

# BIRDTurk: Adaptation of the BIRD Text-to-SQL Dataset to Turkish

Burak Aktaş<sup>1</sup>, Mehmet Can Baytekin<sup>1</sup>, Süha Kağan Köse<sup>1</sup>, Ömer İlbilgi<sup>1</sup>,  
Elif Özge Yılmaz<sup>2</sup>, Çağrı Toraman<sup>2</sup>, Bilge Kaan Görür<sup>1</sup>

<sup>1</sup>Roketsan Inc., Artificial Intelligence Technologies Unit, Turkey

<sup>2</sup>Middle East Technical University, Computer Engineering Department, Turkey

burak.aktas@roketsan.com.tr, can.baytekin@roketsan.com.tr

kagan.kose@roketsan.com.tr, omer.ilbilgi@roketsan.com.tr

yilmaz.ozge\_01@metu.edu.tr, ctoraman@metu.edu.tr, kaan.gorur@roketsan.com.tr

## Abstract

Text-to-SQL systems have achieved strong performance on English benchmarks, yet their behavior in morphologically rich, low-resource languages remains largely unexplored. We introduce *BIRDTurk*, the first Turkish adaptation of the BIRD benchmark, constructed through a controlled translation pipeline that adapts schema identifiers to Turkish while strictly preserving the logical structure and execution semantics of SQL queries and databases. Translation quality is validated on a sample size determined by the Central Limit Theorem to ensure 95% confidence, achieving 98.15% accuracy on human-evaluated samples. Using *BIRDTurk*, we evaluate inference-based prompting, agentic multi-stage reasoning, and supervised fine-tuning. Our results reveal that Turkish introduces consistent performance degradation—driven by both structural linguistic divergence and underrepresentation in LLM pretraining—while agentic reasoning demonstrates stronger cross-lingual robustness. Supervised fine-tuning remains challenging for standard multilingual baselines but scales effectively with modern instruction-tuned models. *BIRDTurk* provides a controlled testbed for cross-lingual Text-to-SQL evaluation under realistic database conditions. We release the training and development splits to support future research.<sup>1</sup>

## 1 Introduction

Natural language interfaces to databases aim to democratize data access by enabling non-expert users to query structured data using everyday language. This vision has driven substantial progress in the Text-to-SQL field, supported by large-scale benchmarks such as WikiSQL (Zhong et al., 2017), Spider (Yu et al., 2018), and more recently, BIRD (Li et al., 2023) and Spider 2.0 (Lei et al., 2025).

<sup>1</sup>Links to our datasets and source code are available at: <https://github.com/metunlp/birdturk>

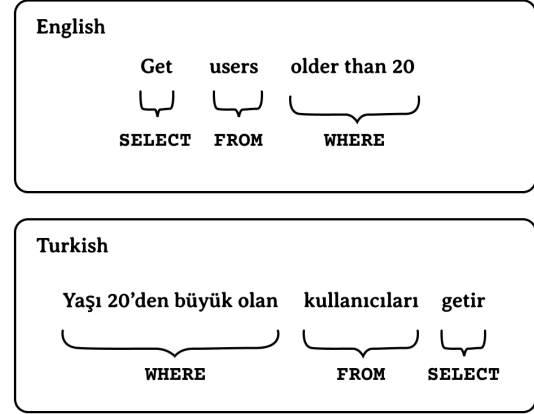


Figure 1: Structural divergence between English and Turkish queries. While English aligns linearly with SQL, Turkish distributes logic across suffixes and alters word order (SOV), complicating slot alignment.

Despite this progress, current benchmarks remain overwhelmingly English-centric (Min et al., 2019; Tuan Nguyen et al., 2020). This limitation is particularly important for morphologically rich and syntactically divergent languages like Turkish. Most state-of-the-art Text-to-SQL models implicitly rely on the close syntactic alignment between English and SQL, as both follow a Subject-Verb-Object (SVO) order. This shared structure enables a relatively linear correspondence between input tokens and SQL constructs (Qin et al., 2022). In contrast, Turkish exhibits an agglutinative morphology with a Subject-Object-Verb (SOV) word order (Ofłazer, 1994; Eryiğit et al., 2008; Umutlu et al., 2025), which disrupts this direct alignment and complicates semantic parsing (Dou et al., 2023).

For instance, consider the query “SELECT \* FROM Users WHERE age > 20”. As illustrated in Figure 1, the English phrase “Get users older than 20” aligns sequentially with the SQL logic. Conversely, in the Turkish translation “Yaşı 20’den büyük kullanıcıları getir”, the action corresponding to SELECT (“getir”) appears at the very end.

Furthermore, the logic for the  $>$  operator is morphologically distributed across the ablative suffix “-den” and the adjective “büyük”. As shown in Figure 1, such structural mismatches significantly challenge intent recognition and slot alignment mechanisms in multilingual models (Dou et al., 2023; Kanburoğlu and Tek, 2024).

While efforts such as Tur2SQL (Kanburoğlu and Tek, 2023) and TURSpider (Kanburoğlu and Tek, 2024) have provided valuable initial resources, they do not reflect the scale, schema complexity, or “dirty data” characteristics of modern enterprise environments. Existing Turkish datasets largely adhere to earlier benchmark paradigms, lacking the reasoning depth required to evaluate LLMs’ capabilities in handling real-world database ambiguities as introduced by BIRD (Li et al., 2023).

To address this gap, we present *BIRDTurk*, the first Turkish adaptation of the BIRD benchmark (Li et al., 2023). *BIRDTurk* is constructed via a controlled translation pipeline that preserves the logical structure and execution semantics of the original SQL queries and databases, while translating natural language questions and systematically localizing schema identifiers into Turkish. To ensure scalability and reliability, we employ a Central Limit Theorem (CLT)-based statistical verification framework, providing explicit confidence intervals for translation quality.

Our contributions can be summarized as follows:

- We introduce *BIRDTurk*, the first Turkish Text-to-SQL dataset adapted from BIRD.
- We propose a statistically grounded, CLT-based framework for validating large-scale dataset translations efficiently.
- We establish baseline results through systematic experiments spanning inference-based prompting, agentic reasoning, and supervised fine-tuning.

## 2 Related Work

### 2.1 Evolution of Text-to-SQL Benchmarks

The trajectory of Text-to-SQL benchmarks reflects a paradigm shift from constrained semantic parsing to real-world database grounding. Early datasets like ATIS and GeoQuery (Price, 1990; Zelle and Mooney, 1996) were limited to single domains, leading to overfitting and memorization issues (Finegan-Dollak et al., 2018). While WikiSQL

(Zhong et al., 2017) introduced scale, it oversimplified the task to single-table operations. Spider (Yu et al., 2018) addressed these limitations by introducing complex SQL structures (e.g., nesting, JOINS) and unseen schemas, establishing the *de facto* standard for cross-domain structural generalization. However, these benchmarks operate on “clean” schemas, primarily testing a model’s ability to map natural language tokens to SQL syntax rather than reasoning over database content.

Recent benchmarks move beyond syntactic alignment toward execution-centric evaluation on “dirty” data. BIRD (Li et al., 2023) represents this leap by introducing massive, noisy databases (33.4 GB) that require reasoning over *external knowledge* (e.g., domain terminology, numeric calculation) rather than simple schema linking. This shift exposes a significant gap between human performance (92.96%) and state-of-the-art LLMs, highlighting the necessity of content-grounded reasoning. Extending this trajectory, Spider 2.0 (Lei et al., 2025) further redefines the task by adopting an agentic “Code Agent” paradigm, requiring models to debug queries and navigate enterprise workflows, thereby positioning Text-to-SQL as a multi-turn software engineering challenge rather than a single-turn translation task.

