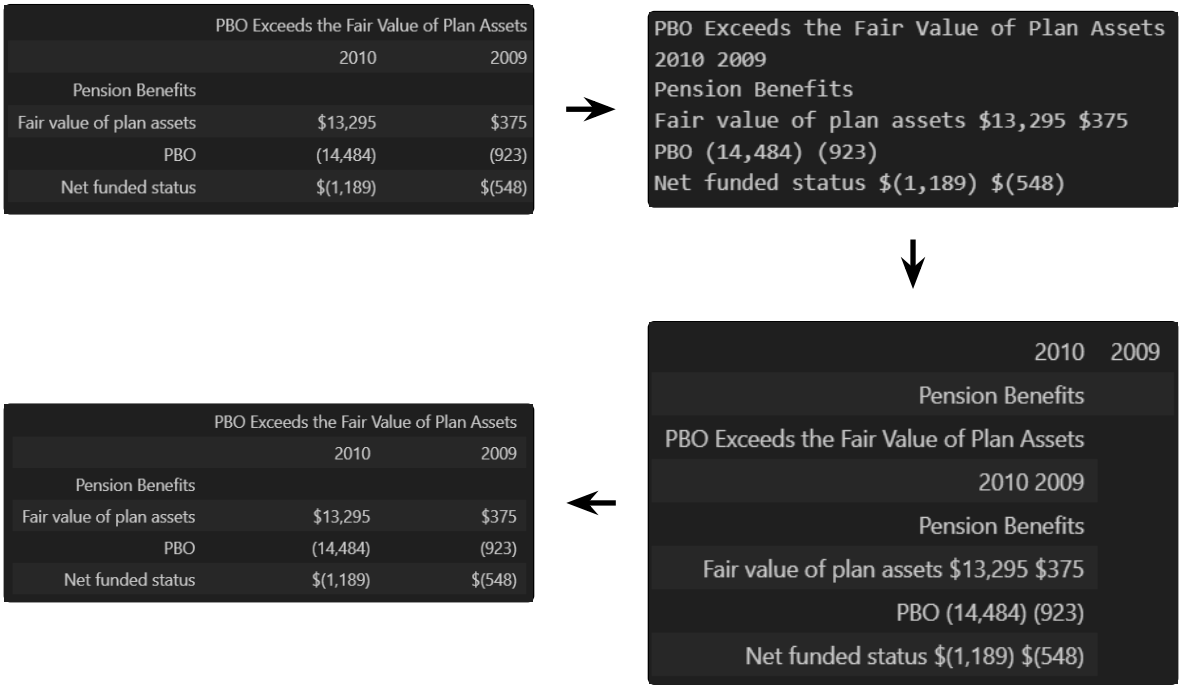
System Prompt(Repair):



	Obese (N=40)	Control (N=40)	p-value*
Fasting insulin (pUI/mL)	7.1±5.2	2.8±1.8	<0.01
Fasting glucose (mg/dL)	82.4±5.9	83.6±6.2	0.38
QUICKI	0.38±0.05	0.43±0.03	<0.01
Total cholesterol (mg/dL)	164.7±25.2	153.4±15.8	0.03
HDL-cholesterol (mg/dL)	59.8±5.9	60.1±4.6	0.46
LDL-cholesterol (mg/dL)	83.1±22.2	77.9±13.7	0.32
HOMA-IR	1.5±1.1	0.6±0.4	p<0.01
BD pre(mm)	2.5±0.3	2.1±0.3	p<0.01
FMD (%)	6.6±3.2	15.6±7.3	p<0.01
PI-C (%)	-15.5±27.2	-31.9±15.5	p<0.01

