RUANITTO ROBERTO DOCINI

A TEXT-TO-SQL APPROACH FOR SMALL LANGUAGE MODELS

(*Defense version, compiled at April 6, 2026*)

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação*.

Orientador: Prof. Luiz Eduardo S. Oliveira.

Coorientador: Prof. Eduardo C. de Almeida.

CURITIBA PR

2025

RESUMO

Grandes modelos de linguagem (LLMs) têm feito avanços significativos no desempenho de tradução texto-para-SQL, transformando consultas em linguagem natural em consultas SQL. O benchmark BIRD, um conjunto de dados entre domínios com 12.751 pares pergunta-SQL distribuídos em 95 bancos de dados, é atualmente o benchmark mais desafiador na área. No entanto, seu ranking é dominado por soluções que dependem de LLMs de código fechado, criando uma barreira financeira para pesquisadores que tentam reproduzir resultados. Em contraste, este trabalho concentra-se em alavancar modelos de código aberto, que são amplamente acessíveis à comunidade de pesquisa, ao invés de alternativas pagas. Primeiro, apresentamos uma metodologia de validação, chamada Wrecking SQL, que modifica incrementalmente o esquema de conjuntos de dados substituindo nomes significativos de colunas e tabelas por nomes sem significado, um problema do mundo real facilmente encontrado em sistemas SQL legados. Exploramos o impacto da nomenclatura de esquemas no desempenho de tradução, investigando especificamente como nomes de colunas e tabelas sem significado afetam a acurácia dos LLMs. Descobrimos que nomenclatura ruim degrada significativamente o desempenho do modelo, sendo os nomes de colunas os de maior impacto. Em seguida, o Wrecking SQL utiliza inferência de nomenclatura de esquema para recuperar o significado semântico do esquema, demonstrando que inferir nomes significativos melhora a acurácia de tradução. Construindo sobre essas descobertas, abordamos a persistente lacuna de desempenho entre modelos de linguagem pequenos e grandes investigando como a complexidade do esquema afeta os resultados de tradução. Apresentamos Schema Distillation, um método que emprega aprendizado em contexto para derivar esquemas simplificados antes da tradução texto-para-SQL, como desnormalização de esquema com garantias de preservação semântica. Nossa abordagem requer apenas uma única chamada a um LLM para simplificação de esquema, seguida por inferência em modelos substancialmente menores e implantáveis localmente. Através de avaliação em 14 modelos (0,5B a 8B de parâmetros) nas bases de dados de desenvolvimento do BIRD, demonstramos melhorias estatisticamente significativas para modelos pequenos ($\leq$3B parâmetros): d de Cohen = 0,97, p = 0,025, com 67% apresentando ganhos de desempenho e melhorias relativas médias de 32%. Os menores modelos beneficiam-se mais dramaticamente: llama3.2:1b alcançando 69,9% de melhoria relativa e qwen3:0.6b apresentando ganhos de 49,3%. Nossas contribuições incluem: 1) Analisar o papel da nomenclatura de esquema na tradução texto-para-SQL usando modelos de código aberto, 2) Mostrar os efeitos prejudiciais de nomenclatura ruim no desempenho, especialmente para nomes de colunas, 3) Propor um método para melhorar a acurácia de tradução inferindo nomes de esquema significativos, e 4) Demonstrar que a simplificação de esquema através de destilação permite que modelos de linguagem pequenos alcancem melhorias significativas de desempenho na tradução texto-para-SQL.

Palavras-chave: Text-to-SQL. Small Language Models. Schema Nomenclature.

ABSTRACT

Large language models (LLMs) have made significant strides in text-to-SQL performance, transforming natural language queries into SQL queries. The BIRD benchmark, a cross-domain dataset with 12,751 question-SQL pairs across 95 databases, is currently the most challenging benchmark in the field. However, its leaderboard is dominated by solutions that rely on closed-source LLMs, creating a financial barrier for researchers attempting to reproduce results. In contrast, this work focuses on leveraging open-source models, which are widely accessible to the research community, rather than paid alternatives. We first present a validation methodology, called Wrecking SQL, that incrementally modifies the schema of datasets by replacing meaningful column and table names with meaningless ones, a real-world problem easily found in legacy SQL systems. We explore the impact of schema nomenclature on translation performance, specifically investigating how meaningless column and table names affect LLM accuracy. We find that poor naming significantly degrades model performance, with column names having the greatest impact. Then, Wrecking SQL uses schema naming inference to recuperate the semantic meaning of the schema, demonstrating that inferring meaningful names improves translation accuracy. Building upon these findings, we address the persistent performance gap between small and large language models by investigating how schema complexity affects translation results. We present Schema Distillation, a method that employs in-context learning to derive simplified schemas before text-to-SQL translation, like schema denormalization with semantic lossless guarantees. Our approach requires only a single call to a LLM for schema simplification, followed by inference on substantially smaller, locally-deployable models. Through evaluation across 14 models (0.5B to 8B parameters) on BIRD dev databases, we demonstrate statistically significant improvements for small models (≤3B parameters): Cohen's d = 0.97, p = 0.025, with 67% showing performance gains and mean relative improvements of 32%. The smallest models benefit most dramatically: llama3.2:1b achieving 69.9% relative improvement and qwen3:0.6b showing 49.3% gains. Our contributions include: 1) Analyzing the role of schema nomenclature in text-to-SQL translation using open-source models, 2) Showing the detrimental effects of poor naming on performance, especially for column names, 3) Proposing a method to improve translation accuracy by inferring meaningful schema names, and 4) Demonstrating that schema simplification through distillation enables small language models to achieve significant performance improvements in text-to-SQL translation.

Keywords: Text-to-SQL. Small Language Models. Schema Nomenclature.

LIST OF FIGURES

2.1 Transformer encoder-decoder architecture showing the flow from input embeddings through multiple encoder and decoder layers (adapted from Vaswani et al. (2017)). 17

2.2 Attention mechanism showing how query, key, and value matrices interact to compute attention weights. 19

2.3 Cognitive Load Theory showing how different types of cognitive load interact with working memory capacity (based on Sweller (1988)). 21

2.4 Knowledge Distillation process showing how a teacher model's knowledge is transferred to a smaller student model. 22

4.1 Wrecking SQL Pipeline 35

4.2 Distillation Pipeline 41

LIST OF TABLES

5.1 Performance comparison 47
5.2 Performance comparison of models across different difficulty levels of query difficulty (Simple, Moderate, and Challenging) 48
5.3 Performance of the models across datasets with columns modifications 48
5.4 Performance comparison 51
5.5 Functional Dependencies: Original vs. Distilled Schema 52
5.6 Performance comparison 53
5.7 Top 5 BIRD Leaderboard Solutions 55

LIST OF ACRONYMS

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BIRD	BIg Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation
D	Default (schema variant)
GPT	Generative Pre-trained Transformer
INFC	Inference Complete
INFS	Inference Simple
KD	Knowledge Distillation
LLM	Large Language Models
LoRA	Low-Rank Adaptation
FFN	Feed-Forward Network
MC	Meaningless Columns (schema variant)
MCT	Meaningless Columns and Tables (schema variant)
MT	Meaningless Tables (schema variant)
NLP	Natural Language Processing
RNN	Recurrent Neural Network
SLM	Small Language Models
SOTA	State-of-the-art
SQL	Structured Query Language

CONTENTS

1	**INTRODUCTION**	**10**
1.1	CHALLENGES	11
1.2	OBJECTIVES	12
1.3	CONTRIBUTIONS	12
1.4	ORGANIZATION OF THE THESIS	13
2	**THEORETICAL BACKGROUND**	**14**
2.1	TEXT-TO-SQL TASK	14
2.2	FOUNDATION MODELS AND LARGE LANGUAGE MODELS	15
2.2.1	Evolution of Language Modeling	15
2.2.2	Transformer Architecture	16
2.2.3	Emergent Behaviors in Large Language Models	18
2.3	PROMPT ENGINEERING	19
2.4	COGNITIVE LOAD THEORY AND MODEL CAPACITY	20
2.4.1	Cognitive Load in Human Cognition	20
2.4.2	Cognitive Load in Language Models	20
2.5	KNOWLEDGE DISTILLATION	21
2.6	DATABASE COMPLEXITY AND DENORMALIZATION	23
2.7	EVALUATION METRICS FOR TEXT-TO-SQL	24
2.8	SUMMARY	25
3	**RELATED WORKS**	**26**
3.1	SCHEMA INFERENCE AND MANIPULATION	26
3.1.1	Machine Learning Based Schema Inference	26
3.1.2	Schema Quality and Naming Patterns	26
3.1.3	Semantic Labeling and Column Name Inference	27
3.2	SMALL LANGUAGE MODELS FOR TEXT-TO-SQL	27
3.2.1	Model Compression and Distillation	27
3.2.2	Prompt Engineering for Resource-Constrained Models	28
3.2.3	Benchmark Performance and Accessibility	28
3.3	DATABASE COMPLEXITY AND QUERY SIMPLIFICATION	28
3.3.1	Schema Complexity Metrics	29
3.3.2	Denormalization Strategies	29
3.4	COGNITIVE LOAD IN AI SYSTEMS	29
3.5	POSITIONING OUR CONTRIBUTIONS	30

4 METHODOLOGY 31
4.1 DATASET AND EVALUATION METRICS 31
4.1.1 Dataset 31
4.1.2 Evaluation metrics 32
4.2 WRECKING SQL - INCREMENTAL VALIDATION METHODOLOGY 32
4.2.1 Model selection 32
4.2.2 Dataset Versions 33
4.2.3 Schema Mapping Implementation 34
4.2.4 Database Modification Process 35
4.2.5 SCHEMA NAMING INFERENCE 36
4.2.6 PROMPT CONSTRUCTION FOR TRANSLATION 38
4.3 SCHEMA DISTILLATION: COMPLEXITY REDUCTION 40
4.3.1 Model selection 40
4.3.2 From Theory to Implementation 40
4.3.3 Distillation Pipeline 41
4.3.4 Distillation Prompt 43
4.3.5 Text-to-SQL Translation Prompt 45
4.4 SUMMARY 46
5 EXPERIMENTS 47
5.1 SCHEMA NOMENCLATURE AFFECT ON TEXT-TO-SQL TRANSLATION . 47
5.2 INFERENCE IMPROVEMENT IN TRANSLATION FOR LEGACY DATABASES 48
5.3 DISTILLATION RESULTS AND ANALYSIS 51
5.3.1 Small Models Show Significant Improvement. 51
5.3.2 Inverse Correlation with Model Size. 52
5.3.3 Understanding Semantic Loss in Schema Distillation 52
5.3.4 Semantic Loss Explains Divergence. 53
5.4 COMPARISON WITH LEADING SOLUTIONS 54
6 CONCLUSION 56
REFERENCES 58

1 INTRODUCTION

Text-to-SQL translation is an important and widely used task in Natural Language Processing (NLP). It consists of transforming natural language questions into executable SQL queries. Among its most popular applications, we can mention database querying for non-technical users, business intelligence systems, and automated report generation. To obtain models capable of performing the aforementioned tasks, the commonly used methodology relies on Large Language Models (LLMs). These models have significantly advanced performance in this domain.

The BIRD benchmark (Li et al., 2024), a cross-domain dataset with 12,751 question-SQL pairs and 95 databases across 37 domains, is currently the most challenging and comprehensive benchmark in the field. The top solutions achieve approximately 77% execution accuracy, which compares the outputs of generated queries with ground-truth queries. Leading approaches include AskData + GPT-4o (77.1%), Contextual-SQL (75.6%), and XiYan-SQL (75.6%) (Gao et al., 2025).

In general, due to the complexity of real-world database schemas and the variability found in natural language expressions, flagship models with billions of parameters are necessary to achieve competitive performance. This requirement for large models has a propensity to create barriers for researchers and practitioners. Projects become difficult or even unfeasible due to the computational costs and the need for expensive API access to closed-source models. Most of the leaderboard is dominated by solutions that rely on closed-source LLMs, creating financial and accessibility barriers for the broader research community.

Some suggestions to work around this problem can be found in recently developed techniques in NLP and database theory. These include schema simplification, knowledge distillation, and prompt engineering. Schema manipulation has been shown to be consistent in improving the performance of several database tasks while decreasing complexity. As for knowledge distillation (Hinton et al., 2015), the technique enables smaller models to learn from larger ones, achieving comparable performance with reduced computational requirements. Lastly, in-context learning with carefully designed prompts has demonstrated that it can guide models to better performance without fine-tuning.

