IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Mathematical Modelling of Engineering Problems
Vol. 13, No. 7, July, 2026, pp. 959-965
Journal homepage: http://iieta.org/journals/mmep

A Comparative Evaluation of T5 and BART Transformer Models for Natural Language to SQL Query Generation on the Spider Benchmark

Ayesha Mafrah A M*, Suresha Mallappa

Department of Studies in Computer Science, University of Mysore, Mysore 570006, India

Corresponding Author Email: ayeshmaffi@gmail.com

Copyright: ©2026 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/mmep.130703

Received: 13 January 2026
Revised: 3 April 2026
Accepted: 16 April 2026
Available online: 30 September 2026

Keywords:
BART, complex SQL generation, natural language interfaces, Natural Language Interfaces to Databases, sequence-to-sequence models, Spider dataset, T5, text-to-SQL

ABSTRACT

Querying relational databases typically demands SQL expertise, which many end-users lack. Natural Language Interfaces to Databases (NLIDB) aim to close this gap by converting plain-language questions into executable SQL. Generating correct SQL becomes particularly hard when queries span multiple joined tables, contain nested subqueries, or rely on aggregation functions, challenges that go well beyond straightforward keyword matching. This paper reports a controlled fine-tuning experiment in which BART and T5, two widely used transformer sequence-to-sequence models, were trained on the Spider benchmark under matching conditions and compared across three metrics: Exact Match (EM), BLEU Score, and Logical Form Accuracy (LFA). BART recorded EM 15.29%, BLEU 36.06%, and LFA 15.43%; T5 recorded 8.29%, 25.03%, and 8.29%, respectively. Both models scored well below schema-aware systems such as RAT-SQL and RYANSQL, and the experimental evidence identifies the absence of schema linking as the chief bottleneck. The study contributes a reproducible baseline and identifies the most promising directions for closing the gap between general-purpose transformers and specialized text-to-SQL parsers.

1. INTRODUCTION

Structured databases are now central to decision-making across industry, government, and research, yet retrieving information from them still requires SQL proficiency that most end-users do not possess. Natural Language Interfaces to Databases (NLIDB) address this barrier directly: they accept a question in plain language and return an executable SQL query, removing the specialist skill requirement and making data more widely accessible.

Translation from natural language to SQL is considerably more difficult than it may first appear. Single-table lookups with straightforward filters are manageable even for older rule-based systems; however, queries that chain multiple tables through joins, nest sub-selects, or group results with HAVING clauses require considerably more. A system must reconcile the ambiguity of natural language with the rigid structural rules of SQL and the specific column and table names of an individual database schema, a combination that exposes the limits of any approach that ignores schema structure.

Among recent sequence-to-sequence architectures, BART [1] and T5 [2] stand out for their broad applicability. BART was pretrained as a denoising autoencoder: its decoder learns to recover original text from corrupted versions, an objective that appears to help with generation quality under noisy or underspecified input. T5 adopts a different approach, casting every NLP task as a text-to-text problem and learning a single unified generation framework. Both have achieved competitive results on translation, summarization, and reading-comprehension tasks. What is less clear is how either model performs when the output must be not free-form text but syntactically valid SQL drawn from an unseen database schema, a question this paper addresses directly.

For this comparison, Spider [3] was used: a benchmark built around 200 databases drawn from 138 distinct domains and comprising 10,181 labelled question–query pairs of varying complexity. Because the test databases are withheld from training, Spider measures genuine cross-domain generalization rather than schema memorization, which makes it a substantially harder challenge than same-domain datasets such as WikiSQL [4].

Schema-aware systems such as RAT-SQL [5] and RYANSQL [6] achieve strong Spider scores largely because they include explicit mechanisms that link each question word to its most relevant schema element. What has received less attention is how far a practitioner can get by simply fine-tuning an off-the-shelf model such as BART or T5 without any of those specialized components. This paper addresses that gap and makes the following contributions:

1. Controlled comparison: Fine-tuning and evaluating BART and T5 under identical conditions on Spider, providing a reproducible baseline for general-purpose transformer models on complex text-to-SQL translation.

2. Metric analysis: Reporting EM, BLEU Score, and Logical Form Accuracy (LFA), and analyzing the implications and known limitations of each metric for structured query evaluation.

3. Gap analysis: Positioning the results against schema-aware baselines to quantify the performance gap attributable to the absence of schema-linking mechanisms.

4. Roadmap for improvement: Identifying the architectural enhancements (schema linking, constrained decoding, and execution-accuracy optimization) most likely to close the observed gap.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the methodology. Section 4 presents the experimental results and discussion. Section 5 concludes the paper.

