OptiSQL: Executable SQL Generation from Optical Tokens

Anonymous ACL submission

Abstract

Executable SQL generation is typically studied in text-to-SQL settings, where tables are provided as fully linearized textual schemas and contents. While effective, this formulation assumes access to structured text and incurs substantial token overhead, which is misaligned with many real-world scenarios where tables appear as visual artifacts in documents or webpages. We investigate whether compact optical representations can serve as an efficient interface for executable semantic parsing. We present OptiSQL, a vision-driven framework that generates executable SQL directly from table images and natural language questions using compact optical tokens. OptiSQL leverages an OCR-oriented visual encoder to compress table structure and content into a small set of optical tokens and fine-tunes a pretrained decoder for SQL generation while freezing the encoder to isolate representation sufficiency. Experiments on a visualized version of Spider 2.0-Snow show that OptiSQL retains strong execution accuracy while reducing table input tokens by an order of magnitude. Robustness analyses further demonstrate that optical tokens preserve essential structural information under visual perturbations.

1 Introduction

Executable SQL generation has been predominantly studied under the text-to-SQL paradigm, where database schemas and table contents are provided as structured textual inputs (Yu et al., 2018; Lei et al., 2025). While effective in controlled settings, this formulation relies on a strong assumption: that clean, machine-readable table text is readily available. In many real-world scenarios, however, tabular data is encountered primarily in visual form, such as scanned documents, PDFs, or web pages, where structured textual schemas are not directly accessible. Converting such inputs into text typically requires multi-stage preprocessing and manual normalization, making text-based pipelines brittle in practice (Kim et al., 2022; Lee et al., 2023).

This mismatch motivates treating tables as visual objects rather than inherently textual ones. Table images naturally encode row-column structure, alignment, and grouping, which are expensive to represent through token-by-token linearization. Recent OCR-oriented vision-language models have shown that document images can be compressed into compact sequences of discrete visual representations while preserving fine-grained semantic and structural information (Wei et al., 2025). In particular, DeepSeek-OCR demonstrates that high-resolution document images can be mapped into a small number of optical tokens suitable for downstream language modeling (Wei et al., 2025). Importantly, these optical tokens capture more than raw text, retaining layout and relational cues critical for table understanding (Wei et al., 2025).

In this work, we leverage optical tokens as an explicit interface for executable semantic parsing. Rather than reconstructing text or schemas, we ask whether a compact visual abstraction of a table is sufficient to support SQL generation under strict token budgets. We introduce **OptiSQL**, a vision-driven framework that generates executable SQL queries directly from table images and natural language questions. OptiSQL uses a pretrained OCR-oriented visual encoder to convert a table image into a fixed-length sequence of optical tokens, and fine-tunes a pretrained autoregressive decoder to generate SQL. The visual encoder is frozen by default, isolating the representational sufficiency of optical tokens and avoiding confounding effects from encoder adaptation.

We evaluate OptiSQL on a visualized version of Spider 2.0 (Lei et al., 2025), using execution accuracy as the primary metric. Through controlled analyses of token budgets and visual perturbations, we show that OptiSQL substantially reduces input length while maintaining strong executable accu-

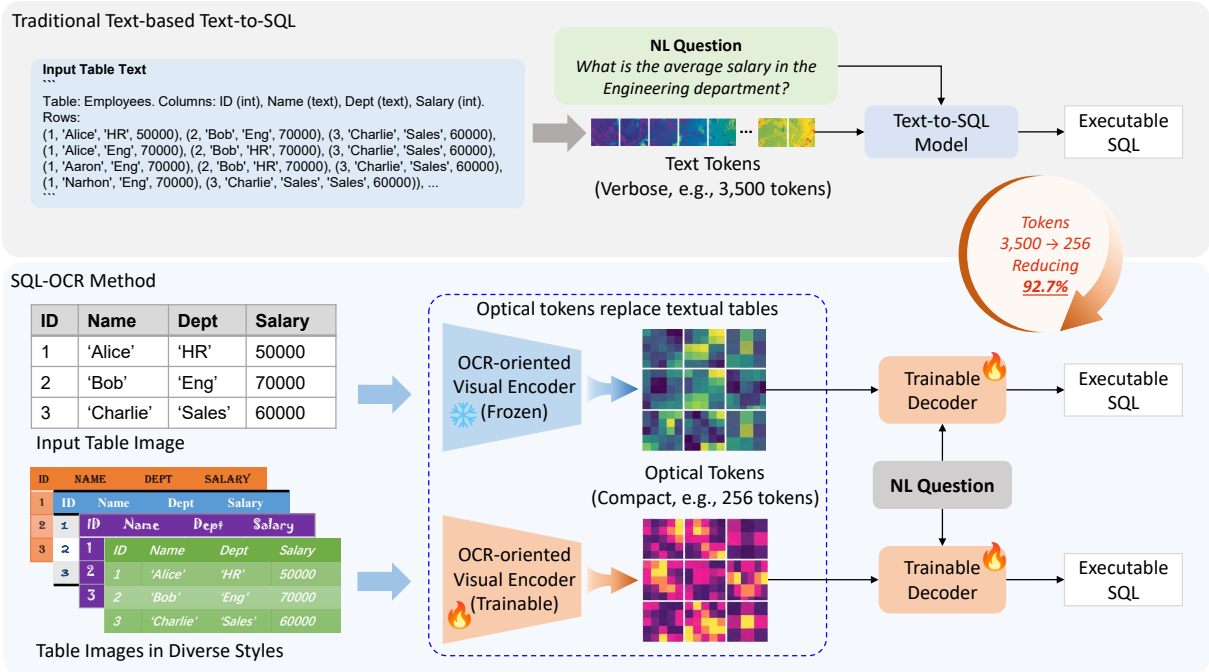

Figure 1: OptiSQL generates executable SQL directly from table images using compact optical tokens. A frozen OCR-oriented visual encoder converts a table image into a fixed-size sequence of optical tokens, which are combined with a natural language question and processed by a trainable decoder. By operating under an explicit token budget, OptiSQL enables an efficient and robust alternative to text-based table encodings.

racy, and that performance degrades predictably as visual information is removed or corrupted. These results indicate that optical tokens provide a practical and analyzable interface for efficiency-oriented executable semantic parsing, especially under long-context cost constraints (Liu et al., 2024). In summary, this work makes the following contributions:

- We propose OptiSQL, a vision-driven framework that replaces explicit textual table encodings with compact optical tokens for executable SQL generation.
- We introduce a controlled evaluation setting that isolates the sufficiency of optical tokens by freezing the visual encoder and adapting only the decoder.
- We demonstrate favorable trade-offs between input efficiency, robustness, and execution accuracy on the semantic parsing task.

