STable Table Generation Framework for Encoder-Decoder Models

Michał Pietruszka[*1,2], Michał Turski[*1,3], Łukasz Borchmann[*1], Tomasz Dwojak[1], Gabriela Pałka[1,3], Karolina Szyndler[1], Dawid Jurkiewicz[1,3], and Łukasz Garncarek[1]

[1]Snowflake
[2]Jagiellonian University
[3]Adam Mickiewicz University

Abstract

Since the output structure of database-like tables can cover a wide range of NLP tasks, we propose a framework for text-to-table neural models applicable to, e.g., extraction of line items, joint entity and relation extraction, or knowledge base population. The permutation-based decoder of our proposal is a generalized sequential method that comprehends information from all cells in the table. The training maximizes the expected log-likelihood for a table's content across all random permutations of the factorization order. During the content inference, we exploit the model's ability to generate cells in any order by searching over possible orderings to maximize the model's confidence and avoid substantial error accumulation, which other sequential models are prone to. Experiments demonstrate a high practical value of the framework, which establishes state-of-the-art results on several challenging datasets, outperforming previous solutions by up to 15%.

1 Introduction

It has been previously shown that encoder-decoder models are capable of unifying a variety of problems involving natural language. In this setting, unification is achieved by casting different tasks as Question Answering with a plain-text answer, i.e., assuming the text-to-text (Kumar et al., 2016; Raffel et al., 2020; McCann et al., 2018; Khashabi et al., 2020) or document-to-text scenario (Powalski et al., 2021; Kim et al., 2022). We argue that the restriction of output type to raw text is suboptimal for the plethora of NLP problems and propose a decoder architecture able to infer *aggregate* data types such as a list of ordered tuples or a database-like table (see Figure 1).

Though the encoder-decoder architecture was formerly used to infer lists (Powalski et al., 2021), named tuples (Dwojak et al., 2020), or even more complex structures (Townsend et al., 2021), it was often achieved in an autoregressive manner, without any architectural changes. A model intended for the generation of *unstructured* text in natural language was used to infer an output with formal *structure*. In contrast, we exploit regularities and relationships within the output data and employ a grammar-constrained decoding process (Section 2.5).

Specifically, we focus on the text-to-table inference with applications to problems such as extraction of line items, key information extraction of multiple properties, joint entity and relation extraction, or knowledge base population. Tables as we understand them are equivalent to database tables and defined as a set of values structured in horizontal rows and vertical columns identifiable by name.

From receipts and invoices, through paycheck stubs and insurance loss run reports, to scientific articles, real-world documents contain explicitly or implicitly tabular data to be extracted. These are not necessarily represented as a table *per se* within the input document, e.g., the currency name on the invoice or policy number on the loss run can be mentioned once and be related to all the line items within. In other cases, the evidence one intends to comprehend and represent as a table may be available in free-text only, as can be found in problems of joint entity and relation extraction (see Figure 1-2). Finally, the data may require some postprocessing, such as the normalization of dates, before returning them to the end-user.

1.1 Limitation of Current Approaches

Admittedly, models based on the transformer encoder-decoder or decoder achieve remarkable results in generating complex, formalized outputs, such as computer programs or JSON files (Chen et al., 2021; Townsend et al., 2021). Nevertheless, we hypothesize that changes leading to the *explicit*

* equal contribution
firstname.lastname@snowflake.com

Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 2454–2472
March 17-22, 2024

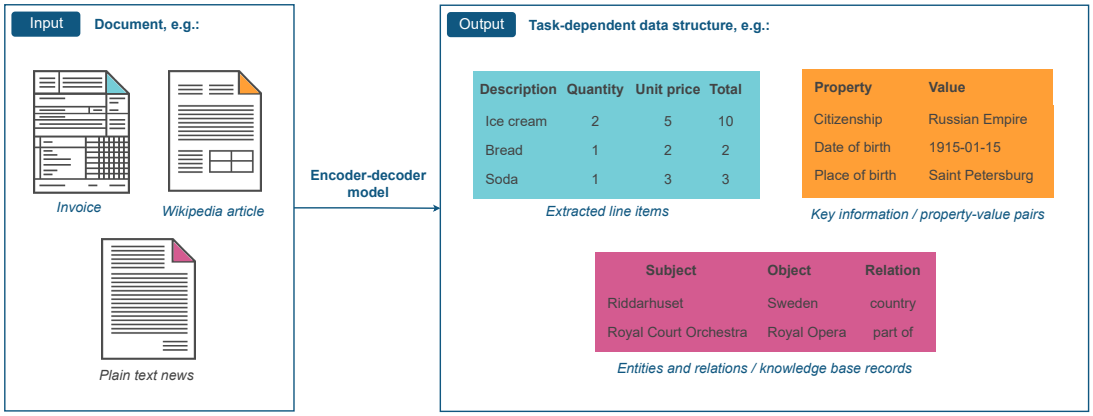

Figure 1: Reinterpreting diverse tasks under a unified paradigm: all these tasks essentially require generating a table based on a given context. While they were not previously seen in this light, we reinterpret them as text-to-table tasks, bringing them together under a single paradigm and directly model the table in the output. This unification has led to significant improvements in each task.

modeling of structured data can outperform the said *implicit* decoding that models long-range syntax dependencies sequentially and does not guarantee the formal validity of produced outputs.

While generating in a particular predefined order (e.g., left-to-right, row-by-row), such approaches have a few drawbacks. Firstly, error propagation that causal models may show after skipping some cells or answering them incorrectly. This flaw may start a chain reaction and directly influence the subsequent cells' generation, causing error propagation and a rapid decline in table quality. Strikingly, an error propagation issue is known in Neural Machine Translation when the right part of the generated sentence used to be worse than the left one (Wu et al., 2018). Therefore, previous approaches to table generation employed preventive measures to keep the table layout under control (Wang et al., 2019) and limit the negative effect of error propagation. Secondly, the answers are forced; the model that cannot give a proper answer consistently has lower confidence and dispersed probability over multiple possibilities. Therefore, we use logit-based confidence to guide the generation process, emergently achieving the property of abstaining from generating answers when the model does not indicate high confidence. Thirdly, the formatting of the table plays a role, and the order of columns may be treated as a hyperparameter in the previous approaches (Wang et al., 2019; Dwojak et al., 2020). For example, performing generation in a predefined and not optimized order may lead to the case when the model is asked about, e.g., date of birth of the person that still needs to be

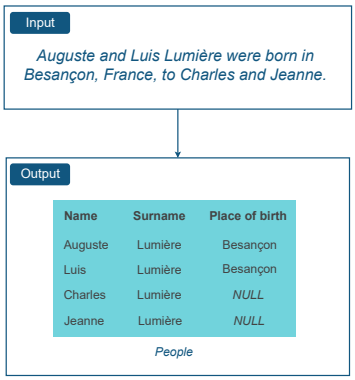

Figure 2: Example of text-to-table generation given plain text input. Concurrent extraction and grouping of the detected entities simplifies the process and may mitigate error accumulation.

specified. Therefore, we want the model to learn the optimal order of the generation as part of the task itself without any implicit human guidance.

Significantly, the advantage the encoder-decoder framework has is that it can cover problems mentioned above in one end-to-end trainable process, thus simplifying the pipeline and reducing the accumulation of errors along the way. At the same time, since extracted data is already in the form the end user requires, one is able to use it directly for downstream application without further processing steps.

1.2 Contribution and Related Works

The specific contribution of this work includes (1) equipping transformer models with permutation-based decoder training to allow comprehending complex, role-dependent relationships in a series of similar objects we represent as a table,

(2) a sequential, grammar-constrained decoding mechanism which generates table content cell-by-cell, in a dynamic, data-dependent order, and (3) introduction of tabular attention bias to the decoder. The novelty of our approach can be better understood in the context of related works.