2. RELATED WORK

Work on NLIDB has a long history, running from hand-crafted grammar systems in the 1980s through to the deep learning and transformer era. The following review groups this work into three phases, ending with an account of how BART and T5 relate to the existing literature.

2.1 Rule-based and statistical approaches

The earliest NLIDB work depended on manually specified grammars. Warren and Pereira [7] built a Prolog-based interpreter that could answer natural-language questions with reasonable fidelity, though porting it to a new database required significant re-engineering. Androutsopoulos et al. [8] documented this fragility in a widely cited survey. Statistical approaches partially automated the process: Popescu et al. [9] showed that question-to-SQL mappings could be learned from annotated examples, and Zelle and Mooney [10] demonstrated that inductive logic programming was well suited for building semantic parsers from data. Zettlemoyer and Collins [11] later showed that probabilistic categorial grammars handled a wider range of query structures, though even their approach struggled when queries involved several joined tables.

2.2 Deep learning approaches

The deep learning era substantially reshaped text-to-SQL research. Zhong et al. [4] introduced Seq2SQL, using reinforcement learning to train on WikiSQL and achieving reasonable performance on single-table queries. The single-table constraint, however, meant that WikiSQL-trained systems had limited relevance for real-world databases. SQLNet [12] avoided the order-sensitivity problem in SQL generation by treating it as a slot-filling task over a fixed query skeleton, which improved consistency. Yavuz et al. [13] targeted WHERE-clause errors specifically and showed that targeted fixes at the clause level translated into meaningful gains at the full-query level. Yu et al. [3] introduced Spider and fundamentally changed the evaluation landscape: a model could no longer rely on memorizing schema-specific patterns, because every test database is unseen during training.

2.3 Transformer-based text-to-SQL models

Self-attention, introduced by Vaswani et al. [14], enabled transformers to build rich contextual representations and quickly became the standard backbone for NLP. BERT [15] showed that bidirectional pretraining on large corpora transfers well to downstream tasks, motivating a wave of transformer-based NLIDB systems. RAT-SQL [5] is particularly relevant here: Wang et al. augmented the standard encoder with relation-aware attention computed over a schema graph, letting the model explicitly represent foreign-key links and column memberships. This architectural choice proved decisive, producing 69.7% execution accuracy on Spider. Choi et al. [6] adopted a complementary approach with RYANSQL, decomposing complex queries recursively into sub-sketches that are filled slot by slot, reaching 60.6% execution accuracy. On the representation side, TaBERT [16] pretrains on aligned sentence–table pairs, TaPas [17] learns to parse questions against tables with weak supervision, and Deng et al. [18] showed that pretraining objectives designed around schema alignment directly improve text-to-SQL accuracy. BART [1] and T5 [2] lie outside this specialized lineage: they were built for general sequence generation and carry no SQL-specific inductive bias, which makes them a useful baseline for quantifying how much schema awareness contributes.

3. METHODOLOGY

Text-to-SQL generation is treated here as a supervised sequence-to-sequence problem: given a natural language question and a linearized database schema, the model must produce a valid SQL string. BART and T5 were each fine-tuned from their publicly available base checkpoints using the same data splits, tokenization scheme, optimizer settings, and number of training steps. The subsections below describe the formal setup, the two models, how schema information was encoded, how data were prepared, and how performance was measured.

3.1 Problem formulation

Denote a natural language question as $Q = \{q_1, q_2, ..., q_n\}$ (n tokens) and its paired SQL query as $S = \{s_1, s_2, ..., s_m\}$ (m tokens). Training seeks a function f_θ: $Q \rightarrow S$ mapping the input question to the target query. Both models were trained by minimizing the token-level cross-entropy loss over the training corpus $D = \{(Q_i, S_i)\}$, $i = 1, \ldots, N$:

$$\mathrm{L}(\theta) = -\Sigma_{\mathrm{j}} \log \mathrm{P}\left(\mathrm{s_j} \mid \mathrm{Q}, \mathrm{s}_{<\mathrm{j}}; \theta\right)$$

where, $P(s_j \mid Q, s_{<j}; \theta)$ is the conditional probability of the j-th output token. The global training objective is:

$$\theta^* = \arg\min_\theta (1/N) \Sigma_i L(\theta; Q_i, S_i)$$

At test time, greedy decoding was used throughout: at each step, the token with the highest predicted probability was appended until the end-of-sequence token was emitted. Beam search was deliberately omitted so that the inference conditions remained consistent with the base configurations reported for the baseline systems.