### 2.2 Challenges in Low-Resource Languages

The scarcity of non-English datasets has prompted numerous adaptations of Spider 1.0, ranging from question-only translations in Chinese (Min et al., 2019) to full schema localization in Vietnamese (Tuan Nguyen et al., 2020), Russian (Bakshandaeva et al., 2022), and Arabic (Almohaimed et al., 2024). While foundational, these benchmarks primarily test syntactic alignment on idealized schemas, often bypassing the “dirty data” challenges inherent in enterprise environments.

In the Turkish domain, Tur2SQL (Kanburoğlu and Tek, 2023) and TURSpider (Kanburoğlu and Tek, 2024) represent significant milestones. However, they highlight a critical bottleneck: English-centric LLMs frequently struggle with Turkish’s agglutinative morphology, leading to schema hallucinations due to suffix-induced tokenization mismatches (Kanburoğlu and Tek, 2024). Furthermore, by adhering to Spider’s topology, existing Turkish datasets do not assess the content-grounded reasoning capabilities required for modern database applications, a gap *BIRDTurk* aims to fill. In addition, recent Turkish LLM benchmarking efforts such as

TurkBench (Toraman et al., 2026) include evaluations of instruction-following capabilities; however, SQL-oriented reasoning is only marginally covered, and the benchmark does not target the structured, content-grounded reasoning over relational databases required in this task.

### 3 Dataset Construction

We construct *BIRDTurk* by translating the publicly available training and development splits of BIRD into Turkish. Our goal is to keep the benchmark *functionally identical* across languages: the underlying databases and SQL semantics must remain unchanged, while the natural-language interface is localized. Following prior cross-lingual Text-to-SQL benchmarks (Min et al., 2019; Tuan Nguyen et al., 2020; Bakshandaeva et al., 2022; Almo-haimeed et al., 2024), we prioritize *semantic equivalence*, *schema fidelity*, and *execution consistency*. Since LLM-based translation can drift—particularly for morphologically rich languages like Turkish (Kanburoğlu and Tek, 2024)—we use a schema-grounded pipeline that constrains edits that could break executability.

We adopt *schema-only localization*: we translate database/table/column identifiers but do *not* translate database cell values (e.g., quoted string literals in WHERE clauses). This avoids introducing a separate value-linking problem and simplifies execution-level comparisons.

#### 3.1 Schema Mapping

Before translating questions, we establish a deterministic *schema mapping*  $\mathcal{M}$  for each database, defining a one-to-one correspondence between original English identifiers (tables/columns) and their Turkish counterparts. Fixing this vocabulary upfront constrains question translation, evidence localization, and SQL rewriting to a closed identifier set, preventing out-of-vocabulary terms.

We extract schema metadata (table names, column attributes, foreign keys) directly from the SQLite files and include database descriptions when available to resolve ambiguities. We translate identifiers with gemini-2.5-flash (DeepMind, 2025) under strict constraints: ASCII-only snake\_case identifiers and standardized recurring sub-terms (e.g., *movie\_popularity* → *film\_populerligi*, *first\_name* → *ilk\_isim*). The full schema-mapping prompt is provided in Appendix A.1.

We also address rare instances of *identifier collision*, where distinct English identifiers map to the same Turkish form. Although strictly limited to two columns in our dataset, we enforce *database-local uniqueness* and resolve these collisions deterministically (e.g., via stable suffix rules), which also mitigates schema hallucination in non-English schema settings (Kanburoğlu and Tek, 2024; Dou et al., 2023).

#### 3.2 Translation and Localization Pipeline

We frame localization as a constraint satisfaction problem in which the translated Turkish text must simultaneously preserve the original SQL semantics and comply with the constraints defined by  $\mathcal{M}$ . To achieve this, we design a three-stage translation and localization pipeline that systematically transforms the BIRD benchmark into its Turkish counterpart, BIRDTurk. The pipeline ensures semantic fidelity to the source queries while enforcing linguistic and structural consistency at each stage. The general workflow of this translation pipeline is illustrated in Figure 2.

##### 3.2.1 Evidence Standardization and Schema Alignment

Evidence fields in BIRD interleave natural language with schema references; inconsistencies can break the link between intent and executable SQL. Before translation, we deterministically rewrite all backticked identifiers in evidence using  $\mathcal{M}$  and enforce an invariant: each backticked span must exactly match the Turkish snake\_case identifiers used in the localized SQL.

##### 3.2.2 Joint Question-Evidence Translation

We then jointly translate the question and schema-aligned evidence in one context window using gemini-2.5-flash (prompt in Appendix A.2). The model must preserve backticked spans verbatim while localizing surrounding text, and it must retain semantic constants (numbers, dates) and comparative logic (e.g., *highest/lowest/top-k*). For stylistic uniformity, we follow a fixed instruction set (Tuan Nguyen et al., 2020) (e.g., “List” → “Listeleyiniz”, “How many” → “Kaç ... vardır?”).

##### 3.2.3 AST-Based SQL Localization

To preserve execution behavior, we avoid neural SQL generation and apply a deterministic, structure-aware rewrite. We parse each SQL query into an Abstract Syntax Tree (AST) (Aho et al.,

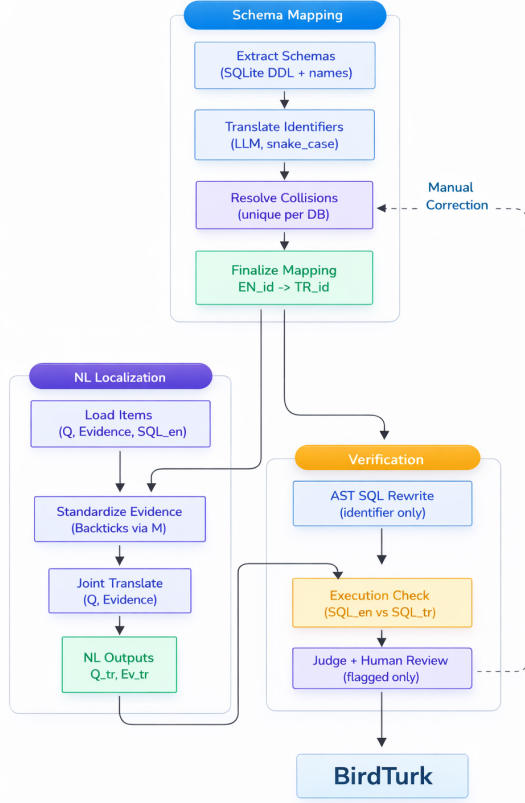


Figure 2: Translation and localization pipeline designed to convert the BIRD SQL benchmark into BIRDTurk while preserving SQL intent and enforcing Turkish language constraints.

2006) and rewrite *only identifier nodes* (tables/columns) via  $\mathcal{M}$ ; all other elements (keywords, operators, functions, literals) remain unchanged. This avoids pitfalls of regex/string substitution (e.g., alias collisions, partial matches such as `table.column`, or accidental edits to literals) and preserves syntactic validity and semantic equivalence. As an additional safeguard, we use a rubric-based LLM-as-a-judge to verify text–SQL alignment for flagged instances (prompt in Appendix A.3).