2 Related Work

2.1 Text-to-SQL Semantic Parsing

Text-to-SQL semantic parsing generates executable SQL queries from natural language questions given textual representations of database schemas and contents (Yu et al., 2018; Shi et al., 2025; Zhou et al., 2024; Lei et al., 2025). Prior work has improved accuracy through schema linking, constrained decoding, and execution-guided learning (Guo et al., 2019; Wang et al., 2025; Liu et al., 2024; Cai et al., 2025). Recent agent-based systems further leverage iterative refinement and execution feedback, with ReFoRCE representing a strong state-of-the-art approach (Deng et al., 2025).

These methods assume clean and complete textual access to schemas and table contents, removing the representational bottleneck that arises in visually grounded settings. We therefore treat text-to-SQL systems as **upper-bound baselines** rather than direct competitors, as they operate under a fundamentally different input assumption.

2.2 OCR-Centric Pipelines

Querying tables embedded in document images is often addressed via OCR-centric pipelines that reconstruct text and table structure before applying text-based parsers (Li et al., 2020; doc; Sheng and Xu, 2024). While modular, such pipelines are sensitive to recognition errors and typically discard spatial and layout information during text linearization, leading to error propagation and high input costs (Guan et al., 2025). We include an OCR pipeline as a reference baseline to highlight the trade-offs between explicit text reconstruction and efficient executable reasoning.

2.3 Optical Tokenization for Document Understanding

Vision-language models such as Pix2Struct and Donut map document images directly to structured outputs and have been applied to table-related tasks (Lee et al., 2023; Kim et al., 2022). More recently, OCR-oriented visual encoders have been proposed to compress document images into compact sequences of optical tokens that preserve semantic and structural information under strict length budgets (Wei et al., 2025). DeepSeek-OCR exemplifies this direction, demonstrating that high-resolution document images can be represented by a small number of informative visual tokens.

Our work builds on optical tokenization as a representation interface but targets a different question: whether compact optical tokens are sufficient for **executable semantic parsing**. Unlike prior work, we explicitly freeze the visual encoder by default and adapt only the decoder, enabling a controlled study of representation sufficiency and efficiency-accuracy trade-offs in SQL generation.

3 Task Definition

We study **executable SQL generation from table images**. Given a table image I and a natural language question q, the goal is to generate an executable SQL query s whose execution on the underlying database returns the correct answer.

3.1 Problem Formulation

Unlike standard text-to-SQL settings, the model is *not* given ground-truth textual schemas, column metadata, or table contents as structured text. All table-related information must be inferred from the visual input. We assume the table image captures the schema cues and cell contents needed to answer the question, and the model does not access database metadata or schema annotations at inference time.

3.2 Efficiency-Oriented Objective

Our central objective is **input efficiency**. Let $|V|$ be the number of optical tokens representing I produced by an OCR-oriented visual encoder, and let $|T|$ be the number of tokens required to represent the same table via textual linearization. We study whether accurate SQL generation remains feasible when $|V| \ll |T|$, and how executable performance varies as we explicitly control the token budget.

3.3 Evaluation

We primarily evaluate with execution accuracy (EXAcc), and additionally report canonical exact match (EX-Can) and efficiency metrics such as optical tokens per query and token savings ratio (TSR); metric definitions are provided in Section 5.

4 Model Architecture

OptiSQL is a vision-driven semantic parsing framework that generates executable SQL queries from table images under explicit input efficiency constraints. As shown in Figure 1, the model consists of three components: (i) a pretrained OCR-oriented visual encoder that converts a table image into a compact sequence of optical tokens, (ii) a unified token interface that combines optical tokens with the natural language question, and (iii) a pretrained decoder adapted for SQL generation.

A key design principle of OptiSQL is to treat optical tokenization as a fixed representation interface. By default, the visual encoder is frozen and only the decoder is fine-tuned for SQL generation. This setting, denoted as FROZENENC, isolates the representational capacity of compact optical tokens and enables a controlled study of whether such representations are sufficient for executable semantic parsing. We additionally consider a FULLFT variant as an ablation to study accuracy and robustness trade-offs introduced by encoder adaptation. Unless otherwise specified, OptiSQL refers to the FROZENENC setting throughout the paper.

4.1 Optical Tokenization via a Visual Encoder

Given a table image I, OptiSQL employs a pretrained OCR-oriented visual encoder E to produce a sequence of optical tokens:

$$V = E(I) = (v_1, v_2, \ldots, v_n), \qquad (1)$$

where each token encodes localized visual, textual, and structural information extracted from the image. The encoder compresses the two-dimensional table layout into a one-dimensional token sequence while preserving regularities such as row-column alignment, cell grouping information, and header-content associations.

Importantly, OptiSQL does not perform explicit text reconstruction or OCR decoding during inference. The optical tokens are treated as a latent interface that implicitly represents table content and structure. By freezing the visual encoder, we

prevent task-specific adaptation and directly evaluate whether the pretrained optical tokens retain sufficient information for executable reasoning.