Decoding of data structures. A few authors attempted the problem of table generation in the encoder-decoder framework. Zhong et al. (2020) proposed a table recognition model consuming input images and decoupled the problem into unconstrained table and cell content generation. In comparison, (1) we use a single constrained decoder comprehending both table structure and its content; (2) we tackle problems of text-to-table inference where the presence of a table at the model input is optional. Recently, Wu et al. (2022) introduced a model relying on constrained decoding of table and tabular embeddings similar to ours. We share their motivation and idea but differ as (1) our method is not restricted to a predefined, row-by-row decoding order and uses a permutation-based training procedure aligned with the use of optimal, model-guided cell permutation during inference; (2) we assume the explicit prediction of the number of rows upfront (before the table decoding starts), instead of allowing the model to stop the generation process after any completed row. The advantage of this approach is discussed in Section 2 and proven by a series of experiments reported in Section 3.

The encoder-decoder model was previously used *as is*, to infer lists and tuples separated with special characters (Powalski et al., 2021; Dwojak et al., 2020). Similarly, Townsend et al. (2021) experimented with the generation of more complex data types represented as XML, JSON, or Python's string representation. In contrast to previous approaches, we do not rely on *implicit* modeling of the formal structure of the output but opt for *explicit* structure generation.

Finally, a text-to-structure approach was recently taken by Lu et al. (2021) for event extraction. The authors used trie-based constrained decoding with event schema injected as the decoder prompt. It resembles our approach to constrained table generation, though they rely on only one proper decoding order resulting from the assumed tree linearization strategy. Moreover, the authors found it challenging to train the structure generation model directly and thus trained it on simple event substructures first. In contrast, we can directly train the structure decoder, and our permutation-based method allows one to generate the structure *flexibly*, in an arbitrary order dynamically guided by the decoding algorithm.

Flexible generation. Even though permutation-based training, which allows for output generation in any order, is of minor usability in the task of LM, it was validated by Stern et al. (2019) for machine translation and by Song et al. (2021) for summarization. Accordingly, Stern et al. (2019) proposed to equip a transformer with the insertion operation, realized by interpreting an additional number generated with the token as the position in the output sequence to which the insertion should be performed. This framework allows for the flexibility of the decoding process, understood as the possibility of stubbing the output sequence with tokens that the model recognizes with high confidence first and then gradually adding more details in the later iterations. In contrast, since the whole output sequence is passed through the decoder anyway, our one cell-decoding step is implemented by sampling all cells at once and then choosing the best-scored ones to be inserted at its location while disregarding others. In the ablation studies we evaluate how the number of cells inserted at once influence the decoding speed and quality, as higher values indicate more cells generated in parallel.

Permutation-based language modeling. The effectiveness of the permutation-based language modeling objective was demonstrated by Yang et al. (2019) who conditioned the BERT-like model to work with the AR objective. However, while the nature of the LM task allowed them to perturb the factorization order of the input sequence arbitrarily, our table-decoding problem requires additional constraints to account for the fact that each cell may consist of several tokens. Thus, the factorization order of blocks of tokens (representing cells) is permuted, while causal order is assumed within the cell. For permutation-invariance and table-awareness on reversed tasks (i.e., table-to-text), we refer the reader to (Wang et al., 2022).

2 STable — Text-to-Table Framework

Serialized representation of the table permits to treat it as a text sequence, and hence, use text-centric methods to perform an autoregressive generation of the output sequence by employing a vanilla Transformer decoder. However, this approach does

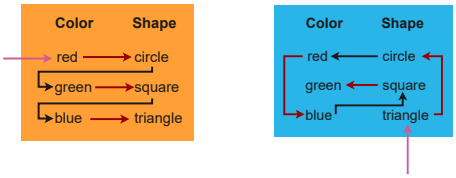

Figure 3: A comparative illustration of the training examples under linearized versus permuted cell ordering. The left panel depicts a typical linearized ordering, following a top-down, left-to-right progression. The right panel presents a permuted ordering example where cells are filled in a non-sequential order.

not exploit the two-dimensional structure of the table as it expands the answer sequentially and utilizes only uni-directional context.

Consequentially, two challenging problems arise. Firstly, how to approach the fact that some information in the table may depend on other cells (e.g., name and surname or the same tax rate for similar items on a receipt) while some may not be dependent (prices of different articles on the shopping list). In general, a model possesses flexibility with respect to this dependence-independence assumption when it can leverage dependencies during decoding but is not forced to do so in any specific order. Our idea (presented in Figure 3) is to solve this problem by delaying the generation of the most challenging and complex answers to later stages and conditioning them on the already generated answer.

Moreover, the decoding must remain free of train-inference discrepancies. Generally, the train-inference alignment means that the state of the table at every step while decoding a particular example must also be possible to achieve in the training phase. Formulating the training that allows for flexible cell generation without providing any additional information remains a non-trivial problem. We rise up to the challenge and demonstrate the solution below.

2.1 Decoding Invariant Under Cell Order

Instead of generating the cell values in a top-down, left-to-right manner as previously seen in the literature (e.g., Wu et al., 2022), we perform the pretraining by maximizing the expected log-likelihood of the sequence of cell values over all possible prediction orders. More specifically, suppose that we are given a document containing a table with

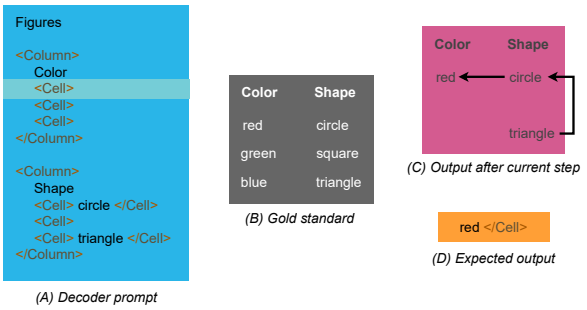

Figure 4: A training example depicting how the answer `red` is produced based on the partially filled cells containing `circle` and `triangle`. (A) The highlighted cell denotes a position where the expected `red </Cell>` should be predicted autoregressively starting from a `<Cell>` token. A successfully decoded cell will lead to the state visible in (C), i.e., the partially decoded gold standard table (B). The generation order of a table is random for each example in the training.

row labels $\mathbf{r} = (r_1, \ldots, r_N)$,[1] and column labels $\mathbf{c} = (c_1, \ldots, c_M)$, which we will collectively denote $\mathbf{h} = (\mathbf{r}, \mathbf{c})$. A linear ordering of the table cells can be represented with a bijection

$$\sigma \colon \{1, 2, \ldots, C\} \to \{1, \ldots, N\} \times \{1, \ldots, M\},$$

where $C = NM$ is the number of cells, so that $\sigma(n) = (i, j)$ are the row and column coordinates of the n-th cell in the ordering. Given such a σ and cell values $\mathbf{v} = (v_{ij})_{i \le N, j \le M}$, we factorize the likelihood of $\mathbf{v}$ given $\mathbf{h}$ as

$$p_\theta(\mathbf{v}|\mathbf{h}) = \prod_{n=1}^{C} p_\theta\big(v_{\sigma(n)} \big| (v_{\sigma(k)})_{k<n}, \mathbf{h}\big), \quad (1)$$

and using this factorization, we maximize the expected log-likelihood

$$\frac{1}{C!} \sum_{\sigma} \sum_{n=1}^{C} \log p_\theta\big(v_{\sigma(n)} \big| (v_{\sigma(k)})_{k<n}, \mathbf{h}\big) \quad (2)$$

over θ. The likelihoods $p_\theta\big(v_{\sigma(n)} \big| (v_{\sigma(k)})_{k<n}, \mathbf{h}\big)$ themselves can be factorized according to the standard auto-regressive approach as

$$p_\theta\big(v_{\sigma(n)} \big| (v_{\sigma(k)})_{k<n}, \mathbf{h}\big) = \\ = \prod_{t=1}^{\ell(v_{\sigma(n)})} p_\theta\big(v^t_{\sigma(n)} \big| (v^i_{\sigma(n)})_{i<t}, (v_{\sigma(k)})_{k<n}, \mathbf{h}\big) \quad (3)$$

[1] In practice, usually there are no row labels; however, in the decoder, the special tokens used for distinguishing rows take this role.

where $\ell(v_{\sigma(n)})$ is the length of $v_{\sigma(n)}$ represented as a sequence of tokens $(v^i_{\sigma(n)})_{i \le L}$. In practice, the expected log-likelihood is estimated by sampling bijections σ at random.

Training example is presented in Figure 4.