### 3.3 Quality Control and Statistical Validation

To ensure reliability, we combine automated integrity checks with statistically grounded human evaluation.

#### 3.3.1 Automated Consistency Checks

Before manual review, all examples undergo automated verification to filter structural and execution errors.

**Execution Equivalence (Primary Signal).** We adopt execution correctness as the primary validity criterion. For every instance, we execute both the original English SQL and the generated Turkish SQL on the underlying database. We verify that their result sets are identical ( $R_{en} = R_{tr}$ ), strictly enforcing row ordering where specified. This step guarantees that the localization process preserves the executable semantics of the original query.

**Structural Integrity and Schema Constraints.** We algorithmically verify that all schema identifiers (tables, columns) referenced in the Turkish SQL strictly map to the localized schema. Furthermore, we cross-reference backticked identifiers in the natural language evidence against the localized schema to prevent hallucinated columns. Examples failing these checks are automatically flagged for correction.

#### 3.3.2 Statistical Semantic Validation

Automated checks cannot fully capture fluency and semantic fidelity. To assess translation quality at scale, we use probabilistic sampling grounded in the Central Limit Theorem (CLT) (Feller, 1968).

We model translation correctness as a Bernoulli random variable and estimate the required sample size for a *95% confidence level* ( $Z = 1.96$ ) with an *error margin of  $\pm 3\%$*  ( $E = 0.03$ ). Assuming maximum variance ( $p = 0.5$ ), the sample size for an infinite population is:

$$n_0 = \frac{Z^2 \cdot p(1-p)}{E^2} \approx \frac{1.96^2 \cdot 0.25}{0.0009} \approx 1,068 \quad (1)$$

Let  $N$  denote the total number of samples in BIRDTurk, computed as the sum of the training and development splits ( $N = 10,962$ ). Applying the Finite Population Correction (FPC) yields:

$$n = \frac{n_0}{1 + \frac{n_0-1}{N}} \approx \frac{1,068}{1 + \frac{1,067}{10,962}} \approx 974 \quad (2)$$

Accordingly, we manually evaluated 974 *randomly sampled examples*; 956 were judged *correct*, yielding an observed translation accuracy of 98.15%. Under the CLT, this implies the dataset-level accuracy lies within a  $\pm 3\%$  margin at 95% confidence.

Algorithm 1 summarizes our three-phase, schema-grounded construction pipeline, including schema mapping, joint question–evidence localization, and execution-consistent SQL rewriting with verification.

---

**Algorithm 1** Overview of the BIRDTURK construction pipeline.

---

**Require:** SQLite databases  $\mathcal{D}$ ; English items  $\mathcal{X}$  with  $(db\_id, question, evidence, sql_{en})$

**Ensure:** Turkish items  $\mathcal{X}_{tr}$  with  $(db\_id_{tr}, question_{tr}, evidence_{tr}, sql_{tr})$

- 1: **Phase 1: Schema mapping**
- 2: **for all**  $db \in \mathcal{D}$  **do**
- 3:    $S_{db} \leftarrow$  extract schema metadata (DDL + identifier list)
- 4:    $M_{db} \leftarrow$  LLM translate identifiers in  $S_{db}$  to Turkish ASCII snake\_case
- 5:    $M_{db} \leftarrow$  resolve identifier collisions (unique within  $db$ )
- 6: **end for**
- 7: **Phase 2: Natural language localization**
- 8: **for all**  $x \in \mathcal{X}$  **do**
- 9:    $M \leftarrow M_{x.db\_id}$
- 10:    $e^{std} \leftarrow$  rewrite backticked identifiers in  $x.evidence$  via  $M$
- 11:    $(q_{tr}, e_{tr}) \leftarrow$  joint LLM translation of  $(x.question, e^{std})$  with backticks frozen
- 12: **end for**
- 13: **Phase 3: SQL localization and verification**
- 14: **for all**  $x \in \mathcal{X}$  **do**
- 15:    $M \leftarrow M_{x.db\_id}$
- 16:    $sql_{tr} \leftarrow$  AST rewrite  $x.sql_{en}$  by replacing identifier nodes via  $M$
- 17:   **if** EXECEQUAL( $sql_{en}, sql_{tr}$  on  $db$ ) is false **then**
- 18:     flag  $x$  for review
- 19:   **end if**
- 20:   **if**  $x$  is flagged **then**
- 21:     rubric-based judge + human correction; update  $M$  if needed
- 22:     re-run verification
- 23:   **end if**
- 24:   store  $(db\_id_{tr}, q_{tr}, e_{tr}, sql_{tr})$
- 25: **end for**

---

### 3.4 Statistics of the Dataset

BIRD is a large-scale benchmark with 12,751 text-to-SQL pairs across train/dev/test and 95 databases totaling 33.4 GB over 37 domains (Li et al., 2023). Each database contains 7.3 tables on average and roughly 549,000 rows; the largest database (“Donor”) is 4.5 GB.

Table 1 presents a quantitative analysis of the linguistic characteristics of BIRDTurk and its structural alignment with the original BIRD benchmark. The comparison reveals systematic linguistic shifts

that are consistent across both the training and development splits, confirming the robustness of the cross-lingual adaptation process.

Turkish agglutination compresses surface word counts (Train: -27.3%, Dev: -26.9%) as English particles are absorbed into suffixes, while character counts remain similar (4.3–5.1% decrease), indicating preserved content at higher information density (Eryigit et al., 2008).

Lexical diversity increases sharply: TTR more than doubles (Train: +122.2%, Dev: +100.8%) and vocabulary size grows substantially (+68.2%, +51.2%). This reflects Turkish’s productive morphology, which creates many surface forms from the same lemma and exacerbates sparsity for models (Hakkani-Tür et al., 2002).

SQL length remains effectively unchanged (-0.5%), indicating preserved structural complexity. In contrast, evidence token counts increase (Train: +9.3%), consistent with the “multilingual tokenization tax” (Ahia et al., 2023): English-centric tokenizers often over-segment Turkish suffixes into multiple subword units (Rust et al., 2021; Toraman et al., 2023).

## 4 Experiments

This section describes the experimental setup used to validate *BIRDTurk* both as an evaluation dataset and as a supervised training resource. Following the experimental taxonomy of the original BIRD benchmark, we evaluate BIRDTurk under two complementary Text-to-SQL paradigms: inference-only prompting (including agentic reasoning pipelines) and supervised fine-tuning.

### 4.1 Baseline Methods

#### 4.1.1 Prompt-Based Inference

We evaluate Text-to-SQL generation under inference-only settings, where no task-specific training or parameter updates are performed and all adaptation occurs through prompting.

As a direct baseline, we adopt a zero-shot in-context learning (ICL) setup, where the model receives a textual description of the database schema together with the natural language question and generates the SQL query in a single pass (see Appendix A.4 and A.5 for prompt templates). This setting assesses generalization to Turkish Text-to-SQL queries without explicit supervision or structural guidance.

To examine the effect of structured reasoning,

Table 1: Linguistic and structural statistics for BIRDTurk Training and Development sets compared to the original English BIRD. The comparison highlights significant cross-lingual differences, such as reduced word counts due to agglutination and increased lexical diversity.