Obese (N=40) Control (N=40) p-value*
Fasting insulin (pUI/mL) 7.1±5.2 2.8±1.8 <0.01
Fasting glucose (mg/dL) 82.4±5.9 83.6±6.2 0.38
QUICKI 0.38±0.05 0.43±0.03 <0.01
Total cholesterol (mg/dL) 164.7±25.2 153.4±15.8 0.03
HDL-cholesterol (mg/dL) 59.8±5.9 60.1±4.6 0.46
LDL-cholesterol (mg/dL) 83.1±22.2 77.9±13.7 0.32
HOMA-IR 1.5±1.1 0.6±0.4 p<0.01
BD pre(mm) 2.5±0.3 2.1±0.3 p<0.01
FMD (%) 6.6±3.2 15.6±7.3 p<0.01
PI-C (%) -15.5±27.2 -31.9±15.5 p<0.01

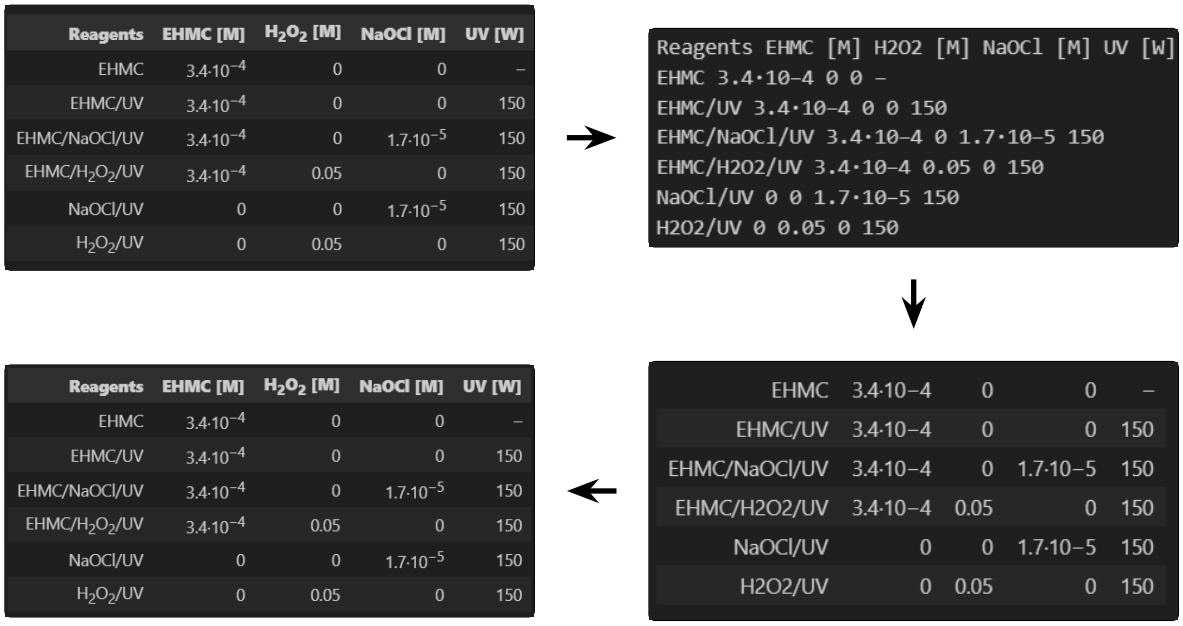
Figure 2: Workflow visualization using images.



	PBO Exceeds the Fair Value of Plan Assets	
	2010	2009
Pension Benefits		
Fair value of plan assets	$13,295	$375
PBO	(14,484)	(923)
Net funded status	$(1,189)	$(548)