The number of optical tokens n is controlled by the encoder configuration, enabling systematic exploration of different token budgets. This explicit control is central to our analysis of efficiency-accuracy trade-offs.

4.2 Unified Input Representation

The decoder input is constructed by concatenating the optical tokens with the tokenized natural language question:

$$X = [V; Q], \tag{2}$$

where Q denotes the question tokens. Optical and textual tokens are projected into a shared embedding space before being processed by the decoder.

No textual schema annotations, column names, or table contents are provided. All table-related information must be inferred from the optical tokens. This design enforces a strict representational bottleneck and distinguishes OptiSQL from text-to-SQL systems and OCR-based pipelines.

4.3 Autoregressive SQL Decoder

OptiSQL uses a pretrained decoder D to generate SQL queries conditioned on the combined input sequence X. At each decoding step t, the model predicts the next SQL token s_t according to:

$$p(s_t \mid s_{<t}, X). \tag{3}$$

The decoder is responsible for producing syntactically valid and executable SQL queries. SQL generation terminates when an end-of-sequence token is produced. While syntax constraints may optionally be applied, the core model relies on learned representations rather than hand-crafted rules.

Under the default FROZENENC setting, only the decoder parameters are updated during training. This design focuses learning on mapping compact optical representations to executable programs, without altering the visual tokenization itself.

4.4 Training Variants

We consider two training variants to disentangle representation sufficiency from encoder adaptation effects.

FROZENENC. In the default setting, the visual encoder is frozen and only the decoder is fine-tuned. This variant serves as the primary configuration for evaluating input efficiency, token budget scaling, visual grounding, and robustness.

FULLFT. As an ablation, we fine-tune both the visual encoder and the decoder. In this setting, we additionally apply robustness-oriented training-time table rendering augmentations. FULLFT is used solely to study the trade-offs between clean accuracy and generalization, and is not the default configuration.

5 Experimental Setup

This section describes the datasets, baselines, evaluation metrics, and experimental protocols. All experiments are designed to address a single question: are compact optical tokens sufficient to support executable SQL generation under strict input efficiency constraints? Unless otherwise specified, all OptiSQL results correspond to the FROZENENC setting, which freezes the encoder.

5.1 Dataset and Task Construction

We evaluate OptiSQL on a visualized version of Spider 2.0-Snow (Lei et al., 2025), a realistic enterprise-oriented text-to-SQL benchmark with complex SQL queries and diverse schemas. For each example, the original table is rendered into a table image, while the natural language question and ground-truth SQL query remain unchanged. No textual schema or table content is provided to the model.

5.2 Baselines

We compare our proposed OptiSQL against representative baselines that reflect different assumptions about table access.

Text-based Text-to-SQL. We include ReFoRCE (Deng et al., 2025), a state-of-the-art text-to-SQL system operating on ground-truth textual schemas and contents. This setting removes the representational bottleneck entirely and therefore serves as an **upper-bound baseline**, albeit with very high token cost.

OCR-based Text-to-SQL Pipeline. We construct a two-stage pipeline that first applies OCR to recover table text and structure, followed by a text-based SQL parser. This baseline reflects a common document understanding workflow and highlights

the effects of OCR errors, schema linearization, and long textual inputs.

Vision-Language Models. We evaluate general-purpose vision-language models (VLMs), including Pix2Struct (Lee et al., 2023) and Donut (Kim et al., 2022), which directly generate SQL from images without explicit OCR pipelines.

5.3 Evaluation Metrics

We evaluate models along four dimensions: executable correctness, syntactic fidelity, input efficiency, and robustness.

Execution Accuracy (EXAcc). Execution accuracy measures whether the generated SQL query produces the correct result when executed against the database:

$$\text{EXAcc} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\text{EX}(s_i, \mathcal{D}_i) = \text{EX}(s_i^*, \mathcal{D}_i)]\,. \tag{4}$$

Execution-based evaluation is preferred as semantically equivalent SQL queries may differ in surface form.

Canonical Exact Match (EX-Can). We additionally report canonical exact match, which compares SQL strings after conservative normalization:

$$\text{EX-Can} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\text{Can}(s_i) = \text{Can}(s_i^*)]\,. \tag{5}$$

Canonicalization removes surface-form differences while avoiding semantic changes. Details are provided in Appendix 1, with an illustrative example in Figure 2.

Robustness Metrics. Robustness is measured as the drop in execution accuracy under visual perturbation $\mathcal{P}$:

$$\Delta_{\mathcal{P}} = \text{EXAcc}_{\text{clean}} - \text{EXAcc}_{\mathcal{P}}. \tag{6}$$

5.4 Experimental Conditions and Ablations

We design a set of experiments to isolate representation sufficiency, visual grounding, and efficiency trade-offs.

Token budget sweep. We vary the optical token budget (e.g., 64, 100, 256, 400) while keeping all other components fixed, enabling systematic analysis of accuracy-efficiency trade-offs.

Example of SQL Canonicalization for EX-Can

Raw prediction s

```
SELECT * FROM users
WHERE age > 18 AND city='LA'
ORDER BY created_at DESC;
```

Canonicalized $\text{Can}(s)$

```
select * from users where city='LA' and
      age>18 order by created_at desc;
```

Figure 2: An example of SQL canonicalization used for EX-Can. The procedure normalizes keyword case, whitespace, punctuation spacing, and reorders flat AND conditions within the same logical level.

FROZENENC vs. FULLFT. We compare the default FROZENENC setting against a FULLFT (where the encoder is also updated) ablation in which both encoder and decoder are fine-tuned. FULLFT is used only to study accuracy and robustness trade-offs and is not the default configuration.

No-image and wrong-image diagnostics. To verify visual grounding, we evaluate NOIMAGE (optical tokens removed) and WRONGTABLE (table images permuted across examples), testing whether models collapse into language-only or template-based generation.