These strategies have shown promising results. They are especially interesting for the problem proposed in this study, considering the existence of large open-source models and the need to make text-to-SQL accessible to practitioners with limited resources.

In this research, we intend to investigate how database schema characteristics affect text-to-SQL translation performance, particularly for smaller models. We focus on two main aspects: schema nomenclature (the naming of tables and columns) and schema complexity (the relational structure with joins and foreign keys). The idea is to investigate and compare several techniques to optimize text-to-SQL translation under resource constraints. We also

aim to establish foundations for democratizing access to this technology by presenting schema distillation.

1.1 CHALLENGES

Despite promising results on benchmarks like BIRD, real-world database deployment presents challenges not adequately represented in current evaluation datasets. One significant issue is the prevalence of legacy systems with poor naming conventions. This is common in banking and large mission-critical systems. While trustworthy, these systems lack modularity, scalability, and flexibility due to their monolithic architecture (Hariharan and Reeshma, 2015). Such systems often use meaningless column and table names like "column1" or "Filler0001" which is the problem that motivated our initial investigation.

A recurrent problem found in such systems is the degradation of LLM performance when schema nomenclature provides no semantic guidance. To study this issue in a controlled way, we adopt an incremental validation protocol (Wrecking SQL) that systematically replaces meaningful names with meaningless ones and measures the impact on execution accuracy. For example, an incremental test can be structured as follows:

i) [table: schools, column: classes]

ii) [table: schools, column: column1]

iii) [table: table1, column: classes]

iv) [table: table1, column: column1]

v) [table: schools, column: taught_subjects]

vi) [table: schools, column: school_classes]

Our experiments revealed that poor naming significantly degrades model performance. Execution accuracy drops by 60-80% when column names are replaced with meaningless identifiers. This finding pointed to something bigger: if naming matters so much, then schema complexity itself might be overwhelming smaller models in ways beyond just nomenclature.

In addition, we identified that even with perfect naming, complex relational schemas impose cognitive burdens on smaller models. When a query requires reasoning across multiple table joins and foreign key relationships, resource-constrained models struggle to track all the necessary information. This understanding led us to investigate Knowledge Distillation concepts applied not to models themselves, but to the database schemas they process.

Finally, we point out that achieving performance improvements for smaller models while preserving semantic correctness presents a fundamental trade-off. By simplifying schemas through denormalization (Shin and Sanders, 2006), we eliminate enforceable functional dependencies and

normal form guarantees. This creates a tension: smaller models benefit from reduced complexity, while larger models may lose exploitable structural information. Understanding, quantifying and exploiting this trade-off to enhance small models performance is central to our research.

1.2 OBJECTIVES

The main objective of this research is the evaluation of schema manipulation techniques on text-to-SQL translation. Specifically, we investigate how schema characteristics affect model performance. We also test whether simplification strategies can enable smaller models to achieve competitive results while using only open-source models accessible to the research community.

For schema nomenclature, we quantify how the loss of semantic cues in table and column names affects execution accuracy, and we evaluate whether name inference from table content can recover part of this loss.

For schema complexity, we investigate whether simplifying relational structure (e.g., reducing join requirements) can enable smaller open-source models to perform better in text-to-SQL translation, while characterizing the semantic trade-offs introduced by simplification.

More specifically, we demonstrate that, by carefully applying schema simplification principles based on cognitive load theory (Sweller, 1988) and denormalization strategies (Shin and Sanders, 2006), it is possible to achieve statistically significant improvements for models with 3B parameters or fewer. We emphasize that while absolute performance remains below flagship solutions (approximately one-quarter to one-third of state-of-the-art accuracy), this approach offers significant economic advantages. It also establishes a foundation for future research directions.

This work addresses three key research questions that build upon each other:

1. How does schema nomenclature affect text-to-SQL translation with LLMs?
2. Can schema naming inference improve text-to-SQL translation in legacy databases?
3. Does schema distillation bring meaningful improvements for smaller models ($\leq$3B parameters) in text-to-SQL translation, and how does this effect vary by model size?

1.3 CONTRIBUTIONS

The main contributions of this research are the proposal and evaluation of schema manipulation techniques that improve text-to-SQL translation for open-source models in resource-constrained scenarios:

- **Published contribution (schema nomenclature).** We introduced Wrecking SQL, an incremental validation methodology to isolate and quantify the impact of schema nomenclature on text-to-SQL translation (Docini et al., 2025).

  - **Empirical findings on naming.** Through systematic experimentation with open-source models, we show that column naming is the most critical schema characteristic affecting translation accuracy; execution accuracy drops by 60–80% when column names are replaced with meaningless identifiers.
  - **Name inference for legacy databases.** We evaluate schema naming inference techniques to partially recover performance in poorly named schemas, comparing simple and complete prompts, with typical gains of 2–4 percentage points.

- **Submitted contribution (schema complexity).** We propose Schema Distillation, a schema simplification pipeline that reduces relational complexity (e.g., join requirements) to better match the capabilities of small language models.
  - **Model-size-dependent effects of simplification.** We show that Schema Distillation yields statistically significant improvements for small models (≤3B parameters): Cohen's d = 0.97, p = 0.025, with 67% showing performance gains and mean relative improvements of 32%.
  - **Trade-off analysis.** We analyze the semantic costs of schema simplification, quantifying the elimination of enforceable functional dependencies and explaining why this can benefit small models while disadvantaging larger ones.

1.4 ORGANIZATION OF THE THESIS

This dissertation is organized as follows: Chapter 2 presents the theoretical concepts that underlie our research. Chapter 3 provides a comprehensive overview of previous studies that addressed issues related to our work. Chapter 4 presents our methodology to build Wrecking SQL incremental validation methodology, detailing our approach to investigating schema nomenclature impact and creating schema distillation Chapter 5 presents the experiments and the complete results from both Wrecking SQL and Schema Distillation experiments. Finally, Chapter 6 discusses our conclusions, implications, limitations, and proposes future research directions.

2 THEORETICAL BACKGROUND

This chapter addresses the theoretical foundations used in this research. We approach the fundamental concepts underlying text-to-SQL translation, large language model architecture, and the cognitive principles that inform our methodology. The theoretical elements associated with each stage are detailed below, providing the conceptual framework necessary to understand our contributions.

2.1 TEXT-TO-SQL TASK

The text-to-SQL problem is defined as follows: given a natural language question and a relational database with a specific schema, the model produces a query in Structured Query Language (SQL) which answers the question posed and returns the requested data. We formulate it in Equation 2.1,

$$Y = f(Q, S\{T, C\}) \tag{2.1}$$

where Y is the valid SQL query automatically generated, Q is the natural language question, S is schema composed by tables (T) and by their respective columns (C), and f is model that takes as input Q and S, and generates the SQL query Y.

This task presents challenges inherent to both language and SQL code complexity. Language-related problems include dealing with lexical, syntactic, semantic, or context-related ambiguity (Allen, 1995). Additionally, systems must handle spelling mistakes and infer information that was omitted but is trivial for humans. Finally, paraphrasing must be considered: when two different statements mean the same thing although written in extremely different ways.

On the SQL side, systems need to deal with strict syntax requirements. The database modeling also matters, as this can lead to errors when accessing columns or tables due to naming conventions.

The first solutions to this task were based on heuristics that required heavy engineering effort. This becomes infeasible when dealing with complex scenarios (Androutsopoulos et al., 1995). Subsequently, the emergence of deep learning shifted the area to sequence-to-sequence solutions. Most approaches trained specific models that learned how to encode and translate natural language and database information into SQL queries (Sutskever et al., 2014).

Following the state-of-the-art in Natural Language Processing (NLP), the best-performing solutions shifted once more. LLMs with in-context learning became the leading technique (Brown et al., 2020). This is the line of work followed by the procedures present in this research, experimenting with cutting-edge models and prompt engineering techniques.

2.2 FOUNDATION MODELS AND LARGE LANGUAGE MODELS

Foundation models are large, general-purpose models trained on massive and diverse datasets, designed to be adapted to many different tasks rather than built for just one specific problem (Bommasani et al., 2021). Unlike traditional models trained for specific tasks, foundation models serve as a base that can be fine-tuned or prompted for various downstream applications.

2.2.1 Evolution of Language Modeling

Language models can be viewed as devices that try to predict how likely a sequence of words is. The evolution of language modeling can be viewed in four sequential stages dating back to the 1990s (Jurafsky and Martin, 2000).

The first stage used statistical language models based upon the Markov assumption. In this context, the Markov assumption means that the probability of the next word is approximated using only a limited window of previous words (an n-gram context), rather than the entire sentence history. Concretely, it assumes that the next-token distribution depends only on the most recent $n - 1$ tokens and ignores earlier history, this is shown in Formula 2.2.

$$P(w_t \mid w_1, \ldots, w_{t-1}) \approx P\big(w_t \mid w_{t-n+1}, \ldots, w_{t-1}\big). \tag{2.2}$$

This assumption makes learning tractable because the model only needs to estimate conditional probabilities for a bounded context (counting occurrences in a corpus). However, it introduces two key limitations. First, **data sparsity**: the number of possible n-grams grows exponentially with n, so many contexts are rarely or never observed, requiring smoothing or backoff techniques. Second, **limited long-range dependencies**: since the context is truncated, n-gram models cannot reliably capture dependencies that span longer distances (e.g., subject–verb agreement across intervening clauses). In practice, these models used the most recent context to predict the next word.

The next stage is marked by neural language models. Recurrent Neural Networks (RNNs) started to be used to predict the possibility of a word sequence (Mikolov et al., 2010). Distributed representation of words as vectors became a significant part of this evolution. This materialized the idea of learning features for words, creating effective representations which impacted many NLP tasks (Mikolov et al., 2013).

In 2017, the seminal article “Attention is All You Need” was published, proposing the transformer architecture based on attention mechanisms (Vaswani et al., 2017). This architecture was adopted and disseminated in the area, becoming the most important advancement that made it possible for the field to reach its current stage.

Subsequently, pre-trained language models emerged. ELMo proposed deep contextualized word representations that modeled complex characteristics like semantics and syntax (Peters

et al., 2018). These representations captured how words could be used in different scenarios. Furthermore, this model was fine-tuned to resolve specific tasks.

Finally, we reach the latest stage: LLMs. These involve scaling enormously both the model size and the corpus on which the model is pre-trained, enhancing the performance of final tasks (Brown et al., 2020).

2.2.2 Transformer Architecture

To understand LLMs and why they are so powerful nowadays, it is necessary to delve into the transformer architecture depicted in the Figure 2.1. This complex algorithm is based on an encoder-decoder structure. The encoder receives an input and generates a sequence of vectors, while the decoder processes it to create a final sequence.

