On Linearizing Structured Data in Encoder-Decoder Language Models: Insights from Text-to-SQL

Yutong Shao and **Ndapa Nakashole**
Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093
{yshao, nnakashole}@eng.ucsd.edu

Abstract

Structured data, prevalent in tables, databases, and knowledge graphs, poses a significant challenge in its representation. With the advent of large language models (LLMs), there has been a shift towards linearization-based methods, which process structured data as sequential token streams, diverging from approaches that explicitly model structure, often as a graph. Crucially, there remains a gap in our understanding of how these linearization-based methods handle structured data, which is inherently non-linear. This work investigates the linear handling of structured data in encoder-decoder language models, specifically T5. Our findings reveal the model's ability to mimic human-designed processes such as schema linking and syntax prediction, indicating a deep, meaningful learning of structure beyond simple token sequencing. We also uncover insights into the model's internal mechanisms, including the ego-centric nature of structure node encodings and the potential for model compression due to modality fusion redundancy. Overall, this work sheds light on the inner workings of linearization-based methods and could potentially provide guidance for future research.

1 Introduction

Motivation. Natural Language Interfaces (NLIs) to computer systems allow the use of everyday language to interact with computer systems, thus lowering technical barriers to advanced computing functionality. Early systems such as SHRDLU (Winograd, 1971) and LUNAR (Woods, 1973) saw limited success due to the limited language processing capabilities of computer systems at the time. After years of steady progress, the strong language processing capabilities of large language models (LLMs) have led to renewed interest in NLIs such as the widely used ChatGPT (Brown et al., 2020). Systems such as ChatGPT already serve as robust NLIs. However, a critical challenge remains in applying the underlying models to specialized and personalized real-world scenarios. The challenge stems from the need for the model to handle "backend data" commonly stored in structured formats such as proprietary databases, knowledge graphs, or dialog states, including intents, slots, and values. We refer to this task as *structured data representation* (SDR)(Shao et al., 2022).

In this study, our primary focus is on a representative SDR task: text-to-SQL parsing (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2012). This task automatically maps natural language queries into SQL commands, thus eliminating or at least reducing the need for programming knowledge. For such a system to be broadly applicable, it must be capable of adapting to new databases, encoding these databases together with user queries, and predicting the corresponding SQL queries.

The Rise of Linearization-based Methods. Recent approaches to text-to-SQL parsing and other SDR tasks fall into two main categories: *linearization-based* and *structure-based* methods (Lin et al., 2020; Scholak et al., 2021; Xie et al., 2022). Structure-based approaches explicitly utilize the inherent structure in the data, often representing it with a graph (Bogin et al., 2019; Wang et al., 2020; Cao et al., 2021; Hui et al., 2022). In contrast, linearization-based methods treat structured data as a token sequence, processed similarly to natural language sentences. The latter have gained traction due to their compatibility with LLMs, which have demonstrated impressive performance across various NLP benchmarks.

Open Problems and Our Contributions. SDR tasks like text-to-SQL remain a challenging problem for LLMs, as they are not completely "solved" by current models (Li et al., 2023). Towards shedding light on ways forward, our main contribution is a detailed exploration of the inner workings of a former state-of-the-art (SOTA) text-to-SQL parser

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 131–156

with T5 backbone (Scholak et al., 2021; Xie et al., 2022). Our analytical approaches include probing classifiers (Köhn, 2015; Gupta et al., 2015; Shi et al., 2016; Conneau et al., 2018), as well as techniques that directly manipulate model intermediates, leveraging the recent causal tracing method (Finlayson et al., 2021; Meng et al., 2022). We find that despite their simplicity, linearization-based methods can effectively represent structured data. Specifically, we show that the prefix-tuned T5 model preserves low-level textual details and enhances understanding of node relationships in structured data. We also show the ego-centric nature of structure node encodings, which primarily contain information relevant to the node itself. Additionally, we uncover a duplicative robustness in modality fusion, indicating potential avenues for model compression. Our study also reveals the model's internal working pipeline, which aligns with human-designed processes like schema linking, syntax prediction, and node selection, suggesting meaningful learning rather than reliance on spurious correlations. The attention mechanism's role in modality fusion and the distinctive functionalities of different layer ranges in the decoder are also revealed. Overall, our research contributes to our understanding of structured data representation in encoder-decoder LLMs.

We opted not to analyze extremely large LMs such as GPT-4, due to the high computational cost of our analytical methods and the opaque nature of their intermediate states. However, given the competitive performance of the model we study, coupled with its sequential input and autoregressive output which are analogous to LLMs, we believe that our findings are general and applicable to models in the same category.

2 Related Work

Structured Data Representation for Text-to-SQL. From prior work, structure-based methods include SchemaGNN (Bogin et al., 2019), RAT-SQL (Wang et al., 2020), LGE-SQL (Cao et al., 2021) and S^2SQL (Hui et al., 2022). Linearization-based methods have been widely studied, including BRIDGE (Lin et al., 2020) and Picard (Scholak et al., 2021). USKG (Xie et al., 2022), which proposes a unified linearization method for all SDR tasks, also falls under the category. Recently, LLMs such as those behind ChatGPT also demonstrated strong performance as a structure linearization-based text-to-SQL method (Li et al., 2023).

Model Behavior Analysis and Interpretation. Previous work include gradient-based methods which check the importance of input features based on their gradient, such as saliency maps (Simonyan et al., 2013). For models reliant on attention mechanisms (Clark et al., 2019), analytical methods exist to examine the significance of individual input units by evaluating attention weights. However, such analyses have faced skepticism in other works (Serrano and Smith, 2019). An alternative line of work is the *probing classifier* approach, in which classifiers are trained on a model's intermediate representations to determine the existence of specific information (Köhn, 2015; Gupta et al., 2015; Shi et al., 2016; Ettinger et al., 2016; Adi et al., 2016; Liu et al., 2019; Tenney et al., 2019; Hewitt and Liang, 2019; Voita and Titov, 2020; Zhu and Rudzicz, 2020; Pimentel et al., 2020; Ravichander et al., 2021b). Though flexible and adaptable to trace various information, probing tests may present challenges in the interpretation or comparison of results (Ravichander et al., 2021a; Belinkov, 2022). A recent line of work involves causal analysis, wherein researchers manipulate the target information within the input and restore intermediate outcomes to a clean state to verify the presence of information (Finlayson et al., 2021; Meng et al., 2022). Our analysis framework builds upon previous probing and causal analysis methods. However, we have adapted and incorporated additional methods specifically to gain insights into tasks related to structured data representation.

3 Preliminaries

The model we investigate in this work is T5-large,[1] with prefix-tuning on the Spider dataset (Yu et al., 2018; Xie et al., 2022). The model is a standard encoder-decoder Transformer architecture.[2] The encoder has two modules per layer: self-attention and linear multilayer perceptron (MLP). A decoder layer has three modules: self-attention, cross-attention, and MLP. To work as a text-to-SQL parser, the model takes the concatenation of user

[1] The prior SOTA method for Spider is Picard + T5-3B (Scholak et al., 2021), in which Picard is a post-hoc decoding algorithm without modifying the model. In the scope of our experiments, we use T5-large instead, as our behavior tests involve significant computational expenses.

[2] As our focus is on the interpretation of the inner workings of the model, we do not use any post-processing heuristic on the model output, like Picard.

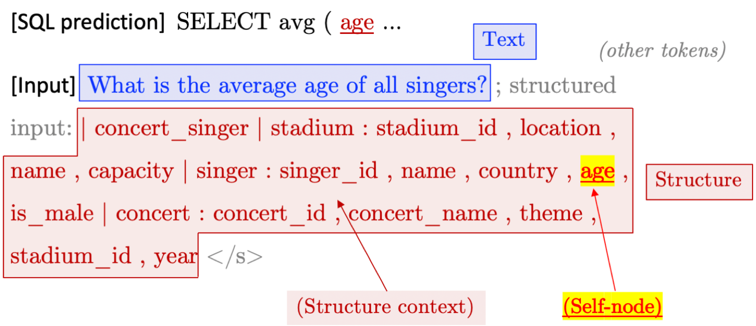


Figure 1: The input to the text-to-SQL parser consists of the query in natural language text (blue), and the relevant structured data (red), other tokens (gray). “self-node,” refers to the input tokens corresponding to the expected output node where a node refers to both column and table names, and “structure-context,” represents all the structured input tokens excluding the self-node. The output is the predicted SQL query (top).

query text and the linear form of a database (DB) structure as input and yields the SQL token sequence as output. The model has been trained with *prefix-tuning* (Li and Liang, 2021). Each attention module within the model is associated with 10 extra key-value entries from the prefix.

Terminology. For clarity of exposition, we introduce the term "structure nodes," or simply "nodes," which collectively refer to both columns and tables. We refer to “layer ranges” in the encoder or decoder, grouping them into to low, middle, and high layers. Both the encoder and decoder have 24 layers. In our discussion, “low layers” refer to layer 0-11, “middle” to 6-17 and “high” to 12-23. In addition, we mention “input sections,” such as text, structure, and prefix. For a structure node, we also have the input section of “self-node,” indicating the input tokens of the anticipated output node, and “structure-context,” representing all the structured input tokens excluding the self-node. Figure 1 illustrates the format of the input to the text-to-SQL parser.

4 Research Questions

Preliminary Intuition Open Questions. The model’s internal mechanics can be intuitively understood as follows: The encoder produces contextualized encodings by fusing both textual and structural input elements. These combined encodings then provide a comprehensive basis upon which the decoder constructs the SQL. However, despite this preliminary understanding, certain aspects remain unclear. Key questions that arise include:

(Q1) What specific information is passed from the encoder to the decoder via the text and structure token encodings? Addressing this allows us to delineate the functions of the encoder and decoder, further aiding in model interpretation.

(Q2) Which parts of the model store the important information? Here “parts” refers to model intermediates within different modules, input sections or layer ranges. This helps in detecting the possible bottlenecks and presents opportunities for model compression on the less important parts.

(Q3) How do the attention modules handle modality fusion? In our exploration of structured data representation in text-only models, we aim to understand how modality fusion is performed. It is expected to occur within the attention modules, as they are the only mechanism enabling different tokens to communicate.

(Q4) What is the internal working pipeline of the model? We aim to check if the thought process of the model mirrors human-designed pipelines, including steps such as schema-linking, syntax prediction, and node selection (Gu et al., 2023; Pourreza and Rafiei, 2023). Additionally, we seek to verify whether the model obtains meaningful knowledge through text-to-SQL training, or it is mainly fitting to spurious correlations.

5 Probing Study

5.1 Probing Tasks

For our first question **Q1**, our initial investigation focused on the information retained by the encoding vectors. For this purpose, we conducted two probing tasks as described below.

Node Name Reconstruction (NR). In this task, we examine the ability of the encoder to retain essential, low-level information about a node by attempting to reconstruct its surface-form name. Due to the tokenization process of T5, a node can be tokenized into multiple sub-tokens. We keep all sub-token encodings as a sequence and pass them to a "probe decoder," which has the same architecture as T5-decoder but is initialized randomly. The probe decoder is trained to reconstruct the node name autoregressively.

Link Prediction (LP). Unlike node name reconstruction, this task assesses the ability of the model to capture higher-order structural information in its encodings. Using the encodings of a pair of nodes,[3] we train a probing classifier to predict the connection between them. The relation definitions follow RAT-SQL (Wang et al., 2020). Examples of these relations include QT-Exact (Question token and Table name Exact match), CC-TableMatch (two Columns belong to the same Table), or null relations like XY-default (a type X node and a type Y node, no special relation). We pool the sub-token encodings of each node into a single vector. We then construct the input vector of two nodes by concatenating their pooled encodings and their element-wise dot-product, i.e. $[e_1; e_2; e_1 * e_2]$ where ";" denotes concatenation and "*" denotes element-wise dot-product. We pass this input vector to a probe classifier, which is either a logistic regressor (LR) or a 2-layer MLP, to predict the relation between the two input nodes.

5.2 Probing Results

For both probing tasks, we utilized the train and dev partitions from the Spider dataset for training and evaluation respectively. The results are presented in Table 1. For Node Name Reconstruction (NR), both prefix-tuned and pretrained T5 exhibit very high reconstruction accuracy. This indicates that the prefix-tuning process did not undermine the ability of the model to preserve the low-level information.

For Link Prediction (LP), our results showed that the encodings from prefix-tuned T5 model outperformed the pre-trained version, suggesting that the prefix-tuning process of the model enhances its understanding of relations between nodes. Interestingly, the pre-trained T5 model also yields high LP accuracy. This suggests that even without tuning, pre-trained models have an implicit capability to process structured text. This observation is consistent with findings from (Ravichander et al., 2021a), where a model can learn features that are not aligned with its primary objective.