2.2 Tabular Attention Bias

We base our attention computation method on the relative bias idea popularized by the T5 model. Given a text consisting of T tokens, in the vanilla T5 model, raw attention scores α_{ij} for tokens i and j (with $0 \le i, j < T$) are modified by introducing a bias term: $\alpha'_{ij} = \alpha_{ij} + \beta_{ij}$ where $\beta_{ij} = W(i-j)$ is a trainable weight, depending on the relative sequential position of these tokens (Raffel et al., 2020).

We modify the decoder's self-attention by extending it with two new bias terms, defined below. The *tabular bias* τ_{ij} encodes the relative position of table cells in which the tokens lie, while the *local sequential bias* λ_{ij} corresponds to the relative sequential position of tokens belonging to the same cell.

$$\tau_{ij} = \begin{cases} R(r_i - r_j) + C(c_i - c_j) & \text{if } r_j > 0 \\ R_0 + C(c_i - c_j) & \text{if } r_j = 0 \end{cases},$$
$$\lambda_{ij} = \begin{cases} L(i-j) & \text{if } (c_i, r_i) = (c_j, r_j) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where (c_i, r_i) are cell coordinates as given by its 1-based column and row indices (with 0 reserved for the header row/column), and $R(k)$, $C(k)$, $L(k)$ and R_0 are trainable weights. The special case with $r_j = 0$ corresponds to the situation when the key/value token lies in the column header, in which case we want to use the same bias independent of the row of the query token, due to the different nature of the relation between two cells, and a cell and its column header. After these adjustments, the final attention score takes the form $\alpha'_{ij} = \alpha_{ij} + \beta_{ij} + \tau_{ij} + \lambda_{ij}$, where β_{ij} is the bias term defined earlier.

2.3 Predicting Number of Groups

Although the previous work of Wu et al. (2022) assumed the table is finalized when the appropriate special token explicitly appears in the output, our systematic study shows that the explicit prediction of the number of groups yields better results (see Section 4 for comparison). This explicit prediction is achieved with a linear layer that consumes the first input token's embedding to perform a prediction on the number of groups. During the training stage, the layer's output is scored against the known number of groups using MSE loss, while during the inference, it is used as a predictor declaring the number of groups to populate the template with.

2.4 Inference with Model-Guided Cell Order

Since the model was trained assuming a permuted factorization of cell ordering, in expectation, the model learned to understand all possible variants of a partially-filled table and predict values for all empty cells. Because each step in the generation process implicates uncertainty that should be globally minimized, we propose to estimate the optimal table decoding algorithm by greedily finding the cell that minimizes this uncertainty at each step.

The decoding employs an outer loop that progresses cell-by-cell, an inner loop that generates each cell that is yet to render, and a selection heuristics that determine which cell, from all the finalized in the inner loop, should be added to the outer loop. The heuristic we use selects the cell containing the token with highest probability among all predicted (Figure 5). The detailed study of this and alternative selection criteria is presented in Appendix C.

In the inner loop, each cell is decoded until the special token determining the end of cell generation is placed. As the inner loop generates each cell autoregressively and independently from other cells, the process can be treated as generating multiple concurrent threads of an answer and is well parallelizable. In the worst case, it takes as many steps as the number of tokens in the most extended cell.

After being selected by a heuristic, the cell from the inner loop is inserted into the outer loop, and made visible to all other cells, while the cells that were not selected are to be reset and continuously generated in the future steps until they are chosen by a heuristic (see pseudocode in Appendix A).

2.5 Grammar-Constrained Decoding

As a result of the model design, incorrect tables cannot be generated. Part of these rules is explicit (e.g., we overwrite logits, so it is impossible to emit particular tokens such as the end-of-cell when no cell is opened), whereas part of the rules results implicitly from the algorithm (template-filling setting, where the well-formulated table is always ensured).

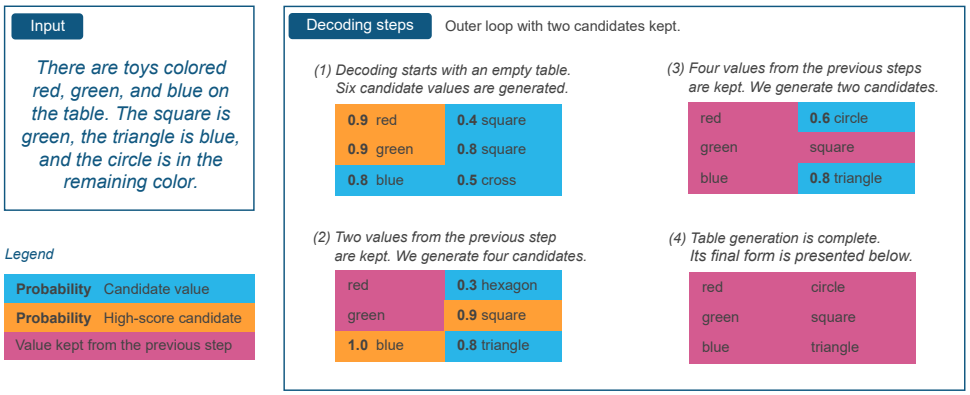

Figure 5: A possible progression of decoding a table given the text on the input. Since the probabilities guide the decoding order, the circle's color that was not explicitly stated in the text is determined at the last step.

Table 1: Results on public and private datasets assuming task-specific metrics. The results of a sequence-to-sequence baseline that learns and generates tables as text are provided in the *Linearized* column. Mean and STD over three runs. The † symbol denotes our TILT training. Underline signifies our model is significantly better than baseline.

Dataset	State-of-the-Art Reference		Linearized	Our Model	
PWC★	T5 2D (Borchmann et al., 2021)	26.8	27.8 ± 1.0	**30.8** ± 0.5	T5 2D + STable
CORD	TILT (Powalski et al., 2021)	**96.3**	92.4 ± 0.7	95.6 ± 0.2	TILT† + STable
Rotowire					
Player	Text-to-Table (Wu et al., 2022)	**86.8**	84.5 ± 0.7	84.5 ± 0.2	
Team	*(BART backbone)*	**86.3**	83.8 ± 0.9	84.7 ± 0.2	T5 + STable
DWIE	KB-both (Verlinden et al., 2021)	**62.9**	60.2 ± 1.5	59.2 ± 1.5	T5 + STable
Recipe…		71.9	60.1 ± 0.3	**75.5** ± 1.6	
Payment…	TILT†	77.0	72.0 ± 2.3	**79.1** ± 0.9	TILT† + STable
Bank…		61.1	58.7 ± 4.9	**69.9** ± 4.8	

3 Experiments

In addition to state-of-the-art reference and our results, we provide scores of the same backbone models (T5, T5 2D, and TILT) while a table linearization strategy follows the assumptions of Wu et al. (2022)'s baselines. Appendix D covers details of training procedure.

Metrics. We rely on the original metrics for all but the DWIE dataset, i.e., GROUP-ANLS for PWC★, F1 for CORD, and non-header exact match cell F1 for Rotowire (other variants proposed by the authors are reported in Table 7 in Appendix D). Use of the original DWIE metric was not possible, as it assumes a step-by-step process. In contrast, we tackle the problem end-to-end, i.e., return *(object, relation, subject)* tuples without detecting all entity mentions within the document and their locations. To ensure a fair comparison, we use the F1 score calculated on triples; that is, we require the model to return the exact match of the triple. Such a setup is very demanding for encoder-decoder models as the convention in DWIE is to require *object* and *subject* to be returned in the longest form of appearance in the document.

Pretraining and Adaptation. Due to the switch to permutative training and the addition of the regression head, there is a significant change in the model objective. Consequently, we anticipated the necessity of the model adaptation phase. It consists of the pretraining stage equivalent to the one conducted by authors of the TILT model (Powalski et al., 2021) extended by Natural Questions (Kwiatkowski et al., 2019) and WebTables[2] datasets. To utilize WebTables we rendered webpages, from which the tables were scraped and taught models to extract table contents from webpages. The said stage is applied to all T5+STable, T5 2D+STable, and TILT+STable models.