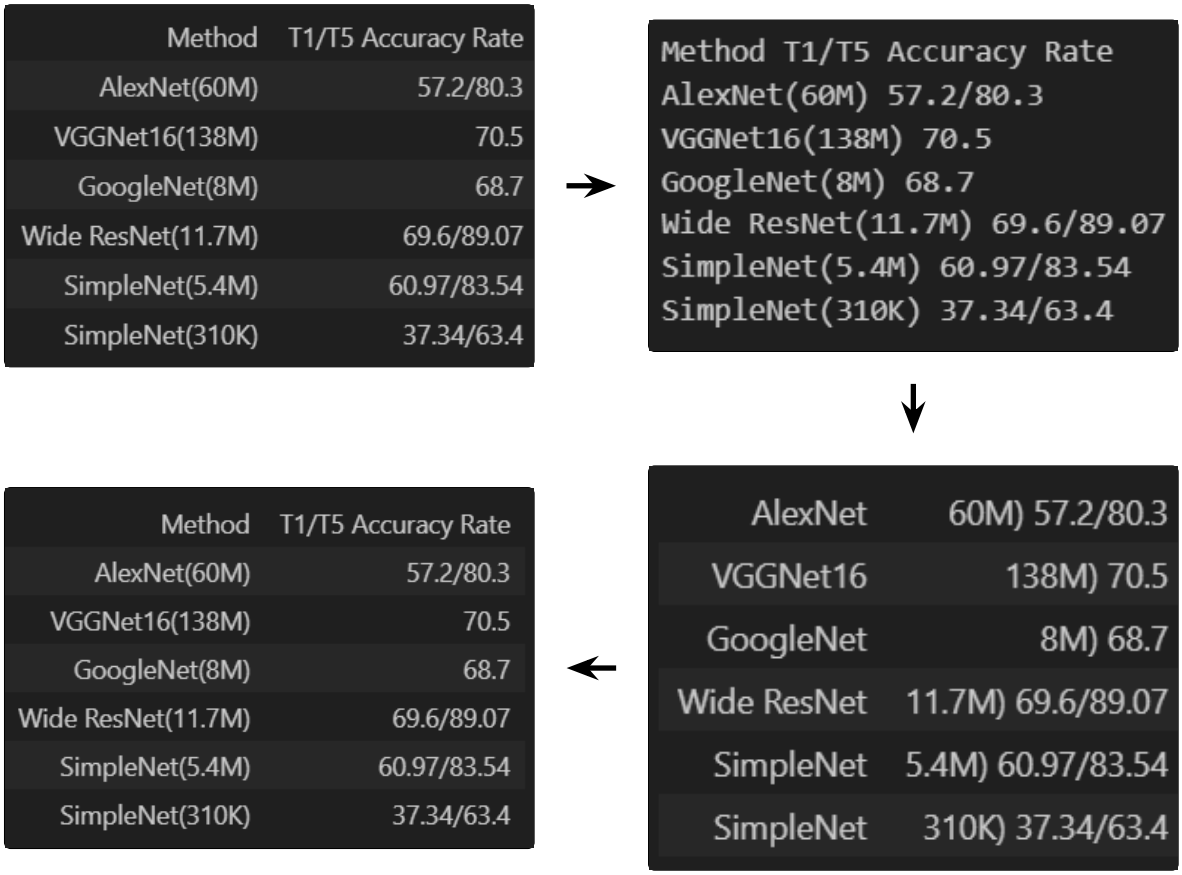
PBO Exceeds the Fair Value of Plan Assets
2010 2009
Pension Benefits
Fair value of plan assets $13,295 $375
PBO (14,484) (923)
Net funded status $(1,189) $(548)

Figure 3: Workflow visualization using images.



Reagents	EHMC [M]	H_2O_2 [M]	NaOCl [M]	UV [W]
EHMC	$3.4{\cdot}10^{-4}$	0	0	–
EHMC/UV	$3.4{\cdot}10^{-4}$	0	0	150
EHMC/NaOCl/UV	$3.4{\cdot}10^{-4}$	0	$1.7{\cdot}10^{-5}$	150
EHMC/H_2O_2/UV	$3.4{\cdot}10^{-4}$	0.05	0	150
NaOCl/UV	0	0	$1.7{\cdot}10^{-5}$	150
H_2O_2/UV	0	0.05	0	150

Figure 4: Workflow visualization using images.



Method	T1/T5 Accuracy Rate
AlexNet(60M)	57.2/80.3
VGGNet16(138M)	70.5
GoogleNet(8M)	68.7
Wide ResNet(11.7M)	69.6/89.07
SimpleNet(5.4M)	60.97/83.54
SimpleNet(310K)	37.34/63.4

Figure 5: Workflow visualization using images.