[3] In this study, a question token is also considered a node.

Model	NR	LP	
	Exact Match	LR acc.	MLP acc.
T5-P-tuned	0.9649	0.8110	0.8600
T5-pretrain	0.9709	0.7929	0.8400
T5-random	0.4918	0.2839	0.3102

Table 1: Node Name Reconstruction (NR) and Link Prediction (LP) probing results. "P-tuned" represents the T5 model with prefix-tuning.

For comparison, the T5-random version showed significantly lower probing performance compared to either T5-prefix-tuned or pre-trained. This confirms that the high performance are not merely due to overfitting noise in high dimensions.

Takeaway 1 (Q1). Regarding the information contained in the encodings, ***prefix-tuned T5 manages to preserve low-level textual details and also improves understanding of node relationships***. Surprisingly, ***pre-trained T5 model also exhibits an intrinsic capacity to handle structured text to some degree.***

6 Direct Model Manipulation

Besides the insights from the probing study, a remaining question is whether the model actually *utilizes* the information encoded in the representation. Since probing experiments did not answer this question, we undertook an exploration of the internal mechanisms of the model by directly manipulating the model intermediates and observing the outcomes. We employ the idea of *causal tracing* (Finlayson et al., 2021; Meng et al., 2022), where specific intermediate information is corrupted or reinstated to analyze its influence on the ultimate prediction. We design our studies in a fine-grained way, computing prediction accuracy at the token-level within a SQL query and categorizing the outcomes based on token types. Token types include *columns*, *tables*, *table aliases*, and *syntax tokens* (including keywords and operators). We focus on columns and syntax tokens, treating columns as being representative of structure node prediction. The results for rest of the token types are in the appendix.

6.1 Encoder States Investigation

We begin by *corrupting* the input embeddings or final encoding vectors of individual tokens or entire input sections. A vector is "corrupted" by replacing

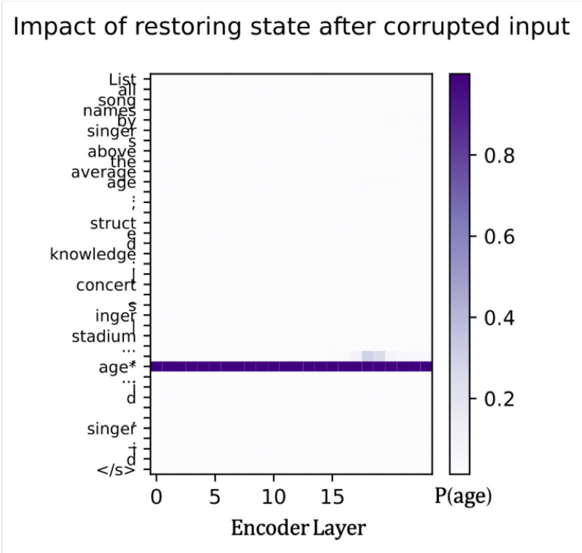


Figure 2: An illustrative sample showing the restoring effect of each encoder intermediate state. The decoder prompt: `SELECT song_name FROM singer WHERE ==> age`. Restoring the self-node hidden state on any layer can recover the correct prediction, while almost all other states do not have such an effect. More samples are available in Figure 6.

it with a zero-vector. The underlying intuition is that, when corrupting the embeddings of a token or section, the information is fully removed from the model. As such, the corruption study evaluates the overall usefulness of the corrupted part. When the final encodings are corrupted, only the information stored in these encodings is lost, thus this corruption study checks the actual information stored in these encodings and its importance.

We compute the average confidence on ground truth tokens for samples where the uncorrupted predictions are correct. The results for corrupting entire sections can be found in Table 2, on the left. First, as a sanity check, we confirm that corrupting the embeddings of the entire text section has a substantial impact on the prediction performance of both columns and syntax tokens. For structure embeddings, corrupting column names suffers a larger drop compared to syntax, consistent with our expectation. For final encodings corruption, we found that when predicting columns, the corruption on merely the self-node is as impactful as the entire structure section. Corruptions on the text part are considerably less effective, and on the structure context it has almost no influence. Shedding light on our question **Q2** regarding which states hold important information, these results indicate that the self-node encoding vectors are the most important when predicting this node. The structure node encodings are "ego-centric," each storing the information pertinent to that node and not others.

We further explored the opposite direction, corrupting the text embeddings and *restoring* the intermediate hidden states. Following ROME (Meng et al., 2022), this approach highlights intermediate states with high *restoring effect*, which refers to the increased probability of correct prediction when restoring the state back to the clean version. These states likely capture crucial information and hold notable importance. We chose to corrupt the embeddings of the text section based on the reasoning that the expected SQL output is entirely specified by the text.

Figure 2 illustrates the restoring effect of all encoder states on a representative sample. We observe that restoring the self-node hidden states at any layer can restore the correct prediction. However, for other tokens, restoring their representations at any layer has minimal impact. This observation reinforces the finding that self-node representations hold greater significance than other tokens when predicting that node. The ability of a single encoding to restore the correct prediction confirms that the low-level textual information is retained within the encoding and effectively utilized by the decoder **(Q1)**.

We also investigated the effects of restoring the final encodings of entire sections with corrupted text embeddings. The findings are presented in Table 2, on the right side. We observed the same trend where restoring only the self-node encoding proves to be more effective than restoring the entire text section or the structural context.

Takeaway 2 (Q1, Q2). The ***encodings of structure nodes are predominantly "ego-centric,"*** containing primarily information relevant to themselves with minimal data about other nodes. Consequently, the ***target node's encodings emerge as the most important*** among all encodings during node prediction.

6.2 Contextual Representations of Structure

We now turn our attention to **Q3**, which explores the attention mechanism and the integration of modalities between text queries and structured input. As can be seen in Table 2, when text embeddings are corrupted, even if their final encodings are restored, the prediction accuracy is still compromised (0.5663), since the structure nodes fail

Section	Embeddings		Final Encodings		Encoding Restore
	Column	Syntax	Column	Syntax	Column
Text	0.2482	0.2704	0.9115	0.4329	0.5663
Struct	0.4084	0.8435	0.4822	0.7056	0.8016
Self-node	0.3916	-	0.5239	-	0.6801
StructCtx.	0.8995	-	0.9848	-	0.1028
all	0.0083	0.0422	0.0943	0.1458	0.9416

Table 2: The effect of corrupting different sections of the input on columns and syntax token. Values are the average confidence scores of the ground truth over all samples where the clean (uncorrupted) prediction is correct.

Corruption part	Columns		Syntax tokens	
	Weights	Logits	Weights	Logits
$T \to S$	0.9671	0.9543	0.9933	0.9919
$S \to T$	0.9071	0.6879	0.9926	0.9845
$T \leftrightarrow S$	0.8416	0.6138	0.9878	0.9788
all	0.2101	0.1502	0.7269	0.7338

Table 3: Attention corruption study on **encoder self-attention** across input sections. For input sections, "T" means text and "S" stands for structure. "$T \to S$" means corrupting the attention weights from text to structure tokens; "all" means corrupting the full attention matrix. On top, "Weights" and "Logits" represent the attention corruption scheme.

to access accurate text information. This ***underscores the important role of text information in the process of encoding structure node, affirming the overall necessity of modality fusion***.

In the following experiments, we aim to study the inner workings of the attention modules by examining and manipulating them through various methods.

6.2.1 Attention Corruption Study

To reveal the inner mechanism of modality fusion, a straightforward first question we propose is: *Where does the fusion primarily take place?* For this, we explore another type of causal study, namely *attention corruption*, in which we deactivate the attention within certain layers and between certain sections by masking the corresponding attention entries. In detail, there are two corruption schemes, by *"weights"* or by *"logits"*. The "weights" setting simply sets the corrupted attention weights to 0, keeping others unchanged. The "logits" setting adjusts attention logits to $-\infty$ before the softmax operation, essentially zeroing out the corrupted sections while ensuring a valid distribution. Both settings are considered in our experiments.

Intuitively, modality fusion is expected to occur within encoder self-attention or decoder cross-attention. We evaluated the prediction accuracy of the model for columns and syntax tokens with our above-mentioned attention corruption, targeting the encoder self-attention or the decoder cross-attention, across different sections and within different layer ranges. Our guiding intuition here is straightforward: for components not engaged in modality fusion, a smaller performance drop is expected.

The results are shown in Table tables 3 and 4. Table 3 reveals an interesting finding under column "Columns - Weights" and on the row "$S \to T$" (structure-to-text[4]): the interference of encoder self-attention from the entire structure section to the text section had a negligible negative impact (0.9071), compared to the performance drop in Table 2 caused by text embeddings corruption (0.5663). Likewise, Table 4 row "Text" reports minimal damage to accuracy (0.9063) when the decoder cross-attention to the text was obstructed.

These findings seem to contradict our previous conclusion that the fusion of text and structure information is vital for the task. We propose an explanation for this inconsistency, suggesting that it stems from the *duplicative robustness* of the model. It means the model has learned certain capabilities in multiple locations, encoder and decoder in our case. To verify this hypothesis, we experiment with jointly corrupting the encoder self-attention from structure to text, and decoder cross-attention to text, essentially combining the corruption effect of the two experiments above. The results are shown in Table 5. We can observe that simultaneously corrupting both leads to a more substantial decline in accuracy. This supports our duplicative robustness hypothesis with regard to modality fusion. Related to **Q3**, this finding provides novel insights into modality fusion. It underscores the robustness of the model, but also hints at potential opportunities for post-hoc model compression.

Takeaway 3 (Q3). The model exhibits *duplicative robustness* in the joint representation of text and structure. ***Both encoder and decoder demonstrate proficient capabilities in fusing text information into structure, highlighting both the internal robustness of the model but also possibilities for compression***.

[4]Th notation of attention "from section A to B," means the attention matrix entries with tokens from A as query q, and tokens from B as keys k and values v.

Corruption part	Columns		Syntax tokens	
	Weights	Logits	Weights	Logits
Text	0.9063	0.9132	0.8517	0.8305
Struct	0.3239	0.4333	0.9543	0.9326
Prefix	0.9114	0.9025	0.8332	0.7676
StructCtx.	0.9429	0.9876	-	-
Self-node	0.3840	0.5382	-	-
all	0.1316	0.0597	0.4590	0.3725

Table 4: Attention corruption study on **decoder cross-attention** from each decoder step to encodings of each input section.

6.2.2 Attention Weights Information

To delve deeper into the functionalities of attention, we examine the potential correlation between attention weights and actual interpretable information. In **Q4** we hypothesize that the model internally performs subtasks akin to those designed by humans. One of them is *schema linking*, which aims to determine the *relevance* of a node, i.e., whether this node should appear in the output SQL. For that, we explore the correlations between the distribution patterns of a column within encoder self-attention and the relevance of the column. In detail, within each sample, we gather the attention weights from each column[5] to different input sections, across different layers and attention heads. We then compare patterns between relevant and non-relevant columns to gauge any distinctive behavior.

The results are presented in Table 6. We indeed observe distinctions between relevant and non-relevant columns. For specific heads and input sections, attention is consistently high for only one type of columns and low for the other. For example, head 8 shows a notably high attention to text and low attention to structural context for relevant nodes. For heads 10 and 11, attention to prefix token #4 is markedly high for non-relevant nodes.

We further confirm the correlation between encoder self-attention weights and node relevance by directly utilizing the attention patterns as features for node relevance classification. Specifically, we focused on the attention positions (layer, head, section) with clear discrepancy between relevant and non-relevant nodes, as mentioned above, and collect the attention weights on these positions to form "input features" for each node. A logistic regressor (LR) was trained to make the relevance prediction using these features. We compared with the implicit predictions made by the full model, where nodes in the generated SQL were predicted as relevant. The results can be found in Table 7. The accuracy and F1 scores of the LR are on par with those of the full model, and significantly better than simple heuristics such as predicting nodes with "exact text match" as relevant. This reaffirms that the encoder self-attention is tightly associated with node relevance, and that the encoder has successfully internalized the schema linking subtask.

[5]For simplicity, in this study we only use the first token of each column as its representative.

Corrupted part	Weights	Logits
Enc.SA-only	0.9071	0.6879
Dec.XA-only	0.9063	0.9132
Enc.SA + Dec.XA	0.6414	0.2987

Table 5: Joint corrupting encoder self-attention (structure to text) and decoder cross-attention (to text), to highlight the duplicative robustness phenomenon.