[2]https://webdatacommons.org/webtables/

Complex Information Extraction. The problem of information extraction involving aggregated data types, where one may expect improvement within the document-to-table paradigm, is prevalent in business cases. Nevertheless, the availability of public datasets here is limited to PWC* (Borchmann et al., 2021; Kardas et al., 2020) and CORD (Park et al., 2019).

In the case of PWC*, the goal is to determine model names, metrics, datasets, and performance, given the machine learning paper as an input. CORD assumes the extraction of line items from images of Indonesian receipts, among others. To determine the gain from our STable decoder, the experiments are conducted with state-of-the-art encoder-decoder models proposed for these datasets (T5 2D and TILT), assuming the same training procedure (Borchmann et al. (2021); Powalski et al. (2021); see Appendix D for details).

Additionally, due to the sparsity of public benchmarks of this kind, we decided to provide results on three confidential datasets. They assume, respectively, (1) the extraction of payments' details from *Payment Stubs*, (2) *Recipe Composition* from documents provided by a multinational snack and beverage corporation, as well as (3) account balances from *Bank Statements*. These are covered in details in Appendix E and addressed by the TILT+STable model with vanilla TILT as a reference.

As summarized in Table 1, we outperformed state-of-the-art information extraction models on several datasets. At the same time, the CORD where we underperform was previously considered solved, e.g., Powalski et al. (2021) point that TILT's output and the reference differed insignificantly. We used it in the experiment as a safety check to determine whether the model can maintain almost-perfect scores after applying the STable decoder. Consequently, we omit it in the ablation studies.

The rest of the experiments were conducted assuming the vanilla T5 model (Raffel et al., 2020) equipped with the STable decoder of our proposal.

Joint Entity and Relation Extraction. To demonstrate the broad applicability of the model, we consider the problem of a joint entity and relation extraction on the example of the DWIE dataset (Zaporojets et al., 2021). Here, the tuples consisting of entities and one of the sixty-five relation types are to be determined given a plain-text news article. Despite not outperforming a multi-step state-of-the-art model, we achieved high scores and were the first to prove that the problem can be successfully approached end-to-end using an encoder-decoder framework. Here, the T5+STable's errors and issues reflect the very demanding assumptions of DWIE, where it is required to return *object* and *subject* in the longest form of appearance in the document.

Reversed Table-to-Text. Finally, following Wu et al. (2022) we evaluate our approach on the Rotowire table-to-text dataset in a reverse direction, i.e., generate tables from text (Wiseman et al., 2017). Consequently, the complex tables reporting teams and player performance are generated given the game description. Results from Table 1 show that our T5+STable model can deliver an improvement over the *Linearized* T5 model on Rotowire Team. The fact that *Linearized* BART from Wu et al. (2022) outperforms our *Linearized* T5 baselines on Rotowire Team and Player datasets by 2.5 and 2.1 points, respectively, suggests that it has a better capacity as a backbone for this task. Several of the ablation studies from the next section were designed to shed light on this subject.

The results of our model (Table 1) demonstrate a significant improvement over the simple sequence-to-sequence generation of tables linearized as sequences on three out of five public datasets. As expected, it yields better results in cases where there is a considerable interdependency between values in a row and no clear, known upfront name distinguishes it from other rows. Note that, e.g., in Rotowire, it suffices to correlate all statistics with team or player name, which is always inferred first due to the employed linearization strategy. The order of columns being decoded is a hyperparameter in the case of linearization. In contrast, the power of STable comes from learning it from the data itself.

4 Ablation Studies

Models were trained three times with different random seeds on the Rotowire, DWIE, and PWC* datasets. To reduce the computational cost, we relied on *base* variants of the models reported in Section 3 – please refer to Appendix D for detailed specifications and results. While results on a single dataset can be considered noisy, aggregation over a diverse set of them helps diminish the randomness's impact and stabilize results on the new ones.

(1) Semi-templated Expansion. To compare our method of group prediction with a regression-free

Table 2: Results of studies (1), (2), (3), and (5). Modified models in relation to complete STable. See Appendix D for per-dataset results.

Model	Score	Change	
Complete STable	62.9 ± 1.0	—	
Semi-templated expansion	61.4 ± 0.1	-1.5	(1)
Fixed causal order	60.0 ± 0.4	-2.9	(2)
Decoding constraint			(3)
Column-by-column	62.4 ± 0.6	-0.5	
Row-by-row	62.1 ± 0.6	-0.8	
L→R and T→B	62.0 ± 0.5	-0.9	
No distant rows	62.2 ± 0.5	-0.7	
Sequential decoder bias only	3.9 ± 0.1	-59.0	(5)
Sequential and header bias	33.2 ± 0.3	-29.7	

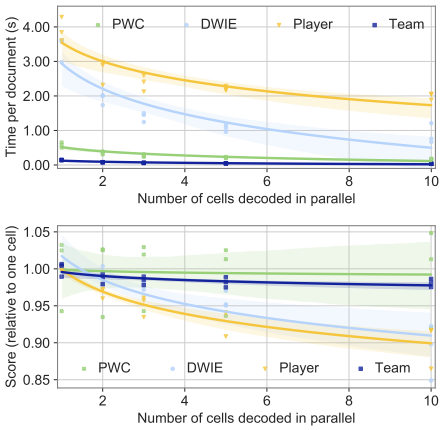

Figure 6: Results of decoding ablation (4). Three runs for 1, 2, 3, 5, and 10 cells decoded in parallel.

alternative, we allow the model to work in a semi-templated manner, where the template is infinite, and the decoding stops when the group with *NULL*-only tokens is generated. For this method, we add such a group at the bottom of the table during the training to comply with the inference. The model performance is significantly below the STable reference, suggesting explicit group number prediction superiority.

(2) Non-Permutative Training. To measure the importance of understanding the bidirectional contexts within the model, we abstain from permutation-based training in our study and choose the predefined factorization order. Here, a *fixed causal order* model that reads tables from left to right and from top to bottom is evaluated. This alternative is in line with text-to-table approach of Wu et al. (2022). As shown in Table 3, the lack of permutative training we introduced in Section 2 leads to significantly worse scores.

(3) Constrained Cell Order. Whereas the permutation-based training allows for great flexibility, the questions posed here are whether limiting the model's ability to discover new cells can be of any value. Methods in this group assure either that the *column-by-column* constrained model predicts the whole column before decoding a new one, the *row-by-row* constrained model predicts the whole row before entering a new one, whereas *L→R and T→B* is a combination of both (ensures row-by-row inference from left to right). The *no distant rows* constraint forces the decoding to have empty cells only on the bottom of each column, thus avoiding skipping cells in the decoding while moving down.

As shown in Table 3, all but column-by-column constraint lead to a decreased scores. At the same time, the mentioned performs on par with STable's model-guided inference (Section 2.4), and both are better than the method with left-to-right decoding order. These results suggest that (1) our method does not require constraining the decoding order, (2) it seems to implicitly incorporate the column-by-column constraint, and (3) it is helpful to be elastic w.r.t. decoding order within the column.

(4) Parallelization of Cell Decoding. As outlined in Section 2.4, one may allow multiple candidates to be kept in each decoding step to shorten the inference time while expecting the performance to degrade to some extent. Results of experiments that follow this observation are presented in Figure 6. One may notice that processing time varies across the considered datasets since it depends mainly on the input sequence length (ranging from $1k$ for Rotowire to $6k$ for PWC) and the sizes of the table to infer (we infer as many as 320 cells for the Player table). Parallelization of cell decoding significantly reduces the total per-document processing time — up to five times for DWIE in the conducted experiments. Interestingly, speed-up does not necessarily lead to a decrease in scores; e.g., in the case of the Team table, there is four times better processing time when ten cells are inferred at once, whereas the scores achieved by the model remain comparable. It can be attributed to the fact that there are almost no inter-cell dependencies and always only two rows to infer — one for each team playing. Since the performance changes w.r.t. this parameter is heavily data-dependent, its value should be obtained experimentally for each dataset separately. Additionally, we argue that it is beneficial to use large values to speed up the

train-time validation as it maintains a correlation with higher-scoring lower parameter values that can be employed during test-time inference.