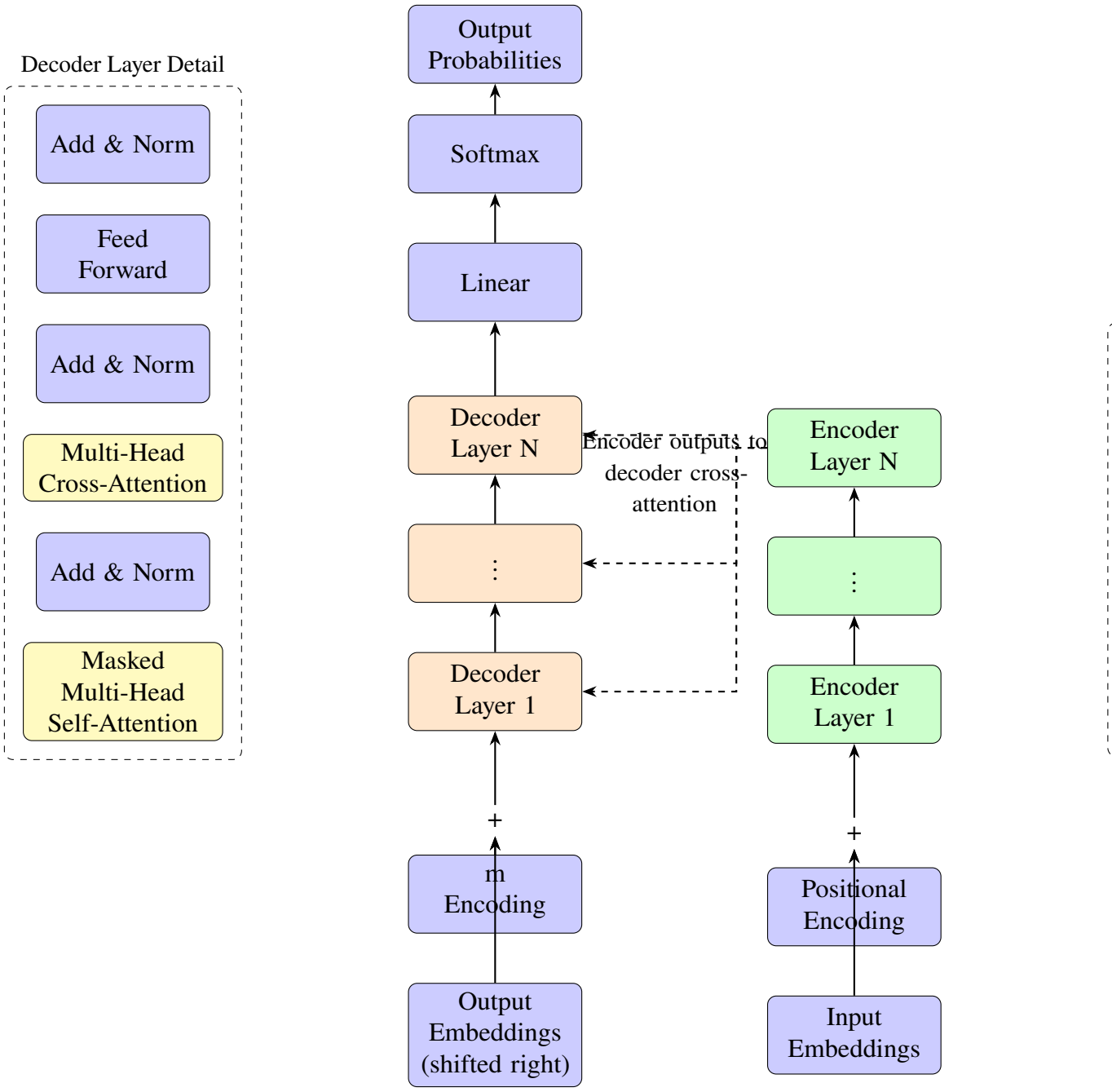

Figure 2.1: Transformer encoder-decoder architecture showing the flow from input embeddings through multiple encoder and decoder layers (adapted from Vaswani et al. (2017)).

In transformers, this structure is stacked N times. The output of one encoder is used as input for the next encoder; the same process occurs for the decoders.

Since self-attention alone is permutation-invariant meaning it does not inherently distinguish different token orders, **positional encoding** is added to the input embeddings so that the model can represent token order. Without it, the encoder would see the same set of tokens regardless of where they appear in the sentence.

Each encoder/decoder layer is not only attention. The **feed-forward network (FFN)** introduces non-linearity and feature transformation at each position. This is important because,

without the FFN, a transformer block would be much closer to a linear model stacking attention-based mixing operations.

Finally, the **Add & Norm** blocks implement residual connections and normalization, helping stability during training and enabling deeper stacks. In the decoder, **masked self-attention** prevents the model from looking ahead to future tokens, while **cross-attention** injects information from the encoder outputs into the decoding process.

2.2.2.1 Self-Attention Mechanism

The major improvement in transformers is the self-attention mechanism. Instead of using recurrent layers, this mechanism solves the problem of losing long-range dependencies. For example, models without it could not grasp the relation between a word at the start of a sentence and one at the end. When processing the last word, all information about the first had been lost, leading to important information being ignored. The attention layer in a transformer network is characterized by the Formula 2.3.

$$\text{Attention} = f(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \tag{2.3}$$

- Q = query: A representation for the word we want to calculate self-attention
- K = key: A representation of each word in the sentence
- V = value: The final representation of all words in the sentence

An input is turned into a vector based on Q, K, and V. These are used to determine how much attention to pay to each word in the sentence with respect to the word being given as input.

The self-attention calculation works as follows. First, we select a specific word in the sentence and construct its representation in the query. Then, we construct the key with the base representation for all words in the sentence. With those two in place, we multiply them. The resulting vector, divided by the dimension of the sentence and plugged into a softmax, gives us the required attention for each word, which is visually represented in Figure 2.2.

This process should be done for each word and summed up to give us final self-attention for the sentence. The process is conducted multiple times in parallel in the many "heads" that the network has. This results in varied vectors that learn different representations and are weighted at the end to unify their knowledge in the final vector.

Moreover, all of this process can be done in parallel. This enables accelerating the learning process. Operations do not need to be handled sequentially, taking only a fraction of the time compared to an RNN processing the same corpus.

2.2.3 Emergent Behaviors in Large Language Models

LLMs have behaviors not present in their predecessors, called emergent behaviors (Wei et al., 2022). These lead to a capacity for solving a wide variety of tasks without any downstream

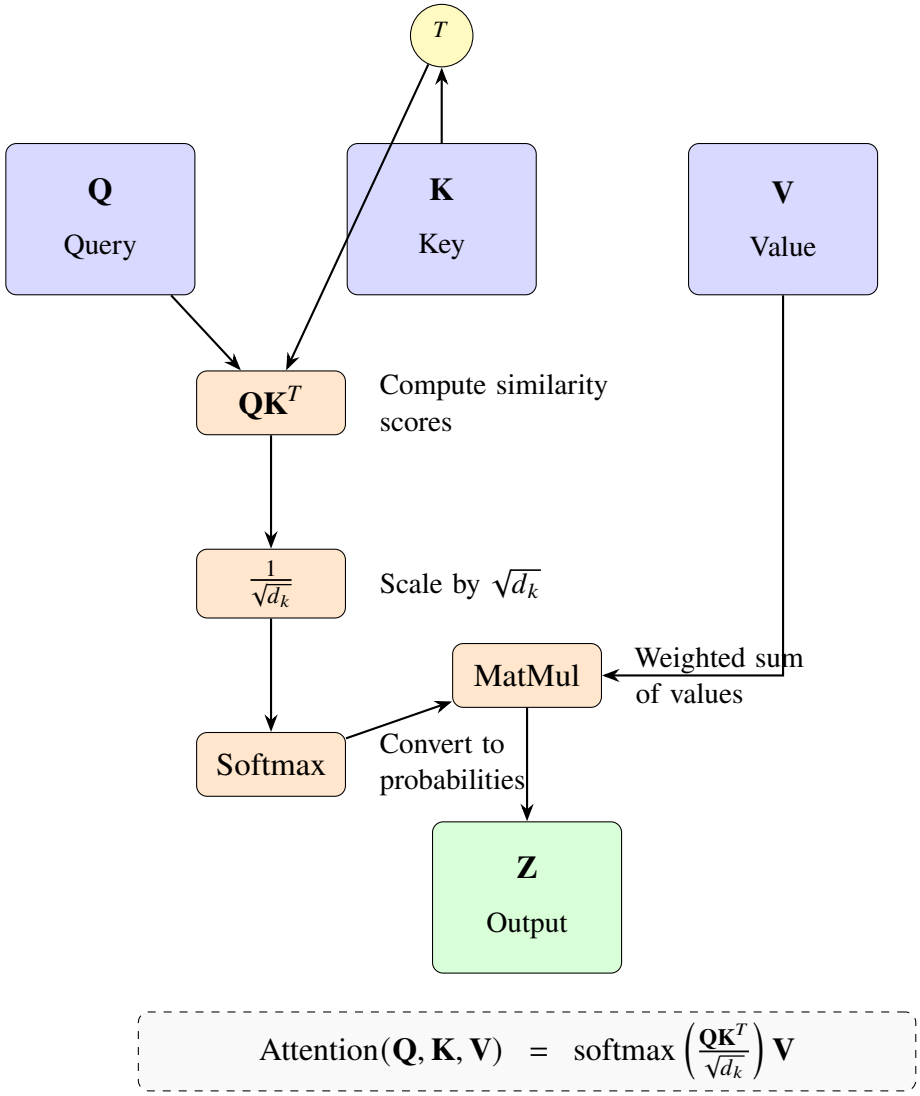

Figure 2.2: Attention mechanism showing how query, key, and value matrices interact to compute attention weights.

fine-tuning for a specific problem. This ability of in-context learning allows that with very few examples, the model presents a fairly good answer to the issue proposed.

2.3 PROMPT ENGINEERING

With the proliferation of LLMs, it has become increasingly common to use them for a large variety of tasks with no fine-tuning (Liu et al., 2023). In this kind of application, after the model is trained on an enormous corpus, we use it in a few-shot manner. We give cues to the LLM alongside the task we want it to solve.

This approach is called prompt engineering because we tweak the arguments we give to the model in order to achieve its best solution at execution time. For example, if we want it to summarize some text, we can pass it two or three sentences that help it understand how we want that done in a specific way.

This structure has proven itself very successful in many cases. Most importantly, it requires much less computational power because there is no fine-tuning involved. This opens up its use for people with modest computational power or sometimes with just access to an inference Application Programming Interface (API).

There are mainly two disseminated techniques that make use of prompt engineering: zero-shot learning and few-shot learning (Brown et al., 2020). Zero-shot learning can be described as using the model to predict a class unseen at training time without supplying any examples. Instead, we give the model a prompt composed of some form of input (a sentence, for example) and a command that describes what we want the model to do with it. In our example, we can say we want it to summarize the sentence.

Few-shot learning, on the other hand, provides a few examples to guide the model. In the case of summarization, we would add to the prompt some phrases and their summaries, then tell the model to replicate the same on an unseen sentence. In the domain of text-to-SQL, most in-context learning techniques use the second approach. This benefits from not having to fine-tune a whole model while still having the ability to supply information for it to perform on a previously unseen task.

2.4 COGNITIVE LOAD THEORY AND MODEL CAPACITY

Understanding why smaller models struggle with complex tasks requires examining cognitive principles. Cognitive Load Theory provides a framework for understanding how information processing capacity limits affect performance.

2.4.1 Cognitive Load in Human Cognition

Cognitive Load Theory, originally proposed by Sweller (1988), postulates that working memory has inherent capacity limits. Miller's Law stated that people can hold about 5 to 9 chunks of information at once (Miller, 1956). When cognitive load exceeds capacity, performance degrades significantly.

The theory distinguishes three types of cognitive load:

- **Intrinsic load**: The inherent difficulty of the material being learned
- **Extraneous load**: Unnecessary cognitive burden imposed by how information is presented
- **Germane load**: Mental effort devoted to processing and schema construction

2.4.2 Cognitive Load in Language Models

Recent research has demonstrated that LLMs exhibit working memory constraints resembling human cognitive load (Liu et al., 2024). Extra information hurts their context processing capacity

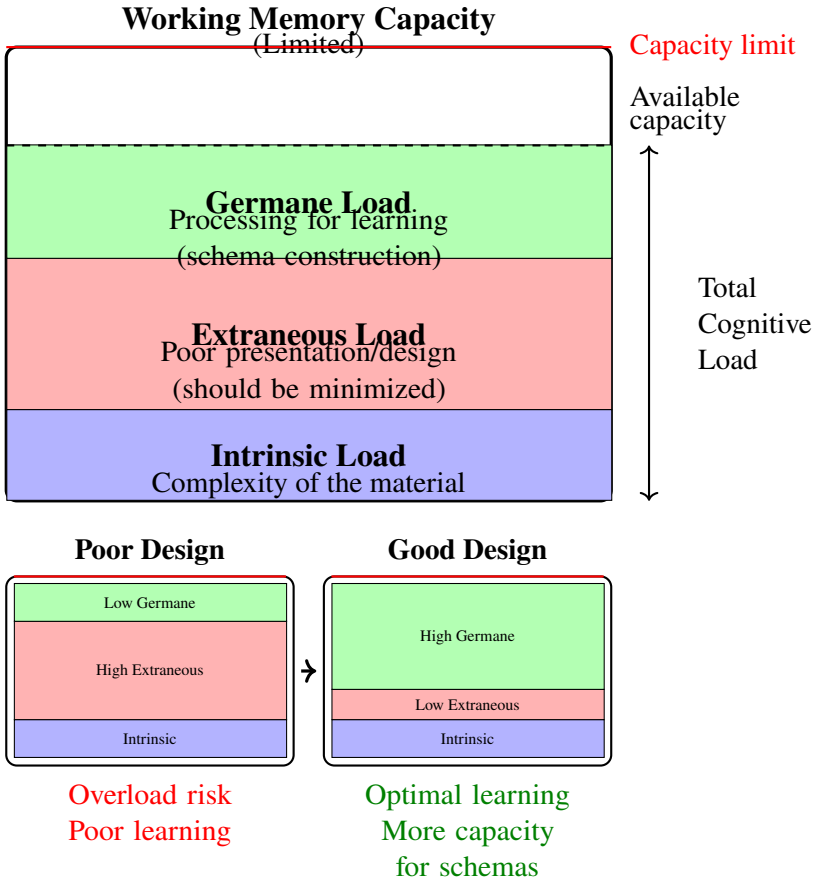

Figure 2.3: Cognitive Load Theory showing how different types of cognitive load interact with working memory capacity (based on Sweller (1988)).

in ways analogous to human working memory limitations. The hypothesis is that complex database schemas impose harmful cognitive load on small models through:

- Multi-table tracking requirements
- Join reasoning across foreign key relationships
- Maintaining multiple entity states simultaneously

Small Language Models (SLMs) with approximately 3B parameters have limited memory to track complex relational structures. Reducing schema complexity by eliminating multi-hop relationships through foreign keys reduces entities to track. This potentially allows SLMs to allocate their limited capacity to SQL syntax generation rather than structural reasoning as we can observe in Figure 2.3, which demonstrates this concept visually.