	Head 7	Head 8	Head 10	Head 11
prefix#0	0.55 / 0.06	0.01 / 0.01	0.04 / 0.01	0.00 / 0.04
prefix#4	0.04 / 0.05	0.00 / 0.00	0.01 / 0.45	0.01 / 0.42
prefix#8	0.01 / 0.01	0.01 / 0.24	0.07 / 0.02	0.12 / 0.03
text	0.00 / 0.01	0.73 / 0.06	0.05 / 0.01	0.00 / 0.00
self	0.25 / 0.57	0.01 / 0.01	0.03 / 0.00	0.04 / 0.05
context	0.10 / 0.24	0.17 / 0.61	0.33 / 0.09	0.22 / 0.23
others	0.01 / 0.01	0.06 / 0.05	0.27 / 0.07	0.52 / 0.15

Table 6: Attention weights from a column to each section, averaged for all relevant / non-relevant columns. Values in red are high for relevant nodes, and in blue are high for non-relevant nodes. Due to space constraints we only show results for Encoder layer 23, on a subset of heads and prefix tokens, on the dev set. More results are provided in Table 13.

Takeaway 4 (Q3, Q4). Encoder self-attention weights carry distinguishing information about node relevance. ***This implies the ability of the encoder to perform the schema linking subtask.***

6.2.3 End-to-End SQL Performance and Error Analysis

To verify the above findings, which are based on token-level prediction results, we extended our experiments using the same corruption settings to evaluate the *end-to-end SQL prediction* performance. SQL predictions are measured by the Exact Match and Execution Match metrics, in line with the original Spider leaderboard (Yu et al., 2018). In these experiments, we introduced corruptions on different layer ranges and targeting specific sections such as the encoder self-attention between the text and structure, and decoder cross-attention to text and structure. Additionally, we added *decoder self-attention* corruption, which yielding interesting findings.

Relevance	Attn. + LR	Full model	Heuristics
Accuracy	0.9729	0.9842	0.9403
Precision	0.8669	0.9316	0.8616
Recall	0.8535	0.8994	0.4634
F1	0.8602	0.9152	0.6026

Table 7: Column relevance prediction results. The P/R/F1 is computed for "relevant" columns due to their sparsity (<10%) among all columns.

Module	Corruption type	Layers	Section	Exact	Exec
Enc.SA	Logits	Low	$S \to T$	0.6538	0.6654
Enc.SA	Logits	Low	$T \to S$	0.6518	0.6721
Enc.SA	Logits	Low	$T \leftrightarrow S$	0.6422	0.6634
Enc.SA	Logits	High	$S \to T$	0.4072	0.4362
Enc.SA	Logits	all	$S \to T$	0.2234	0.2369
Enc.SA	Weights	all	$S \to T$	0.5145	0.5387
Dec.XA	Logits	all	Text	0.0706	0.0812
Dec.XA	Logits	all	Struct	0.0648	0.0638
Dec.SA	Weights	all	all	0.0000	0.0000
-	(Clean)	-	-	0.6692	0.6809

Table 8: End-to-end SQL performance with different attention corruption settings. Rows are selected for discussion; full results are available in Table 20 in the appendix. SA: self-attention; XA: cross-attention.

The results are provided in Table 8. For encoder self-attention (Enc.SA), the trends are consistent with previous observations. For example, the performance of "logits" corruption is much lower than "weights" (0.2369 vs. 0.5387 on Exec-match). Interestingly, corruption on the lower layers resulted in almost no decrease in performance, even for section "$T \leftrightarrow S$" ($0.6809 \to 0.6634$), hinting at opportunities for model pruning.

For decoder cross-attention (Dec.XA), introducing corruption to either text or structure drastically reduces performance. This is attributable to the compromised ability to predict syntax tokens and structure nodes, respectively. To further understand the actual behavior of the model **(Q4)**, we conducted manual error analysis on a subset of 50 samples. The results are shown in Figure 3, with supplementary details provided in Table tables 21 and 22 in the appendix. We observe that when cross attention to text is blocked, the predominant errors are "clause-semantic errors," most commonly missing a condition or aggregation function. On the other hand, blocking structure section primarily results in node selection errors, where the model hallucinates on node names. This finding verifies the specialized capabilities of the decoder for SQL syntax prediction and node selection, functioning independently of each other. This reinforces the conclusion from Takeaway **4** for **Q4** that the model mirrors human-designed pipelines, including schema linking, syntax prediction, and node selection.

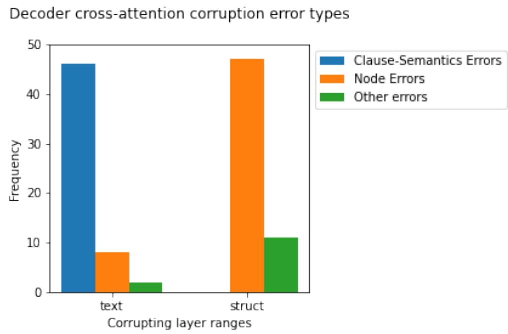


Figure 3: Error type analysis on **decoder cross-attention** corruption on the text or structure part.

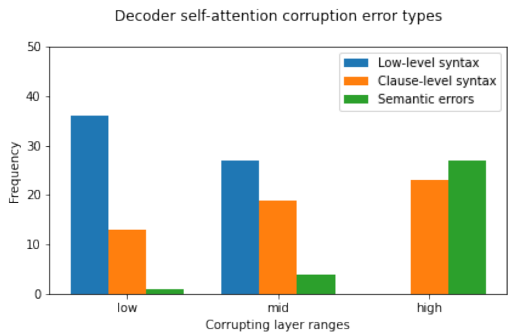


Figure 4: Error type analysis on **decoder self-attention** corruption on various layer ranges.

Takeaway 5 (Q4) The model shows the ***ability to perform different subtasks corresponding to human-designed pipelines: schema linking in the encoder self-attention, syntax prediction in decoder cross-attention to text, and node selection in decoder cross-attention to structure***.

Decoder Self-attention Study. In Takeaway **5**, we confirmed the model's subtask handling capabilities and identified corresponding submodules. Delving deeper, we aim to pinpoint the layers where these processes occur, relating back to **Q2** concerning the storage of information within the layer dimension. Hypothetically, different layer ranges in the decoder have distinct responsibilities. Intuitively, lower layers would focus more on syntax prediction, while higher layers would concentrate more on node selection, reflecting the intrinsic order of these subtasks.

To verify this, we conducted additional experiments by corrupting the *decoder self-attention*, effectively blocking all incoming information from previous timesteps in the decoder, within different layer ranges. The goal was to identify the information that is either already available or still

Q: **What is the average horsepower for all cars produced before 1980 ?**
Pred: `SELECT avg(horsepower) FROM cars_data WHERE year prior 1980`
Gold: `SELECT avg(horsepower) FROM cars_data WHERE year < 1980`
Q: **What are the codes of template types that have fewer than 3 templates?**
Pred: `SELECT template_type_code FROM templates GROUP BY template_type_code HAVING COUNT(*) less than 3`
Gold: `SELECT template_type_code FROM templates GROUP BY template_type_code HAVING COUNT(*) < 3`
Q: **What are the names of airports in Aberdeen?**
Pred: `SELECT airportname FROM airports WHERE city is Aberdeen`
Gold: `SELECT airportname FROM airports WHERE city = "Aberdeen"`
Q: **List the title of all cartoons in alphabetical order.**
Pred: `SELECT title FROM cartoon arranged alphabetically`
Gold: `SELECT title FROM cartoon ORDER BY title`
Q: **How many type of governments are in Africa?**
Pred: `SELECT COUNT(different governmentform) FROM country WHERE continent = Africa`
Gold: `SELECT COUNT(DISTINCT governmentform) FROM country WHERE continent = "Africa"`

Table 9: Samples of corrupting **decoder self-attention** on **high layers**. In **red** are natural phrases generated by the model that semantically match the SQL syntax counterparts in blue.

missing at each layer range. The overall results are presented in Table 8, where, as expected, the end-to-end performance significantly deteriorated. For deeper insights, we performed manual error analysis on each layer range, including low, middle, and high, on a subset of 50 samples. The results are shown in Figure 4, with more details in Table tables 23 and 24 in the appendix. We observe a clear spectrum of error types distributed across the corrupted layer ranges. For low layers, the errors are predominantly "low-level syntax errors" such as unpaired brackets or quotes. For middle layers, many errors are "clause-level errors" where missing clauses or operators invalidate the SQL. For high layers, the SQL predictions are generally better formed syntactically, and the errors tend to pertain to higher-level semantics of the SQL. Interestingly, often the SQL error is confined to a clause where the SQL grammars are replaced by *natural language phrases with similar semantics*. Examples of this phenomenon are provided in Table 9. This intriguing observation is a strong indicator that the model has learned to align the semantics of SQL with natural language, and these representations are already obtained within the lower layers. Meanwhile, the overall behavior discrepancies between layer ranges support our intuitive hypotheses on the distinct functionalities of different layer ranges in the decoder.

Takeaway 6 (Q2, Q4) The decoder follows human intuitions that low layers focus more on syntax prediction and high layers focus more on node selection. Remarkably, we found that ***the model learns to align the semantics of SQL with natural language, despite no training on naturalized SQL versions***, such as SemQL or NatSQL (Guo et al., 2019; Gan et al., 2021). This suggests that the model learns meaningful knowledge rather than merely exploiting spurious correlations in the dataset.

7 Conclusion

We conducted a comprehensive study on the internal behavior of an encoder-decoder language model, specifically T5, text-to-SQL parser. Through both probing and manipulation of internal states, we provide insight into various aspects such as the information transfer between encoder and decoder, the storage of crucial data within the model, the functions of attention mechanisms, the process of modality fusion, the internal processing pipeline of the model, and the intrinsic alignment of SQL and natural language semantics. Our findings can inform and guide future research in text-to-SQL and related structured data representation tasks.

8 Limitations

Our study is limited to one type of pretrained language model architecture and did not consider a broader spectrum of models. The scope of our study was also limited by the computational resources required for analyzing larger models and the availability of model intermediates. Thus, future research directions include: 1) Exploring similar studies across different pretrained language model architectures, such as the widely adopted decoder-only models (Brown et al., 2020). ii) Examining the impact of model scaling, both in terms of increased parameters and data volume, following the insights from scaling laws (Hoffmann et al., 2022). iii) Extending the study to other structured data tasks, such as speech-to-SQL (Shao et al., 2023) or text-to-plots (Shao and Nakashole, 2020; Wang et al., 2021). and a variety of structured data sources such as knowledge graphs (Nakashole et al., 2010, 2011; Nakashole, 2013; Nakashole and Weikum, 2012; Nakashole et al., 2013; Mitchell et al., 2015; Kumar et al., 2017; Moon et al., 2019; Tuan et al., 2022) and tables (Yin et al., 2020; Herzig et al., 2020; Yu et al., 2021; Yang et al., 2022).

Our study was carried out only on the Spider dataset for text-to-SQL. However, we do not consider this to be a significant limitation, since our objective was to interpret the model behavior rather than proposing and validating a novel model.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2016. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. ArXiv, abs/1608.04207.

Yonatan Belinkov. 2022. Probing classifiers: Promises, shortcomings, and advances. Computational Linguistics, 48(1):207–219.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-SQL parsing. ArXiv, abs/1905.06241.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In NeurIPS.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. In ACL.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of bert's attention. In ACL 2019, pages 276–286.

Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018. What you can cram into a single $&!#% vector: Probing sentence embeddings for linguistic properties. In ACL, pages 2126–2136.

Allyson Ettinger, Ahmed Elgohary, and Philip Resnik. 2016. Probing for semantic evidence of composition by means of simple classification tasks. In RepEval@ACL 2016, pages 134–139.

Matthew Finlayson, Aaron Mueller, Sebastian Gehrmann, Stuart M. Shieber, Tal Linzen, and Yonatan Belinkov. 2021. Causal analysis of syntactic agreement mechanisms in neural language models. In ACL, pages 1828–1843. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John H. Drake, and Qiaofu Zhang. 2021. Natural SQL: Making SQL easier to infer from natural language specifications. In EMNLP Findings, pages 2030–2042.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. AllenNLP: A deep semantic natural language processing platform.

Zihui Gu, Ju Fan, Nan Tang, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Sam Madden, and Xiaoyong Du. 2023. Interleaving pre-trained language models and large language models for zero-shot NL2SQL generation. ArXiv, abs/2306.08891.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In ACL, pages 4524–4535.