(5) Tabular Attention Biases. In comparison with the initially introduced two relations (between cells and within cells), removing them and using only 1D global bias disrupts the permutation-based training as the model scores degrade since it cannot assign answers to correct columns. However, additional incorporation of the header name (by attending only to row with headers, $r_j = 0$ in Equation 4) leads to significant improvement, but it is still below the full model. Detailed analysis showed that the model could not benefit from 1D global bias, as (1) the distance between cells and header is too large for the first cells in the training since they are randomly chosen from any position within the table, and (2) a table itself is considerably bigger, as in permutation-based training we assumed that every cell in the table is generated, while for the linearized model, the headers are generated by the model, and a part of them can be skipped, thus reducing the size of the table. The consistent improvements on four datasets indicate that proposed tabular attention biases enhance table-modeling efforts.

5 Limitations

The state-of-the-art performance of STable is its foremost advantage, while the constraining factors come from different aspects. Of them, the generated sequence's length seems to incure the most long-term cost during inference, while the increase in training time per example is a short-term obstacle. The underlying issue is that the full table context negatively influences the computational cost of the attention on the decoder side. This however is also the case for the family of encoder-decoder models generating the whole table such as these proposed by Wu et al. (2022) or Townsend et al. (2021). A possible solution here is a model with table context limited to the row and column a given table cell belongs to. Such a change would have a positive impact on the memory consumption in the decoder, as self-attention complexity decreases from $\mathcal{O}(NM)$ to $\mathcal{O}(N+M)$, where N, M denotes the number of rows and columns respectively. The exploitation of this optimization is an interesting future direction.

To navigate the intricacy of the order employed by the STable framework, we performed a systematical analysis that did not conclude in finding a visible decoding pattern that could be described formally beyond the observation already provided in Figure 5 and in constrained-decoding ablations. Studying the generation order in the context of data calls for designing a new explainability-related method, which is not in the scope of this work.

6 Summary

We equipped the encoder-decoder models consuming text (T5, T5 2D) and documents (TILT) with the capabilities to generate tables in a data-dependent order. Firstly, an aligned training procedure based on permuting factorization order of cells was presented, and secondly, the parallelizable decoding process that fills the table with values in a flexible and unconstrained order was proposed. The important design choices for both contributions were motivated by an extensive ablation study. The proposed STable framework demonstrates its high practical value by yielding state-of-the-art results on PWC$^\bigstar$ and outperforming linearized models on CORD and Rotowire Team datasets, as well as outperforming reference models on several confidential datasets. The highest gains due to the permutative training were accomplished on the PWC$^\bigstar$ dataset, where 4.0 points ($26.8 \rightarrow 30.8$) amounts to 14.9% relative improvement, while the 8.8 point gain on Bank Statements ($61.1 \rightarrow 69.9$) exceeds 14.4% relative improvement.

Acknowledgments

The Smart Growth Operational Programme partially supported this research under projects no. POIR.01.01.01-00-0877/19-00 (*A universal platform for robotic automation of processes requiring text comprehension, with a unique level of implementation and service automation*) and POIR.01.01.01-00-1624/20 (*Hiper-OCR - an innovative solution for information extraction from scanned documents*).

References

Łukasz Borchmann, Michał Pietruszka, Tomasz Stanislawek, Dawid Jurkiewicz, Michał Turski, Karolina Szyndler, and Filip Graliński. 2021. DUE: End-to-end document understanding benchmark. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan,

Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Tomasz Dwojak, Michał Pietruszka, Łukasz Borchmann, Jakub Chłędowski, and Filip Graliński. 2020. From dataset recycling to multi-property extraction and beyond. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 641–651, Online. Association for Computational Linguistics.

Marcin Kardas, Piotr Czapla, Pontus Stenetorp, Sebastian Ruder, Sebastian Riedel, Ross Taylor, and Robert Stojnic. 2020. AxCell: Automatic extraction of results from machine learning papers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8580–8594, Online. Association for Computational Linguistics.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. UNIFIEDQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1896–1907, Online. Association for Computational Linguistics.

Geewook Kim, Teakgyu Hong, Moonbin Yim, JeongYeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. 2022. Ocr-free document understanding transformer. In *Computer Vision – ECCV 2022*, pages 498–517, Cham. Springer Nature Switzerland.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1378–1387, New York, New York, USA. PMLR.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*.

Yaojie Lu, Hongyu Lin, Jin Xu, Xianpei Han, Jialong Tang, Annan Li, Le Sun, Meng Liao, and Shaoyi Chen. 2021. Text2Event: Controllable sequence-to-structure generation for end-to-end event extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2795–2806, Online. Association for Computational Linguistics.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *CoRR*, abs/1806.08730.

Seunghyun Park, Seung Shin, Bado Lee, Junyeop Lee, Jaeheung Surh, Minjoon Seo, and Hwalsuk Lee. 2019. CORD: A consolidated receipt dataset for post-ocr parsing. In *Document Intelligence Workshop at NeurIPS*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Rafał Powalski, Łukasz Borchmann, Dawid Jurkiewicz, Tomasz Dwojak, Michał Pietruszka, and Gabriela Pałka. 2021. Going full-TILT boogie on document understanding with text-image-layout transformer. In *Document Analysis and Recognition – ICDAR 2021*, pages 732–747, Cham. Springer International Publishing.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(1).

Kaiqiang Song, Bingqing Wang, Zhe Feng, and Fei Liu. 2021. A new approach to overgenerating and scoring abstractive summaries.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5976–5985. PMLR.

Benjamin Townsend, Eamon Ito-Fisher, Lily Zhang, and Madison May. 2021. Doc2dict: Information extraction as text generation. *CoRR*, abs/2105.07510.

Severine Verlinden, Klim Zaporojets, Johannes Deleu, Thomas Demeester, and Chris Develder. 2021. Injecting knowledge base information into end-to-end joint entity and relation extraction and coreference resolution. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1952–1957, Online. Association for Computational Linguistics.

Fei Wang, Zhewei Xu, Pedro Szekely, and Muhao Chen. 2022. Robust (controlled) table-to-text generation with structure-aware equivariance learning.

Xing Wang, Zhaopeng Tu, Longyue Wang, and Shuming Shi. 2019. Self-attention with structural position representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1403–1409, Hong Kong, China. Association for Computational Linguistics.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Lijun Wu, Xu Tan, Di He, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. Beyond error propagation in neural machine translation: Characteristics of language also matter.

Xueqing Wu, Jiacheng Zhang, and Hang Li. 2022. Text-to-Table: A new way of information extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2518–2533, Dublin, Ireland. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Klim Zaporojets, Johannes Deleu, Chris Develder, and Thomas Demeester. 2021. DWIE: An entity-centric dataset for multi-task document-level information extraction. *Information Processing & Management*, 58(4):102563.

Xu Zhong, Elaheh ShafieiBavani, and Antonio Jimeno Yepes. 2020. Image-based table recognition: Data, model, and evaluation. In *Computer Vision – ECCV 2020*, pages 564–580, Cham. Springer International Publishing.

A Table Decoding Algorithm

The algorithm presented above operates on the output of the encoder model and reuses the cached encoded representations that are considered to be a part of the DECODERMODEL for brevity. Another important characteristic of the DECODERMODEL introduced for conciseness of the pseudocode is that it produces all cell tokens and handles the sequential text decoding on its own.

The decoding employs an OUTERLOOP, parametrized by the k parameter (denoting the parallelization of cell decoding) that progresses cell-by-cell, the INNERLOOP function that generates each cell that is yet to render, and OUTERCRITERION — a selection heuristics that determine which cell, from all the finalized in the inner loop, should be added to the outer loop. The INNERCRITERION is a heuristic we utilize that selects the cell with the maximum probability for its tokens' predictions (Figure 5).

In the INNERLOOP, each cell is decoded until the special token determining the end of cell generation is placed. As the INNERLOOP generates each cell autoregressively and independently from other cells, the process can be treated as generating multiple concurrent threads of an answer and is well parallelizable. In the worst case, it takes as many steps as the number of tokens in the most extended cell.

After the selection by the OUTERCRITERION heuristic, the cell from the inner loop is inserted into the outer loop, and made visible to all other cells, while the cells that were not selected are to be reset and continuously generated in the future steps until they are chosen by the OUTERCRITERION heuristics.