2.5 KNOWLEDGE DISTILLATION

Knowledge Distillation (KD) is a fundamental technique in machine learning. A smaller "student" model learns to replicate a larger "teacher" model's capabilities (Hinton et al., 2015). This

concept can be more easily understood and exemplified by the diagram in Figure 2.4. The key insight of KD is that compact models benefit from pre-processed information rather than raw complex inputs. The teacher model's learned representations contain "dark knowledge" that can be transferred to the student. This transfer happens through soft targets (probability distributions) rather than hard labels.

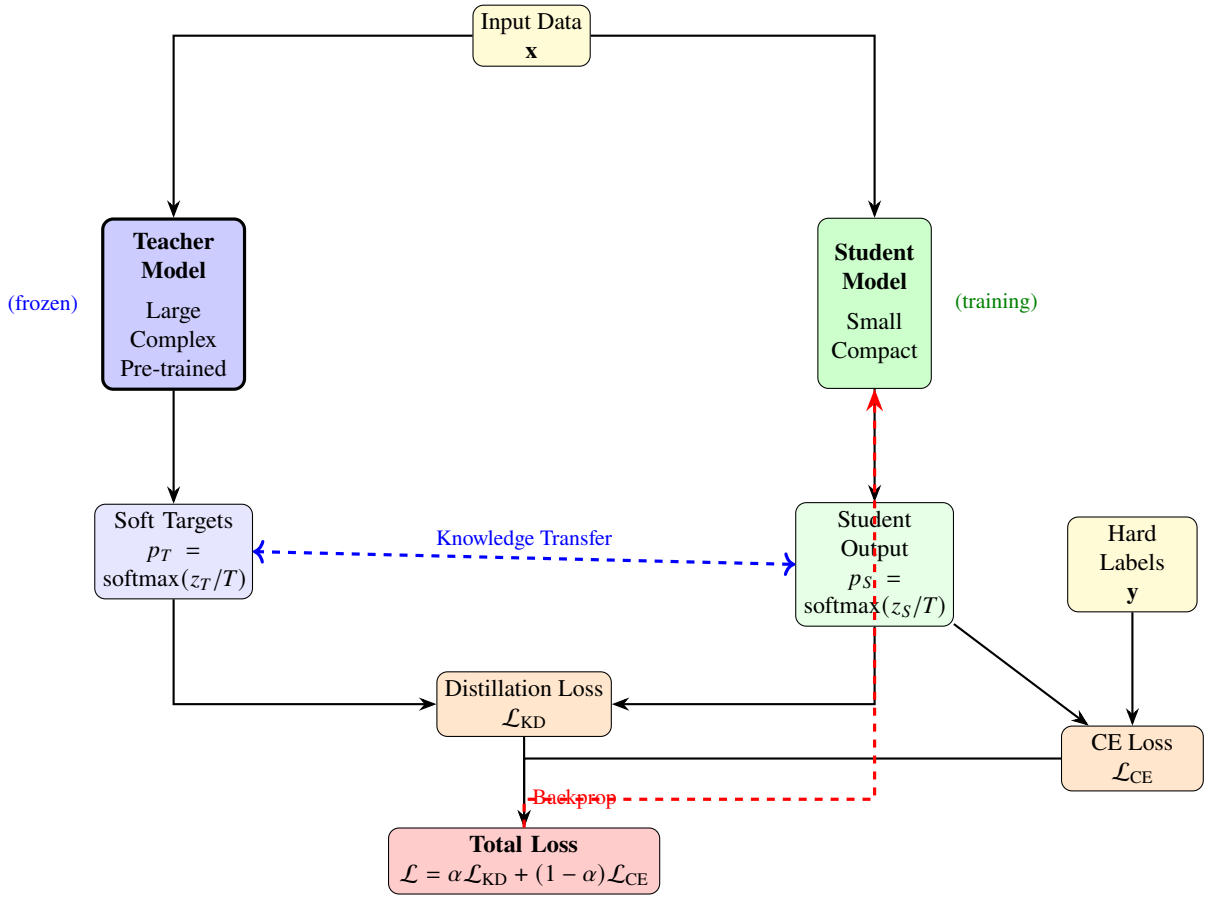

Figure 2.4: Knowledge Distillation process showing how a teacher model's knowledge is transferred to a smaller student model.

Figure 2.4 summarizes this training flow. We can read it from top to bottom: first, the same input **x** is fed to both models. The **teacher** produces a probability distribution over outputs, and we treat this distribution as supervision for the **student**. This is what the figure calls **soft targets** (p_T). A temperature T is applied to make the distribution less confident, so it carries more information than a one-hot label.

Second, the student produces its own distribution (p_S) for the same input. The distillation loss $\mathcal{L}_{\text{KD}}$ compares p_S to p_T and penalizes the student whenever it deviates from the teacher.

Third, the student is also trained with the standard cross-entropy loss $\mathcal{L}_{\text{CE}}$ against the ground-truth label **y**. The final objective is a weighted sum of both losses. During optimization, gradients flow only to the student (the teacher remains frozen), which is why the backpropagation arrow points exclusively to the student.

In the context of LLMs, KD serves as a pivotal methodology for transferring advanced capabilities from proprietary models like GPT-4 to open-source counterparts like LLaMA and Mistral. This enables more efficient smaller models while preserving essential performance (Xu et al., 2024).

Drawing inspiration from this paradigm, distillation concepts can be applied not to the models themselves but to the database schemas they process. If we simplify the database structure before translation, removing relational complexity that burdens small models, they might achieve performance closer to their larger counterparts. This would happen without the computational and financial overhead of larger models.

2.6 DATABASE COMPLEXITY AND DENORMALIZATION

Understanding database complexity requires analyzing both structural characteristics and normalization-related factors. Following the approach proposed by Calero et al. (2001), database complexity is quantified through a set of complementary metrics. We first define the individual components of complexity and subsequently combine them into an overall complexity measure, as expressed in Equation 2.7.

- **Relational Complexity** (C_R): quantifies the number of foreign key relationships that must be traversed, capturing the extent of multi-hop reasoning required across tables. This metric is defined in Equation 2.4.

- **Structural Complexity** (C_S): represents the number of tables that must be tracked within the schema, reflecting the overall structural size of the database. This is formalized in Equation 2.5.

- **Attribute Dispersion** (C_A): measures the average number of join operations per query, indicating how dispersed attributes are across relations. This metric is defined in Equation 2.6.

$$C_R = \sum_{i=1}^{n} |FK_i| \tag{2.4}$$

$$C_S = |T| \tag{2.5}$$

$$C_A = \frac{1}{|Q|} \sum_{q \in Q} |JOIN(q)| \tag{2.6}$$

$$C_{total} = \alpha C_R + \beta C_S + \gamma C_A \tag{2.7}$$

Denormalization is the process of adding redundancy to a normalized database to improve query performance (Shin and Sanders, 2006). While normalization (2NF, 3NF, Boyce-Codd Normal Form - BCNF) aims to minimize redundancy and ensure data integrity through constraints, denormalization strategically introduces redundancy to reduce query complexity.

Key denormalization strategies include:

- **Table merging**: Combining related tables to eliminate joins
- **Column duplication**: Copying frequently accessed columns across tables
- **Pre-computed aggregates**: Storing calculated values

The fundamental trade-off: denormalization sacrifices enforceable semantic constraints (foreign keys, normal forms) for cognitive and computational simplicity.

2.7 EVALUATION METRICS FOR TEXT-TO-SQL

To evaluate text-to-SQL translation performance, we employ execution accuracy. This is the standard metric used in major benchmarks such as BIRD and Spider (Yu et al., 2019). It measures the proportion of queries where the predicted SQL output matches the ground truth query results. We show it in Equation 2.8 and define the indicator function in Equation 2.9.

$$\text{Execution Accuracy} = \frac{\sum_{n=1}^{N} f(Q_n, \hat{Q}_n)}{N} \tag{2.8}$$

$$f(Q_n, \hat{Q}_n) = \begin{cases} 1 & \text{if } Q_n = \hat{Q}_n \\ 0 & \text{if } Q_n \neq \hat{Q}_n \end{cases} \tag{2.9}$$

In both Equations, Q_n represents the results obtained by the nth ground truth SQL, $\hat{Q}_n$ represents the results from the nth SQL predicted by the model, and N is the total number of text-to-SQL pairs in the benchmark.

To validate that observed improvements are statistically significant rather than due to chance, standard statistical tests are employed:

- **Paired t-test**: Compares performance of the same models on original vs. distilled schemas
- **Wilcoxon signed-rank test**: Non-parametric alternative when data doesn't meet normality assumptions
- **Cohen's d**: Effect size measure indicating practical significance beyond statistical significance. The calculation is illustrated in Equation 2.10 where values of 0.2, 0.5, and 0.8 represent small, medium, and large effect sizes respectively (Cohen, 1988).

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s_{pooled}} \tag{2.10}$$

2.8 SUMMARY

This chapter presented the theoretical foundations underlying our research. We covered the text-to-SQL task formulation, the evolution of foundation models culminating in modern transformer architectures, prompt engineering techniques for in-context learning, cognitive load theory as applied to model capacity, knowledge distillation principles, database complexity metrics and denormalization strategies, and evaluation methodologies.

These theoretical elements provide the conceptual framework necessary to understand our methodology presented in the following chapters. The next chapter discusses related work and positions our contributions within the existing literature.

3 RELATED WORKS

This chapter presents the state of the art in text-to-SQL translation, focusing on two distinct perspectives: schema manipulation techniques that address database structure challenges, and methods for enabling smaller models to achieve competitive performance. We examine how previous research has tackled similar problems and position our contributions within this landscape.

3.1 SCHEMA INFERENCE AND MANIPULATION

Database schema understanding has been a persistent challenge in text-to-SQL translation. Multiple approaches have emerged to address different aspects of this problem. The work in this area can be broadly categorized into schema inference from data, schema quality assessment, and schema optimization for query generation.

3.1.1 Machine Learning Based Schema Inference

Numerous community initiatives have targeted schema inference through diverse methodologies. Dani et al. (2024) introduced schema inference as a scalable SQL function. They demonstrated that inference could be integrated directly into database systems rather than requiring external preprocessing. Their approach focused on automated column type detection and relationship discovery. However, it did not address the semantic naming challenges we encounter in legacy systems.

Frozza et al. (2020) adapted NoSQL schema inference techniques to relational databases. They proposed a process that could handle semi-structured data. While their methodology showed promise for discovering implicit structures, it assumed some level of semantic information in column names. This is precisely what we cannot rely on in legacy banking systems with names like “Filler0001.”

In a different direction, Klisura and Rios (2024) explored schema extraction attacks in text-to-SQL systems. They demonstrated that models could inadvertently leak schema information. Their security-focused work highlighted an interesting property: LLMs are quite good at inferring schema structure from data patterns. This capability is leveraged in our naming inference approach.

3.1.2 Schema Quality and Naming Patterns

SNAILS (Luoma and Kumar, 2025) examines schemas through naming pattern “naturalness.” The work introduces metrics to quantify how well column and table names align with common database design conventions. Their work established that naming quality correlates with query

generation success. However, they focused on assessment rather than remediation. We build on this insight by not only validating that poor naming degrades performance but also proposing methods to recover from it.

STCExtract (Cvijetić and Radivojević, 2023) introduced a two-phase framework for table structure reconstruction. The approach first identifies column types and then infers relationships. Their machine learning approach showed that structure could be recovered even from heavily degraded schemas. However, they did not evaluate the impact on downstream text-to-SQL tasks, which is our primary concern.