3.2 Model architecture

T5-base and BART-base are both encoder–decoder transformers, but their pretraining objectives and input conventions differ in ways that are relevant to SQL generation; these differences are summarized in Table 1. Figure 1 shows the end-to-end workflow used in this study for both models, and Figure 2 outlines the data preprocessing and training pipeline described in Section 3.3.

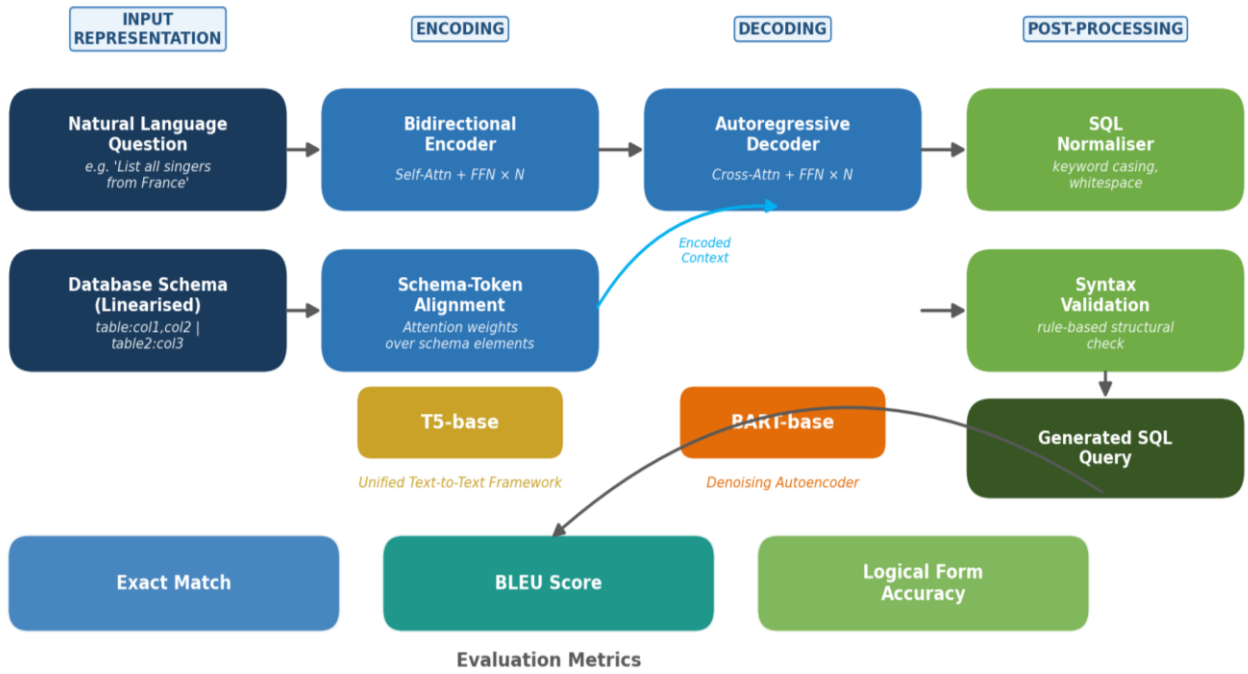


Figure 1. End-to-end architectural workflow of T5 and BART for Natural Language Interfaces to Databases (NLIDB) text-to-SQL generation