B Negative Result: Prevention of Column Order Leakage

In the approach outlined in Section 2, the sequence of column labels $\mathbf{c}$, on which the likelihoods are conditioned, may leak additional unwanted information to the decoder. If the data in the document are indeed formatted as a table, and the order of labels in $\mathbf{c}$ matches the column order, the model might learn to extract cells by location, instead of using the actual semantics of the cell label. However, during inference, while we know which entities we want to extract from the document, we are not given the order in which they appear, which can be perceived as a serious train-inference discrepancy.

To remedy this problem, we tried to further modify the training objective (See Figure 7). Denote by $\mathcal{C}$ the set of all non-empty sequences of distinct column labels. Instead of all the cells $\mathbf{v}$, we can predict only the cells $\mathbf{v_c}$ corresponding to a sequence $\mathbf{c} \in \mathcal{C}$ of columns, in the order defined by the order of columns in $\mathbf{c}$. The expected log-likelihood over all $\mathbf{c} \in \mathcal{C}$ can be then expressed as

$$\log p_\theta(\mathbf{v}|\mathbf{h}) = \frac{1}{|\mathcal{C}|} \sum_{\mathbf{c} \in \mathcal{C}} \log p_\theta(\mathbf{v_c}|\mathbf{r}, \mathbf{c}), \quad (5)$$

where $p_\theta(\mathbf{v_c}|\mathbf{r}, \mathbf{c})$ decomposes according to the discussion in Section 2.

In practice, we found it to have no relevant impact on the training process. It did not lead to significant changes in evaluation scores when used in the supervised pretraining stage or on a downstream task. Consequently, we abandoned the idea and did not use it for any of the models reported in the paper. This study helps us state that the model learns the semantics of the cell labels without a need for regularization.

C Inner/Outer Loop Decision Criteria

The heuristic we test selects the cell in the outer loop based on the minimal or maximal inner score. Such inner score is calculated in three different ways: by taking the minimal, maximal, and mean of the token's logits score. The results, presented in Table 3, point to the lesser importance of choosing the inner scoring method, while the choice of the outer loop heuristics impacts results more significantly. The former is the desired behavior since the algorithm we proposed in Section 2.4 is based on the assumption that it is beneficial to decode cells starting from those with the model's highest confidence. On the other hand, as there is a significant variance depending on the dataset chosen (see Appendix D), these and other inference parameters can be subject to cost-efficient, task-specific hyperparameter optimization.

D Details of Experiments and Ablation Studies

All models were trained three times with different random seeds. We relied on *large* variants of the models for experiments in Table 1, and on *base* variants for the ablation studies. These are ana-

Algorithm 1 Table Decoding Algorithm of our proposal.

procedure OUTERLOOP(k)
    $T \leftarrow 0_{n,m,l}$ ▷ $n \times m$ table with l padding tokens per cell
    $C \leftarrow 0_{n,m}$ ▷ current cell status (decoded or not)
    while SUM(C) $< nm$ **do** ▷ while there is a cell to decode
        $T', L \leftarrow$ INNERLOOP(T, C) ▷ create complete table candidate T' and cell scores
        $\mathcal{B} \leftarrow$ OUTERCRITERION(L) ▷ sequence of coordinates sorted according to scores
        for $c \leftarrow 1, k$ **do** ▷ for k best cells from T'
            $i, j \leftarrow \mathcal{B}_c$ ▷ get coordinates
            $T_{i,j} \leftarrow T'_{i,j}$ ▷ ...copy values to table T accordingly
            $C_{i,j} \leftarrow 1$ ▷ ...and mark the appropriate cell as already decoded
        end for
    end while
    return T
end procedure

procedure INNERLOOP(T, C)
    $L \leftarrow 0_{n,m}$ ▷ scores for each cell in $n \times m$ table
    $T' \leftarrow T$ ▷ inner loop's table copy
    parfor $i \leftarrow 1, n$ **do** ▷ for each table row
        parfor $j \leftarrow 1, m$ **do** ▷ ...and each table cell processed in parallel
            if $C_{i,j} = 0$ **then** ▷ ...if it was not decoded yet
                $s, t \leftarrow$ DECODERMODEL(T, i, j) ▷ produce cell tokens t and their scores s
                $L_{i,j} \leftarrow$ INNERCRITERION(s) ▷ aggregate per-token scores into cell score
                $T'_{i,j} \leftarrow t$ ▷ update table copy
            end if
        end parfor
    end parfor
    return (T', L)
end procedure

procedure INNERCRITERION(s)
    /* Any $\mathbb{R}^n \rightarrow \mathbb{R}$ function. STable assumes max, but we test other in the ablation studies. */
end procedure

procedure OUTERCRITERION(L)
    /* Some $\mathbb{R}^{m \times n} \rightarrow (\mathbb{N} \times \mathbb{N})^{mn}$ function returning a permutation of indices of the input
    matrix L. STable assumes sort of matrix coordinates according to descending values of its
    elements, but we test other functions in the ablation studies. */
end procedure

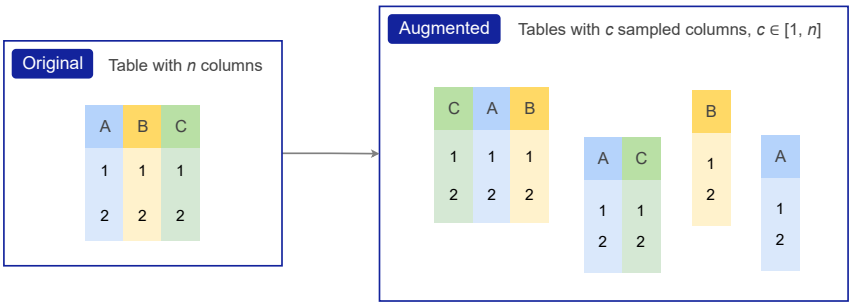

Figure 7: Change in training illustrated as augmentation of permuted sub-tables from the original table.

Table 3: Results of studies on decision criteria. Modified models in relation to complete STable. See Appendix D for per-dataset results.

Model		Score	Change
Complete STable		62.9 ± 1.0	—
Criteria (inner, outer)			
min	max	61.7 ± 0.7	-1.2
mean	max	62.7 ± 0.7	-0.2
mean	min	60.8 ± 0.7	-2.1
min	min	62.1 ± 0.4	-0.8
max	min	61.2 ± 0.2	-1.7

lyzed in Table 3 given the average results over Rotowire, PWC★, and DWIE datasets (see Table 4 for detailed scores).

Hyperparameters. We use task-independent hyperparameters that roughly follow these proposed by the authors of the T5 model for its finetuning, as during our initial experiments, they turned out to be a robust default (see Table 5).

Maximal input sequence lengths were chosen in such a way a fair comparison with reference models was ensured. In particular, we use T5+2D's limit despite the fact one can achieve better results when consuming a more significant part of the input document. Similarly, the max number of updates follows the limit in reference models except for the DWIE dataset, where the state-of-the-art solution is based on the incomparable multi-step pipeline. See Table 6 for these task-specific details.

Software and hardware. All experiments and benchmarks were performed on DGX-A100 servers equipped with eight A100-SXM4-80GB GPUs that feature automatic mixed precision. Our models and references were implemented in PyTorch 1.8.0a0 (Paszke et al., 2019) with CUDA 11.4 and NVIDIA drivers 470.82.01.

E Business Datasets

Due to the sparsity of public benchmarks for complex information extraction, we decided to provide results on three confidential datasets. They assume, respectively, (1) the extraction of payments' details from *Payment Stubs*, (2) *Recipe Composition* from documents provided by multinational snack and beverage corporation, as well as (3) account balances from *Bank Statements*. Their details are covered in the present section and Table 8.

Recipe Composition. The problem faced is extracting proprieties of food ingredients from confidential food manufacturer's documentation. This dataset contains 165 annotated fragments from 55 documents, three pieces for each document, with annotations sourced from the corporation's CRM system.