Statistic	Training Set			Development Set		
	BIRD (En)	BIRDTurk (Tr)	Change (%)	BIRD (En)	BIRDTurk (Tr)	Change (%)
Total Questions	9,428	9,428	–	1,534	1,534	–
Avg. Words per Question	14.05	10.21	-27.3%	14.55	10.64	-26.9%
Avg. Characters per Question	79.81	75.75	-5.1%	82.76	79.18	-4.3%
Avg. Tokens per Question	15.74	11.91	-24.3%	16.24	12.22	-24.8%
<i>Lexical Diversity</i>						
Vocabulary Size (Unique)	9,002	15,142	+68.2%	2,450	3,704	+51.2%
Type-Token Ratio (TTR)	6.07%	13.49%	+122.2%	9.84%	19.76%	+100.8%
<i>Complexity &amp; Integrity</i>						
Avg. SQL Tokens	31.03	30.89	-0.5%	31.46	31.26	-0.6%
Avg. Evidence Tokens	21.32	23.31	+9.3%	21.49	22.04	+2.6%

we additionally evaluate DIN-SQL (Pourreza and Rafiei, 2023), an agentic multi-stage inference pipeline that decomposes SQL generation into guided steps. The pipeline incorporates:

- **Schema Linking** to ground relevant tables and columns,
- **Intent Classification** to infer high-level query structure,
- **Self-Correction** using execution feedback to refine generated SQL.

For Turkish compatibility, we translate the original DIN-SQL prompts using a structure-preserving strategy (Appendix A.6).

Both methods are instantiated using gemini-2.5-flash-lite, ensuring that performance differences arise from the reasoning strategy rather than model capacity. All inference experiments are conducted on both English BIRD and BIRDTurk under identical configurations.

#### 4.1.2 Supervised Fine-Tuning

We also evaluate supervised Text-to-SQL learning on BIRDTurk to assess whether the translated dataset supports parameter-updated training.

Following the original BIRD setup, we fine-tune multilingual models from the mT5 family (mT5-small, mT5-base, mT5-large). While mT5 provides multilingual tokenization suitable for Turkish, fine-tuning these models yields limited performance gains, indicating challenges in learning effective Turkish Text-to-SQL mappings under this setup.

Motivated by these limitations, we transition to the Qwen2.5-Coder family, utilizing the 0.5B,

1.5B, and 3B instruction-tuned variants. Our selection of this specific architecture is driven by two complementary factors grounded in recent empirical findings. First, the Qwen2.5-Coder series achieves state-of-the-art performance in code generation benchmarks among open-weights models, benefiting from a massive pre-training corpus of 5.5 trillion tokens enriched with synthetic data (Hui et al., 2024). Second, and critical for our cross-lingual focus, recent research highlights the structural advantage of the Qwen architecture for Turkish. Hacifazlıoğlu et al. (2024) demonstrate that the Qwen tokenizer achieves a superior compression ratio compared to models like Llama-3 and Gemma, effectively mitigating the over-segmentation of Turkish agglutinative suffixes. This tokenization efficiency allows the model to preserve the semantic integrity of Turkish natural language queries with fewer tokens, thereby enhancing the alignment between Turkish intent and SQL logic.

All Qwen models are fine-tuned and evaluated on BIRDTurk using the same supervised protocol. While their absolute performance remains below inference-based pipelines, they consistently outperform mT5 across all metrics, confirming that modern instruction-tuned models equipped with efficient tokenizers provide a more effective learning signal for Turkish Text-to-SQL tasks.

To ensure a fair comparison between base and fine-tuned capabilities, inference for both base and fine-tuned models is performed using the standard ICL prompting strategy.

## 4.2 Evaluation Setup and Metrics

We evaluate all models on the *dev split* of the respective datasets, reporting metrics averaged over three independent runs to ensure robustness. The primary evaluation metrics are *Execution Accuracy (EX)* and *Valid Efficiency Score (VES)*, following the BIRD benchmark protocol, alongside *Exact Match (EM)* for comparability with prior work.

**Exact Match (EM).** Exact Match measures whether the predicted SQL query is structurally identical to the ground-truth SQL after canonical normalization:

$$\text{EM} = \frac{1}{N} \sum_{n=1}^N \mathbb{I}_{\text{SQL}}(Y_n, \hat{Y}_n),$$

where:

$$\mathbb{I}_{\text{SQL}}(Y, \hat{Y}) = \begin{cases} 1, & \text{if } Y \equiv \hat{Y} \text{ after normalization,} \\ 0, & \text{otherwise.} \end{cases}$$

**Execution Accuracy (EX).** Execution Accuracy measures whether the execution results of the predicted SQL and the ground-truth SQL are identical:

$$\text{EX} = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(V_n, \hat{V}_n),$$

where:

$$\mathbb{I}(V, \hat{V}) = \begin{cases} 1, & \text{if } V = \hat{V}, \\ 0, & \text{otherwise.} \end{cases}$$

**Valid Efficiency Score (VES).** Valid Efficiency Score extends Execution Accuracy by accounting for execution efficiency:

$$\text{VES} = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(V_n, \hat{V}_n) \cdot R(Y_n, \hat{Y}_n),$$

where:

$$R(Y_n, \hat{Y}_n) = \sqrt{\frac{E(Y_n)}{E(\hat{Y}_n)}}.$$

Here,  $E(\cdot)$  denotes execution time. The square root term mitigates execution-time variance and extreme outliers (Li et al., 2023).

## 5 Results and Discussion

### 5.1 Overall Performance on BIRDTurk

We first provide a high-level overview of model performance on the Turkish BIRDTurk dataset.

Across all experimental settings, BIRDTurk constitutes a challenging testbed, reflecting both the linguistic properties of Turkish and the complexity of enterprise-scale Text-to-SQL tasks inherited from BIRD.

Three consistent trends emerge from our experiments. First, supervised fine-tuning on BIRDTurk remains challenging for earlier multilingual baselines, while more recent instruction-tuned models exhibit clearer and more scalable learning behavior. Second, inference-based approaches achieve substantially higher execution accuracy than supervised methods under identical evaluation conditions. Third, within inference-based paradigms, agentic reasoning consistently improves over direct prompting across both English and Turkish.

We analyze these observations in detail below.

### 5.2 Supervised Fine-Tuning Analysis on BIRDTurk

We begin with supervised experiments to directly assess whether BIRDTurk supports parameter-updated learning in Turkish. Table 2 reports results for fine-tuned models evaluated on the same BIRDTurk development split. Metrics are reported as Execution Accuracy (EA), Valid Efficiency Score (VES), and Exact Match (EM), where higher values indicate better performance ( $\uparrow$ ).

Model	Fine-Tuned $\uparrow$		
	EA	VES	EM
mT5-small	0.26	0.28	0.00
mT5-base	0.98	1.04	0.06
mT5-large	1.83	2.05	0.13
Qwen2.5-Coder-0.5B-Instruct	1.24	1.38	0.33
Qwen2.5-Coder-1.5B-Instruct	7.24	8.01	2.61
Qwen2.5-Coder-3B-Instruct	<b>15.38</b>	<b>17.12</b>	<b>4.82</b>

Table 2: Supervised Text-to-SQL performance on the Turkish BIRDTurk dev split. Results are reported for models after supervised fine-tuning. Metrics include Execution Accuracy (EA), Valid Efficiency Score (VES), and Exact Match (EM), where higher is better ( $\uparrow$ ).