5.5 Robustness Augmentation and Evaluation

We study robustness under both training-time augmentation and inference-time perturbation. The two are intentionally separated to distinguish robustness gained through encoder adaptation from robustness intrinsic to the optical token interface.

Training-time augmentation (FULLFT). When fine-tuning the visual encoder (FULLFT), we apply rendering-based augmentations that preserve table semantics while altering visual appearance. These augmentations are designed to encourage invariance to superficial layout and style variations, and are applied on-the-fly during training.

Specifically, we use: (i) *Style variation*, including changes in font family and size, spacing, border thickness, zebra striping, cell padding, and column width; and (ii) *Table transposition*, which swaps rows and columns while relocating headers accordingly. Examples of such augmentations are shown in Figure 3. The supervision signal (ground-truth

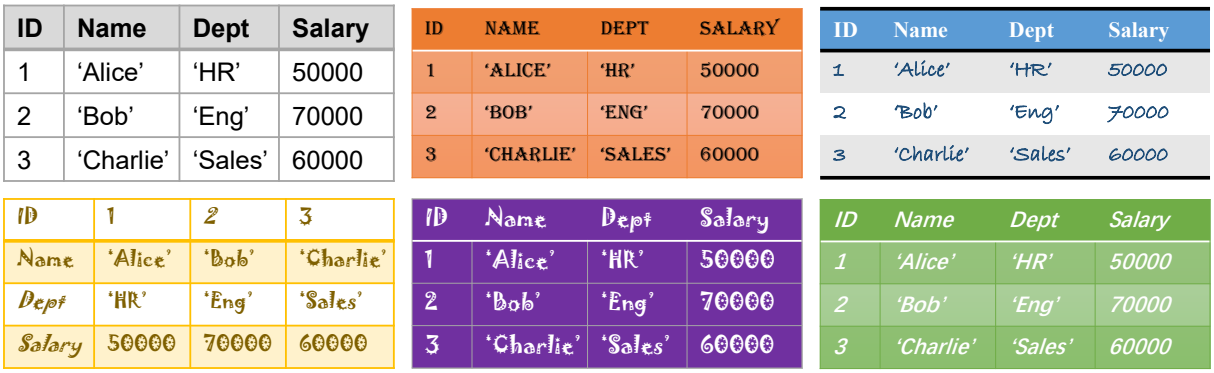

Figure 3: Different visual renderings of the same table content. Variations include changes in font style, color, layout, borders, and spacing, while preserving identical schema and cell values. These styles are used to study robustness to superficial visual changes.

SQL) remains unchanged. Unless otherwise specified, these augmentations are enabled only in the FULLFT ablation and are not used in the default FROZENENC setting.

Inference-time robustness evaluation. To evaluate robustness independently of retraining, we apply controlled test-time perturbations to the rendered table images and measure the resulting drop in execution accuracy:

$$\Delta_{\mathcal{P}} = \text{EXAcc}_{\text{clean}} - \text{EXAcc}_{\mathcal{P}}.$$

All perturbations preserve table content except where explicitly noted.

StyleShift. *StyleShift* samples a different rendering template at test time while keeping the underlying table content fixed. This perturbation assesses invariance to superficial visual factors such as font choice, spacing, and grid styling, without altering textual or relational information.

HeaderMask. *HeaderMask* partially occludes column headers to disrupt explicit schema cues while preserving table body cells. We mask only the header row region using opaque rectangular fills. The masking ratio is capped at at most one-third of the header width per table. Mask locations are sampled uniformly over contiguous header spans (or multiple short spans). Empirically, larger masking ratios lead to unstable behavior and can prevent convergence, and are therefore excluded.

Visual grounding diagnostics. In addition to robustness perturbations, we include two diagnostic settings to test visual grounding. NOIMAGE removes optical tokens (or replaces them with a learned null token) while keeping the question unchanged. WRONGTABLE permutes table images across examples, producing mismatched table-question pairs. These settings are not intended as realistic noise models, but as stress tests to verify that SQL generation does not collapse into language-only or template-based behavior.

Default parameterizations for all perturbations are summarized in Appendix A.4.

5.6 Implementation Details

OptiSQL uses a pretrained OCR-oriented visual encoder to produce compact optical tokens. Unless otherwise specified, the encoder is frozen and only the autoregressive decoder is fine-tuned using token-level cross-entropy loss over SQL targets. For FULLFT, both encoder and decoder are updated and robustness-oriented training-time augmentations are enabled. All models are evaluated under identical decoding and runtime settings.

6 Results and Analysis

We analyze OptiSQL with a focus on representation sufficiency rather than leaderboard performance. Specifically, we ask whether compact optical tokens can preserve sufficient structural and semantic information to support the generation of executable SQL under explicit efficiency constraints. Unless otherwise specified, OptiSQL refers to the default FROZENENC setting.

6.1 Main Results

Table 1 reports execution accuracy, canonical exact match, robustness, and table-related token cost across representative baselines. For all methods, we report only tokens used to encode table information, excluding question and output tokens.

Text-only systems achieve the highest execution

Model	Input Table	Tokens	EXAcc	EX-Can	EXAcc St.	EXAcc HM
ReFoRCE (Deng et al., 2025)	Text	3,500	**0.78**	**0.86**	**0.86**	**0.86**
DeepSeek-OCR (Wei et al., 2025)	OCR text	3,500	0.62	0.70	0.58	0.45
Pix2Struct (Lee et al., 2023)	Image	800	0.55	0.63	0.50	0.38
Donut (Kim et al., 2022)	Image	900	0.57	0.65	0.48	0.35
OptiSQL (FrozenEnc)	Image	**256**	0.66	0.76	0.72	0.60
OptiSQL (FullFT)	Image	256	0.68	0.77	0.65	0.55

Table 1: Table Tokens count only tokens used to represent table information, excluding question and output tokens. ReFoRCE is a text-only upper-bound baseline. EXAcc St. represents the style-shift setting, and EXAcc HM denotes the header-masking setting. Best results are in **bold**, second-best are underlined.

accuracy when ground-truth textual schemas and table contents are available. In particular, ReFoRCE benefits from lossless access to explicit schema structure and therefore serves as an **upper-bound reference** rather than a directly comparable system. This setting assumes strictly stronger input supervision than any image-based formulation.