For each file, there are five tables to be extracted. The first one describes the ingredient's physical and chemical parameters (i.e., parameter name, testing method, range of allowed values, unit of measurement, and testing method details). The second one describes sub-components of the ingredient (i.e., its quantity, name, allergens, ingredient function, and country of origin). The third table informs about the presence of allergens (e.g., their names and binary information about their presence). The last two tables contain a quantity of the allergens (e.g., names and their qualities) as sub-components and caused by contamination retrospectively.

The first table needs to be extracted from the first document fragment, the second table – from the second fragment, and the three last tables – from the third document fragment. Input documents feature tables and fulfilled forms, where properties are presented in the form of text or check-boxes.

The analysis of expected outputs shows a high level of variability concerning the factors of table length (1 to 60 rows) and answer type (either a

Table 4: Per-dataset results of studies (1), (2), (3), and (4). Modified models in relation to Complete STable.

Model	RW Player	RW Team	PWC★	DWIE	
Complete STable (reference)	82.7 ± 0.3	84.1 ± 0.7	27.5 ± 2.2	56.0 ± 1.4	
Semi-templated expansion	80.4 ± 0.5	84.1 ± 0.5	25.0 ± 0.8	56.1 ± 1.0	(1)
Fixed causal order	83.2 ± 0.4	84.3 ± 0.3	26.3 ± 1.6	46.5 ± 0.5	(2)
Decoding constraint					(3)
Column-by-column	82.5 ± 0.4	84.0 ± 0.5	28.4 ± 1.5	54.8 ± 0.8	
Row-by-row	80.2 ± 0.4	83.8 ± 0.4	27.6 ± 1.6	56.8 ± 0.8	
L→R and T→B	83.1 ± 0.5	84.1 ± 0.7	27.7 ± 1.8	53.2 ± 0.5	
No distant rows	82.7 ± 0.5	83.8 ± 0.6	28.1 ± 1.0	54.2 ± 1.2	
Decision criteria (inner × outer)					(4)
min max	81.9 ± 0.4	83.7 ± 0.5	26.5 ± 2.0	54.2 ± 0.8	
mean max	83.0 ± 0.3	83.8 ± 0.8	27.8 ± 1.4	56.1 ± 1.1	
mean min	81.2 ± 1.1	83.7 ± 0.6	26.4 ± 1.9	51.9 ± 0.5	
min min	82.8 ± 0.6	83.8 ± 0.5	27.6 ± 1.3	54.0 ± 0.5	
max min	82.3 ± 0.3	84.5 ± 1.0	20.7 ± 1.6	52.7 ± 0.4	
Sequential decoder bias only	0.3 ± 0.1	0.6 ± 0.3	14.1 ± 0.3	0.6 ± 0.1	(5)
Sequential and header bias	16.0 ± 0.4	45.1 ± 0.4	27.7 ± 2.0	44.2 ± 1.2	

Table 5: Task-independent hyperparameters used across all experiments.

Hparam	Dropout	Batch	Learning rate	Weight decay	Label smoothing	Optimizer
Value	.1	64	1e-3	1e-5	.1	AdamW

Table 6: Task-dependent hyperparameters and training details. (*) Length equal to the one consumed by the baseline model.

Dataset	Max steps		Max input
	Ablation	Final	length
PWC★	500	1,000	6,144*
Rotowire	3,000	8,000	1,024
CORD	—	36,000	1,024
DWIE	4,000	8,000	2,048
Recipe Composition	—	400	2600
Payment Stubs	—		
Bank Statements	—	200	7000

binary value, number, complex chemical name, or a more extended description).

Payment Stubs. The second of our private datasets consists of 110 American payment stubs, i.e., documents obtained by an employee regarding the salary received.

We aim to extract employee and employer names, dates, and payment tables, where each row consists of payment type, hours worked, and payment amount. Since documents come from different companies, their layouts differ significantly.

Due to the straightforward form of information to be extracted, a single annotator annotated each document. We state these were annotated ethically by our paid co-workers.

Bank Statements. The last dataset consists of 131 annotated bank statements. The goal here is to extract bank and customer name, date of issue, and table of account balances (e.g., account number, balance at the beginning of the period, and balance at the end).

Data to be comprehended is partially presented in the document's header and partially in multiple forms (each for one account).

Similar to the Payment Stubs dataset, documents here were issued by different banks and represent a broad spectrum of layouts. The annotation process was the same as for the Payment Stubs dataset.

F Adaptation to Table Structure Recognition Task

Our method by design does not generate the table header since we assume that the names of the datapoints to infer are given in advance. To tackle problems such as table structure recognition where the set of possible header values is not limited, one needs to slightly modify the proposed solution. However, we do not consider it a serious limitation as the required modification is relatively straightforward, and for the sake of completeness, we describe it below.

To adjust the proposed method to be applicable to the task of Table Structure Recognition, one must

Table 7: Detailed results of experiments on reversed Rotowire dataset. See Wu et al. (2022) for metrics' specification.

	Row header F1			Column header F1			Non-header F1		
	Exact	Chrf	BERT	Exact	Chrf	BERT	Exact	Chrf	BERT
Team	94.9	95.2	97.8	88.9	85.8	88.7	84.7	85.6	90.3
Player	93.5	95.3	95.1	88.1	91.2	94.5	84.5	86.8	90.4

Table 8: Summary of the confidential datasets.

	Recipe Composition	Payment Stubs	Bank Statements
train documents	119	80	111
val documents	16	10	10
test documents	30	20	10
avg doc len (words)	0.6k	0.3k	1.3k
max doc len (words)	1.6k	2k	4, 9k
avg doc len (characters)	3.3k	2k	8.3k
max doc len (characters)	10k	14.2k	37.9k
properties total	64	11	10
properties in tables (tables columns)	64	4	4
properties outside of tables	0	7	6
mean number of table rows	12	5	2
max number of rows	60	15	5
mean length of cell (characters)	12	8	9
max length of cell (characters)	308	44	36

understand the differences in framing the problem between the tasks here.

Table Structure Recognition or Table Extraction aims to generate headers and the table content based on the document with the table provided explicitly. STable described in the main part of this paper can generate the table given any text and its position on pages. This capacity generalizes well to any input, including when the table is provided on the input. The difference is that the output form in STable assumes the headers are known upfront, while for Table Structure Recognition, inferring them is a part of the task. STable can achieve such capabilities to solve the Table Structure Recognition task by (1) adding a linear layer to predict the number of columns, (2) treating headers as the values to be inferred in the first row, (3) using dummy names of the columns, e.g., "first column," "second column," and (4) increasing the predicted number of rows by 1.

In this setup, the model will predict the number of columns and the number of rows, while the first row will represent the values of header names. The dummy headers will have to be removed during postprocessing, and the values in the first row should be treated as valid headers.

G Sample Input-Output Pairs

PWC* (Borchmann et al., 2021). Input in the PWC* consists of born-digital, multipage PDF files containing an article from the machine learning field. The expected output is a list of tuples describing achieved results on arbitrary datasets (see Figure 8).

CORD (Park et al., 2019). Input in the dataset is a single scanned or photographed receipt. From our point of view, the output here is twofold — there are simple data points that can be considered key-value pairs and data points that take the structured form of line items. We approach the problem as the generation of two tables from the document — one for each data kind (see Figure 9).

DWIE (Zaporojets et al., 2021). Input in the dataset is a plain-text article. The final goal is to extract the normed object, relation, and subject triples (though the original formulation assumes several intermediate stages). Triples are always complete (i.e., there are no NULL values, as we understand them (see Figure 10 for an example).

Reversed Rotowire (Wu et al., 2022). Input in the reversed Rotowire dataset, as reformulated by (Wu et al., 2022), is a plain-text sport news article. The task is to generate tables with team and

player statistics. The number of rows in the *Team* table is from zero (if no team is mentioned in the text) to two, whereas the number of rows in the *Player* is highly variable and content-dependent. Figure 11 present sample pair of document and tables to generate.