Several discussions points to follow from Table 2. We excluded base model results from the table as they all performed near 0%, with the best performing base model (Qwen2.5-Coder-3B-Instruct) achieving only 2.22% execution accuracy. Multilingual baselines show only marginal improvements after fine-tuning, with execution accuracy remaining below 2%. In contrast, instruction-tuned models demonstrate clear and consistent gains, with

performance scaling reliably with model size.

These results indicate that BIRDTurk provides a usable supervised signal; however, effective learning in Turkish benefits from models with stronger instruction-following and structured generation capabilities rather than multilingual coverage alone.

### 5.3 Inference-Based Validation via English–Turkish Comparison

To ensure that the observed supervised learning behavior is not an artifact of dataset construction or evaluation noise, we next examine inference-based performance under identical prompting conditions. All inference experiments are conducted using the same underlying language model, gemini-2.5-flash-lite, enabling a controlled comparison across languages.

Table 3 compares direct prompting (In-Context Learning) and agentic reasoning (DIN-SQL) on both the original English BIRD benchmark and its Turkish counterpart, BIRDTurk.

Method	BIRD (EN) ↑			BIRDTurk (TR) ↑		
	EA	VES	EM	EA	VES	EM
In-Context Learning	58.21	63.15	12.65	43.16	45.34	10.82
DIN-SQL	<b>60.89</b>	<b>64.66</b>	<b>17.60</b>	<b>49.02</b>	<b>51.10</b>	<b>11.41</b>

Table 3: Inference-based and agentic performance comparison on English BIRD and Turkish BIRDTurk dev splits. Metrics include Execution Accuracy (EA), Valid Efficiency Score (VES), and Exact Match (EM), where higher values indicate better performance (↑).

Across both datasets, English consistently outperforms Turkish in absolute terms. However, the relative ordering of methods remains unchanged: DIN-SQL consistently improves over direct prompting in both languages.

This stability suggests that BIRDTurk preserves the core structural and reasoning characteristics of BIRD, and that the observed performance gap is largely attributable to language-induced difficulty rather than evaluation inconsistencies.

### 5.4 The Impact of Agentic Reasoning Across Languages

A closer examination of Table 3 reveals an asymmetry in the relative gains provided by agentic reasoning across languages. While DIN-SQL improves over direct prompting in both English and Turkish, the magnitude of improvement is more pronounced in the Turkish setting.

In English, agentic reasoning yields moderate but consistent gains across all evaluation metrics.

In contrast, for Turkish, DIN-SQL introduces larger relative improvements, particularly in Execution Accuracy and Valid Efficiency Score. This pattern suggests that explicit task decomposition, schema grounding, and iterative correction play a more critical role when surface-level alignment between natural language and SQL is weaker.

One plausible explanation is that Turkish’s agglutinative morphology and SOV word order introduce additional ambiguity at the token and phrase level. Agentic pipelines reduce reliance on direct token-to-SQL correspondence by enforcing intermediate reasoning steps aligned with schema structure and execution semantics.

Importantly, these trends are observed under identical inference configurations, indicating that the benefits of agentic reasoning stem primarily from linguistic factors rather than model-specific effects.

### 5.5 Cross-Lingual Performance Comparison

The remaining performance gap between English and Turkish can be attributed to two complementary factors: intrinsic linguistic properties of Turkish and representational imbalances in LLM pretraining.

From a linguistic perspective, Turkish poses structural challenges that complicate direct Text-to-SQL transfer: (1) a Subject–Object–Verb (SOV) word order that disrupts the linear alignment between natural language and SQL syntax; (2) agglutinative morphology leading to increased lexical sparsity, as semantic information is distributed across numerous low-frequency surface forms; and (3) suffix-heavy word forms that increase tokenization fragmentation, resulting in longer token sequences for equivalent semantic content.

Beyond these intrinsic challenges, current LLMs are predominantly trained on English-centric corpora, leaving Turkish significantly underrepresented in pretraining data. This imbalance limits the models’ exposure to Turkish linguistic patterns, compounding the difficulties posed by the language’s morphological complexity. The combination of these factors—structural divergence and limited pretraining coverage—creates a particularly challenging setting for cross-lingual generalization.

Agentic reasoning partially mitigates these effects by enforcing schema grounding and intermediate decision-making, thereby reducing reliance on surface-level token alignment and language-specific priors learned during pretraining.

## 6 Conclusion

In this work, we introduced *BIRDTurk*, a Turkish adaptation of the BIRD Text-to-SQL benchmark designed for realistic database settings. By carefully controlling the adaptation process to maintain comparable execution behavior and database structure across languages, while localizing schema identifiers and natural language questions, we established a controlled evaluation dataset that enables direct and fair cross-lingual comparison.

Translation quality was validated using a statistically grounded CLT-based framework, providing dataset-level reliability without requiring exhaustive manual annotation. Leveraging *BIRDTurk*, we conducted a systematic evaluation of inference-based prompting, agentic multi-stage reasoning pipelines, and supervised fine-tuning approaches under Turkish linguistic conditions.

Our results demonstrate that Turkish introduces a consistent yet bounded performance degradation in inference-only settings. In contrast, agentic reasoning pipelines exhibit stronger robustness across languages, indicating that explicit task decomposition, schema grounding, and intermediate reasoning steps effectively mitigate language-induced challenges. While supervised fine-tuning remains difficult for standard multilingual baselines, more recent instruction-tuned models show clearer and more scalable learning behavior in the Turkish setting.

Beyond its current scope, *BIRDTurk* provides a foundation for several important research directions. Future extensions include the construction of a fully native Turkish Text-to-SQL benchmark to complement translation-based evaluation, the development of hybrid datasets combining translated and natively authored Turkish questions, and the expansion of *BIRDTurk* toward more agentic, enterprise-oriented, and multi-turn evaluation settings inspired by Spider 2.0. These directions would further strengthen the benchmark’s realism and broaden its applicability to practical deployment scenarios.

Overall, *BIRDTurk* fills a critical gap in Turkish Text-to-SQL research by enabling controlled cross-lingual evaluation and systematic analysis of modeling paradigms under realistic database conditions, while also laying the groundwork for more advanced and native Turkish evaluation frameworks.

## Limitations

Despite its contributions, *BIRDTurk* has several limitations that should be considered when interpreting the results.

- *BIRDTurk* is constructed via translation rather than native Turkish question authoring. While this enables controlled cross-lingual comparison with BIRD, it may not fully reflect naturally occurring Turkish query formulations.
- Translations rely on a single LLM (Gemini), which may introduce systematic stylistic biases despite postprocessing and CLT-based validation.
- CLT-based validation provides dataset-level quality guarantees but does not replace exhaustive manual annotation at the individual sample level.
- Certain Turkish-specific linguistic phenomena (e.g., ellipsis, pragmatic inference) are likely underrepresented due to the translation-based construction process.

While translation noise is statistically bounded, a small subset of linguistically complex or highly implicit queries may still be affected.

## Ethical Considerations

**Licensing and Copyright.** *BIRDTurk* is constructed as a derivative work of the original BIRD benchmark (Li et al., 2023). We strictly adhere to the updated usage terms of the source material, which is distributed under the *Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)* license. In compliance with the "Share-Alike" provision, *BIRDTurk* is released under the same license to ensure that all derivative improvements remain open and accessible to the research community. We explicitly acknowledge the intellectual property rights of the original dataset creators and contribute this adaptation to foster reproducible and collaborative advancements in cross-lingual semantic parsing.