Abhijeet Gupta, Gemma Boleda, Marco Baroni, and Sebastian Padó. 2015. Distributional vectors encode referential attributes. In EMNLP 2015, pages 12–21.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. ArXiv, abs/2004.02349.

John Hewitt and Percy Liang. 2019. Designing and interpreting probes with control tasks. In EMNLP 2019, pages 2733–2743.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. 2022. An empirical analysis of compute-optimal large language model training. In NeurIPS 2022.

Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Yanyang Li, Bowen Li, Jian Sun, and Yongbin Li. 2022. S^2SQL: Injecting syntax to question-schema interaction graph encoder for text-to-SQL parsers. ArXiv, abs/2203.06958.

Arne Köhn. 2015. What's in an embedding? analyzing word embeddings through multilingual evaluation. In EMNLP, pages 2067–2073.

Anurag Kumar, Bhiksha Raj, and Ndapandula Nakashole. 2017. Discovering sound concepts and acoustic relations in text. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017, pages 631–635. IEEE.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Chenhao Ma, Kevin C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. ArXiv, abs/2305.03111.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In ACL 2021.

Xi Victoria Lin, R. Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In EMNLP Findings.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019. Linguistic knowledge and transferability of contextual representations. In NAACL 2019, pages 1073–1094. Association for Computational Linguistics.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual knowledge in GPT. ArXiv, abs/2202.05262.

Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr., Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapandula Nakashole, Emmanouil A. Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. 2015. Never-ending learning. In AAAI, pages 2302–2310. AAAI Press.

Seungwhan Moon, Pararth Shah, Anuj Kumar, and Rajen Subba. 2019. OpenDialKG: Explainable conversational reasoning with attention-based walks over knowledge graphs. In ACL.

Ndapandula Nakashole. 2013. Automatic extraction of facts, relations, and entities for web-scale knowledge base population. Ph.D. thesis, Saarland University.

Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. 2010. Find your advisor: Robust knowledge gathering from the web. In WebDB 2010. ACM.

Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. 2011. Scalable knowledge harvesting with high precision and high recall. In Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011, pages 227–236. ACM.

Ndapandula Nakashole and Gerhard Weikum. 2012. Real-time population of knowledge bases: Opportunities and challenges. In AKBC-WEKEX@NAACL-HLT 2012, pages 41–45. Association for Computational Linguistics.

Ndapandula Nakashole, Gerhard Weikum, and Fabian M. Suchanek. 2013. Discovering semantic relations from the web and organizing them with PATTY. SIGMOD Rec., 42(2):29–34.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830.

Tiago Pimentel, Josef Valvoda, Rowan Hall Maudslay, Ran Zmigrod, Adina Williams, and Ryan Cotterell. 2020. Information-theoretic probing for linguistic structure. In ACL 2020, pages 4609–4622.

Mohammad Reza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. ArXiv, abs/2304.11015.

Abhilasha Ravichander, Yonatan Belinkov, and Eduard Hovy. 2021a. Probing the probing paradigm: Does probing accuracy entail task relevance? In EACL, pages 3363–3377, Online. Association for Computational Linguistics.

Abhilasha Ravichander, Yonatan Belinkov, and Eduard H. Hovy. 2021b. Probing the probing paradigm: Does probing accuracy entail task relevance? In EACL 2021, pages 3363–3377.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. ArXiv, abs/2109.05093.

Sofia Serrano and Noah A. Smith. 2019. Is attention interpretable? In ACL, pages 2931–2951, Florence, Italy. Association for Computational Linguistics.

Yutong Shao, Arun Kumar, and Ndapa Nakashole. 2022. Structured data representation in natural language interfaces. IEEE Data Eng. Bull., 45(3):68–81.

Yutong Shao, Arun Kumar, and Ndapa Nakashole. 2023. Database-aware ASR error correction for speech-to-sql parsing. In ICASSP 2023, pages 1–5.

Yutong Shao and Ndapa Nakashole. 2020. Chartdialogs: Plotting from natural language instructions. In ACL 2020, pages 3559–3574.

Xing Shi, Inkit Padhi, and Kevin Knight. 2016. Does string-based neural MT learn source syntax? In EMNLP 2016, pages 1526–1534.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. CoRR, abs/1312.6034.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. 2019. What do you learn from context? probing for sentence structure in contextualized word representations. In ICLR 2019.

Yi-Lin Tuan, Sajjad Beygi, Maryam Fazel-Zarandi, Qiaozi Gao, Alessandra Cervone, and William Yang Wang. 2022. Towards large-scale interpretable knowledge graph reasoning for dialogue systems. In ACL Findings.

Elena Voita and Ivan Titov. 2020. Information-theoretic probing with minimum description length. In EMNLP 2020, pages 183–196.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: relation-aware schema encoding and linking for text-to-SQL parsers. In ACL, pages 7567–7578.

Yihan Wang, Yutong Shao, and Ndapandula Nakashole. 2021. Interactive plot manipulation using natural language. In NAACL.

Terry Winograd. 1971. Procedures as a representation for data in a computer program for understanding natural language. Technical report, Massachusetts Institute of Technology.

William A Woods. 1973. Progress in natural language understanding: an application to lunar geology. In Proceedings of the June 4-8, 1973, national computer conference and exposition, pages 441–450.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2022. UnifiedSKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. ArXiv, abs/2201.05966.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 2022. TableFormer: Robust transformer modeling for table-text encoding. In ACL.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In ACL.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2021. Grappa: Grammar-augmented pre-training for table semantic parsing. ArXiv, abs/2009.13845.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In EMNLP, pages 3911–3921. Association for Computational Linguistics.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In Proceedings of the national conference on artificial intelligence, pages 1050–1055.

Luke S Zettlemoyer and Michael Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. arXiv preprint arXiv:1207.1420.

Zining Zhu and Frank Rudzicz. 2020. An information theoretic view on selecting linguistic probes. In EMNLP 2020, pages 9251–9262.

A Details of Dataset and Model

Spider Dataset The Spider dataset has 7000 samples in the training set and 1034 samples in the dev set (Yu et al., 2018). Unless mentioned otherwise, our analytical studies were conducted on the full dev set.

Text-to-SQL Model The model we study is T5-large with prefix-tuning, implemented in the USKG project (Xie et al., 2022). Regarding the model size, it is almost the same as T5-large which has around 770M parameters.

Data Processing The tokenizer we use is the T5 tokenizer imported from HuggingFace, consistent with the model.

Probing Models For logistic regression, we use the Logistic Regression class from the Scikit-learn package (Pedregosa et al., 2011), with hyperparameter $C = 1.0$. For neural models including MLP (for the LP probing task) and T5-decoder (for the NR probing task), we use AllenNLP with Pytorch backend (Gardner et al., 2017). The MLP probe for the LP task has 2 linear layers with a middle dimensionality of 64 and activation function LeakyReLU (slope = 0.01). It is trained with Adam with initial learning rate $1e^{-4}$. The "probe decoder" for the NR task uses T5-large decoder architecture and pretrained parameters (without prefix-tuning). It is fine-tuned with Adam with initial learning rate $1e^{-5}$.

B Extra Details of Analysis

B.1 Probing Study

For Link Prediction (LP), we made adjustments to balance the frequencies of relation labels, given that there are multiple dominant classes with "default" labels (indicating no specific relation). For each input sample, which consists of a textual sentence and its corresponding structural input, we use only K node-pairs from each relation class. In our study, we simply set $K = 1$.

B.2 Causal Tracing Study

When the ground truth unit is multi-token, no matter node name or syntax tokens, we compute the probability of each sub-token in the ground truth with teacher-forcing, and compute the minimum probability (bottleneck) among all sub-tokens as the probability of the entire unit.

When we study behavior of columns in the input, such as the effect of "self-node" or the attention to other sections, we exclude the column whose name appears more than once in the structured input, since it is non-trivial to define "self-node" in such cases. In detail, it is unclear whether another node with the exact same name should be considered as "self-node".

We corrupt representation vectors to zero-vectors instead of adding a random noise vector as done in (Meng et al., 2022). This is because we observed in our preliminary studies that adding a random noise does not change the model prediction in most cases, different from the behaviors of the studied GPT-2 model in (Meng et al., 2022).

C Extra Results

C.1 Attention Blocking Effect

In comparing the outcomes of corrupting "weights" versus "logits" in Table tables 3 and 4, an interesting trend emerges. In Table 3,when corrupting "$S \rightarrow T$" (and similarly "$T \leftrightarrow S$") in encoder self-attention, the "logits" corruption caused a higher detriment than "weights" (0.9071 vs. 0.6879). This means that reallocating encoder self-attention weights to other less-attended sections can severely hamper performance. Conversely, Table 4 does not display the same pattern. When corrupting decoder cross-attention to "Text", the accuracy under the "logits" setting is not inferior to "weights" but slightly superior (0.9063 vs. 0.9132). For other sections like "Struct" or "Self-node", the "logits" accuracy is also higher than the "weights" setting. Besides, such patterns are only observed for column prediction and not for syntax tokens.

We attribute these trends to what we term the *blocking effect* of encoder self-attention, where a high attention weights serves a dual purpose: transmitting information from that section while simultaneously inhibiting other sections from conveying information. This effect is also verified by our finding on the correlation of attention weights and node relevance, which proves the attention weights carry distinguishing information. This explains the lowered performance when reallocating the attention to other sections, and justifies the existence of blocking effect in encoder self-attention.

C.2 Attention per Layer Range

Exploring another aspect of **Q3** (attention functionalities), we check the importance of attention within different layer ranges. It also touches on **Q4** regarding the inner pipeline of the model. Referring back to Table tables 3 and 4, it is evident that for the encoder self-attention, corruption in low layers yields a substantially milder effect compared to high layers. This trend is consistent for both structure nodes and syntax prediction. This suggests that, within the encoder, the high-layer self-attention takes an overall lead on the contextualization process. We further conducted direct observations on a subset of samples, examining the attention distribution from a column token to all tokens. We found that low-layer self-attention is more "self-concentrated", meaning it is largely attending to the self-node tokens. In contrast, attention in high layers is generally more distributed towards other tokens. Relevant visualizations can be found in Figure 7. Another evidence is from the attention distribution patterns (Table 13). High layers exhibit more distinctive attention positions than low layers. This further implies that high layers play a more pivotal role in gathering information.

For the decoder cross-attention, the comparison between low and high layers is less clear. One possible explanation is that different layer ranges have distinct responsibilities. We will further discuss this topic in later sections.

D Experiment Environment

CPU: 48 × Intel(R) Xeon(R) Gold 6136 CPU @ 3.00GHz

GPU: 4 × NVIDIA GeForce GTX 1080 Ti

CUDA: Version = 10.2

OS: Ubuntu 16.04.6 LTS (Xenial)

Relation	Freq	T5-P-tuned	T5-pretrained
qq_dist(-2)	500	0.7725	0.7367
qq_dist(-1)	500	0.5164	0.5196
qq_dist(0)	500	0.9159	0.9684
qq_dist(1)	500	0.4990	0.5125
qq_dist(2)	500	0.7787	0.7324
qc_default	500	0.9098	0.8793
qt_default	500	0.9715	0.9547
cq_default	500	0.8877	0.8598
cc_default	500	0.6264	0.6385
cc_foreign_key_forward	471	0.8986	0.8925
cc_foreign_key_backward	471	0.9104	0.9031
cc_table_match	500	0.6274	0.6141
cc_dist(0)	500	0.9210	0.9440
ct_default	500	0.7035	0.6549
ct_primary_key	465	0.9110	0.8665
ct_table_match	500	0.7522	0.7249
ct_any_table	500	0.9950	0.9980
tq_default	500	0.9639	0.9660
tc_default	500	0.6730	0.6046
tc_primary_key	465	0.8862	0.8580
tc_table_match	500	0.7543	0.6991
tc_any_table	500	0.9891	0.9960
tt_default	471	0.5612	0.5571
tt_foreign_key_forward	439	0.6683	0.6073
tt_foreign_key_backward	439	0.6396	0.6369
tt_dist(0)	500	0.9620	0.9320
qcCEM	264	0.8910	0.8673
cqCEM	264	0.8922	0.8729
qtTEM	199	0.9466	0.9300
tqTEM	199	0.9249	0.9193
qcCPM	281	0.8885	0.8405
cqCPM	281	0.9024	0.8364
qtTPM	51	0.8367	0.8200
tqTPM	51	0.8315	0.8444
qcNUMBER	81	0.9701	0.9405
cqNUMBER	81	0.9529	0.9277
qcTIME	15	0.8889	0.7500
cqTIME	15	0.8889	0.7692
qcCELLMATCH	125	0.8250	0.7410
cqCELLMATCH	125	0.7686	0.7903

Table 10: Link prediction (LP) probing results (F1-score) per relation type, using logistic regression (LR) as probing method. For detailed definitions of each relation, please refer to (Wang et al., 2020).