In contrast, OCR-based pipelines exhibit a substantial performance drop. Despite reconstructing text explicitly, OCR errors and schema linearization introduce compounding noise and long input sequences. General-purpose VLMs further underperform on executable accuracy, suggesting that document-level generation objectives do not directly translate to structured program synthesis.

Under a strictly weaker and more realistic assumption, where table information is provided only as images and accessed through compact optical tokens, OptiSQL achieves the strongest execution accuracy among all non-text-only approaches. With only 256 optical tokens, OptiSQL approaches the performance of OCR pipelines while using an order of magnitude fewer table tokens. These results demonstrate that optical tokens produced by an OCR-oriented encoder preserve sufficient structural and semantic information to support executable SQL generation under severe compression.

Robustness and encoder adaptation. Both FrozenEnc and FullFT degrade under test-time perturbations, with header masking consistently more damaging than style variation. While FullFT slightly improves clean accuracy, it exhibits larger robustness drops, indicating overfitting to surface rendering patterns even with training-time augmentation. This trade-off motivates our choice of FrozenEnc as the default setting for studying representation sufficiency and efficiency.

Setting	EXAcc (%)	EX-Can (%)
Clean Image	66	76
NoImage	15 -83.33%	28 -63.16%
RandomImage	22 -66.67%	30 -60.53%
WrongTable	6 -90.91%	12 -84.21%

Table 2: EXAcc and EX-Can under visual grounding diagnostics. Removing, randomizing, or mismatching table images causes substantial performance degradation, highlighting the importance of correct visual grounding.

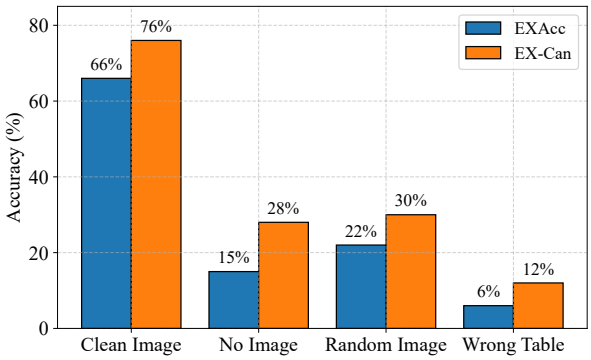

Figure 4: Visual grounding diagnostics for OptiSQL (FrozenEnc). EXAcc and EX-Can under clean and perturbed visual inputs. Bars show accuracy. Performance degrades sharply when visual grounding is disrupted.

6.2 Visual Grounding Diagnostics

To verify that OptiSQL relies on visual table information rather than language priors, we conduct strong diagnostic ablations that remove or corrupt the visual input. Table 2 reports execution accuracy for FrozenEnc under these conditions.

Removing optical tokens (NoImage) causes a dramatic collapse in execution accuracy, confirming that SQL generation does not reduce to a language-only or template-based process. Replacing the table image with random or mismatched images further degrades performance, demonstrating

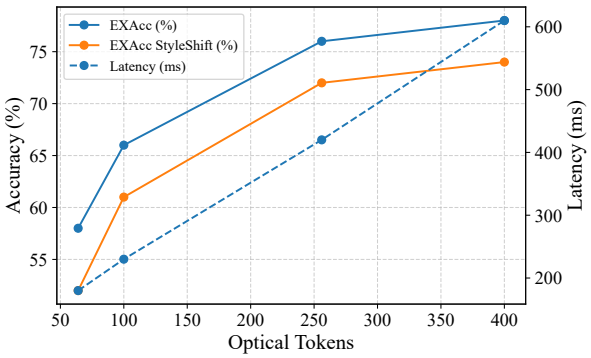

Figure 5: Execution accuracy and inference latency as a function of the optical token budget. Increasing the number of optical tokens improves execution accuracy with diminishing returns beyond 256 tokens, while latency grows approximately linearly, revealing a clear efficiency-accuracy trade-off.

that OptiSQL is sensitive to table-specific visual content rather than generic visual cues. Together, these results provide direct evidence that optical tokens act as a necessary conditioning signal for executable reasoning.

6.3 Efficiency-Accuracy Trade-offs

We next analyze how performance scales with the optical token budget. Figure 5 shows execution accuracy and latency across budgets.

Execution accuracy increases monotonically as the token budget grows, with diminishing returns beyond approximately 256 tokens. This identifies a clear efficiency sweet spot where OptiSQL achieves strong executable accuracy at moderate runtime cost. Notably, robustness trends remain stable across budgets, indicating that compression primarily limits representational capacity rather than altering sensitivity to visual perturbations.

Scope of Comparison. OptiSQL is not designed to compete directly with leaderboard systems on Spider 2.0, which assume full access to ground-truth textual schemas and contents. Such comparisons would conflate differences in task formulation with performance in modeling. Instead, we include a strong text-only system as an upper-bound reference to contextualize the efficiency-accuracy trade-off introduced by optical representations. Our objective is to study whether executable reasoning remains feasible when table information is accessed through a highly compressed visual channel.

6.4 Summary

Overall, these results show that compact optical tokens can serve as an effective and analyzable interface for executable semantic parsing. Compared to textual representations, they offer substantial compression at a controlled accuracy cost. Compared to OCR pipelines, they avoid explicit reconstruction and reduce error propagation. Compared to general-purpose vision-language models, OptiSQL benefits from OCR-oriented tokenization and task-specific decoder adaptation. Rather than replacing text-based settings when ground-truth schemas are available, optical tokens enable a principled exploration of efficiency-oriented semantic parsing under realistic input constraints.