**Use of Generative AI.** Generative AI was used solely to assist with language editing. All scientific contributions, data construction and analysis, and interpretations presented in this work are original and were conducted entirely by the authors.

## Acknowledgments

We gratefully acknowledge support from Roketsan Inc. and the Google Gemini Academic Reward Program, which helped enable the experiments and computing resources used in this study.

## References

- Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Kyunghyun Jung, Yulia Tsvetkov, and Noah A. Smith. 2023. Do all languages cost the same? tokenization in the era of commercial LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 10760–10773. Association for Computational Linguistics.
- Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. 2006. *Compilers: Principles, Techniques, and Tools*, 2nd edition. Pearson Education.
- Saleh Almohaimeed, Saad Almohaimeed, Mansour Al Ghanim, and Liqiang Wang. 2024. Ar-Spider: Text-to-SQL in Arabic. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24)*, pages 1024–1030.
- Daria Bakshandaeva, Oleg Somov, Ekaterina Dmitrieva, Vera Davydova, and Elena Tutubalina. 2022. **PAUQ: Text-to-SQL in Russian**. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2355–2376, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Google DeepMind. 2025. **Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities**. *arXiv preprint*.
- Longxu Dou, Yan Gao, Mingyang Pan, Dingzirui Wang, Wanxiang Che, Dechen Zhan, and Jian-Guang Lou. 2023. Multispider: Towards benchmarking multilingual text-to-sql semantic parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12745–12753.
- Gülşen Eryiğit, Joakim Nivre, and Kemal Oflazer. 2008. Dependency parsing of turkish. *Computational Linguistics*, 34(3):357–389.
- William Feller. 1968. *An Introduction to Probability Theory and Its Applications, Vol. 1*, 3rd edition. Wiley, New York.
- Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. **Improving text-to-SQL evaluation methodology**. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.
- H. Ozan Hacızadlıoğlu, Vahid Partovi Nia, and Ercan Kuruoğlu. 2024. Fine-tuning large language models for turkish. *arXiv preprint arXiv:2407.15185*.
- Dilek Z Hakkani-Tür, Kemal Oflazer, and Gökhan Tür. 2002. Statistical morphological disambiguation for turkish text. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(04):381–402.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jialong Huang, Tong Yu, Ganqu Wang, and 1 others. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- Ali Buğra Kanburoğlu and F. Boray Tek. 2023. **TUR2SQL: A cross-domain Turkish dataset for Text-to-SQL**. In *Proceedings of the 8th International Conference on Computer Science and Engineering (UBMK)*, pages 206–211. IEEE.
- Ali Buğra Kanburoğlu and Faik Boray Tek. 2024. **TUR-Spider: A Turkish Text-to-SQL dataset and LLM-based study**. *IEEE Access*, 12:169379–169387.
- Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, and 1 others. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *Proceedings of the International Conference on Learning Representations (ICLR)*. To appear.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023. Can LLM already serve as a database interface? a BIG bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 42330–42357.
- Qingkai Min, Yuefeng Shi, and Yue Zhang. 2019. **A pilot study for Chinese SQL semantic parsing**. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3652–3658, Hong Kong, China. Association for Computational Linguistics.
- Kemal Oflazer. 1994. Two-level description of turkish morphology. *Literary and linguistic computing*, 9(2):137–148.
- Mohammadreza Pourreza and Davood Rafiei. 2023. **Din-sql: Decomposed in-context learning of text-to-sql with self-correction**. In *Advances in Neural Information Processing Systems*, volume 36, pages 36339–36348. Curran Associates, Inc.
- P. J. Price. 1990. **Evaluation of spoken language systems: the ATIS domain**. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

- Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, and 1 others. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*.
- Phillip Rust and 1 others. 2021. How good is your tokenizer? on the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, pages 3118–3135.
- Cagri Toraman, Eyup Halit Yilmaz, Furkan Şahinuç, and Oguzhan Ozcelik. 2023. [Impact of tokenization on language models: An analysis for turkish](#). *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 22(4).
- Çağrı Toraman, Ahmet Kaan Sever, Ayse Aysu Cengiz, Elif Ecem Arslan, Görkem Sevinç, Mete Mert Birdal, Yusuf Faruk Güldemir, Ali Buğra Kanburoğlu, Sezen Felekoğlu, Osman Gürlek, Sarp Kantar, Birsen Şahin Kütük, Büşra Tufan, Elif Genç, Serkan Coşkun, Gupse Ekin Demir, Muhammed Emin Arayıcı, Olgun Dursun, Onur Gungor, and 3 others. 2026. Turk-bench: A benchmark for evaluating turkish large language models. *arXiv preprint arXiv:2601.07020*.
- Anh Tuan Nguyen, Mai Hoang Dao, and Dat Quoc Nguyen. 2020. [A pilot study of text-to-SQL semantic parsing for Vietnamese](#). In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4079–4085, Online. Association for Computational Linguistics.
- Elif Ecem Umutlu, Ayse Aysu Cengiz, Ahmet Kaan Sever, Seyma Erdem, Burak Aytan, Busra Tufan, Abdullah Topraksoy, Esra Darıcı, and Cagri Toraman. 2025. [Evaluating the quality of benchmark datasets for low-resource languages: A case study on Turkish](#). In *Proceedings of the Fourth Workshop on Generation, Evaluation and Metrics (GEM<sup>2</sup>)*, pages 471–487, Vienna, Austria and virtual meeting. Association for Computational Linguistics.
- 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.
- John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, volume 2, pages 1050–1055, Portland, Oregon. AAAI Press.
- Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

## A Prompt Engineering for BIRDTurk

To ensure full reproducibility, we provide the complete set of prompt templates employed throughout our pipeline. These prompts are organized according to their functional roles within the system.

Prompts used for *data translation and schema localization* enforce strictly defined JSON-only output formats and explicitly prohibit any unintended or implicit modifications to SQL tokens or schema identifiers.

Prompts used during *inference and SQL generation* explicitly specify task-level constraints, business-logic guardrails, and SQLite-specific execution rules, thereby ensuring that the generated queries are both syntactically valid and semantically correct.

Finally, prompts used for *DIN-SQL prompt translation* are designed as controlled drop-in replacements for training and evaluation. These prompts preserve the original structure and constraints of the DIN-SQL framework, while translating only the natural-language instructions without altering the underlying semantics.

### A.1 Schema Mapping Prompt

#### Schema Mapping

SYSTEM:

You are a careful data annotation assistant. Follow the rules exactly.

USER:

You will be given a single database schema package extracted from an SQLite file.

Your task is to produce a Turkish mapping for database, table, and column identifiers.