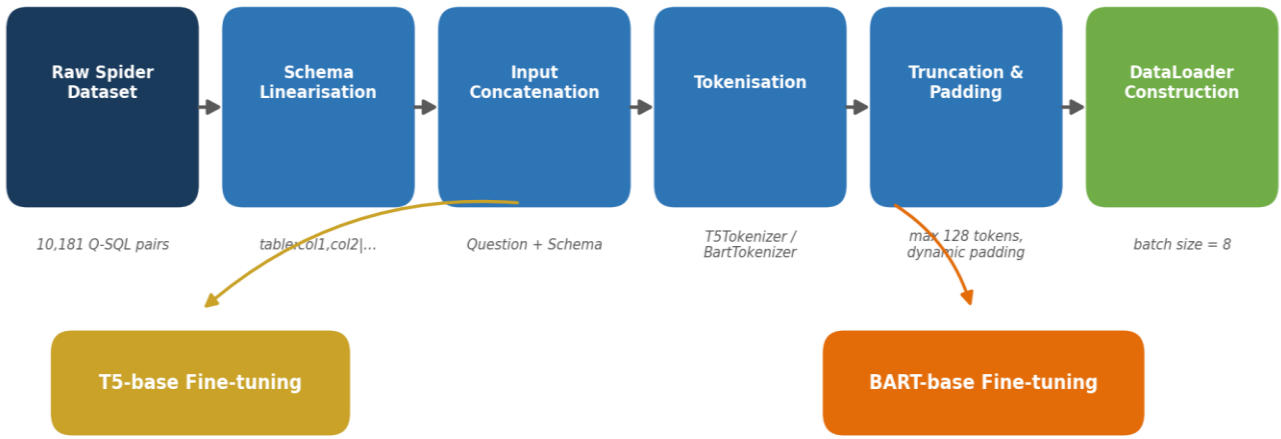


Figure 2. Data preprocessing and model training pipeline

Table 1. Comparative architectural features of T5-base and BART-base

Feature	T5-Base	BART-Base
Pretraining Objective	Masked span prediction (text-to-text)	Denoising autoencoder (reconstruction)
Schema Awareness	Implicit via text-to-text format	Implicit via bidirectional encoder
Encoder Type	Bidirectional self-attention	Bidirectional self-attention
Decoder Strategy	Structured sequence generation	Sequence generation with denoising
Key Strength	Versatility across NLP tasks	Robustness to input variability
Key Limitation	Sensitive to hyperparameter tuning	Slower convergence than T5

3.2.1 T5

T5 [2] handles every NLP task through the same text-to-text interface: the encoder reads a task-prefixed string, and the decoder outputs the answer as a token sequence. For text-to-SQL, the prefix "translate English to SQL: [question] context: [schema]" was used. The bidirectional encoder attends over the entire input, including the linearized schema, building contextual representations before the decoder begins generating SQL tokens one at a time. Because there is no task-specific output layer, T5 applies no SQL grammar constraint during decoding; it can generate any token sequence and relies entirely on the training signal to learn valid SQL structure.

3.2.2 BART

BART [1] was pretrained as a denoising autoencoder: input text is deliberately corrupted through operations including token deletion, random masking, text infilling, and sentence reordering, and the model learns to reconstruct the original sequence. This reconstruction objective is thought to make the decoder more robust when the input is noisy or semantically underspecified. For fine-tuning, the natural language question and the linearized schema were concatenated into a single input string and passed to the bidirectional BART encoder, and the decoder was trained to output the corresponding SQL. No other modifications were made to the model architecture.

3.2.3 Schema representation

Each database schema was converted to a flat string by listing table names followed by their columns, separated by pipe characters: “table1 : col1 , col2 | table2 : col3 , col4”. This string was prepended to the natural language question to form a single encoder input. No foreign-key annotations, special separator tokens, or explicit question-to-column alignment were added. This simple schema representation was chosen deliberately, because the experiment was designed to measure how well vanilla fine-tuning performs without schema-linking augmentation.

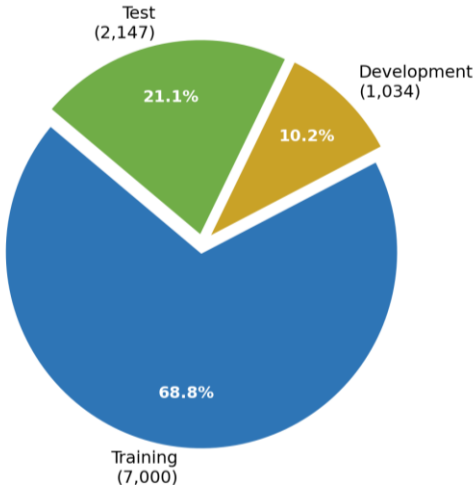


Figure 3. Spider dataset split proportions (Total: 10,181 examples)

3.3 Data preparation

Spider was obtained through the Hugging Face Datasets library. Its fixed splits of 7,000 training, 1,034 development, and 2,147 test examples (Figure 3) were preserved so that the reported results are directly comparable with published work. To monitor training progress, the 7,000 training examples were further split 90/10 (6,300 / 700, fixed random seed) to give a held-out training-time validation set (Figure 4). The original 1,034-example development set was kept entirely separate and used only for the final metric calculations reported in Section 4.