7 Conclusion

We study executable SQL generation under a realistic setting where tables are available only as visual artifacts rather than structured text. Instead of reconstructing tables into long textual inputs, we investigate whether compact optical tokens can serve as an efficient interface for semantic parsing. Experiments on a visualized version of Spider 2.0-Snow show that optical tokens achieve substantial input compression while preserving strong execution accuracy. Compared to OCR-based pipelines and general-purpose vision-language models, OptiSQL offers a more favorable efficiency-accuracy trade-off under identical input assumptions. These results suggest that optical tokenization is a practical alternative to text-centric table representations when structured schemas are unavailable, and highlight representation choice as a key design for executable reasoning under long-context constraints.

Limitations

OptiSQL relies on a pretrained visual encoder, and its performance is bounded by the encoder's ability to represent textual and structural information in table images. Inputs with severe noise, handwriting, or distortions may not be well handled. Moreover, our evaluation focuses on single-table queries in a visualized Spider 2.0-Snow setting. Extending the approach to multi-table queries will require modeling inter-table relations and joins, which we leave to future work. Finally, we primarily adopt a frozen-encoder training protocol to study representation sufficiency. While fine-tuning the encoder can improve accuracy, it may reduce robustness, suggesting a trade-off that warrants investigation. We focus exclusively on executable SQL generation; the applicability of compact optical tokens to other structured generation tasks remains open.

References

Doc2Table — doc2table.com. https://www.doc2table.com/home. [Accessed 06-01-2026].

Qifeng Cai, Hao Liang, Chang Xu, Tao Xie, Wentao Zhang, and Bin Cui. 2025. Text2sql-flow: A robust sql-aware data augmentation framework for text-to-sql. *Preprint*, arXiv:2511.10192.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. ReFoRCE: A text-to-sql agent with self-refinement, consensus enforcement, and column exploration. *Preprint*, arXiv:2502.00675.

Shuhao Guan, Moule Lin, Cheng Xu, Xinyi Liu, Jinman Zhao, Jiexin Fan, Qi Xu, and Derek Greene. 2025. PreP-OCR: A complete pipeline for document image restoration and enhanced OCR accuracy. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15413–15425, Vienna, Austria. Association for Computational Linguistics.

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. *Preprint*, arXiv:1905.08205.

Geewook Kim, Teakgyu Hong, Moonbin Yim, JeongYeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. 2022. Ocr-free document understanding transformer. In *European Conference on Computer Vision (ECCV)*.

Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. 2023. Pix2struct: Screenshot parsing as pretraining for visual language understanding. *Preprint*, arXiv:2210.03347.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. *Preprint*, arXiv:2411.07763.

Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, Ming Zhou, and Zhoujun Li. 2020. TableBank: Table benchmark for image-based table detection and recognition. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1918–1925, Marseille, France. European Language Resources Association.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Lei Sheng and Shuai-Shuai Xu. 2024. Pdftable: A unified toolkit for deep learning-based table extraction. *Preprint*, arXiv:2409.05125.

Jie Shi, Bo Xu, Jiaqing Liang, Yanghua Xiao, Jia Chen, Chenhao Xie, Peng Wang, and Wei Wang. 2025. Gen-SQL: Efficient text-to-SQL by bridging natural language question and database schema with pseudo-schema. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3794–3807, Abu Dhabi, UAE. Association for Computational Linguistics.

Guifeng Wang, Yuanfeng Song, Meng Yang, Tao Zhu, Xiaoming Yin, and Xing Chen. 2025. Beyond query-level comparison: Fine-grained reinforcement learning for text-to-sql with automated interpretable critiques. *Preprint*, arXiv:2511.22258.

Haoran Wei, Yaofeng Sun, and Yukun Li. 2025. Deepseek-ocr: Contexts optical compression. *Preprint*, arXiv:2510.18234.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Fan Zhou, Siqiao Xue, Danrui Qi, Wenhui Shi, Wang Zhao, Ganglin Wei, Hongyang Zhang, Caigai Jiang, Gangwei Jiang, Zhixuan Chu, and Faqiang Chen. 2024. Db-gpt-hub: Towards open benchmarking text-to-sql empowered by large language models. *Preprint*, arXiv:2406.11434.

A Appendix

This appendix provides implementation and protocol details that are omitted from the main text for conciseness. In particular, we describe (1) token accounting and TSR computation (table-only), (2) dataset visualization and rendering pipeline following DeepSeek-OCR style inputs, (3) robustness perturbation generation, (4) execution evaluation protocol in SQLite with failure taxonomy, (5) training and decoding hyperparameters for FROZENENC and FULLFT, and (6) additional canonicalization examples and intentionally excluded rules.

A.1 Canonicalization for EX-Can

We compute Canonical Exact Match (EX-Can) by applying a conservative canonicalization function $\text{Can}(\cdot)$ to both predicted and gold SQL strings, and then performing exact string comparison. Canonicalization removes surface-form differences that are unlikely to change semantics, while avoiding aggressive rewrites that may introduce incorrect equivalence.

More examples. Below are common cases handled by our pipeline:

- **Whitespace and case:** `SELECT * FROM T` → `select * from t`.
- **Operator spacing:** `a=1 AND b =2` → `a=1 and b=2`.
- **Flat boolean ordering (same level only):** `where age>18 and city='LA'` ↔ `where city='LA' and age>18`.

Rules we intentionally do *not* apply. We do not reorder JOINs, do not normalize nested boolean structures (beyond flat chains), do not rewrite subqueries, and do not apply algebraic transformations such as distributing predicates across OR clauses. These operations can change semantics or require database-aware reasoning, and are therefore excluded.