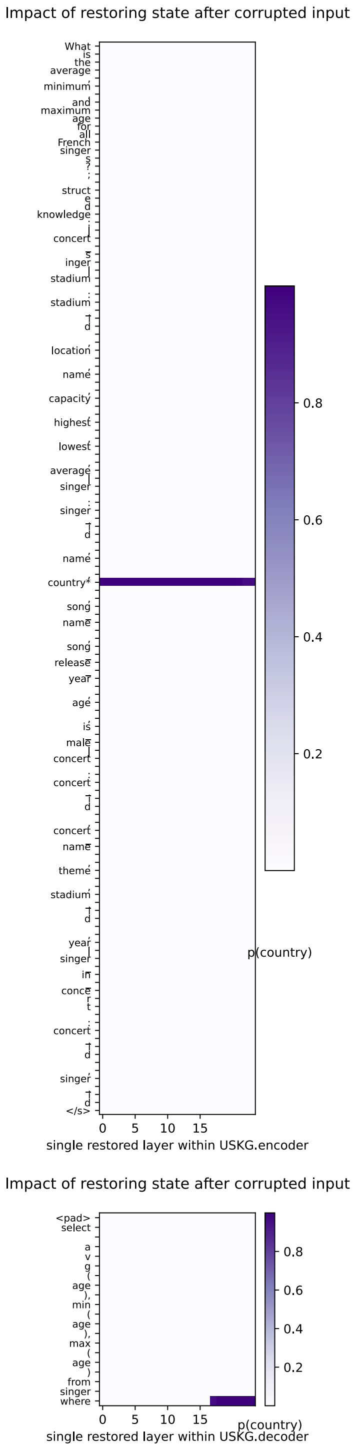
Column	Clean Text	DC. Text	Crpt. Text
Clean struct	0.9905	0.9611	0.8505
DC. struct	0.6152	0.4952	0.0489
Crpt. struct	0.3916	0.3469	0.0241

Table	Clean Text	DC. Text	Crpt. Text
Clean Struct	0.9906	0.9802	0.9687
DC. Struct	0.4472	0.3965	0.1020
Crpt. Struct	0.2241	0.1958	0.0015

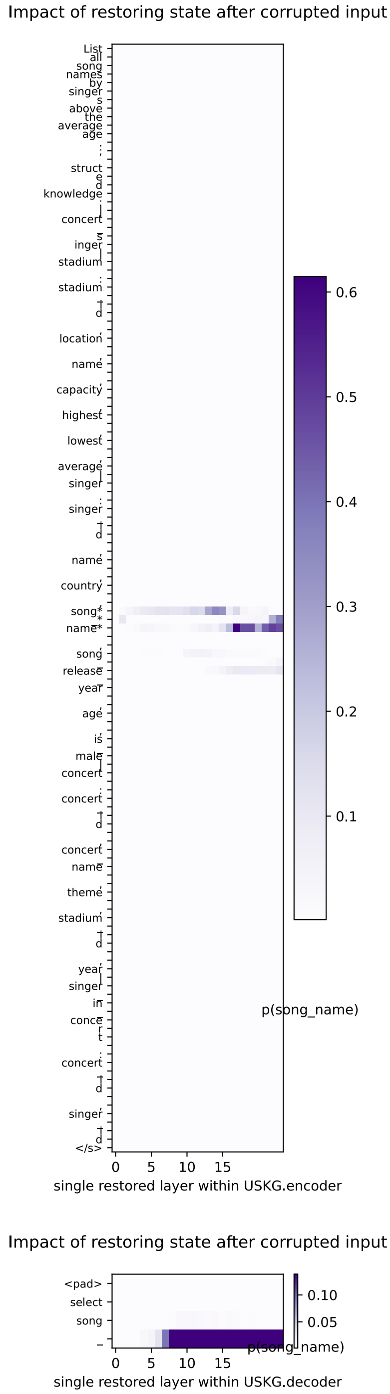
Table 11: Results of "dirty context encodings". "DC." means "dirty context", i.e. this section is encoded with its own embeddings clean but the other part embeddings corrupted; the other part's final encodings are restored to the clean state. Effectively, in this setting we obtain encodings without information from the context. "Crpt." means "corrupted", i.e. the input embeddings of this section is corrupted.

	Both+	Clean+	SCC+	None	Total
Column	1333	103	66	500	2002
Table	1424	123	105	31	1683
Table alias	1509	191	64	275	2039

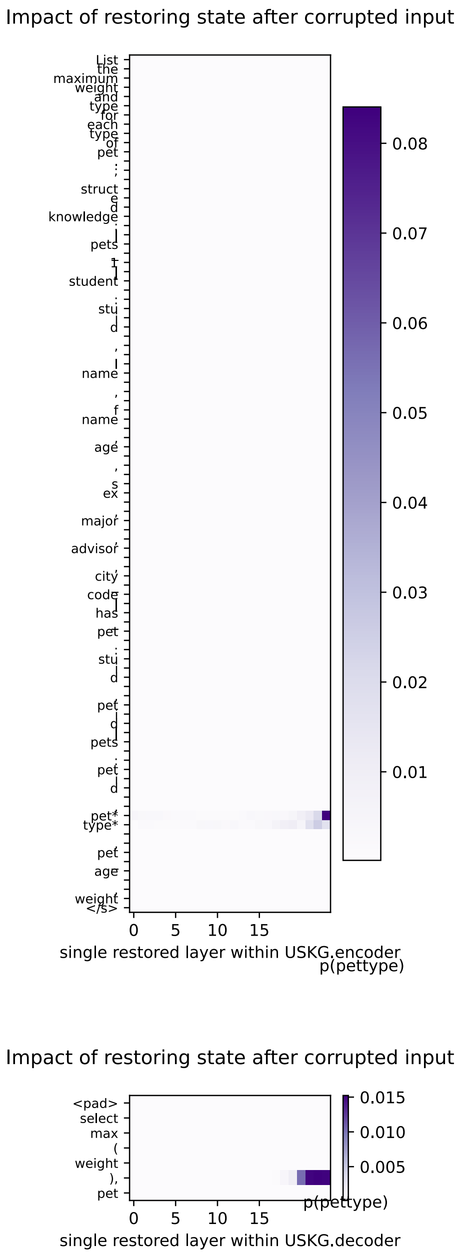
Table 12: Struct context corruption (SCC) effect. Clean+ means clean prediction is correct, with SCC it is wrong. Vice versa, SCC+ means clean prediction is wrong, but SCC makes it correct. We see the number of cases in which such corruption has positive / negative effects are on the same level, indicating that the two factors (information removal & distraction removal) both exist in structure context corruption.



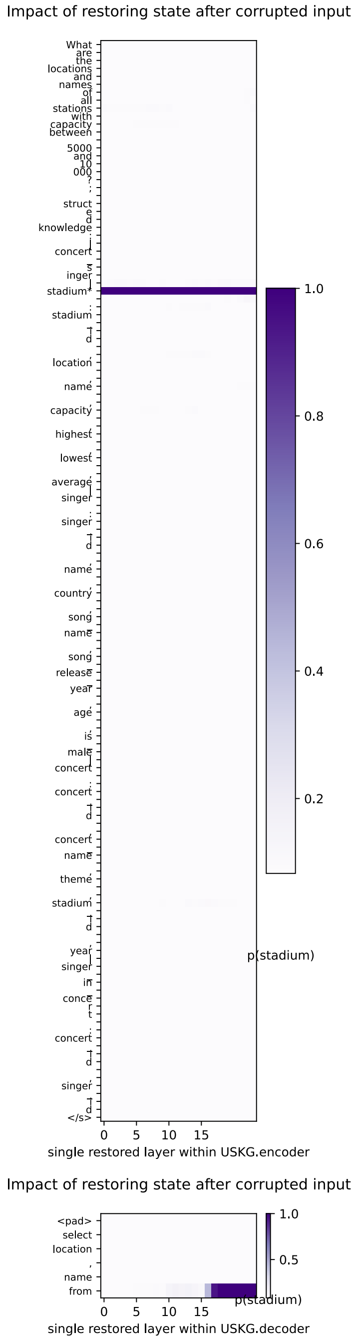
(column-1)



(column-2)

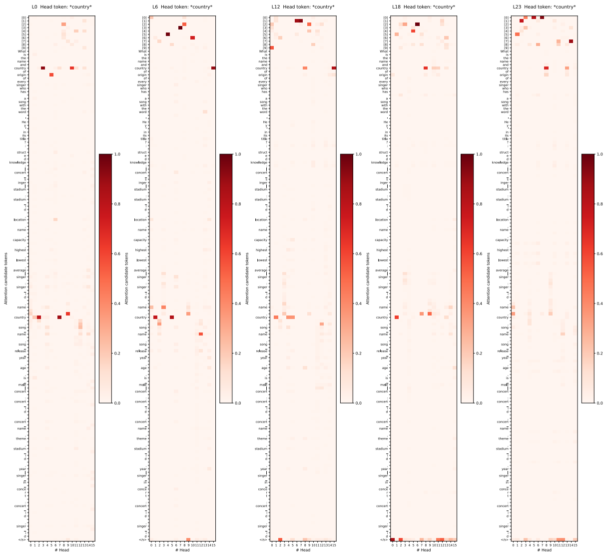


(column-3)

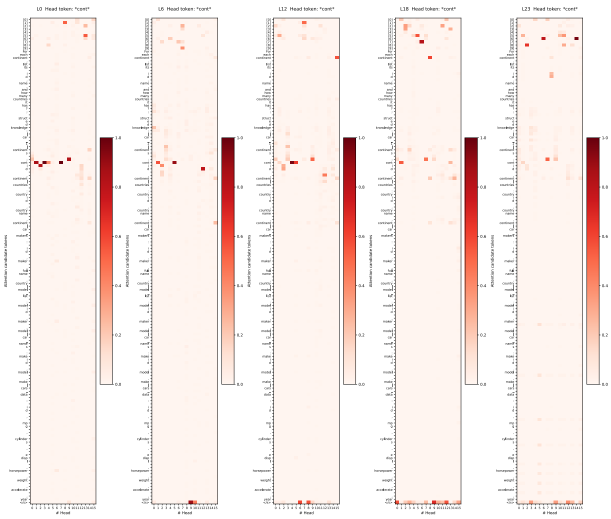


(table-1)

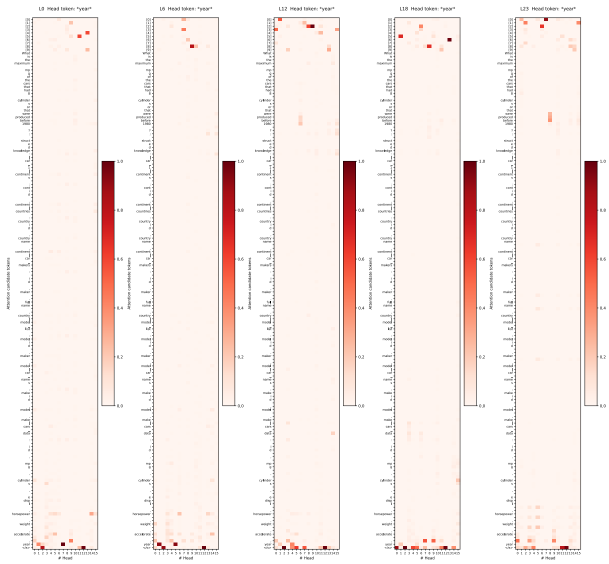
Figure 6: Encoder state restoration effectiveness. Multi-token nodes are usually harder to recover by restoring a single state. Supplementary for Figure 2.



(column-1)



(column-2)



(column-3)

Figure 7: Encoder self-attention direct visualization.