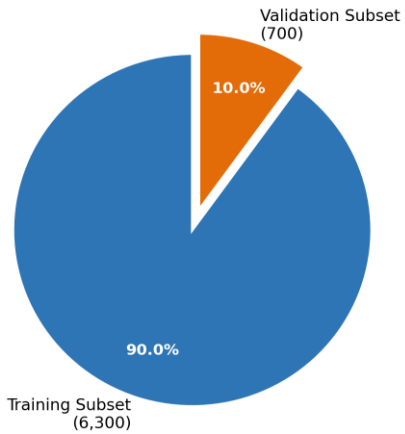


Figure 4. Adapted 90:10 training-validation split of the Spider training set

Table 2. Data preparation and preprocessing configuration

Stage	Detail	Value / Tool
Dataset source	Hugging Face Datasets library	Spider v1.0
Training subset	90% of Spider training split	6,300 examples
Validation subset	10% of Spider training split (fixed seed)	700 examples
Evaluation set	Spider development set (held-out)	1,034 examples
Tokenizer (T5)	T5Tokenizer (sentencepiece)	T5-base vocabulary
Tokenizer (BART)	BartTokenizer (BPE)	BART-base vocabulary
Max sequence length	Truncated to 128 tokens	128
Padding strategy	Dynamic batch-level padding	DataCollatorForSeq2Seq

Input strings were formed by placing the linearized schema before the question, separated by whitespace. Sequences longer than 128 tokens were truncated from the right to fit within GPU memory; this threshold was the maximum feasible on the available hardware and is acknowledged as a source of information loss for complex schemas. Dynamic padding via DataCollatorForSeq2Seq was used so that batch tensors were padded only to the length of the longest example in each batch, reducing wasted computation. The complete preprocessing pipeline is illustrated in Figure 2, and the corresponding configuration is listed in Table 2.

3.4 Evaluation metrics

Exact Match (EM): after normalizing both generated and gold queries (standardizing whitespace and keyword case), EM measures the fraction that are character-for-character identical. The metric is unforgiving, since a correct query with columns listed in a different order fails, but it is the standard Spider leaderboard metric and is therefore essential for comparison with published work.

$$EM = (\text{Number of Exact Matches} / \text{Total Queries}) \times 100$$

BLEU Score: Originally designed for machine translation [19], BLEU counts how often n-gram sequences in the generated output also appear in the reference. For SQL, BLEU is a coarse proxy: a query can share most keywords with the gold query while still being structurally incorrect. BLEU is included to maintain comparability with prior studies that report it, but it is treated as a secondary measure.

$$BLEU = BP \cdot \exp(\Sigma^{n} \omega^{n} \log p^{n})$$

LFA: LFA normalizes both predicted and reference queries by fixing keyword case, collapsing whitespace, and sorting commutative clause elements, then compares the resulting strings. A query that is semantically identical to the gold query but formatted differently passes LFA where EM would reject it. Execution accuracy, which runs both queries against the actual database and compares the returned rows, would be the most meaningful test of correctness, but it was not feasible with the available infrastructure; this absence is acknowledged as a limitation.

3.5 Training configuration

Fine-tuning was carried out using the Hugging Face Transformers library [20] and PyTorch [21] on Google Colab Pro (NVIDIA Tesla T4 GPU). AdamW was used as the optimizer with a weight decay of 0.01, a learning rate of 5×10^{-5} selected from a small grid search over $\{1 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}\}$, and a batch size of 8. Training ran for 10 epochs with a linear warm-up over the first 10% of steps followed by linear decay. A checkpoint was saved after every epoch, and the checkpoint with the lowest held-out validation loss was used for all reported evaluations. The complete training and implementation configuration is summarized in Table 3.

Table 3. Training hyperparameters and implementation configuration

Parameter	Value	Notes
Models	BART-base, T5-base	Hugging Face hub
Optimizer	AdamW	Weight decay 0.01
Learning rate	5×10^{-5}	Grid search over $\{1 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}\}$
Batch size	8	GPU memory constraint
Epochs	10	Best checkpoint selected on validation loss
Max sequence length	128 tokens	Input and output truncation
Decoding strategy	Greedy decoding	No beam search
LR schedule	Linear warmup + linear decay	Warmup: 10% of steps
Hardware	NVIDIA Tesla T4	Google Colab Pro
Framework	Transformers 4.×, PyTorch 2.×	Python 3.9

4. RESULTS AND DISCUSSION

4.1 Training dynamics

Figures 5 and 6 show the training and validation loss over 10 epochs for BART and T5, respectively. Training loss decreased steadily for both models with no sign of instability. BART reached a lower training loss than T5 from approximately epoch 5 onward and maintained this margin until the end of training. For both models, validation loss plateaued at approximately epoch 7, suggesting that further training at the same learning rate would not have improved generalization. BART's faster descent likely reflects its reconstruction-based pretraining: learning to regenerate clean text from corrupted input may initialize the decoder weights in a state better suited to sequence generation than the span-masking objective used by T5.