A.2 Token Accounting and TSR (Table-only)

In the main text we report input efficiency with table-only token accounting. That is, we measure how many tokens are needed to represent *only the table*, and we ignore question tokens and output tokens for TSR and related statistics. This choice isolates the compression benefit of optical tokenization on tabular content.

Algorithm 1: Canonicalization pipeline for EX-Can

Input: SQL string s
Output: Canonicalized SQL string $\text{Can}(s)$

Optional preprocessing:
$s \leftarrow \textsc{StripComments}(s)$; `// remove SQL comments if present`
$s \leftarrow \textsc{NormalizeUnicode}(s)$; `// normalize Unicode characters`

Step 1 (Case normalization):
$s \leftarrow \textsc{LowercaseKeywords}(s)$; `// or uppercase, applied consistently`

Step 2 (Whitespace normalization):
$s \leftarrow \textsc{CollapseWhitespace}(s)$; `// spaces/newlines/tabs → single space`
$s \leftarrow \textsc{Trim}(s)$

Step 3 (Punctuation spacing):
$s \leftarrow \textsc{NormalizeOpSpacing}(s)$; `// e.g., a=1 ↔ a = 1`

Step 4 (Redundant parentheses):
$s \leftarrow \textsc{RemoveRedundantParens}(s)$; `// only if precedence is preserved`

Step 5 (AND/OR condition ordering):
1. Parse s into an abstract syntax tree (AST): $T \leftarrow \textsc{ParseSQL}(s)$
2. For each `WHERE` clause node in T:
   (a) Identify maximal *flat* boolean chains at the same syntactic level (handle `AND` and `OR` separately)
   (b) Canonicalize each condition in the chain to a string key
   (c) Sort conditions lexicographically within each chain
   (d) Reconstruct the chain without re-associating nested boolean expressions
3. Serialize the modified AST back to SQL: $s \leftarrow \textsc{SerializeSQL}(T)$

return s

Table-text token length. Let $\text{Lin}(\cdot)$ denote a deterministic textual linearization of the table (schema plus cell values) used by the text-only baseline and by the OCR pipeline after reconstruction. We compute

$$L_{\text{table-text}}(i) = |\text{Tok}(\text{Lin}(\text{table}_i))|\,, \qquad (7)$$

where $\text{Tok}(\cdot)$ is the tokenizer of the decoder (same tokenizer used for SQL generation), and $|\cdot|$ counts tokens.

Optical token length. For OptiSQL, let $V_i = E(I_i)$ be the optical token sequence produced by the OCR-oriented encoder. We compute

$$L_{\text{opt}}(i) = |V_i|. \qquad (8)$$

Under a fixed token budget configuration (for example 256), $L_{\text{opt}}(i)$ is bounded and typically constant

for most samples, except when dynamic tiling is enabled.

Token Savings Ratio (TSR). We define TSR strictly on the table representation:

$$\text{TSR} = \frac{\mathbb{E}_i\left[L_{\text{table-text}}(i)\right]}{\mathbb{E}_i\left[L_{\text{opt}}(i)\right]}. \tag{9}$$

For interpretability, we also report the per-query averages $\mathbb{E}[L_{\text{table-text}}]$ and $\mathbb{E}[L_{\text{opt}}]$.

Implementation notes.

- We use the *same* tokenizer across baselines to avoid tokenizer-induced artifacts.
- $\text{Lin}(\cdot)$ is deterministic, including fixed column ordering and row ordering, so that TSR is stable across runs.
- For OCR pipelines, $L_{\text{table-text}}$ is computed on the reconstructed text fed into the downstream text-to-SQL model. This number is therefore comparable to the text-only baseline.

A.3 Visualized Spider 2.0-Snow Construction and Rendering

We build a visualized version of Spider 2.0-Snow (Lei et al., 2025) by rendering database tables into images while keeping natural language questions and gold SQL unchanged. Rendering follows the image-input conventions used by DeepSeek-OCR (Wei et al., 2025) to ensure compatibility with OCR-oriented optical tokenization.

Table extraction. For each example, we identify the target table(s) required by the gold SQL. In our single-table setting, we extract the referenced table into a structured grid with:

- a header row (column names),
- body cells (stringified values with type-aware formatting),
- optional row index column (disabled by default, enabled only for transpose augmentation).

Base rendering pipeline. We render tables via an HTML-CSS template and export to PNG:

1. Serialize the table grid into HTML.
2. Apply a CSS theme (fonts, border rules, padding, alignment).
3. Use a headless browser to rasterize the HTML into a high-resolution PNG with a white background and tight crop.
4. Apply model-side preprocessing expected by the encoder (RGB conversion, resize with aspect ratio preserved, and padding if needed).

Unless otherwise specified, FROZENENC uses a single base style at both training and evaluation.

Style templates (for FULLFT augmentation). For training-time style augmentation (FULLFT only), we sample from a set of style templates that vary:

- font family and size, boldness,
- cell padding, row height, and column width,
- border thickness and grid visibility,
- zebra striping and header emphasis.

Figure 3 shows representative examples in the main text.

Table transpose augmentation (FULLFT only). We optionally apply a transpose transform to encourage invariance to layout changes while preserving table semantics. Concretely, we render a transposed view in which each original column becomes a labeled row, and we add an index header for original rows. This produces a visually distinct but information-preserving representation. The supervision remains the same gold SQL because the underlying data content and header-value associations are preserved in the rendered view.

A.4 Robustness Perturbations

We evaluate robustness using test-time perturbations that modify the rendered table image without retraining. All perturbations preserve table content except for explicit occlusion in *HeaderMask*.

StyleShift. *StyleShift* samples a different rendering template at test time while preserving the same grid content. This assesses invariance to superficial style factors.

HeaderMask. *HeaderMask* occludes a portion of column headers. We mask only the header row region (not body cells) with a solid rectangle fill. The masking ratio is capped at $1/3$ of the header region. Empirically, larger masking ratios cause optimization instability and can prevent convergence.