Dev-L1	Head 0	Head 1	Head 2	Head 3	Head 4	Head 5	Head 6	Head 7	Head 8	Head 9	Head 10	Head 11	Head 12	Head 13	Head 14	Head 15
prefix#0	0.00 / 0.00	0.02 / 0.03	0.04 / 0.03	0.01 / 0.01	0.00 / 0.00	0.09 / 0.11	0.02 / 0.02	0.00 / 0.00	0.01 / 0.01	0.01 / 0.01	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.04 / 0.05	0.01 / 0.02
prefix#1	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.02 / 0.01	0.01 / 0.02	0.03 / 0.03	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.00	0.05 / 0.05	0.00 / 0.00
prefix#2	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00	0.04 / 0.05	0.09 / 0.08	0.11 / 0.06	0.01 / 0.01	0.00 / 0.00	0.01 / 0.01	0.02 / 0.01	0.00 / 0.00	0.03 / 0.02	0.01 / 0.01	0.01 / 0.01	0.01 / 0.02	0.05 / 0.05
prefix#3	0.01 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.03 / 0.03	0.03 / 0.02	0.04 / 0.03	0.00 / 0.00	0.03 / 0.03	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.01	0.02 / 0.04
prefix#4	0.01 / 0.01	0.01 / 0.01	0.01 / 0.01	0.00 / 0.00	0.05 / 0.04	0.02 / 0.02	0.02 / 0.02	0.00 / 0.00	0.01 / 0.02	0.00 / 0.00	0.00 / 0.00	0.00 / 0.01	0.00 / 0.00	0.00 / 0.00	0.02 / 0.02	0.02 / 0.03
prefix#5	0.02 / 0.02	0.08 / 0.09	0.00 / 0.00	0.01 / 0.01	0.04 / 0.04	0.02 / 0.02	0.03 / 0.03	0.00 / 0.00	0.01 / 0.00	0.06 / 0.04	0.00 / 0.01	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.01 / 0.01	0.00 / 0.01
prefix#6	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.02 / 0.03	0.00 / 0.00	0.00 / 0.01	0.03 / 0.03	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.01	0.04 / 0.03	0.00 / 0.00
prefix#7	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.03 / 0.04	0.06 / 0.02	0.02 / 0.01	0.00 / 0.00	0.08 / 0.09	0.00 / 0.00	0.00 / 0.00	0.07 / 0.08	0.00 / 0.00	0.00 / 0.00	0.02 / 0.01	0.01 / 0.01
prefix#8	0.05 / 0.06	0.06 / 0.05	0.01 / 0.02	0.00 / 0.00	0.00 / 0.00	0.02 / 0.03	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.02	0.00 / 0.00	0.00 / 0.00	0.02 / 0.02	0.00 / 0.01
prefix#9	0.01 / 0.01	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.01	0.01 / 0.02	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.01 / 0.02	0.00 / 0.00	0.00 / 0.00	0.02 / 0.03	0.02 / 0.05
text	0.05 / 0.01	0.04 / 0.03	0.00 / 0.00	0.02 / 0.01	0.03 / 0.01	0.06 / 0.05	0.12 / 0.09	0.01 / 0.00	0.21 / 0.11	0.00 / 0.00	0.41 / 0.02	0.09 / 0.01	0.00 / 0.00	0.42 / 0.04	0.05 / 0.03	0.35 / 0.06
self	0.33 / 0.34	0.44 / 0.39	0.81 / 0.80	0.20 / 0.22	0.19 / 0.18	0.09 / 0.08	0.02 / 0.02	0.26 / 0.22	0.00 / 0.00	0.63 / 0.66	0.28 / 0.50	0.11 / 0.08	0.45 / 0.42	0.13 / 0.33	0.09 / 0.09	0.01 / 0.01
context	0.50 / 0.51	0.28 / 0.33	0.09 / 0.10	0.69 / 0.67	0.50 / 0.53	0.40 / 0.48	0.50 / 0.57	0.04 / 0.04	0.40 / 0.49	0.26 / 0.26	0.30 / 0.47	0.54 / 0.63	0.39 / 0.45	0.37 / 0.55	0.49 / 0.53	0.38 / 0.60
others	0.02 / 0.01	0.02 / 0.02	0.03 / 0.03	0.03 / 0.01	0.02 / 0.01	0.06 / 0.06	0.17 / 0.15	0.69 / 0.73	0.19 / 0.19	0.00 / 0.00	0.01 / 0.00	0.10 / 0.10	0.15 / 0.12	0.04 / 0.05	0.12 / 0.10	0.12 / 0.11

(a) Dev-L1

Dev-L12	Head 0	Head 1	Head 2	Head 3	Head 4	Head 5	Head 6	Head 7	Head 8	Head 9	Head 10	Head 11	Head 12	Head 13	Head 14	Head 15
prefix#0	0.00 / 0.00	0.06 / 0.07	0.04 / 0.06	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.07 / 0.10	0.00 / 0.00	0.00 / 0.00	0.01 / 0.06	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00
prefix#1	0.00 / 0.01	0.01 / 0.05	0.00 / 0.00	0.03 / 0.04	0.00 / 0.00	0.00 / 0.00	0.10 / 0.05	0.26 / 0.14	0.00 / 0.01	0.00 / 0.00	0.01 / 0.02	0.02 / 0.01	0.00 / 0.01	0.08 / 0.11	0.00 / 0.00	0.00 / 0.00
prefix#2	0.16 / 0.12	0.01 / 0.02	0.01 / 0.01	0.02 / 0.02	0.00 / 0.00	0.00 / 0.00	0.10 / 0.05	0.18 / 0.21	0.01 / 0.01	0.15 / 0.18	0.00 / 0.00	0.04 / 0.04	0.05 / 0.10	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00
prefix#3	0.08 / 0.03	0.00 / 0.01	0.00 / 0.01	0.00 / 0.00	0.00 / 0.00	0.01 / 0.00	0.04 / 0.02	0.00 / 0.00	0.01 / 0.01	0.07 / 0.04	0.03 / 0.01	0.03 / 0.01	0.00 / 0.01	0.01 / 0.01	0.01 / 0.02	0.01 / 0.05
prefix#4	0.07 / 0.04	0.06 / 0.08	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.05 / 0.12	0.00 / 0.00	0.00 / 0.00	0.02 / 0.01	0.02 / 0.01	0.14 / 0.08	0.00 / 0.00	0.03 / 0.02	0.01 / 0.01	0.00 / 0.00
prefix#5	0.05 / 0.05	0.06 / 0.05	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.07 / 0.03	0.05 / 0.06	0.00 / 0.00	0.00 / 0.01	0.00 / 0.01	0.11 / 0.06	0.00 / 0.00	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00
prefix#6	0.01 / 0.01	0.02 / 0.06	0.13 / 0.09	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.02	0.00 / 0.00	0.00 / 0.01	0.01 / 0.02	0.01 / 0.02	0.01 / 0.01	0.00 / 0.00	0.03 / 0.02	0.00 / 0.00
prefix#7	0.02 / 0.01	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.00	0.02 / 0.02	0.00 / 0.00	0.20 / 0.19	0.01 / 0.01	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00
prefix#8	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.04 / 0.02	0.00 / 0.01	0.00 / 0.00	0.07 / 0.09	0.00 / 0.00	0.00 / 0.01
prefix#9	0.11 / 0.12	0.01 / 0.01	0.00 / 0.00	0.02 / 0.03	0.00 / 0.00	0.00 / 0.00	0.00 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.03 / 0.05	0.02 / 0.01	0.03 / 0.03	0.00 / 0.00	0.00 / 0.00
text	0.03 / 0.02	0.02 / 0.02	0.00 / 0.00	0.00 / 0.00	0.02 / 0.01	0.04 / 0.02	0.11 / 0.08	0.01 / 0.01	0.25 / 0.05	0.00 / 0.00	0.18 / 0.11	0.01 / 0.01	0.00 / 0.00	0.14 / 0.08	0.13 / 0.07	0.71 / 0.18
self	0.18 / 0.20	0.61 / 0.44	0.33 / 0.39	0.27 / 0.32	0.57 / 0.49	0.25 / 0.28	0.01 / 0.01	0.11 / 0.09	0.00 / 0.00	0.28 / 0.23	0.02 / 0.01	0.07 / 0.07	0.44 / 0.38	0.04 / 0.03	0.15 / 0.13	0.01 / 0.01
context	0.16 / 0.25	0.10 / 0.14	0.04 / 0.05	0.63 / 0.58	0.36 / 0.44	0.40 / 0.44	0.27 / 0.39	0.06 / 0.07	0.41 / 0.56	0.07 / 0.06	0.51 / 0.46	0.45 / 0.56	0.23 / 0.23	0.46 / 0.45	0.55 / 0.61	0.18 / 0.44
others	0.10 / 0.13	0.03 / 0.05	0.41 / 0.36	0.01 / 0.02	0.04 / 0.06	0.29 / 0.25	0.22 / 0.23	0.21 / 0.29	0.31 / 0.36	0.18 / 0.26	0.15 / 0.24	0.07 / 0.07	0.24 / 0.24	0.13 / 0.18	0.10 / 0.11	0.09 / 0.31

(b) Dev-L12

Dev-L23	Head 0	Head 1	Head 2	Head 3	Head 4	Head 5	Head 6	Head 7	Head 8	Head 9	Head 10	Head 11	Head 12	Head 13	Head 14	Head 15
prefix#0	0.00 / 0.00	0.03 / 0.01	0.00 / 0.00	0.09 / 0.04	0.04 / 0.02	0.09 / 0.14	0.11 / 0.07	0.55 / 0.06	0.01 / 0.01	0.07 / 0.03	0.04 / 0.01	0.00 / 0.04	0.06 / 0.32	0.02 / 0.00	0.00 / 0.00	0.01 / 0.00
prefix#1	0.03 / 0.03	0.02 / 0.04	0.13 / 0.09	0.00 / 0.00	0.01 / 0.00	0.00 / 0.00	0.06 / 0.03	0.00 / 0.00	0.00 / 0.01	0.01 / 0.00	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.00	0.04 / 0.02
prefix#2	0.08 / 0.01	0.00 / 0.20	0.00 / 0.00	0.04 / 0.00	0.00 / 0.00	0.00 / 0.01	0.12 / 0.01	0.00 / 0.00	0.00 / 0.01	0.03 / 0.03	0.00 / 0.00	0.00 / 0.00	0.07 / 0.08	0.03 / 0.00	0.00 / 0.00	0.01 / 0.00
prefix#3	0.00 / 0.00	0.00 / 0.03	0.14 / 0.01	0.10 / 0.00	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.04	0.05 / 0.01	0.03 / 0.01	0.00 / 0.00	0.00 / 0.00	0.12 / 0.09	0.00 / 0.00
prefix#4	0.02 / 0.01	0.01 / 0.01	0.09 / 0.16	0.10 / 0.02	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.04 / 0.05	0.00 / 0.00	0.01 / 0.00	0.01 / 0.45	0.01 / 0.42	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.05 / 0.01
prefix#5	0.00 / 0.00	0.17 / 0.04	0.01 / 0.00	0.01 / 0.00	0.00 / 0.00	0.00 / 0.00	0.04 / 0.02	0.01 / 0.04	0.00 / 0.00	0.00 / 0.21	0.08 / 0.24	0.02 / 0.05	0.00 / 0.03	0.02 / 0.01	0.01 / 0.00	0.00 / 0.01
prefix#6	0.01 / 0.00	0.03 / 0.20	0.01 / 0.03	0.08 / 0.01	0.00 / 0.00	0.00 / 0.06	0.06 / 0.23	0.00 / 0.00	0.00 / 0.00	0.00 / 0.01	0.01 / 0.00	0.00 / 0.00	0.03 / 0.02	0.01 / 0.09	0.10 / 0.17	0.00 / 0.00
prefix#7	0.01 / 0.01	0.01 / 0.11	0.02 / 0.00	0.00 / 0.00	0.00 / 0.03	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.14	0.02 / 0.01	0.03 / 0.00	0.01 / 0.00	0.00 / 0.00	0.23 / 0.01	0.00 / 0.00
prefix#8	0.02 / 0.05	0.01 / 0.01	0.06 / 0.14	0.00 / 0.00	0.04 / 0.02	0.00 / 0.00	0.03 / 0.01	0.01 / 0.01	0.01 / 0.24	0.00 / 0.00	0.07 / 0.02	0.12 / 0.03	0.00 / 0.00	0.32 / 0.06	0.07 / 0.02	0.01 / 0.02
prefix#9	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.07	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.07 / 0.00	0.00 / 0.00
text	0.01 / 0.00	0.07 / 0.04	0.00 / 0.00	0.01 / 0.00	0.02 / 0.01	0.02 / 0.01	0.05 / 0.02	0.00 / 0.01	0.73 / 0.06	0.00 / 0.00	0.05 / 0.01	0.00 / 0.00	0.00 / 0.00	0.10 / 0.01	0.01 / 0.00	0.10 / 0.05
self	0.46 / 0.40	0.09 / 0.04	0.41 / 0.37	0.07 / 0.21	0.29 / 0.24	0.03 / 0.11	0.02 / 0.03	0.25 / 0.57	0.01 / 0.01	0.25 / 0.26	0.03 / 0.00	0.04 / 0.05	0.28 / 0.19	0.14 / 0.11	0.10 / 0.10	0.04 / 0.02
context	0.28 / 0.44	0.37 / 0.22	0.06 / 0.07	0.21 / 0.53	0.31 / 0.44	0.77 / 0.60	0.36 / 0.50	0.10 / 0.24	0.17 / 0.61	0.29 / 0.13	0.33 / 0.09	0.22 / 0.23	0.15 / 0.12	0.25 / 0.59	0.26 / 0.56	0.46 / 0.68
others	0.08 / 0.04	0.18 / 0.05	0.08 / 0.12	0.30 / 0.18	0.26 / 0.21	0.07 / 0.05	0.14 / 0.08	0.01 / 0.01	0.06 / 0.05	0.32 / 0.15	0.27 / 0.07	0.52 / 0.15	0.39 / 0.22	0.11 / 0.13	0.03 / 0.02	0.25 / 0.18