4.2 Performance metrics

Table 4 summarizes results on the Spider development set; Figure 7 plots the same data for easier comparison.

BART outperformed T5 on every metric, approximately doubling the EM and LFA scores. The most likely explanation is BART's pretraining task: reconstructing heavily corrupted text sequences may predispose the decoder toward generating well-formed structured output. The identical EM and LFA scores of T5 (0.0829) indicate that its errors were not formatting discrepancies that normalization could resolve; the generated queries were structurally incorrect rather than superficially variant. For BART, the small difference between EM (0.1529) and LFA (0.1543) shows that a minor subset of its failures were surface-level formatting issues recovered by LFA normalization, pointing to occasional tokenization artefacts rather than structural errors.

4.3 Comparison with schema-aware baselines

Table 5 places BART and T5 in context by listing figures from published baselines. A direct comparison is not strictly valid, because RAT-SQL and RYANSQL report execution accuracy on the Spider test set whereas this study reports EM and LFA on the development set; nevertheless, the figures are useful for gauging the magnitude of the performance gap.

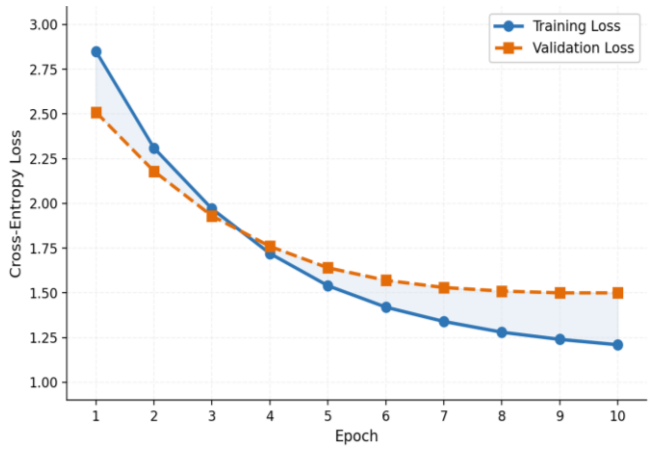


Figure 5. Training and validation loss for BART-base across 10 epochs

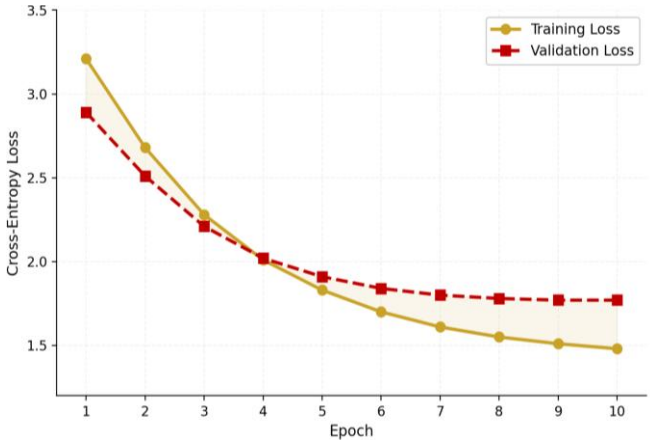


Figure 6. Training and validation loss for T5-base across 10 epochs

Table 4. Performance of T5-base and BART-base on the Spider development set

Metric	T5-Base	BART-Base	Difference (Δ)
Exact Match (EM)	0.0829 (8.29%)	0.1529 (15.29%)	+7.00 pp
BLEU Score	0.2503 (25.03%)	0.3606 (36.06%)	+11.03 pp
Logical Form Accuracy	0.0829 (8.29%)	0.1543 (15.43%)	+7.14 pp

Note: pp = percentage points.

Table 5. Contextual comparison of T5 and BART with published baselines on Spider (— = unavailable)

System	Exact Match (EM)	BLEU	Logical Form Accuracy (LFA)	Schema-Linked	Execution Accuracy (Development Set)	Studies
Seq2SQL	0.594	—	0.483	No	—	[4]
RAT-SQL	0.572	—	—	Yes	69.7%	[5]
RYANSQL	0.582	—	—	Yes	60.6%	[6]
BART-base (this work)	0.1529	0.3606	0.1543	No	Not evaluated	—
T5-base (this work)	0.0829	0.2503	0.0829	No	Not evaluated	—