3.1.3 Semantic Labeling and Column Name Inference

The work most closely related to our naming inference approach is SeLaB (Trabelsi et al., 2021). It implements BERT (Devlin et al., 2019) for semantic labeling, exploiting column values and row contexts for column name inference. SeLaB demonstrated that contextualized embeddings could capture semantic relationships between data patterns and appropriate labels.

However, SeLaB necessitated dedicated task-specific training on labeled datasets of (column_data, column_name) pairs. This requires substantial annotation effort and domain expertise. In Wrecking SQL we build upon these contributions by employing in-context learning with LLMs rather than supervised training. We capitalized on pre-training knowledge to reconstruct BIRD benchmark schemas.

The present work advances this research significantly. While in Wrecking SQL we attempt to generate complete schemas from scratch without prior knowledge of column or table names, with Distillation we implement a fundamentally divergent strategy. We commence with an existing schema and systematically refine it into an enhanced, optimized structure. This achieves superior execution accuracy relative to the original, constituting significant progress beyond our baseline.

3.2 SMALL LANGUAGE MODELS FOR TEXT-TO-SQL

The text-to-SQL landscape has been dominated by large flagship models. The BIRD benchmark leaderboard overwhelmingly features solutions built on GPT-4, Claude, and other closed-source models exceeding 100B parameters. However, a growing body of research addresses the challenge of enabling smaller, more accessible models to achieve competitive performance.

3.2.1 Model Compression and Distillation

Knowledge Distillation has been successfully applied to compress LLMs while preserving performance across various NLP tasks. Hinton et al. (2015) established the foundational principle: smaller models can achieve comparable performance by learning from a teacher model's soft probability distributions rather than hard labels alone.

In the context of text-to-SQL specifically, several approaches have emerged. Some researchers have fine-tuned smaller models on large synthetic datasets generated by flagship models. This effectively distills the larger model's text-to-SQL capabilities. Others have focused on architectural innovations that allow smaller models to process longer contexts more efficiently. This addresses one limitation that particularly affects schema-heavy tasks. Techniques like LoRA (Hu et al., 2021) have enabled parameter-efficient fine-tuning of smaller models.

However, these approaches still require significant training compute for fine-tuning. Our Schema Distillation takes a complementary approach: rather than compressing the model, we simplify the task itself by reducing schema complexity. This allows smaller models to succeed without additional training.

3.2.2 Prompt Engineering for Resource-Constrained Models

The rise of in-context learning has led to extensive research on prompt engineering techniques that maximize smaller model performance (Liu et al., 2023). Few-shot prompting, chain-of-thought reasoning, and structured output formatting have all shown promise in improving text-to-SQL translation without model fine-tuning.

Particularly relevant is research on schema linking prompts. These help models identify relevant tables and columns before query generation. Such techniques reduce the cognitive burden by breaking the task into subtasks. Our approach shares this philosophy but goes further: we restructure the schema itself to eliminate the need for complex linking reasoning altogether.

3.2.3 Benchmark Performance and Accessibility

The BIRD benchmark represents the current state-of-the-art evaluation for text-to-SQL. Its cross-domain dataset of 12,751 question-SQL pairs and 95 databases across 37 domains makes it comprehensive. However, the benchmark's leaderboard tells a concerning story about accessibility. As of our experiments, the top 10 solutions all relied on closed-source flagship models. The associated API costs make reproduction prohibitively expensive for many researchers.

Some recent work has focused specifically on open-source alternatives. Several teams have reported results using Llama, Mistral, and Qwen families of models in their larger versions. Performance typically lags 20-30 percentage points behind flagship solutions. These gaps motivate our research: we seek not to compete directly with flagship models but to identify techniques that maximize what smaller models can achieve within their capacity constraints.

3.3 DATABASE COMPLEXITY AND QUERY SIMPLIFICATION

Beyond the ML and NLP communities, database research has long studied schema complexity and its impact on query performance and usability.

3.3.1 Schema Complexity Metrics

Calero et al. (2001) proposed foundational metrics for database complexity. They quantified relational complexity through foreign key counts, structural complexity through table counts, and attribute dispersion through average joins per query. These metrics have been validated across numerous studies as predictors of query difficulty, maintenance costs, and error rates.

We adapt these metrics to the text-to-SQL context. The hypothesis is that the same complexity dimensions that challenge human developers and database engines also burden language models. This is particularly true for smaller ones with limited capacity.

3.3.2 Denormalization Strategies

Shin and Sanders (2006) extensively studied denormalization techniques for data warehouses. They documented the trade-offs between redundancy and query simplicity. Their work demonstrated that strategically introducing redundancy through table merging and pre-computed joins could dramatically reduce query complexity. This comes at the cost of increased storage and update complexity.

Critically, their work focused on database engine performance rather than AI model performance. However, the principle translates: if denormalization helps database engines by reducing join operations, it should similarly help language models. The reasoning steps required during SQL generation are reduced.

Our Schema Distillation applies denormalization principles specifically optimized for small language models. Unlike traditional denormalization focused on physical performance, we denormalize to reduce cognitive load. We accept complete loss of enforceable functional dependencies in exchange for simplified reasoning paths.

3.4 COGNITIVE LOAD IN AI SYSTEMS

An emerging line of research examines whether cognitive load theory (Sweller, 1988), originally developed for human learning, applies to language model behavior.

Recent work has demonstrated that LLMs exhibit working memory constraints resembling human cognitive load (Liu et al., 2024). Extra information hurts their context processing capacity in ways analogous to human working memory limitations. Models struggle with tasks requiring them to track many entities simultaneously. They also struggle to maintain multiple relationships across long contexts.

This research suggests that reducing extraneous cognitive load should benefit model performance. Complex schemas with many tables and foreign keys impose precisely this type of load. The models must track which table contains which columns, which foreign keys enable which joins, and how to navigate multi-hop relationships.

Our work extends this cognitive load perspective specifically to database schemas. The hypothesis is that schema complexity imposes measurable cognitive burden that disproportionately affects smaller models with limited capacity.

3.5 POSITIONING OUR CONTRIBUTIONS

Given this landscape, our research makes several distinct contributions that address gaps in existing work:

On schema nomenclature: While previous work identified that naming quality matters (Luoma and Kumar, 2025) and demonstrated name inference is possible (Trabelsi et al., 2021), we provide the first systematic evaluation. We show how poor naming degrades text-to-SQL performance across different model sizes and query difficulties. Moreover, we show that in-context learning can partially recover from poor naming without task-specific training.

On schema complexity: Database research has long studied denormalization for performance (Shin and Sanders, 2006), but has not examined its effects on language model reasoning. We demonstrate that complexity reduction principles from database theory can be adapted to benefit AI models, particularly smaller ones.

On resource-constrained models: While much recent work has focused on compressing large models (Hinton et al., 2015) or generating synthetic training data, we take the complementary approach of simplifying the task itself. Schema Distillation requires only a single flagship model call for schema simplification. This is followed by inference on substantially cheaper models, a different economic trade-off than existing approaches.

On validation methodology: We introduce incremental validation for schema characteristics and its affects in model performance focused only on nomenclature. With Schema Distillation we extend this to structural complexity. We validate that improvements correlate inversely with model size. We demonstrate the capacity-complexity interaction principle empirically.

Critically, our work establishes proof-of-concept foundations rather than claiming competitive performance with flagship solutions. Our goal is to understand when and why schema manipulation can help smaller models, and to map the trade-offs involved, positioning these techniques as steps toward democratizing text-to-SQL.

The next chapter details our Wrecking SQL methodology alongside with the Distillation method. We show how we operationalize these ideas through incremental validation of schema nomenclature effects.

4 METHODOLOGY

This chapter details the methodology applied in our research. We start by describing the dataset and the evaluation metrics used throughout the experiments. Then, we present the Wrecking SQL methodology, focused on isolating the impact of schema nomenclature on text-to-SQL translation. Finally, we describe Schema Distillation, which targets schema complexity reduction to help small language models perform better in the task.

4.1 DATASET AND EVALUATION METRICS

4.1.1 Dataset

Our experiments used the BIRD dataset. It is a challenging benchmark that more accurately reflects real-life databases compared to Spider (Yu et al., 2019). Spider, created in 2018 by Yale researchers, includes 200 databases across 138 domains with 10,181 questions and 5,693 unique SQL queries. It classifies queries by difficulty (easy, medium, hard, and extra hard) based on the number of SQL components.

BIRD is a recent benchmark that adds complexity with features like missing values, incorrect queries, improper columns, and unexpected encoding. It tests models on a split that includes complex SQL functions and knowledge reasoning. The benchmark contains 12,751 questions divided into simple (30%), moderate (60%), and challenging (10%) levels.

Evaluation focuses on question understanding, knowledge reasoning, data complexity, and query complexity. A simple query in BIRD might involve a basic SELECT statement or even a query with a single JOIN clause, we can see an example of it in the Listing 4.1

```
SELECT T2.MailStreet
FROM frpm AS T1
INNER JOIN school AS T2 ON T1.CDSCode = T2.CDSCode
ORDER BY T1.'FRPM Count (K-12)' DESC
LIMIT 1
```

Listing 4.1: Example of a simple BIRD query

While an example of a challenging query found in the dataset involves a number of joins and nested queries, as exemplified in Listing 4.2.

```
SELECT AVG(single_bond_ctd)
FROM (
   SELECT T3.molecule_id, COUNT(T1.bond_tp) AS single_bond_ctd
   FROM bond AS T1
   INNER JOIN atom AS T2 ON T1.molecule_id = T2.molecule_id
   INNER JOIN molecule AS T3 ON T3.molecule_id = T2.molecule_id
   WHERE T1.bond_tp = '-' AND T3.label = '+'
```

```
8     GROUP BY T3.molecule_id
9 ) AS subquery
```

Listing 4.2: Example of a challenging BIRD query

All three types of queries are divided among 95 individual databases. The number of tables varies from 3 to 64 and the number of columns per database varies from 4 to 115.

4.1.2 Evaluation metrics

To compare results, we used **execution accuracy**, the same evaluation metric defined in our theoretical background (Equations 2.8–2.9). This is also the standard metric reported in the BIRD and Spider leaderboards. Execution accuracy measures the proportion of questions for which the SQL predicted by the model produces the same result set as the ground-truth SQL, and therefore directly captures end-to-end correctness. In our setting, this is the most appropriate choice because our interventions (schema renaming and schema distillation) are designed to affect whether the generated query executes to the correct answer, not merely whether its surface form matches the gold query. In addition to reporting execution accuracy, we interpret the observed gains/losses using the schema complexity metrics introduced in Chapter 2: relational complexity (C_R, Equation 2.4), structural complexity (C_S, Equation 2.5), and attribute dispersion (C_A, Equation 2.6), as well as their combined form (C_{total}, Equation 2.7). These metrics provide the explanatory bridge between our interventions (e.g., removing foreign keys or collapsing joins) and the execution-accuracy outcomes reported in Chapter 5. Finally, when we claim that an improvement is meaningful (especially in the Schema Distillation analysis), we follow the statistical evaluation described in Chapter 2 by reporting significance tests (paired t-test and Wilcoxon signed-rank test, depending on assumptions) and effect size via Cohen's d (Equation 2.10).

4.2 WRECKING SQL - INCREMENTAL VALIDATION METHODOLOGY

In the Wrecking SQL pipeline, we create the corpus with multiple versions of the original dataset. Each version is incrementally modified by removing specific parts of the schema description or inferring the columns using an LLM. At the end of the corpus construction, we have six distinct versions of the dataset. These are also the stages of our incremental methodology. Our goal is analyzing the degradation in execution accuracy.

4.2.1 Model selection

We used the latest models available at the start of our experiments and updated with new releases to stay relevant. Our implementation uses the Hugging Face and Ollama libraries, enabling us to self-host models locally.
For the Wrecking SQL experiments we used:

- Command-R [7B]
- Gemma2 [2B, 9B]
- Granite3.1-dense [2B, 8B]
- Llama3.1 [8B]
- Llama3.2 [1B]
- Phi4 [14B]
- Qwen2.5-coder [1.5B, 14B]
- Qwen2.5 [14B]

We did not fine-tune the models, leveraging their inherent capabilities. The selection was driven by reproducibility and practicality: we focused on widely available open-source models that could be self-hosted within our hardware limitations. We also included different generations of the same family (e.g., Llama3.1 and Llama3.2) to keep a perspective on how these models evolve across releases.