Model Name	GriTS$_{top}$	GriTS$_{con}$	TEDS	Lev-TED
Mistral-7B (LIFT)	0.79	0.86	0.96	0.96
Mistral-24B (LIFT)	0.82	0.89	0.97	0.97
Phi-4 (LIFT)	0.77	0.84	0.94	0.94
Qwen-3 (LIFT)	0.73	0.78	0.90	0.92
Llama-1B (LIFT)	0.68	0.71	0.87	0.87
Llama-3B (LIFT)	0.73	0.78	0.92	0.93
Mistral-7B (EEFT)	0.78	0.83	0.94	0.94
Mistral-24B (EEFT)	0.82	0.88	0.96	0.97
Phi-4 (EEFT)	0.71	0.81	0.94	0.94
Qwen-3 (EEFT)	0.65	0.68	0.73	0.75
Llama-1B (EEFT)	0.70	0.73	0.87	0.86
Llama-3B (EEFT)	0.75	0.80	0.90	0.91

Table 9: Percentage of inputs improved over single call for GPT 4o for different metrics

Model Name	GriTS$_{top}$	GriTS$_{con}$	TEDS	Lev-TED
Mistral-7B (LIFT)	0.25	0.32	0.25	377.74
Mistral-24B (LIFT)	0.27	0.36	0.26	408.32
Phi-4 (LIFT)	0.24	0.31	0.24	387.65
Qwen-3 (LIFT)	0.20	0.25	0.21	349.12
Llama-1B (LIFT)	0.16	0.18	0.19	270.28
Llama-3B (LIFT)	0.20	0.25	0.22	342.30
Mistral-7B (EEFT)	0.24	0.30	0.24	371.39
Mistral-24B (EEFT)	0.26	0.35	0.26	412.67
Phi-4 (EEFT)	0.21	0.27	0.21	336.95
Qwen-3 (EEFT)	0.15	0.19	0.14	236.17
Llama-1B (EEFT)	0.17	0.20	0.20	285.96
Llama-3B (EEFT)	0.21	0.26	0.22	350.29

Table 10: Average absolute improvement in metric against a single call to GPT 4o

```
You are an expert in interpreting various table formats. Your task is to
    generate fixed HTML tables from unstructured text, and optionally a table
    that may or may not contain errors.
```

User Prompt(Repair):

```
I have an unstructured text representation of a table and an html
    representation of the same table.
```

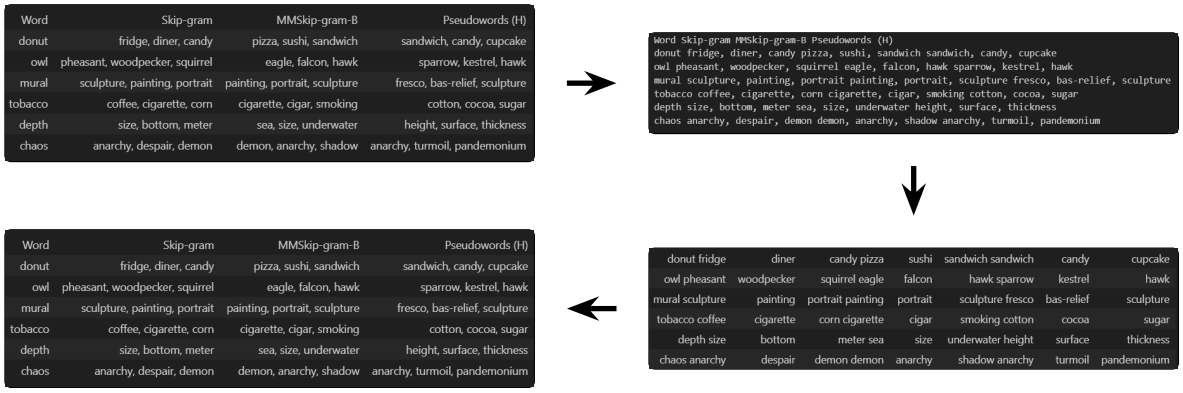


Figure 6: Workflow visualization using images.