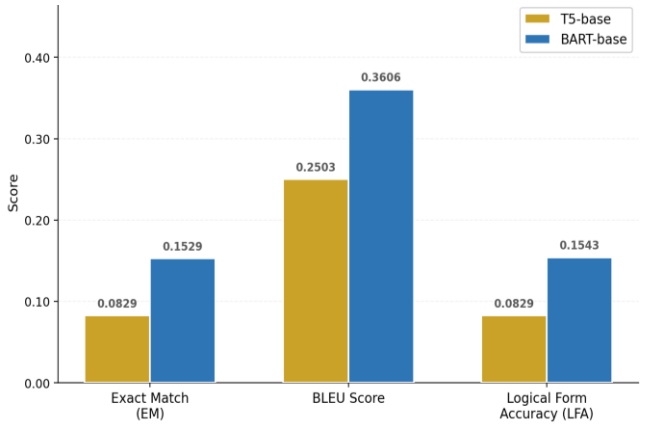


Figure 7. Performance metrics comparison: T5-base vs. BART-base on Spider development set

The gap is substantial: BART's EM of 15.29% lies approximately 42 percentage points below the EM of RAT-SQL and RYANSQL. The most direct explanation is the absence of schema linking. When a question contains a phrase such as "most recent order", a schema-aware model identifies which column in the relevant table carries date information; without that lookup, the model must infer it from the linearized string, and in cross-domain settings, where table and column names vary widely across databases, this inference is frequently wrong.

4.4 Error analysis

To move beyond aggregate metrics, 50 development-set examples on which both models failed were manually inspected. Three recurring error patterns emerged, as shown in Figure 8.

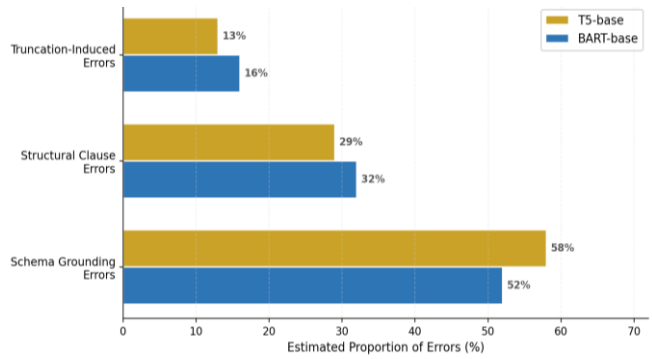


Figure 8. Estimated error category distribution for BART-base and T5-base

Schema grounding errors (BART ~52%, T5 ~58% of failures): the predicted query named a column or table that was plausible given the question but did not exist or was not the correct choice in the specific database. These errors clustered in databases where several tables share structurally similar column names, which is precisely the situation in which explicit question-to-schema alignment would help most. This error type is the direct consequence of absent schema linking.

Structural errors in complex clauses (BART ~32%, T5 ~29% of failures): nested subqueries, HAVING clauses with aggregation, and multi-table JOIN conditions regularly produced malformed SQL from both models. T5 exhibited this pattern more often, which is consistent with its lower EM and LFA scores and suggests that its generation strategy handles hierarchical SQL structure less reliably than the reconstruction-oriented decoder of BART.

Truncation-induced errors (BART ~16%, T5 ~13% of failures): when the combined question-and-schema string exceeded 128 tokens, the tail of the schema was removed. If the relevant table or column appeared near the end of the linearized schema, the model had no access to it and produced incomplete or incorrect table references. Increasing the maximum sequence length, subject to available GPU memory, would directly reduce this error category.

4.5 Limitations

Four limitations should be stated explicitly. First, execution accuracy was not measured; some queries that fail EM may still return the correct answer, and the converse is also possible, so EM and LFA give an incomplete picture of practical usefulness. Second, the 128-token cap truncates a measurable fraction of complex inputs, creating a systematic disadvantage for longer queries. Third, the comparison in Table 5 is not like-for-like: evaluation sets, metrics, and preprocessing differ between this work and the published baselines. Fourth, BLEU was designed for free-form text and can overstate the quality of structurally incorrect SQL that contains the correct keywords; readers should weight EM and LFA more heavily than BLEU when interpreting the results.

5. CONCLUSION

This paper set out to measure how well BART-base and T5-base perform on the Spider text-to-SQL benchmark when fine-tuned without any schema-aware architectural additions. Under matched experimental conditions, BART outperformed T5 on all three metrics, approximately doubling the EM and LFA scores, an advantage attributed to its reconstruction-based pretraining. Both models, however, remained far below schema-aware systems such as RAT-SQL and RYANSQL. Manual inspection of 50 failed examples showed that the majority of errors stemmed from the model selecting the wrong schema element, a problem that schema linking is specifically designed to solve.