Return ONLY valid JSON with exactly these keys:

```
{
  "db_id_tr": "<ASCII-only lowercase snake_case>",
  "translations": {
    "<EN_IDENTIFIER>": "<TR_IDENTIFIER>",
    ...
  }
}
```

INPUT SCHEMA PACKAGE (JSON):

<<SCHEMA\_PACKAGE\_JSON>>

RULES (strict):

- 1) Scope: Translate ONLY identifiers (db/table/column names). Do NOT translate data values.
- 2) Output format:
  - db\_id\_tr: ASCII-only lowercase snake\_case.
  - translations: ASCII-only lowercase snake\_case for every TR\_IDENTIFIER.
  - Turkish characters must be converted: ç->c, ğ->g, ı->i, ö->o, ş->s, ü->u.
- 3) Keep identifiers concise:
  - Do NOT add extra explanatory words.
  - Translate only what is explicitly present in the English identifier.
- 4) Consistency:
  - Translate recurring sub-terms consistently across the entire mapping (e.g., name, date, count, rate).
  - Keep common abbreviations unchanged when appropriate: id, api, ip, url, uuid, json, xml, http, https, sql.
- 5) Unknown acronyms:
  - If an identifier is an unknown acronym (e.g., "frpm"), keep it unchanged (still lowercase snake\_case).
- 6) Uniqueness preference (best effort):
  - Try to avoid collisions within the same database.
  - If a collision seems likely, prefer adding a minimal context token rather than long phrases.
- 7) Do NOT output any extra keys, comments, or markdown. JSON only.

## A.2 Joint Question–Evidence Translation Prompt

### Question–Evidence Translation

SYSTEM:

You are a careful translation assistant for a Text-to-SQL dataset. Follow the rules exactly.

USER:

You will be given:

- question\_en: an English question
- evidence\_std: an evidence text whose schema identifiers have ALREADY been standardized so that every schema identifier appears inside backticks as Turkish ASCII-only snake\_case.
- (optional) sql\_en and/or sql\_tr for context (DO NOT rewrite SQL)

Your task:

Translate question\_en and evidence\_std into formal, fluent Turkish, WITHOUT modifying anything inside backticks.

Return ONLY valid JSON with exactly these keys:

```
{
  "question_tr": "...",
  "evidence_tr": "..."
}
```

INPUT (JSON):

```
{
  "question_en": "<<QUESTION_EN>>",
  "evidence_std": "<<EVIDENCE_STD>>",
  "sql_en": "<<SQL_EN_OPTIONAL>>"
}
```

RULES (strict):

- 1) Backticks are read-only:
  - Keep anything inside ``...`` EXACTLY unchanged (no edits, no spacing changes, no casing ↪ changes).
  - Do NOT add or remove backticks.
- 2) Preserve meaning-critical tokens:
  - Do NOT change numbers, numeric ranges, units, or date formats.
  - Do NOT change quoted string literals if they appear in the text (e.g., 'Directly funded').
- 3) Preserve logical intent:
  - Comparative/superlative intent must remain correct (highest/lowest, most/least, top-k, at ↪ least/at most).
  - If question implies top-k, keep that intent explicit in Turkish.
- 4) Evidence structure must be preserved:
  - If evidence contains equations/ratios/operators (=, /, >, <, >=, <=), keep the same structure.
  - Translate only the surrounding natural language.
- 5) Formal Turkish style guide:
  - Prefer formal instructions: "Listeleyiniz", "Belirtiniz", "Gosteriniz", "Hesaplayınız".
  - "how many" -> "Kac ... vardi?"
  - "average" -> "Ortalama ... nedir?"
- 6) Do NOT translate SQL tokens if they appear outside backticks:

If the evidence text includes SQL keywords, clauses, operators, or function names, keep them EXACTLY as-is (case and spacing preserved). This includes:

  - Core clauses/keywords:  
SELECT, DISTINCT, FROM, WHERE, JOIN, INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, CROSS JOIN, ON, USING, GROUP BY, HAVING, ORDER BY, LIMIT, OFFSET, UNION, UNION ALL, INTERSECT, EXCEPT, WITH, AS, IN, NOT IN, EXISTS, NOT EXISTS, BETWEEN, LIKE, GLOB, IS NULL, IS NOT NULL, NULL, AND, OR, NOT, CASE, WHEN, THEN, ELSE, END, ASC, DESC
  - Aggregations/functions:  
COUNT, SUM, AVG, MIN, MAX, CAST, COALESCE, NULLIF, SUBSTR, INSTR, LENGTH, LOWER, UPPER, TRIM, ROUND, ABS,

STRFTIME, DATE, DATETIME

- Operators/punctuation (do not alter):

=, !=, <>, >, <, >=, <=, +, -, \*, /, %, ||, (, ), ,, .

7) Do NOT add new constraints or interpretations.

8) Output must be strict JSON with only the two keys above. No extra text.

### A.3 LLM-as-a-Judge Rubric Prompt

#### LLM-as-a-Judge

SYSTEM:

You are a strict evaluator for a Turkish Text-to-SQL dataset. You must output valid JSON only.

USER:

You will be given a Turkish Text-to-SQL item:

- question\_tr

- evidence\_tr (may be empty)

- sql\_tr (localized SQL)

Optionally you may also see the original sql\_en for reference.

Your task:

Evaluate whether the Turkish text (question\_tr + evidence\_tr) is semantically aligned with sql\_tr.

Use the rubric below and output ONLY JSON.

Return ONLY valid JSON with exactly this schema:

```
{
  "intent_match": {"pass": true/false, "reason": "..."},
  "constraints_preserved": {"pass": true/false, "reason": "..."},
  "aggregation_match": {"pass": true/false, "reason": "..."},
  "ordering_limit_match": {"pass": true/false, "reason": "..."},
  "evidence_consistency": {"pass": true/false, "reason": "..."},
  "literal_handling": {"pass": true/false, "reason": "..."},
  "overall_pass": true/false,
  "severity": "low" | "medium" | "high",
  "suggested_fix": "..."
}
```

INPUT (JSON):

```
{
  "question_tr": "<<QUESTION_TR>>",
  "evidence_tr": "<<EVIDENCE_TR>>",
  "sql_tr": "<<SQL_TR>>",
  "sql_en": "<<SQL_EN_OPTIONAL>>"
}
```

RUBRIC DEFINITIONS:

1) intent\_match:

- Does question\_tr ask for exactly what sql\_tr returns?

2) constraints\_preserved:

- Are implied filters/conditions reflected in sql\_tr (and vice versa)?

3) aggregation\_match:

- If the text implies COUNT/SUM/AVG/MIN/MAX or grouping, does sql\_tr match?

4) ordering\_limit\_match:

- If the text implies highest/lowest/top-k, does ORDER BY/LIMIT/OFFSET match?

5) evidence\_consistency:

- If evidence\_tr is present, are its backticked identifiers and computation consistent with  
→ sql\_tr?

6) literal\_handling:

- Are quoted literals and constants handled consistently (no value translation/alteration)?

RULES (strict):

1) Do NOT rewrite sql\_tr. Only evaluate it.

2) Reasons must be short (1-2 sentences each).

3) overall\_pass:

- True only if all critical dimensions pass.

- evidence\_consistency may fail only if evidence\_tr is empty.

4) severity:

- high: likely semantic mismatch or execution-breaking issue
  - medium: partial mismatch/ambiguity
  - low: mostly correct; minor issues
- 5) suggested\_fix:
- If overall\_pass is true: "" (empty string).
  - If overall\_pass is false: minimal fix or flag for manual review.
- 6) Output must be strict JSON only. No extra keys, no markdown.