4.2.2 Dataset Versions

The incremental modifications to create the six dataset versions are:

1. **Default (D)**: Dataset without any modifications, serving as our baseline.
2. **Meaningless Columns (MC)**: Dataset where all columns have meaningless names.
3. **Meaningless Tables (MT)**: Dataset where all tables have meaningless names.
4. **Meaningless Columns and Tables (MCT)**: Dataset where both columns and tables have meaningless names.
5. **Inferred Names Simple (INFS)**: Dataset with inferred column names using simple prompting.
6. **Inferred Names Complete (INFC)**: Dataset with inferred column names using a more complex prompt.

By meaningless, we refer to names that lack any semantics or connection to the content of tables or columns. Such naming conventions are commonly found in legacy production systems (Hariharan and Reeshma, 2015). These systems were often designed for efficiency and rapid problem-solving rather than for future analytical or AI applications that rely on meaningful naming.
For example, legacy banking systems often use column names like `Filler0001`, `Filler0002`, etc. This practice accommodates schema evolution but comes at the expense of column semantics.

We apply similar naming across the dataset with a schema mapping: the original dataset schema pointing to the modified schema. For instance, a column named `playername` becomes `column1`, and a table named `restaurant_menu` becomes `table1`.

4.2.3 Schema Mapping Implementation

Our methodology requires a pair of dataset and golden SQL queries for training and testing. Modifications were applied only to the databases, not to golden queries. This allows us to compare results from both modified and default data sources. In Listing 4.3 we observe an example of the table description file from the dataset authors, modified by us, creating our schema mapping implementation. It contains database IDs, table names, column names, types, and keys.

```
{
  "debit_card_specializing": {
    "columns": {
      "customers": {
        "CustomerID": "column_0",
        "Segment": "column_1",
        "Currency": "column_2"
      },
      "gasstations": {
        "GasStationID": "column_0",
        "Country": "column_2"
      }
    },
    "tables": {
      "customers": "table_0",
      "gasstations": "table_1"
    }
  }
}
```

Listing 4.3: Schema mapping structure for the debit_card_specializing database

As the JSON shows, we start from the original file, get the database_id, and construct an object to map all names. We separate columns and tables. Columns are mapped to dummy names (e.g., column1, column2) based on their order in each table. Tables are similarly named (e.g., table1, table2) based on their position in the database description file.

This logic is applied to every database in the benchmark. This creates a unified mapper for building prompts used in inference, with new names and databases. The generated SQL runs and returns result tuples to be compared with the golden queries. It is important to highlight that this method can be extrapolated to any other SQL database with very little effort.

4.2.4 Database Modification Process

With the "meaningless" and "inferred" mappers ready, we built new database versions for model-generated queries using the sqlite3 library in Python. We created two methods: `alter_table_names` and `alter_column_names`. The `alter_table_names` method uses the ALTER TABLE SQL clause to rename tables based on the mapper. This is straightforward in SQLite shown in Listing 4.4

```
ALTER TABLE customers RENAME TO table_0;
ALTER TABLE gasstations RENAME TO table_1;
```

Listing 4.4: Table renaming example

The `alter_column_names` method, illustrated in Listing 4.5, is more complex because SQLite does not support renaming columns directly. Instead, we duplicate each table with the same schema but with renamed columns, then insert the original data.

```
-- Create new table with renamed columns
CREATE TABLE customers_new (
    column_0 integer PRIMARY KEY,
    column_1 text,
    column_2 text
);

-- Copy data from original table
INSERT INTO customers_new SELECT * FROM customers;

-- Drop original table and rename new one
DROP TABLE customers;
ALTER TABLE customers_new RENAME TO customers;
```

Listing 4.5: Column renaming process

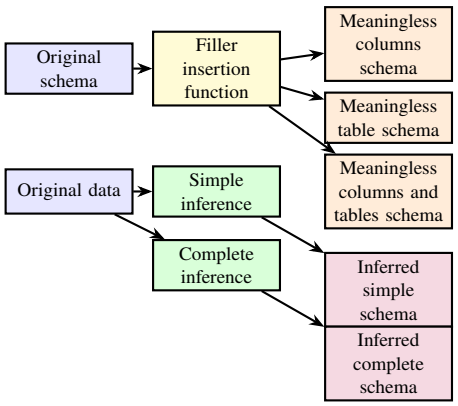

Figure 4.1: Wrecking SQL Pipeline

These functions create multiple datasets used in the Wrecking SQL pipeline which can be seen in Figure 4.1. This includes a combined version with renamed columns and tables,

4.2.5 SCHEMA NAMING INFERENCE

Inferred names follow the mapper structure shown previously. Instead of meaningless names, we use prompts supplied to LLMs to infer names based on table content. We use two in-context learning approaches using few-shot techniques: simple and complete.

4.2.5.1 Simple Inference (INFS)

The simple approach provides column data for name inference. We sample values from each column and ask the LLM to generate an appropriate label. The prompt structure can be found in Listing 4.6. This approach is fast and requires minimal context. It is suitable for columns where the values themselves are self-explanatory (numeric IDs, dates, simple categorical values).

```
You are supplied with the content of a specific column from a database.
You are tasked with rephrasing the name of the column to better reflect
its content. Remember that this name should be simple and also descriptive.

The content of the column is as follows: "{content}"

Your response should be a simple new string that represents the content
of the column. Just that, nothing more is accepted.

Respond with a single string, without explanations.
String must be a single phrase linked by underscores, like "new_column_name".
Give just the name, NOTHING MORE IS ACCEPTED

Follow the examples below:

content: "1, 2, 3, 4, 5"
new column name: "numbers"

content: "apple, banana, orange, pear"
new column name: "fruits"

Now based on the previous examples,
do the inference for the information below:

content: "{content}"
new column name:
```

Listing 4.6: Simple inference prompt template

4.2.5.2 Complete Inference (INFC)

The complete approach provides both column content and full row content for broader reasoning context. This allows the LLM to understand how columns relate to each other within the same

table, the prompt we use is located in Listing 4.7. This approach is more expensive (requires more tokens per inference) but produces better results for columns where the meaning depends on context. It also helps when values alone are ambiguous.

```
You are supplied with the content of a specific column from a database
and also the content of the table at which this column is located. Based
on this information, you are tasked with rephrasing the name of the column
to better reflect its content. I want you to consider the content of the
table and the column to come up with a new name.

The content is of extreme importance, so make sure to consider it when
coming up with the new name. Remember that this name should be simple
and descriptive.

Give just the name, NOTHING MORE IS ACCEPTED

table content: "{table_content}"
column content: "{column_content}"

Follow the examples below:

table content: [
   ["hot", "sunny", "south", "beach"],
   ["cold", "rainy", "north", "mountain"],
   ["mild", "cloudy", "west", "forest"]
]
column content: ["hot", "cold", "mild"]
new column name: "temperature"

table content: [
   ["hot", "sunny", "south", "beach"],
   ["cold", "rainy", "north", "mountain"],
   ["mild", "cloudy", "west", "forest"]
]
column content: ["sunny", "rainy", "cloudy"]
new column name: "weather"

Think of this new name as a way to summarize what is in the column. The
idea is that by reading it you will have a good idea of what is in the
column. Because of that it should be semantically related to the content
of the column and the table, very content specific.

Your response should be a simple new string that represents the content
of the column. Just that, nothing more is accepted.

Respond with a single string, without explanations.
Avoid names that are devoid of semantic meaning and very general. The name
```

```
should be specific to the content of the column and the table.
String must be a single phrase linked by underscores, like "new_column_name".
```

Listing 4.7: Complete inference prompt template

4.2.5.3 Inference Process

For each column in each table of each database:

1. Sample up to 10 distinct non-null values from the column
2. For INFS: Pass these values to the LLM with the simple prompt
3. For INFC: Also collect full rows and pass both to the LLM with the complete prompt
4. Parse the LLM response to extract the suggested name
5. Validate the name (check for forbidden characters, length constraints)
6. Update the schema mapper with the inferred name

We used Gemini 1.5 Pro for inference due to its strong performance on naming tasks and relatively low cost.

4.2.6 PROMPT CONSTRUCTION FOR TRANSLATION

To compare models across different created databases, we employed prompt engineering. This guides the model toward the correct solution by providing relevant information directly in the input, without fine-tuning the base LLM.

4.2.6.1 Zero-Shot Translation Prompts

In our experiments, we used a zero-shot approach. This means no examples were provided alongside the instructions. Each of our prompts consists of a triplet: an instruction, a schema, and the question to be translated. This triplet is fed into the model. It prompts the model to generate an SQL query that satisfactorily answers the question, as demonstrated in Listing 4.8.

```
Based on the database schema below and the question, create a SQL query
that will return the desired result:

----------------------
DATABASE SCHEMA
CREATE TABLE table_0 (
   column_0 integer,
   column_1 text,
   column_2 text,
   PRIMARY KEY (column_0)
```

```
);
CREATE TABLE table_1 (
   column_0 integer,
   column_1 integer,
   column_2 text,
   column_3 text,
   PRIMARY KEY (column_0)
);
CREATE TABLE table_2 (
   column_0 integer,
   column_1 text,
   PRIMARY KEY (column_0)
);
CREATE TABLE table_3 (
   column_0 integer,
   column_1 date,
   column_2 text,
   column_3 integer,
   column_4 integer,
   column_5 integer,
   column_6 integer,
   column_7 integer,
   column_8 real,
   PRIMARY KEY (column_0)
);
CREATE TABLE table_4 (
   column_0 integer,
   column_1 text,
   column_2 real,
   FOREIGN KEY (column_0) REFERENCES table_0(column_0)
);

QUESTION: Which country had the gas station that sold the most expensive
product id No.2 for one unit?
----------------------
```

Listing 4.8: Zero-shot translation prompt structure

In this prompt format, we have the following organization:

- Lines 1–3 provide the instruction to the LLM
- Lines 5–42 define the schema using standard SQL CREATE TABLE statements
- Lines 43–45 present the question to be translated

For each question in the benchmark, we use this structured prompt to generate the corresponding SQL query automatically. The prompt structure remains identical across all dataset versions. Only the schema content changes (meaningless names, inferred names, or original names).

4.3 SCHEMA DISTILLATION: COMPLEXITY REDUCTION

In the second part of our research, to tackle the remainder of our research questions, we present Schema Distillation as a concrete implementation of complexity reduction for small language models going towards the scientific contribution of this work of of validating its benefits.

4.3.1 Model selection

For the Schema Distillation experiments, the applied SLMs were:

- Gemma3 [1B, 4B]
- Granite3.3 [2B]
- Llama3.1 [8B]
- Llama3.2 [1B, 3B]
- Mistral [7B]
- Qwen2.5-coder[0.5B, 1.5B, 3B]
- Qwen3 [0.6N, 1.7B, 4B, 8B]

We did not fine-tune the models, leveraging their inherent capabilities. Since this phase of the research came after Wrecking SQL, we updated the model set to reflect the most cutting-edge open-source releases available at the time, while still keeping the selection reproducible and compatible with our hardware limitations.

4.3.2 From Theory to Implementation

Our approach operationalizes the reduction of the complexity metrics through three core reduction strategies: eliminating relational complexity ($C_R \rightarrow 0$) by removing foreign key constraints and merging related tables into denormalized views, reducing structural complexity (C_S transformation) where queries access only a single view rather than multiple tables and lastly by minimizing attribute dispersion ($C_A \rightarrow 0$) by co-locating related attributes within views.
Although we implement these principles through in-context learning with SLMs, the underlying complexity reduction strategies could also be implemented through automated schema transformation algorithms (rule-based systems analyzing query patterns), fine-tuned distillation models which are specialized transformers trained on schema simplification, query-driven denormalization engines (systems monitoring workloads and creating denormalized structures), or hybrid human-AI approaches (database administrators using our metrics as guidelines).
Any implementation achieving similar complexity reduction ($C_R \rightarrow 0$, $C_A \rightarrow 0$) should produce similar model-size-dependent effects. Our in-context learning approach demonstrates feasibility

and economic viability (single flagship call + small model inference), but the fundamental contribution is demonstrating that these complexity reduction principles benefit small models which is validated through statistical analysis (Section 6) showing p = 0.025, Cohen's d = 0.97 for models ≤3B parameters. Schema Distillation is a pipeline to manipulate any database schema before applying **ANY** text-to-SQL translation method, positioning itself as an add-on component.

4.3.3 Distillation Pipeline

For our experimentation we implemented a multi-step pipeline, illustrated in Figure 4.2 that leverages the original questions and the original database schema to create a better and more semantically meaningful schema. By semantically meaningful we mean, a schema which helps the language model understand better the database relations and simplifies connections between tables and columns. Once that we are mainly using small models, who have more limited capabilities, the pipeline transforms the original schema into its distilled version. We start by selecting the original questions and the original schema for that database.