Three directions for future work follow from these findings.

The most impactful is the addition of schema linking, either by constructing a schema graph with relation-aware attention as in RAT-SQL, or by applying an alignment step that marks which question words correspond to which schema elements before encoding. A second step is extending the sequence length beyond 128 tokens, which would remove the category of errors that arise solely from truncated schema information. Third, training with an execution-accuracy objective, rather than relying solely on cross-entropy over token sequences, would push the model toward outputs that are not merely superficially similar to the gold query but functionally correct when executed against real data. Taken together, these experiments show that general-purpose transformers offer a practical and accessible starting point for NLIDB development, while making clear that schema-aware engineering is still needed to close the gap with purpose-built text-to-SQL systems.

REFERENCES

[1] Lewis, M., Liu, Y., Goyal, N., et al. (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the ACL, pp. 7871-7880. https://doi.org/10.18653/v1/2020.acl-main.703

[2] Raffel, C., Shazeer, N., Roberts, A., et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 21(1): 5485-5551.

[3] Yu, T., Zhang, R., Yang, K., et al. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Proceedings of EMNLP 2018, pp. 3911-3921. https://doi.org/10.18653/v1/D18-1425

[4] Zhong, V., Xiong, C., Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103. https://doi.org/10.48550/arXiv.1709.00103

[5] Wang, B., Shin, R., Liu, X., Polozov, O., Richardson, M. (2020). RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In Proceedings of the 58th Annual Meeting of the ACL, pp. 7567-7578. https://doi.org/10.18653/v1/2020.acl-main.677

[6] Choi, D., Shin, M.C., Kim, E., Shin, D.R. (2021). RYANSQL: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. Computational Linguistics, 47(2): 309-332. https://doi.org/10.1162/coli_a_00403

[7] Warren, D.H.D., Pereira, F.C.N. (1982). An efficient easily adaptable system for interpreting natural language queries. American Journal of Computational Linguistics, 8(3-4): 110-122.

[8] Androutsopoulos, I., Ritchie, G.D., Thanisch, P. (1995). Natural language interfaces to databases—An introduction. Natural Language Engineering, 1(1): 29-81. https://doi.org/10.1017/S135132490000005X

[9] Popescu, A.M., Etzioni, O., Kautz, H. (2003). Towards a theory of natural language interfaces to databases. In Proceedings of the 8th ACM IUI, pp. 149-157. https://doi.org/10.1145/604045.604070

[10] Zelle, J.M., Mooney, R.J. (1996). Learning to parse database queries using inductive logic programming. In Proceedings of AAAI-96, pp. 1050-1055.

[11] Zettlemoyer, L.S., Collins, M. (2012). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In Proceedings of UAI 2005, pp. 658-666.

[12] Xu, X., Liu, C., Song, D. (2017). SQLNet: Generating structured queries from natural language without reinforcement learning. arXiv preprint arXiv:1711.04436. https://doi.org/10.48550/arXiv.1711.04436

[13] Yavuz, S., Gür, I., Su, Y., Yan, X. (2018). What it takes to achieve 100% condition accuracy on WikiSQL. In Proceedings of EMNLP 2018, pp. 1702-1711. https://doi.org/10.18653/v1/D18-1197

[14] Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, pp. 5998-6008.

[15] Devlin, J., Chang, M.W., Lee, K., Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171-4186. https://doi.org/10.18653/v1/N19-1423

[16] Yin, P., Neubig, G., Yih, W., Riedel, S. (2020). TaBERT: Pretraining for joint understanding of textual and tabular data. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 8413-8426. https://doi.org/10.18653/v1/2020.acl-main.745

[17] Herzig, J., Nowak, P.K., Müller, T., Piccinno, F., Eisenschlos, J.M. (2020). TaPas: Weakly supervised table parsing via pre-training. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 4320-4333. https://doi.org/10.18653/v1/2020.acl-main.398

[18] Deng, X., Awadallah, A.H., Meek, C., Polozov, O., Sun, H., Richardson, M. (2021). Structure-grounded pretraining for text-to-SQL. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1337-1350. https://doi.org/10.18653/v1/2021.naacl-main.105

[19] Papineni, K., Roukos, S., Ward, T., Zhu, W.J. (2002). BLEU: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, pp. 311-318. https://doi.org/10.3115/1073083.1073135

[20] Wolf, T., Debut, L., Sanh, V., et al. (2020). Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38-45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

[21] Paszke, A., Gross, S., Massa, F., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, Canada, pp. 8026-8037.