(c) Dev-L23

Train-L23	Head 0	Head 1	Head 2	Head 3	Head 4	Head 5	Head 6	Head 7	Head 8	Head 9	Head 10	Head 11	Head 12	Head 13	Head 14	Head 15
prefix#0	0.00 / 0.00	0.02 / 0.01	0.00 / 0.00	0.08 / 0.04	0.04 / 0.02	0.05 / 0.12	0.13 / 0.05	0.52 / 0.05	0.01 / 0.01	0.08 / 0.03	0.04 / 0.01	0.00 / 0.05	0.02 / 0.35	0.02 / 0.00	0.00 / 0.00	0.02 / 0.00
prefix#1	0.05 / 0.07	0.02 / 0.03	0.11 / 0.09	0.00 / 0.00	0.01 / 0.00	0.00 / 0.00	0.03 / 0.03	0.00 / 0.00	0.00 / 0.00	0.01 / 0.00	0.02 / 0.01	0.00 / 0.00	0.01 / 0.00	0.00 / 0.00	0.02 / 0.00	0.04 / 0.03
prefix#2	0.10 / 0.01	0.00 / 0.17	0.00 / 0.00	0.04 / 0.00	0.00 / 0.00	0.00 / 0.01	0.10 / 0.01	0.00 / 0.00	0.00 / 0.01	0.04 / 0.03	0.00 / 0.00	0.00 / 0.00	0.10 / 0.11	0.02 / 0.00	0.00 / 0.00	0.01 / 0.00
prefix#3	0.00 / 0.00	0.01 / 0.03	0.18 / 0.01	0.10 / 0.00	0.00 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.03	0.05 / 0.01	0.02 / 0.01	0.00 / 0.00	0.00 / 0.00	0.11 / 0.09	0.00 / 0.00
prefix#4	0.02 / 0.01	0.01 / 0.01	0.08 / 0.21	0.11 / 0.02	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.05 / 0.06	0.00 / 0.00	0.01 / 0.00	0.01 / 0.45	0.00 / 0.49	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.06 / 0.01
prefix#5	0.00 / 0.00	0.13 / 0.04	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.03 / 0.01	0.02 / 0.04	0.00 / 0.00	0.00 / 0.21	0.08 / 0.25	0.01 / 0.03	0.00 / 0.02	0.03 / 0.01	0.01 / 0.00	0.00 / 0.01
prefix#6	0.01 / 0.00	0.02 / 0.20	0.01 / 0.03	0.12 / 0.01	0.00 / 0.00	0.00 / 0.06	0.03 / 0.21	0.00 / 0.00	0.01 / 0.01	0.00 / 0.01	0.01 / 0.00	0.00 / 0.00	0.05 / 0.03	0.01 / 0.08	0.05 / 0.14	0.00 / 0.00
prefix#7	0.00 / 0.01	0.01 / 0.16	0.01 / 0.00	0.00 / 0.00	0.00 / 0.05	0.00 / 0.00	0.00 / 0.00	0.01 / 0.00	0.00 / 0.00	0.00 / 0.17	0.02 / 0.01	0.04 / 0.00	0.01 / 0.00	0.00 / 0.00	0.22 / 0.01	0.00 / 0.00
prefix#8	0.03 / 0.05	0.01 / 0.00	0.03 / 0.11	0.00 / 0.00	0.05 / 0.02	0.01 / 0.01	0.02 / 0.02	0.01 / 0.01	0.00 / 0.24	0.00 / 0.00	0.07 / 0.02	0.12 / 0.02	0.00 / 0.00	0.33 / 0.07	0.09 / 0.03	0.01 / 0.02
prefix#9	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.00 / 0.07	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	0.06 / 0.00	0.01 / 0.01
text	0.01 / 0.00	0.07 / 0.04	0.00 / 0.00	0.01 / 0.00	0.03 / 0.01	0.02 / 0.01	0.06 / 0.02	0.00 / 0.01	0.75 / 0.05	0.00 / 0.00	0.05 / 0.01	0.00 / 0.00	0.00 / 0.00	0.10 / 0.01	0.01 / 0.00	0.10 / 0.04
self	0.44 / 0.39	0.09 / 0.04	0.44 / 0.35	0.06 / 0.20	0.30 / 0.23	0.03 / 0.12	0.02 / 0.03	0.24 / 0.58	0.01 / 0.01	0.21 / 0.24	0.03 / 0.00	0.04 / 0.05	0.30 / 0.19	0.13 / 0.10	0.11 / 0.11	0.05 / 0.02
context	0.27 / 0.42	0.41 / 0.22	0.06 / 0.07	0.18 / 0.54	0.32 / 0.45	0.81 / 0.61	0.39 / 0.53	0.12 / 0.24	0.15 / 0.62	0.28 / 0.12	0.34 / 0.10	0.25 / 0.20	0.12 / 0.10	0.26 / 0.59	0.28 / 0.58	0.45 / 0.69
others	0.07 / 0.03	0.19 / 0.05	0.07 / 0.12	0.29 / 0.17	0.24 / 0.20	0.07 / 0.06	0.15 / 0.08	0.01 / 0.01	0.06 / 0.05	0.35 / 0.16	0.27 / 0.06	0.51 / 0.13	0.38 / 0.20	0.10 / 0.12	0.03 / 0.03	0.25 / 0.17

(d) Training-L23

Table 13: Encoder attention weights to each sections from all relevant / non-relevant nodes on each head. Values in red are high for relevant nodes, and in blue are high for non-relevant nodes. (The criteria is based on heuristics: $x/y - y/x + x - y > 2.0$) Supplementary for Table 6. The trend is almost the same between training and dev set.

Corruption part	Low layers	High layers	All layers
Prefix	0.9802	0.9226	0.8499
Text	0.9862	0.9331	0.8594
Self-node	0.9625	0.8054	0.6572
Struct	0.9526	0.7498	0.5507
Text+struct	0.9478	0.6707	0.4909
StructContext	0.9876	0.9473	0.9195
Text+StructContext	0.9821	0.8757	0.6738
All	0.9376	0.5974	0.4509

(a) Columns

Corruption part	Low layers	High layers	All layers
Prefix	0.9796	0.8033	0.5895
Text	0.9913	0.9775	0.9551
Self-node	0.9862	0.8126	0.4924
Struct	0.9631	0.7350	0.3310
Text+struct	0.9545	0.6817	0.2948
StructContext	0.9844	0.9384	0.7964
Text+StructContext	0.9799	0.8890	0.5685
All	0.8988	0.4484	0.2337

(b) Tables

Corruption part	Low layers	High layers	All layers
Prefix	0.9923	0.9541	0.9259
Text	0.9942	0.9856	0.9746
Self-node	0.9917	0.9833	0.9519
Struct	0.9874	0.9468	0.9137
Text+struct	0.9869	0.9405	0.9117
StructContext	0.9924	0.9501	0.9384
Text+StructContext	0.9911	0.9436	0.9275
All	0.9751	0.9278	0.9078

(c) Table aliases

Table 14: Attention corruption study on attentions from node-of-interest to each section, the results for all node types.

Corruption part	Weights			Logits		
	Low window	High window	All layers	Low layers	High layers	All layers
$T \to S$	0.9908	0.9787	0.9671	0.9836	0.9622	0.9543
$S \to T$	0.9929	0.9647	0.9071	0.9847	0.8381	0.6879
$T \leftrightarrow S$	0.9889	0.9413	0.8416	0.9762	0.7769	0.6138
$T \to P$	0.9805	0.9460	0.8856	0.9694	0.9251	0.8703
$S \to P$	0.9782	0.9268	0.8459	0.9680	0.8821	0.8317
$TS \to P$	0.9707	0.7953	0.5793	0.9524	0.7073	0.5778
$T \to T$	0.9882	0.9491	0.8048	0.9578	0.9028	0.7631
$S \to S$	0.9574	0.6447	0.5213	0.6859	0.5708	0.4601
all	0.8502	0.3578	0.2101	0.5213	0.2521	0.1502

(a) Columns

Corruption part	Weights			Logits		
	Low window	High window	All layers	Low layers	High layers	All layers
$T \to S$	0.9908	0.9784	0.9737	0.9888	0.9774	0.9732
$S \to T$	0.9906	0.9755	0.9319	0.9829	0.8792	0.6925
$T \leftrightarrow S$	0.9868	0.9654	0.9106	0.9777	0.8498	0.6302
$T \to P$	0.9887	0.9792	0.9471	0.9867	0.9718	0.9469
$S \to P$	0.9695	0.7746	0.6130	0.9642	0.7444	0.6515
$TS \to P$	0.9625	0.7064	0.4773	0.9547	0.6173	0.4781
$T \to T$	0.9864	0.9833	0.9441	0.9732	0.9802	0.9304
$S \to S$	0.9350	0.6739	0.3604	0.5448	0.5519	0.3727
all	0.8045	0.3939	0.1594	0.4762	0.2360	0.1012

(b) Tables

Corruption part	Weights			Logits		
	Low window	High window	All layers	Low layers	High layers	All layers
$T \to S$	0.9971	0.9953	0.9933	0.9965	0.9943	0.9919
$S \to T$	0.9980	0.9949	0.9926	0.9977	0.9918	0.9845
$T \leftrightarrow S$	0.9965	0.9920	0.9878	0.9958	0.9895	0.9788
$T \to P$	0.9945	0.9739	0.9585	0.9947	0.9695	0.9549
$S \to P$	0.9962	0.9853	0.9829	0.9961	0.9857	0.9830
$TS \to P$	0.9921	0.9459	0.8976	0.9924	0.9401	0.9022
$T \to T$	0.9903	0.9830	0.9432	0.9872	0.9753	0.9268
$S \to S$	0.9834	0.9769	0.9320	0.9674	0.9727	0.9582
all	0.9257	0.8382	0.7269	0.8961	0.7883	0.7338

(c) Syntax tokens

Table 15: Attention corruption study on encoder self-attentions across input sections. For corruption part, "T" means text, "S" for structure, "P" for prefix; "$T \to S$" means corrupting the attention weights from text to structure tokens, i.e. text tokens as q and structure tokens as k. On top, "Weights" and "Logits" represent the attention corruption scheme. Supplementary for Table 3.

Corruption part	Weights			Logits		
	Low layers	High layers	All layers	Low layers	High layers	All layers
Text	0.9588	0.9815	0.9063	0.9565	0.9835	0.9132
Struct	0.8746	0.3577	0.3239	0.8067	0.5050	0.4333
Prefix	0.9260	0.9806	0.9114	0.9199	0.9746	0.9025
Struct ctx.	0.9420	0.9867	0.9429	0.9551	0.9981	0.9876
Self-node	0.9505	0.3911	0.3840	0.9320	0.5857	0.5382
All	0.5805	0.1613	0.1316	0.3284	0.1017	0.0597

(a) Column

Corruption part	Weights			Logits		
	Low layers	High layers	All layers	Low layers	High layers	All layers
Text	0.9895	0.9905	0.9840	0.9907	0.9909	0.9868
Struct	0.7901	0.0539	0.0672	0.7465	0.3822	0.3137
Prefix	0.9273	0.9900	0.9115	0.8963	0.9878	0.8784
Struct ctx.	0.9140	0.9940	0.9824	0.9236	0.9997	0.9954
Self-node	0.9100	0.1794	0.1738	0.9075	0.4587	0.4272
All	0.5965	0.0328	0.0341	0.4779	0.0985	0.0706

(b) Table

Corruption part	Weights			Logits		
	Low layers	High layers	All layers	Low layers	High layers	All layers
Text	0.9820	0.9377	0.8517	0.9770	0.9103	0.8305
Struct	0.9568	0.9970	0.9543	0.9317	0.9963	0.9326
Prefix	0.9608	0.9486	0.8332	0.9545	0.8971	0.7676
All	0.7780	0.8878	0.4590	0.6694	0.8221	0.3725

(c) Syntax tokens

Table 16: Attention corruption study on decoder cross-attentions to each input section. Supplementary for Table 4.