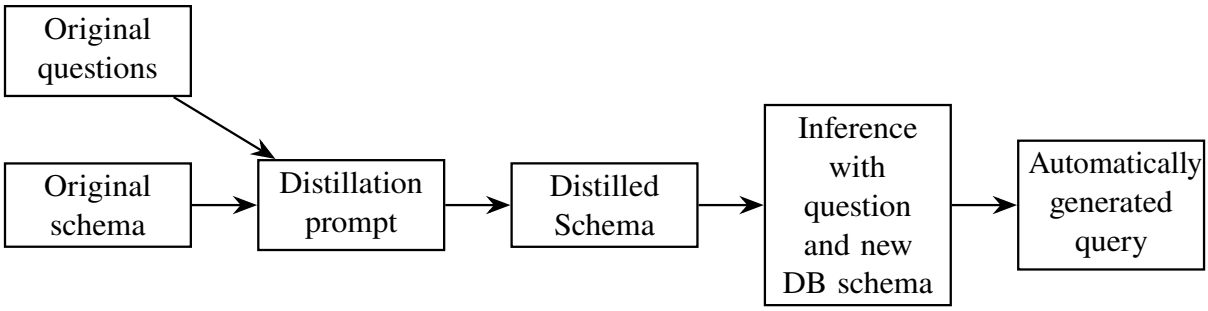

Figure 4.2: Distillation Pipeline

Once we have that, all this information is injected into a prompt that is tailored to digest all this information and make information more palatable based on the set of questions that are asked on the database. This prompt is processed by a flagship LLM in a single call to produce a new Distilled schema, for the purpose of this research the model that performed better in this task were Gemini 2.5 Pro. An example from the old schema is present on Listing 4.9.

```
1 ---------------------
2 CREATE TABLE Examination (
3 ID integer, Examination Date date, aCL IgG real, aCL IgM real, ANA integer,
      ANA Pattern text,aCL IgA integer,Diagnosis text,KCT text,
4 RVVT text, LAC text, Symptoms text, Thrombosis integer, FOREIGN KEY (ID)
     REFERENCES Patient(ID)
5 );
6 CREATE TABLE Patient (
7 ID integer, SEX text, Birthday date, Description date, First Date date,
     Admission text, Diagnosis text, PRIMARY KEY (ID)
8 );
9 CREATE TABLE Laboratory (
```

```
ID integer, Date date, GOT integer, GPT integer, LDH integer, ALP integer,
    TP real, ALB real, UA real, UN integer, CRE real, T-BIL real,
T-CHO integer, TG integer, CPK integer, GLU integer, WBC real, RBC real,
    HGB real, HCT real, PLT integer, PT real, APTT integer, FG real,
PIC integer, TAT integer, TAT2 integer, U-PRO text, IGG integer, IGA
    integer, IGM integer, CRP text, RA text, RF text, C3 integer,
C4 integer, RNP text, SM text, SC170 text, SSA text, SSB text, CENTROMEA
    text, DNA text, DNA-II integer,
FOREIGN KEY (ID) REFERENCES Patient(ID),);
----------------------
```

Listing 4.9: Original schema example

While the schema inferred by the LLM using our few-shot prompt for simplification, can be seen in Listing 4.10.

```
----------------------
CREATE VIEW autoimmune_disease_profile (patient_id, sex, birthday,
    event_date, age_at_event, diagnosis, symptoms, ana_pattern,
ana_value, rheumatoid_factor, c_reactive_protein, C3, C4, anti_rnp_antibody,
     anti_sm_antibody, anti_scl70_antibody, anti_ssa_antibody,
anti_ssb_antibody, anti_centromere_antibody, anti_dna_antibody,
    anti_dna_ii_antibody, IGG, IGA, IGM);

CREATE VIEW coagulation_and_thrombosis_profile (patient_id, sex, birthday,
    event_date, age_at_event, Diagnosis, thrombosis_level,
platelet_count, prothrombin_time, activated_partial_thromboplastin_time,
    fibrinogen, kaolin_clotting_time, russell_viper_venom_time,
lupus_anticoagulant);

CREATE VIEW metabolic_health_profile (patient_id, sex, birthday,
    admission_status, lab_date, age_at_lab_test, got_ast, gpt_alt,
lactate_dehydrogenase, alkaline_phosphatase, total_protein, albumin,
    uric_acid, urea_nitrogen, creatinine, total_bilirubin,
total_cholesterol, triglycerides, creatine_phosphokinase, glucose);

CREATE VIEW patient_demographics (patient_id, sex, birthday,
    patient_description_date, first_visit_date, admission_status,
initial_diagnosis);

CREATE VIEW patient_examination_results (patient_id, sex, birthday,
    admission_status, examination_date, age_at_examination, acl_igg,
acl_igm, acl_iga, ana_value, ana_pattern, diagnosis, symptoms,
    thrombosis_level, KCT, RVVT, LAC);

CREATE VIEW patient_lab_results (patient_id, sex, birthday,
    admission_status, lab_date, age_at_lab_test, GOT, GPT, LDH, ALP, TP,
```

```
ALB, UA, UN, CRE, t_bil, t_cho, TG, CPK, GLU, WBC, RBC, HGB, HCT, PLT, PT,
    APTT, FG, PIC, TAT, TAT2, u_pro, IGG, IGA, IGM, CRP, RA,
RF, C3, C4, RNP, SM, SC170, SSA, SSB, CENTROMEA, DNA, dna_ii);
----------------------
```

Listing 4.10: Distilled schema example

It is important to note that we chose to use **views** rather than creating new tables. This decision avoids the need to refactor the entire database, since views allow us to generate the processed structures while keeping the original data intact. This approach prevents migration issues and makes the method suitable for deployment in enterprise environments with minimal overhead. We replace the original schema for the Distilled schema in the the text-to-SQL inference method and also create the new views on the database to run the generated query and extract the results. After this pipeline occurs, in our case, we proceed to the evaluation steps, which compares golden SQL results to the ones produced by the Distillation pipeline.

4.3.4 Distillation Prompt

To achieve a simplified database schema for our small language model to do text-to-SQL translation, we employ in-context learning with a one-shot technique, where we use a prompt with the database information and questions to produce a Distilled version for the smaller model.

```
[ORIGINAL SCHEMA]
{original_schema}

[QUESTIONS]
{question_group}

Based on this schema, and the SQL questions, merge the data on the select
    and filters to a single view,
that will be queried and will return the same result. The endgoal here is
    to create a comprehensive
view that returns the same result. All the questions must be replied by
   the consolidated view.
At the end of the view, write how would be the query that returns the
   same result in this new view.
Start the example queries with --. Always create new views, not tables.
   Your intention is to integrate
the data from the questions into the least number of views that can be
   queried to get the same results
as the original queries. The part of getting the same results is key.
Your main intention is: Put all the data of the parcial queries in the
   views, with the same names as
in the original schema. The views must be self-contained and not require
   any additional tables to answer
the questions. THis is extremely important. This is the main requirement
   of this query.
```

```
17
18 Bellow you can see an example of what I want you to do:
19
20 Question: What's the finish time for the driver who ranked second in 2008
       's Australian Grand Prix?
21 This is the SQL that answers it:
22
23 SELECT T1.time FROM results AS T1 INNER JOIN races AS T2 ON T1.raceId =
       T2.raceId WHERE T1.rank = 2
24 AND T2.name = 'Australian Grand Prix' AND T2.year = 2008
25
26 One way to generalize this question simplifying the final query using the
        original schema is:
27 CREATE VIEW finish_times_by_year_position AS
28 SELECT DISTINCT T1.time finish_time, T1.rank position_in_race, T2.name
       race_name,
29    T3.forename driver_first_name, T3.surname driver_last_name, T3.
          nationality driver_nationality,
30    T2.year year_of_race
31 FROM results T1 LEFT JOIN races AS T2 ON T1.raceId = T2.raceId
32 LEFT JOIN drivers T3 ON T1.driverId = T3.driverId
33
34 This transforms the query to:
35 SELECT finish_time FROM finish_times_by_year_position WHERE year_of_race
       = 2008
36 AND race_name = 'Australian Grand Prix'
37
38 While helps to reply to other related questions by generalizing it with
       the table. Make sure to rename
39 properly the new fields, make them semantically relevant and simple. Now
       its your turn.
40 Also always use distinct to avoid duplicates. Try to make the most
       reasonable generalization possible,
41 while keeping the data required to answer the questions. It is key to
       keep the data required to answer the questions, the same results must
       be
42 returned. JOIN at most 3 entities in the same view. If more are required,
        create a new view. I want for
43 you to have at minimum 5 views, that fills different purposes
       highlighting different group of questions.
44 Folow the example above strictly. Pre-calculate the joins, filters,
       selection and groups in the views.
45 Always use snake_case for the views and the fields.
46 **IMPORTANT** -> The names of the views must be semantically relevant to
       the data they contain.
47 You are obligated to create ONLY 5 VIEWS. NOTHING MORE.
48 **IMPORTANT** -> Each view should answer a niche set of questions.
```

Listing 4.11: Distillation prompt

In Listing 4.11 we can examine the prompt format we used, lines 1-2 supply the original database schema, lines 4–5 supply the original questions, and lines 7–48 are the set of instructions including the one-shot to direct the model to produce the new Distilled schema instructing it to reproduce the same reasoning displayed on the example. This prompt runs once for each database and produce a set of new digested view that will be used in the text-to-SQL translation step.

4.3.5 Text-to-SQL Translation Prompt

To compare models across databases, we used prompt engineering to guide the model without fine-tuning the base SLM. Our zero-shot approach, shown in Listing 4.12, involved no examples with the instructions. Each prompt includes an instruction, schema, and question, which the model uses to generate a SQL query.

```
1 "Based on the database schema below and the question, create a SQL query:
2 [DATABASE SCHEMA] --> {schema}
3 [QUESTION] --> {question}
4 Pay attention to craft a valid SQL query. The SQL query must be in SQLite
      syntax.
5 Using only the tables and columns present in the schema above. Use only
    columns from the section DATABASE SCHEMA.
```

Listing 4.12: Prompt to do the SQL translation

Also when using the new schema, we asked to the SLM to focus on one table only, as it is the intention of our Distillation prompt, to remove the need for any kind of joins. For that, we add the a line at the end of the prompt, to reinforce for using a single view when creating the SQL as we can see in Listing 4.13

```
1 You dont need to use joins, only use a single table to reply to the
    question.
2 Pay attention to the column and table names.
3 You will never need to use joins, select single table to reply to the
    question
4 and craft the SQL query based on that.
```

Listing 4.13: Prompt restriction

This single addition is used to direct the model in using the new schema as we intent it to, telling it not to try to do any joins, as our new schema is not meant for that and works under the single table query constraint.

4.4 SUMMARY

This chapter presented the methodology for the two steps of our incremental research, displaying in order the steps taken to make our experiments possible. The methodology enables us to answer our research questions with clarity and confidence, which will be displayed in the following section.

5 EXPERIMENTS

In this section we present the experiments and the results of the two parts of our research. First focusing on the comparisons between the different kinds of schemas and afterwards the affects of Distillation in the translation.

5.1 SCHEMA NOMENCLATURE AFFECT ON TEXT-TO-SQL TRANSLATION

In this experiment, we quantify how schema nomenclature affects text-to-SQL translation accuracy. We apply the Wrecking SQL incremental validation protocol: starting from the default schema (D), we progressively obfuscate table names (MT), column names (MC), and both (MCT), and then measure the impact on execution accuracy.

Model	**Default**	**MT**	**MC**	**MCT**
Command-R7b [7B]	23.01%	16.29%	6.45%	2.34%
Gemma2 [2B]	10.03%	7.17%	2.73%	1.49%
Gemma2 [9B]	22.09%	21.05%	6.38%	2.54%
Granite3.1-dense [2B]	5.01%	4.23%	1.69%	1.10%
Granite3.1-dense [8B]	11.92%	10.69%	2.15%	1.89%
Llama3.1 [8B]	20.92%	17.99%	6.06%	2.47%
Llama3.2 [1B]	7.36%	3.58%	2.99%	2.15%
Phi4 [14B]	23.46%	22.29%	8.08%	1.89%
Qwen2.5-coder[1.5B]	12.38%	7.88%	2.60%	1.89%
Qwen2.5-coder[14B]	29.92%	29.59%	11.79%	3.25%
Qwen2.5 [14B]	22.03%	23.14%	7.43%	2.28%

Table 5.1: Performance comparison of the models across the different variants of the dataset

Table 5.1 summarizes execution accuracy across the four schema variants (Default, MT, MC, and MCT) for each evaluated model. The results show a consistent pattern: the best performance comes from the default database, followed by the database with non-significant table names (MT). Performance declines sharply when column names are altered (MC), and the lowest performance is observed when both column and table names are modified (MCT). Next, we analyze whether this degradation depends on query difficulty, by breaking the results down by BIRD difficulty tier in Table 5.2 that shows all difficulty tiers suffer when column names are altered. Even simple questions are affected, indicating that proper column naming is crucial for effective text-to-SQL systems. Taken together, Table 5.1 (aggregate results) and Table 5.2 (difficulty breakdown) support the central conclusion of this experiment: while LLMs can often tolerate non-semantic table names, meaningful column names are critical for successful text-to-SQL translation, and their degradation produces substantial performance loss.