Perturbation	Default parameters
StyleShift	sample 1 template uniformly from style pool
HeaderMask	mask ratio $r \sim \text{Unif}(0, 1/3)$; header row only
NoImage	remove optical tokens, keep question tokens
WrongTable	permute table images within evaluation batch or dataset

Table 3: Default robustness perturbation parameters used in evaluation.

- **Mask target:** header row cells only.
- **Mask ratio cap:** at most $1/3$ of header width per table.
- **Sampling:** uniformly sample a contiguous span (or multiple short spans) over header cells, then draw opaque rectangles.

NoImage and WrongTable. NOIMAGE removes optical tokens (or replaces them with a learned null token) while keeping the question unchanged. WRONGTABLE permutes table images across examples, producing mismatched table-question pairs. Both are diagnostic tests for visual grounding.

Suggested perturbation parameterization. Table 3 summarizes default parameters used in our experiments.

A.5 Execution Evaluation Protocol in SQLite

We evaluate executable correctness by executing both predicted SQL and gold SQL in SQLite, using the same database instance per example. Execution timeouts are treated as failures.

Engine and safety. We use SQLite as the DB engine. For each query:

1. open a read-only connection to the corresponding database,
2. execute the gold SQL to obtain the reference result R^*,
3. execute the predicted SQL to obtain R if execution succeeds.

Timeout handling. We enforce a wall-clock timeout (for example 2 seconds) per query using a process-level timeout mechanism. If a query exceeds the timeout, it is marked as *Timeout* and counted as execution failure. After excluding device instability, we treat timeouts as model-induced failures, typically caused by missing predicates or pathological joins.

Outcome category	Rate (%)
Correct execution (EXAcc = 1)	76
Executable but wrong result	10
Timeout	2
Non-executable (syntax or schema error)	12
Total failures (EXAcc = 0)	24
Non-executable among failures	50

Table 4: Execution outcome breakdown under SQLite. Among failures, non-executable predictions are about half.

Result normalization. We compare results after normalization:

- treat row order as irrelevant unless an explicit `ORDER BY` exists in both queries,
- canonicalize numeric formatting (float tolerance if needed),
- normalize NULL and string encodings.

Failure taxonomy and additional statistics. Beyond EXAcc, we compute:

- **ValidSQL**: percentage of predicted SQL that executes successfully within timeout,
- **NonExecutable**: percentage with SQL syntax errors, missing tables or columns, or other runtime errors,
- **Timeout**: percentage exceeding the timeout.

We observed that among the failed cases (EXAcc $= 0$), non-executable predictions account for roughly 50%. Table 4 reports a representative breakdown consistent with this observation. These values should be updated with the final run logs, but the taxonomy and reporting format remain the same.

Common failure causes. We find two dominant sources:

- **Non-executable:** schema linking errors (wrong column names, missing table aliases), malformed quoting, and invalid aggregation usage.
- **Executable but wrong:** incorrect join paths, missing filter predicates, and wrong aggregation or grouping.

Timeouts are commonly triggered by missing WHERE predicates leading to full scans, or unintended Cartesian products.

Hyperparameter	FROZENENC	FULLFT
Optimizer	AdamW	AdamW
β_1, β_2	(0.9, 0.95)	(0.9, 0.95)
Weight decay	0.1	0.1
Warmup ratio	0.03	0.05
LR schedule	cosine decay	cosine decay
Grad clip norm	1.0	1.0
Precision	bf16 (or fp16)	bf16 (or fp16)

Table 5: Common optimization settings.

Learning rate	FROZENENC	FULLFT
Decoder LR	2×10^{-4}	1×10^{-4}
Encoder LR	0 (frozen)	5×10^{-6}
Encoder LR relative scale	0	20× smaller than decoder

Table 6: Recommended learning rates for FROZENENC and FULLFT.

A.6 Training and Decoding Hyperparameters (FROZENENC vs FULLFT)

This section provides a complete hyperparameter specification used for the two training variants: FROZENENC (default) updates only the decoder, and FULLFT updates both encoder and decoder with training-time augmentations.

Optimizer and schedule. We use AdamW for all experiments. We recommend cosine decay with linear warmup. Gradient clipping is applied for stability.

Learning rates. FROZENENC uses a higher decoder LR since only the decoder is trained. FULLFT uses a much smaller encoder LR to reduce overfitting to visual styles and to preserve the pretrained optical tokenization behavior.

Batching and steps. We recommend setting global batch size via gradient accumulation to fit GPU memory. A practical configuration is:

- **Global batch size:** 256 examples (via accumulation).
- **Training steps:** 20k to 30k for FROZENENC, 10k to 20k for FULLFT (higher cost per step).

Sequence lengths. We cap input and output lengths to stabilize training and ensure comparable latency reporting.

- **Optical tokens:** $\{64, 100, 256, 400\}$ depending on budget.
- **Question length cap:** 128 tokens.
- **Max input length:** 512 tokens (optical plus question plus special tokens).
- **Max output length:** 256 tokens for SQL generation.

Decoding strategy. We use deterministic decoding for reliable executable evaluation.

- **Default:** beam search with beam size 4, temperature 0, max output length 256.
- **Latency reporting:** greedy decoding (temperature 0) with the same max output length.

Augmentations in FULLFT. Training-time augmentations described in Appendix A.3 and Appendix A.4 (style variation and transpose) are enabled only for FULLFT. All other settings are kept identical to FROZENENC to isolate the effect of encoder adaptation.

A.7 Notes on Baseline Token Accounting

To avoid confusion in Table 1, we emphasize that the OCR pipeline baseline consumes *text* tokens after OCR reconstruction and schema linearization. Therefore, its Tokens/Query reflects the length of reconstructed table text (table-only) rather than optical tokens. In the main table, we recommend labeling this row as *OCR-text + NL* and explicitly stating that its token count is computed under the same table-text token accounting defined in Appendix A.2.