Text match	Exact (791)			Partial (443)			No match (202)		
	Low layers	High layers	All layers	Low layers	High layers	All layers	Low layers	High layers	All layers
$T \to S$	0.9942	0.9868	0.9784	0.9820	0.9699	0.9433	0.9882	0.9654	0.9555
$S \to T$	0.9977	0.9838	0.9461	0.9954	0.9525	0.8613	0.9806	0.9289	0.8456
$T \leftrightarrow S$	0.9954	0.9640	0.8725	0.9839	0.9239	0.8003	0.9773	0.8993	0.7890
$T \to P$	0.9915	0.9601	0.9269	0.9811	0.9346	0.8350	0.9552	0.9183	0.8206
$S \to P$	0.9895	0.9599	0.9307	0.9568	0.9437	0.8441	0.9660	0.8388	0.6529
all	0.9148	0.4417	0.2527	0.7743	0.3159	0.2053	0.7489	0.1492	0.0524

Table 17: Attention corruption study on **encoder self-attentions** across input sections (for column prediction). Comparison of results between target columns with different text-matching situations.

Text match	Exact (791)			Partial (443)			No-match (202)		
	Low layers	High layers	All layers	Low layers	High layers	All layers	Low layers	High layers	All layers
Text	0.9558	0.9845	0.9006	0.9578	0.9817	0.8859	0.9645	0.9759	0.9252
Struct	0.9740	0.5157	0.4410	0.7737	0.1094	0.1353	0.7410	0.1828	0.1946
Prefix	0.9809	0.9896	0.9641	0.9415	0.9744	0.9325	0.8188	0.9670	0.8052
StructCtx.	0.9739	0.9924	0.9610	0.9468	0.9572	0.8852	0.8813	0.9901	0.9369
Self-node	0.9926	0.5288	0.5167	0.8994	0.2956	0.2533	0.8979	0.1813	0.1999
all	0.7176	0.2169	0.1877	0.4343	0.0809	0.0672	0.3974	0.0915	0.0530

Table 18: Attention corruption study on **decoder cross-attentions** to each input section (for column prediction). Comparison of results between target columns with different text-matching situations.

Syntax-tok	$t \rightarrow s$	$s \rightarrow t$	$t \leftrightarrow s$	$t \rightarrow p$	$s \rightarrow p$	$ts \rightarrow p$	$t \rightarrow t$	$s \rightarrow s$	**All**	**Eff_cnt**	**All_cnt**	**Eff_rate**
!=	0.9984	0.9995	0.9825	0.8485	0.9998	0.9070	0.3025	0.9193	0.0284	20	20	1.0000
(	0.9999	0.9794	0.9946	0.9719	0.9795	0.9685	0.8983	0.9467	0.1856	98	675	0.1452
)	0.9257	0.9926	0.9716	0.9914	0.9074	0.8420	0.9244	0.4715	0.1108	13	23	0.5652
*	1.0000	0.9999	1.0000	0.9997	0.9999	0.9998	0.9999	0.9982	0.1802	10	381	0.0262
=	0.9784	0.9923	0.9786	0.9378	0.8940	0.7439	0.9466	0.5933	0.1149	128	968	0.1322
>	1.0000	1.0000	1.0000	0.9509	0.9990	0.9652	0.8193	0.9845	0.0219	68	101	0.6733
>=	1.0000	1.0000	1.0000	0.9151	1.0000	0.9048	0.2593	0.9964	0.1703	11	30	0.3667
and	0.7931	0.8981	0.6727	0.4962	0.8409	0.3468	0.3603	0.6749	0.0554	29	39	0.7436
as	0.9212	0.8936	0.8349	0.7992	0.5034	0.3457	0.8478	0.3288	0.1002	52	952	0.0546
asc	0.9723	0.9096	0.9437	0.4939	0.8596	0.3612	0.6133	0.6725	0.0408	19	19	1.0000
avg	0.9996	0.9940	0.9972	0.7660	0.9996	0.3619	0.7743	0.9513	0.0360	63	65	0.9692
between	0.9999	0.9999	1.0000	0.7788	0.9999	0.3407	0.0009	0.9999	0.0000	6	6	1.0000
count	0.9911	0.9977	0.9960	0.9641	0.9850	0.7469	0.8609	0.8691	0.0220	294	406	0.7241
desc	0.9939	0.9992	0.9942	0.9198	0.9191	0.7296	0.9044	0.6997	0.1144	82	164	0.5000
distinct	0.8663	0.8651	0.8284	0.6270	0.9118	0.4787	0.6194	0.7636	0.0014	26	26	1.0000
except	0.9517	0.9847	0.9415	0.6213	0.9311	0.0783	0.4062	0.7137	0.0002	21	21	1.0000
from	0.9924	0.9703	0.9671	0.9781	0.9519	0.6276	0.8407	0.5194	0.1384	263	1196	0.2199
group	0.9895	0.9925	0.9771	0.8332	0.9838	0.5694	0.9167	0.7823	0.0282	241	265	0.9094
having	0.9978	0.9924	0.9894	0.5962	0.9993	0.3893	0.8985	0.9348	0.0262	77	81	0.9506
in	0.9973	0.9995	0.9992	0.9995	0.9253	0.5859	0.7717	0.3267	0.1096	8	50	0.1600
intersect	0.9649	0.9938	0.9522	0.2385	0.9592	0.0575	0.4348	0.8243	0.0001	34	34	1.0000
join	0.9099	0.6489	0.4976	0.7286	0.3030	0.1984	0.7286	0.1891	0.0158	44	496	0.0887
like	1.0000	1.0000	1.0000	0.9997	0.9997	0.8233	0.2700	0.8809	0.0062	12	12	1.0000
limit	0.9932	0.9854	0.9684	0.8400	0.9992	0.7895	0.8076	0.7534	0.1531	26	177	0.1469
max	0.8483	0.9925	0.9741	0.8141	0.9458	0.4659	0.5214	0.9008	0.0633	29	30	0.9667
min	0.9962	1.0000	1.0000	0.9906	0.9993	0.9377	0.7412	0.9999	0.0450	14	18	0.7778
not	0.9884	0.9515	0.9360	0.9510	0.9067	0.6624	0.8928	0.7208	0.0117	45	46	0.9783
or	0.9788	0.9728	0.9582	0.7773	0.9856	0.9181	0.4193	0.9718	0.0311	34	34	1.0000
order	0.9964	0.9891	0.9826	0.7079	0.9802	0.4836	0.8542	0.8393	0.0680	197	221	0.8914
sum	0.9054	0.9988	0.8969	0.8431	0.9998	0.6812	0.4672	0.9511	0.0119	20	22	0.9091
union	0.9990	0.8977	0.8037	0.2034	0.7390	0.3218	0.0174	0.7111	0.0001	6	6	1.0000
where	0.9820	0.9900	0.9702	0.9035	0.9576	0.7533	0.8924	0.7699	0.0858	350	516	0.6783

Table 19: The accuracy of each syntax token when corrupting encoder self-attention. This table shows the detailed effect of such corruptions. Eff_rate means for all occasions with this token as ground truth, the percentage of predictions being affected (became wrong) by the corruption.

Layer	Section	Weights		Logits	
		Exact	Exec	Exact	Exec
Low	$S \to T$	0.6683	0.6789	0.6538	0.6654
Low	$T \to S$	0.6586	0.6779	0.6518	0.6721
Low	$T \leftrightarrow S$	0.6538	0.6712	0.6422	0.6634
Mid	$S \to T$	0.6180	0.6344	0.5019	0.5184
Mid	$T \to S$	0.6470	0.6692	0.6451	0.6586
Mid	$T \leftrightarrow S$	0.5957	0.6141	0.4294	0.4468
High	$S \to T$	0.5938	0.6141	0.4072	0.4362
High	$T \to S$	0.6248	0.6431	0.6151	0.6412
High	$T \leftrightarrow S$	0.5242	0.5387	0.3250	0.3559
all	$S \to T$	0.5145	0.5387	0.2234	0.2369
all	$T \to S$	0.5938	0.6190	0.5783	0.6103
all	$T \leftrightarrow S$	0.3926	0.4236	0.1509	0.1712
(clean)	(clean)	0.6692	0.6809	0.6692	0.6809

(a) Encoder self-attention

Layer	Section	Weights		Logits	
		Exact	Exec	Exact	Exec
Low	Text	0.5135	0.5406	0.4400	0.4700
Low	Struct	0.3327	0.3472	0.2592	0.2679
Mid	Text	0.2737	0.2950	0.1818	0.2002
Mid	Struct	0.3762	0.3801	0.2147	0.2128
High	Text	0.4836	0.4255	0.3994	0.3627
High	Struct	0.0068	0.0068	0.1015	0.1044
all	Text	0.0996	0.1015	0.0706	0.0812
all	Struct	0.0010	0.0010	0.0648	0.0638
(clean)	(clean)	0.6692	0.6809	0.6692	0.6809

(b) Decoder cross-attention

Layer	Exact	Exec
Low	0.0416	0.0329
Mid	0.0696	0.0841
High	0.2089	0.2157
all	0.0000	0.0000

(c) Decoder self-attention. Only experimented with blocking all tokens and using "weights" corruption type, essentially zeroing out the self-attention output vector.

Table 20: End-to-end SQL performance with different attention corruption settings. Supplementary for Table 8.

Error class	Definition
	S-class: Clause-level Semantics Errors
S0	missing / wrong aggregator
S1	missing / wrong condition clause or ordering
S2	missing / wrong literal value
	N-class: Node (column / table name) Errors
N0	invalid (hallucinated) node name, either token or natural language phrases
N1	using '*' for an actual column
N2	valid but wrong node name (not including '*')
	J-class: Join-chain Errors
J0	extra join, but still correct
J1	missing alias reference, may cause ambiguous-column error depending on the schema
	A-class: Low-level Syntax Errors
A0	Unpaired brackets / quotes
A1	Misspelled keyword
A2	Non-ending token
	B-class: Clause-level Syntax Errors
B0	Missing / extra / misplaced / partial clauses (causing syntax error)
B2	Alias error (t1 -> t1.col, t1 -> t1st, errors like this)
B3	Missing / extra operator (causing syntax error)
	C-class: Other High-level Semantics Errors
C0	Natural language expression of SQL, or unquoted string values
C3	(Not really an error) - equivalent or alternative correct answer

Table 21: Decoder cross-attention corruption error analysis: detailed categories and descriptions.

Category	**S**			**J**		**N**			**A**			**B**			**C**	
	S0	**S1**	**S2**	**J0**	**J1**	**N0**	**N1**	**N2**	**A0**	**A1**	**A2**	**B0**	**B2**	**B3**	**C0**	**C3**
all-text	31	21	1	0	1	0	6	2	0	0	0	0	0	0	0	1
all-struct	0	0	0	0	0	46	0	1	1	1	1	6	1	2	1	0

Table 22: Decoder cross-attention corruption error analysis: detailed break down.

Error class	Definition
	A-class: Low-level Syntax Errors
A0	Unpaired brackets / quotes
A1	Misspelled keyword (some keywords like 'avg', 'distinct' could be tokenized)
A2	Non-ending token
	B-class: Clause-level Syntax Errors
B0	Missing / extra / misplaced clauses (causing syntax error)
B2	Alias error (t1 -> t1.col, t1 -> t1st, errors like these)
B3	Missing / extra operator
	C-class: High-level Semantics Errors
C0	Natural language expression of SQL, or unquoted string values
C1	Wrong node name with similar surface form
C2	Valid SQL but wrong semantics from user query
C3	(Not really an error) - equivalent or alternative correct answer

Table 23: Decoder self-attention corruption error analysis: detailed categories and descriptions.

Category	A			B				C			
	A0	**A1**	**A2**	**B0**	**B1**	**B2**	**B3**	**C0**	**C1**	**C2**	**C3**
low	31	19	5	10	1	4	0	0	0	1	0
mid	27	0	0	16	0	0	3	1	0	1	2
high	0	0	0	14	0	1	9	21	4	1	2

Table 24: Decoder self-attention corruption error analysis: detailed break down.