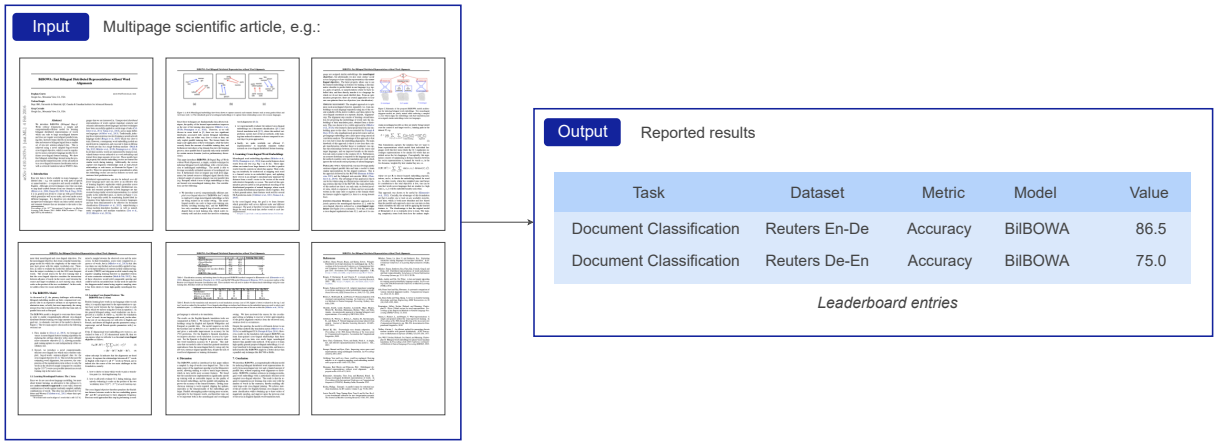

Figure 8: An example from PWC★ dataset considered in the document-to-table paradigm.

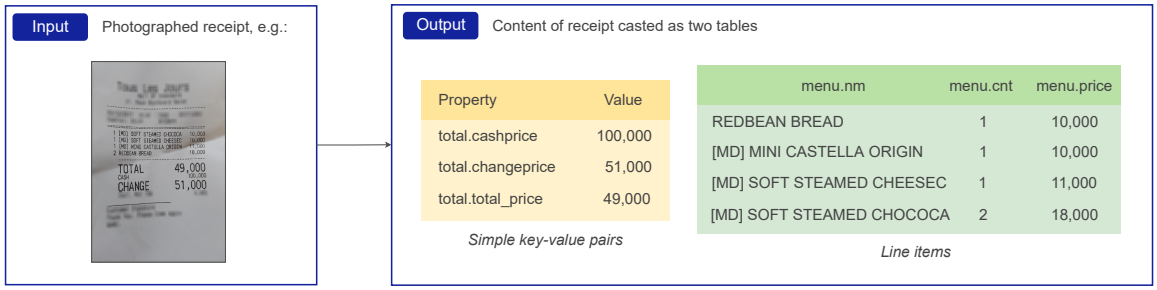

Figure 9: Sample document from CORD dataset and its expected output as interpreted in our approach.

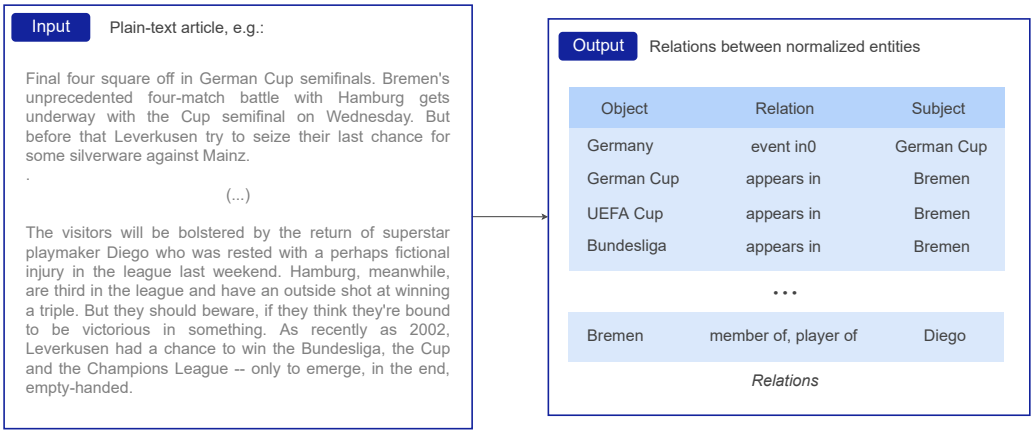

Figure 10: Sample input-output pair from the DWIE dataset. The table was shortened and consisted of 29 rows in our approach. Suppose multiple relations appear in the same direction between the pair of object-subject. In that case, we predict a list of them in a single cell, reducing the number of rows generated (see the example of the Bremen-Diego pair).

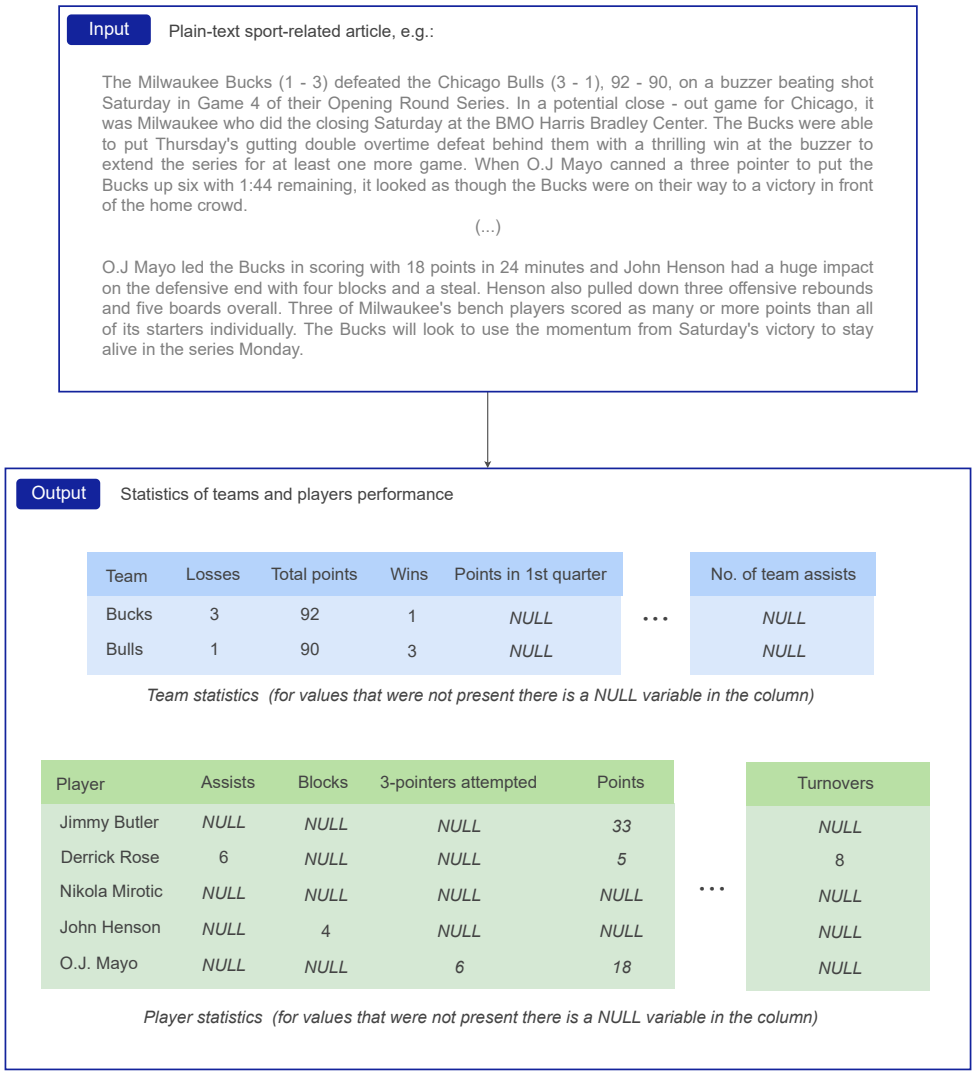

Figure 11: Input-output example from the reversed Rotowire dataset. We present shortened forms of tables than in real have 13 columns for Team and 20 columns for Player tables. Note that there is a *NULL* value in the column for values not present in the input text.

The horse face emoji we feature is a part of Noto Emoji distributed under the Apache License 2.0.
Copyright by Google Inc. No animals were harmed in the making of this article.