Model Name	GriTS_{top}	GriTS_{con}	TEDS	Lev-TED
GPT 4o (Single Call)	[0.6672, 0.6893]	[0.5499, 0.5724]	[0.6789, 0.6949]	[465.2874, 519.4812]
Mistral-7B (LIFT)	[0.9196, 0.9305]	[0.8713, 0.8859]	[0.9322, 0.9394]	[88.2496, 108.8308]
Mistral-24B (LIFT)	[0.9471, 0.9551]	[0.9185, 0.9291]	[0.9481, 0.9541]	[58.9844, 73.2232]
Phi-4 (LIFT)	[0.9090, 0.9205]	[0.8631, 0.8770]	[0.9193, 0.9269]	[94.3027, 113.8553]
Qwen-3 (LIFT)	[0.8769, 0.8888]	[0.8078, 0.8242]	[0.8969, 0.9060]	[123.6902, 150.2994]
Llama-1B (LIFT)	[0.8327, 0.8479]	[0.7352, 0.7547]	[0.8740, 0.8845]	[195.4633, 233.1382]
Llama-3B (LIFT)	[0.8715, 0.8846]	[0.8015, 0.8189]	[0.8986, 0.9077]	[134.8696, 161.7537]
Mistral-7B (EEFT)	[0.9117, 0.9224]	[0.8585, 0.8725]	[0.9227, 0.9303]	[102.5411, 123.4352]
Mistral-24B (EEFT)	[0.9379, 0.9468]	[0.9052, 0.9168]	[0.9413, 0.9480]	[64.0791, 77.9916]
Phi-4 (EEFT)	[0.8805, 0.8937]	[0.8239, 0.8416]	[0.8910, 0.9017]	[137.9231, 168.1007]
Qwen-3 (EEFT)	[0.8697, 0.8879]	[0.7786, 0.8017]	[0.8448, 0.8587]	[145.3816, 176.4601]
Llama-1B (EEFT)	[0.8479, 0.8623]	[0.7533, 0.7722]	[0.8822, 0.8918]	[175.3041, 209.9542]
Llama-3B (EEFT)	[0.8819, 0.8942]	[0.8127, 0.8296]	[0.9013, 0.9097]	[122.2198, 143.7246]

Table 11: Bootstrap Confidence Intervals calculated at 95% with 1000 bootstrap samples, computed for all the metrics of interest

Dataset	**Train**	**Val**	**Test**
PubTabNet	3464	422	877
FinTabNet	2714	361	805
SciTSR	2789	350	914

Table 12: Dataset Contribution to Splits

```
Instructions:
-Analyze the given ''Raw Input **unstructured** Text'' and html table to
    accurately identify rows, columns, and data cells.
-Ensure that the output table maintains the same row and column structure as
    the given ''Raw Input Text''.
-Keep the table rows and columns in the same order and structure as they appear
     in the given ''Raw Input Text''.
-Include all content from the current given ''Raw Input Text'' without omission.

-Do not add any content in the fixed HTML table if it is not present in the
    given ''Raw Input Text''.
-Ensure that the fixed HTML table is well-formed and valid, and enclosed inside
     <table></table> tags.
''Raw Input Text'':
{{Input Text Here}}

''HTML Table'':
{{Broken HTML Table Here}}
```

System Prompt(Explicitation):

```
You are an expert in interpreting various table formats. Your task is to
    generate fixed HTML tables from unstructured text.
```

User Prompt(Explicitation):

```
I have an unstructured text representation of a table.
Instructions:
-Analyze the given ''Raw Input **unstructured** Text'' and html table to
    accurately identify rows, columns, and data cells.
-Ensure that the output table maintains the same row and column structure as
    the given ''Raw Input Text''.
```

```
-Keep the table rows and columns in the same order and structure as they appear
     in the given ''Raw Input Text''.
-Include all content from the current given ''Raw Input Text'' without omission.

-Do not add any content in the fixed HTML table if it is not present in the
    given ''Raw Input Text''.
-Ensure that the fixed HTML table is well-formed and valid, and enclosed inside
     <table></table> tags.
''Raw Input Text'':
{{Input Text Here}}
```