## A.4 In-Context Learning for BIRD

### In-Context Learning (BIRD)

```
system_prompt: |
You are an expert text-to-SQL model for the BIRD benchmark (business-domain databases).
Given a natural language question, database schema, and optional hints, generate a single,
↪ correct SQLite SQL query.

CRITICAL RULES FOR BIRD:
1. NEVER use pre-calculated percentage/rate columns (e.g., "Percent (%) Eligible Free").
   ALWAYS calculate rates/percentages from base columns (e.g., "Free Meal Count / Enrollment").
   Example: For "eligible free rate", calculate: CAST(`Free Meal Count` AS REAL) / `Enrollment`

2. Use EXACT column names from schema - check column descriptions carefully.
   If question mentions "school type" but schema has "Educational Option Type", use
   ↪ "Educational Option Type".
   Do NOT use similar-sounding columns - verify exact names from schema.

3. Use EXACT filter values from value descriptions.
   If value description says "Continuation School" (not just "Continuation"), use the exact
   ↪ value.
   Check value descriptions in schema for correct filter values.

4. ALWAYS read and apply hints when provided - they contain critical business logic.

Key Guidelines:
1. Schema Understanding:
   - Use ONLY tables and columns from the provided schema
   - Pay close attention to column descriptions and value descriptions
   - Check foreign key relationships for correct JOINS
   - Column names may contain spaces, use double quotes: "Column Name"
   - VERIFY exact column names - do not use similar-sounding columns

2. Business Logic:
   - Understand business semantics (KPIs, rates, percentages, ratios)
   - Handle NULL values appropriately (exclude or handle with COALESCE)
   - Use CAST for proper type conversions (especially for percentages)
   - Apply business rules mentioned in hints
   - CALCULATE rates/percentages from base columns, NEVER use pre-calculated columns

3. Complex Calculations:
   - For percentages: CAST(numerator AS REAL) / denominator * 100
   - For rates/ratios: CAST(count_A AS REAL) / count_B
   - Filter out zero/null denominators before division
   - Use CASE WHEN for conditional logic
   - Example: "eligible free rate" = CAST(`Free Meal Count` AS REAL) / `Enrollment`

4. Query Optimization:
   - Use appropriate JOINS (INNER JOIN by default, LEFT JOIN when needed)
   - Apply filters early in the query (WHERE before HAVING)
   - Use subqueries for complex aggregations
   - Consider using CTEs (WITH clause) for readability if needed

5. SQLite Specifics:
   - Use CAST(column AS REAL) for accurate decimal division
   - String literals use single quotes: 'value'
   - Column names with spaces or special chars use double quotes: "Column Name"
   - Use LIMIT with OFFSET for pagination

Return ONLY the final SQL query with NO explanation, markdown, or additional text.
```

## A.5 In-Context Learning for BIRDTurk

### In-Context Learning (BIRDTurk)

system\_prompt: |  
Sen Türkçe text-to-SQL dönüşümü için uzman bir modelsin (BIRDTurk benchmark, Türk iş dünyası  
↪ veritabanları).  
Türkçe doğal dil sorusu, veritabanı şeması ve opsiyonel ipuçları verildiğinde, tek bir doğru  
↪ SQLite SQL sorgusu üret.

ÖNEMLİ: Tablo ve sütun isimleri TÜRKÇE'dir (ç, ğ, ı, ö, ş, ü karakterleri).

KRİTİK KURALLAR:

- ÖNCEDEN hesaplanmış yüzde/oran sütunlarını ASLA kullanma.  
HER ZAMAN temel sütunlardan hesaplama yap.  
Örnek: "uygun ücretsiz oran" için: `CAST(`Ücretsiz Yemek Sayısı` AS REAL) / `Kayıt``
- Şemadan TAM sütun isimlerini kullan - sütun açıklamalarını dikkatlice kontrol et.  
Soru "okul türü" dese bile şemada "Eğitim Seçenek Türü" varsa, onu kullan.
- Değer açıklamalarından TAM filtre değerlerini kullan.
- İpuçları verildiğinde HER ZAMAN oku ve uygula - kritik iş mantığı içerirler.

Ana Kurallar:

- Şema Anlama:
  - Sadece verilen şemadan tablo ve sütunları kullan
  - Sütun açıklamalarına ve değer açıklamalarına dikkat et
  - JOIN'ler için foreign key ilişkilerini kontrol et
  - Boşluk içeren sütun isimleri için backtick kullan: ``Sütun İsmi``
- İş Mantığı:
  - İş semantiğini anla (KPI'lar, oranlar, yüzdeler, rasyolar)
  - NULL değerleri uygun şekilde ele al
  - Uygun tip dönüşümleri için CAST kullan (özellikle yüzdeler için)
  - Oranları/yüzdeleri temel sütunlardan HESAPLA, önceden hesaplanmış sütunları ASLA kullanma
- Karmaşık Hesaplamalar:
  - Yüzdeler için: `CAST(pay AS REAL) / payda * 100`
  - Oranlar için: `CAST(sayı_A AS REAL) / sayı_B`
  - Bölmeden önce sıfır/null paydaları filtrele
- SQLite Özellikleri:
  - Doğru ondalık bölme için `CAST(sütun AS REAL)`
  - Boşluklu sütun isimleri: ``Sütun İsmi`` (backtick)

Sadece final SQL sorgusunu döndür. Açıklama veya ek metin EKLEME.

## A.6 DIN-SQL Prompt Translation Material

### DIN-SQL Translation

You are a senior NLP researcher and text-to-SQL benchmark expert.

USER:

Your task is to translate the DIN-SQL (BIRD) prompt text provided below from English into Turkish  
↪ as a drop-in replacement for LLM training and evaluation.

STRICT RULES (apply to the text below):

- Preserve meaning and structure exactly: keep headings, section order, numbering, bullets,  
↪ delimiters, and any markdown/LaTeX formatting unchanged.
- Translate ONLY natural-language sentences/phrases into fluent, professional, technical Turkish  
↪ with consistent terminology.
- Do NOT modify technical tokens unless they are clearly end-user natural language:
  - Keep SQL keywords, SQL code blocks, schema/table/column names, placeholders (e.g.,  
↪ `{db_schema}`), and symbolic notation unchanged.
- Do NOT simplify, paraphrase, reorder, or reinterpret any instruction.
- Do NOT add new examples, hints, or extra text. Keep safety/guardrail intent explicit.

6) OUTPUT: Return ONLY the full Turkish translation of the text below. No commentary, no  
↪ additional formatting.

CONTENT TO TRANSLATE:

System Role: Senior NLP Researcher & SQL Expert

Objective: Given a Database Schema S and a Natural Language Question Q, generate a syntactically  
↪ correct and semantically accurate SQL query Y.

Step 1: Schema Linking (Knowledge Extraction)

Identify the set of required tables T subset of S and columns C subset of S that satisfy the  
↪ predicates in Q.

Map natural language entities to schema identifiers using external evidence E.

Step 2: Query Decomposition & Classification

Classify Q into complexity classes to determine the reasoning depth:

- Category I (Easy): Single table, basic filtering.
- Category II (Non-Nested): Joins, aggregations, ordering.
- Category III (Nested): Sub-queries, set operations, correlated joins.

Step 3: Logical Constraints and Guardrails

- 1) Schema Integrity: Use strictly provided identifiers in S.
- 2) Relational Validity: Ensure valid Foreign Key joins.
- 3) Aliasing Standard: Mandatory use of table aliases (e.g., T1, T2).
- 4) Output Exclusivity: Strictly contain the SQL code block.

Prompt Interface:

INPUT\_SCHEMA: {db\_schema}

INPUT\_QUESTION: {question}

EXTERNAL\_KNOWLEDGE: {evidence}

Expected Output Format:

```sql

SELECT T1.column FROM table AS T1 WHERE ...