Model	Simple		Moderate		Challenging	
	Default	**MC**	**Default**	**MC**	**Default**	**MC**
Command-R7b [7B]	30.91%	8.21%	11.39%	3.87%	9.72%	3.47%
Gemma2 [2B]	14.81%	3.13%	2.79%	2.58%	2.7%	0.69%
Gemma2 [9B]	28.54%	8.10%	12.47%	4.08%	11.80%	2.77%
Granite3.1-dense [2B]	6.91%	2.37%	1.72%	0.86%	3.47%	0.0%
Granite3.1-dense [8B]	16.97%	2.70%	4.08%	1.72%	4.86%	0.0%
Llama3.1 [8B]	27.02%	7.45%	10.75%	4.51%	14.58%	2.08%
Llama3.2 [1B]	10.48%	3.24%	2.58%	3.44%	2.77%	0.0%
Phi4 [14B]	30.59%	10.27%	13.54%	4.30%	9.72%	6.25%
Qwen2.5-coder[1.5B]	17.18%	2.81%	4.94%	2.58%	5.55%	1.38%
Qwen2.5-coder[14B]	38.48%	15.67%	16.55%	6.66%	18.05%	3.47%
Qwen2.5 [14B]	29.29%	9.40%	10.96%	4.08%	11.11%	5.55%

Table 5.2: Performance comparison of models across different difficulty levels of query difficulty (Simple, Moderate, and Challenging)

5.2 INFERENCE IMPROVEMENT IN TRANSLATION FOR LEGACY DATABASES

To address the importance of column names in in-context learning LLM-based translation, we developed a novel method to automatically generate meaningful names based on table content. Using the schema mapper structure previously shown, we leveraged LLMs' in-context capabilities to create descriptive names and replace the dummy names for semantically relevant ones. As demonstrated in the methodology two prompts were used: a **simple** one providing sample of the column content and requesting it to generate the appropriate labels and a more **complete** one including both column content and full rows for reasoning.

These two methods were applied to each column in the experimental dataset to generate a mapping, which was then used as the new column names to assess their impact on accuracy.

	MC	**INFS**	**INFC**
Command-R7b [7B]	6.45%	8.53%	10.62%
Gemma2 [2B]	2.73%	3.06%	3.58%
Gemma2 [9B]	6.38%	8.86%	10.43%
Granite3.1-dense [2B]	1.69%	2.02%	1.95%
Granite3.1-dense [8B]	2.15%	4.04%	5.41%
Llama3.1 [8B]	6.06%	7.62%	7.88%
Llama3.2 [1B]	2.99%	1.56%	2.02%
Phi4 [14B]	8.08%	9.64%	9.77%
Qwen2.5-coder[1.5B]	2.60%	2.67%	4.49%
Qwen2.5-coder[14B]	11.79%	13.88%	14.40%
Qwen2.5 [14B]	7.43%	9.12%	9.51%

Table 5.3: Performance of the models across datasets with columns modifications

As shown in Table 5.3, most of the LLMs performed better with the new set of names inferred through prompt engineering compared to the meaningless names. We observed a 2% to 4%

improvement in translation accuracy, indicating that inferring and improving column names can enhance performance, even in databases operating under the worst-case scenario.

Although the dataset version with renamed methods performed better than the one with meaningless names, particularly the more comprehensive version that relies on full row information, there are still several challenges to overcome before matching the performance of the default dataset. Issues such as data ambiguity within the columns and rows, as well as highly specific domain knowledge embedded in the table, are not fully captured by the LLM when attempting to recreate the names. Addressing these challenges is essential to improving the efficiency of the method.

To better understand the complexity of inferring names, we can examine examples that emerged from our experiments. We can bring here one of the most successful inferences that is the columns that are the unique identifiers. We show a portion of the debit_card_specializing mapper to exemplify our point in Listing 5.1.

```
{
   "debit_card_specializing": {
      "columns": {
         "customers": {
            "CustomerID": "transaction_id",
            ...
            ...
         },
         "gasstations": {
            "GasStationID": "product_id",
            ...
            ...
         },
         "products": {
            "ProductID": "product_type_id",
            ...
            ...
         },
         "transactions_1k": {
            "TransactionID": "transaction_id",
            ...
            ...
         },
         "yearmonth": {
            "CustomerID": "customer_id",
            ...
            ...
         }
      },
   }
}
```

Listing 5.1: Real columns Vs Inferred columns (sucess case)

As we can see in the JSON example, all the unique keys from the database were identified as the identifiers that they are. Moreover more than half of them were recreated with a name very close to the original names, showing us that with the appropriate data it is possible to get close to unique identifiers names from a "conventional" dataset. Usually this happens when a table has a few columns that are identifiers (primary key and foreign keys). On the other hand when we have many of those, in a specific table, the inference does not yield great results as displayed in Listing 5.2.

```
{
   "card_games": {
         "columns": {
            "cards": {
               "id": "card_types",
               "cardKingdomFoilId": "card_release_year",
               "cardKingdomId": "card_release_year_21565",
               "mcmId": "card_number",
               "mcmMetaId": "card_rarity_371",
               "mtgArenaId": "card_rarity_53928",
               "mtgjsonV4Id": "card_unique_id",
               "mtgoFoilId": "card_multiverse_id",
               "mtgoId": "card_multiverse_id_51847",
               "multiverseId": "card_id",
               "otherFaceIds": "card_rarity_69766",
               "scryfallId": "card_identifiers",
               "scryfallIllustrationId": "card_unique_id_51436",
               "scryfallOracleId": "card_unique_id_74168",
               "tcgplayerProductId": "card_power_toughness",
               "type": "card_types_84018",
               "uuid": "card_names_63896",
            },
         }
      }
}
```

Listing 5.2: Real columns Vs Inferred columns (worst case scenario)

The cards table in the card_games database has a lot of identifiers, and during the prompt inference, we have trouble in creating a new name for the columns. In this example, we have a lot of domain specific information in the columns names, which makes a lot tougher to differentiate each one and assign the correct nomenclature. The point of domain specific information is the central problem here, for example, cardKingdomFoilId column, to arrive at this name is extremely difficult it is extremely linked to the origin of the data, which is very niche. That, will be the direction of our future work, understanding how to create solutions that can deal with that characteristic and craft the specific required knowledge to solve this shortcoming.

5.3 DISTILLATION RESULTS AND ANALYSIS

We present the Distillation results in two parts: first, statistical analysis establishing model-size-dependent effects; second, functional dependency analysis quantifying the underlying semantic trade-offs.

5.3.1 Small Models Show Significant Improvement.

Model	**Original schema**	**Distilled schema**	**Absolute difference**	**Percentage difference**
gemma3:1b	3.5%	3.32%	-0.18%	-5.14%
gemma3:4b	16.36%	17.86%	1.5%	9.17%
granite3.3:2b	9.71%	12.25%	2.54%	26.16%
llama3.1:8b	18.31%	19.29%	0.98%	5.35%
llama3.2:1b	2.99%	5.08%	2.09%	69.90%
llama3.2:3b	13.49%	14.79%	1.3%	9.64%
mistral:7b	12.39%	13.42%	1.03%	8.31%
qwen2.5-coder:0.5b	5.21%	6.06%	0.85%	16.31%
qwen2.5-coder:1.5b	10.29%	12.58%	2.29%	22.25%
qwen2.5-coder:3b	18.70%	18.44%	-0.26%	-1.39%
qwen3:0.6b	3.71%	5.54%	1.83%	49.33%
qwen3:1.7b	13.55%	12.97%	-0.58%	-4.28%
qwen3:4b	20.01%	17.20%	-2.81%	-14.04%
qwen3:8b	23.27%	18.57%	-4.7%	-20.20%

Table 5.4: Performance comparison of the models in both schemas

The experimental results for the performance comparison in Table 5.4 demonstrate heterogeneous outcomes across the full spectrum of evaluated models. Overall, approximately 64% (9 out of 14) of models show improvements with the distilled schema, though this pattern does not reach statistical significance across the complete sample (Wilcoxon signed-rank test, $p = 0.24$).
However, a focused analysis reveals that **schema distillation produces statistically significant benefits specifically for smaller models**. When restricting analysis to models with $\leq$3B parameters (N=9), the improvements become statistically significant using a paired t-test ($p = 0.025$, Cohen's $d = 0.97$), indicating a large effect size. Within this small model subset, 67% (6 out of 9) exhibit improved performance, with relative gains ranging from 5.35% to 69.90% and a mean relative improvement of 32.27% among positively-affected models.
The models demonstrating the greatest performance gains are those with the smallest parameter counts, llama3.2:1b (69.90% relative improvement), qwen3:0.6b (49.33%), and granite3.3:2b (26.16%). This systematic trend, combined with the large effect size (Cohen's $d = 0.97$), provides strong evidence that schema distillation is particularly beneficial for resource-constrained models. On the other hand, larger models (>3B parameters) show mixed results, with several exhibiting degraded performance. This suggests that while smaller models benefit from reduced schema complexity, larger models may lose critical information through the distillation process.

5.3.2 Inverse Correlation with Model Size.

The systematic gradient from +69.90% (llama3.2:1b) through intermediate improvements (+26.16% for granite3.3:2b, +9.64% for llama3.2:3b) down to degradation (-20.20% for qwen3:8b, -14.04% for qwen3:4b). Improvement magnitude correlates inversely with model parameter count. This trend validates the capacity-complexity interaction principle (Section 2.4.1), demonstrating that the effect is predictable and capacity-dependent, not prompt-specific or database-specific. While we do not compute C_{total} numerically for all databases, our distillation approach systematically achieves the theoretical goals: $C_R \to 0$ (all foreign keys eliminated), C_S transformation (normalized tables $\to$ specialized views, each query uses single view), and $C_A \to 0$ (average joins per query reduced to zero). This consistent complexity reduction across 11 diverse dev BIRD databases demonstrates that our approach implements the theoretical principles systematically, validating that the findings represent generalizable science rather than prompt-specific engineering.

5.3.3 Understanding Semantic Loss in Schema Distillation

To understand why smaller models improve while larger models degrade, we analyzed the semantic information lost during schema distillation in the example database belonging to the schemas presented in previous sections. Our manual analysis of functional dependencies (FDs), displayed in Table 5.5, in both schemas reveals that **100% of enforceable functional dependencies are eliminated**.

Dependency Type	**Original**	**Distilled**	**Status in Distilled Schema**
Primary Key FDs	3	0	All lost - views have no enforceable keys
Foreign Key FDs	2	0	Both lost - relationships merged into views
Transitive FDs	2	0	Lost - no join paths remain
Implicit FDs in Views	0	6	Present but unenforceable
Enforceable FDs	**7**	**0**	**100% loss**

Table 5.5: Functional Dependencies: Original vs. Distilled Schema

What is Lost. By converting normalized tables with explicit foreign keys into denormalized views, Schema Distillation eliminates: foreign key constraints where examinations can no longer be enforced to reference valid patients; cardinality constraints defining one-to-many relationships; primary key dependencies where patient demographics now repeat across every row instead of appearing once (e.g., a patient with 10 examinations and 20 lab tests has sex/birthday duplicated 30 times); and normal form guarantees (3NF/BCNF), as distilled views violate normalization by introducing massive redundancy. While distilled views contain 6 implicit FDs, these lack database enforcement: views cannot have primary/foreign keys, relationships between `patient_id` across views are ambiguous, and no mechanism prevents orphaned data.

Why This Explains Our Results. This complete semantic loss creates the observed trade-off. Smaller models benefit from cognitive simplification: no complex join reasoning, no tracking which table contains which attributes, and all data co-located in flat views. Conversely, larger models can exploit normalized schema semantics where foreign keys signal join requirements, primary keys indicate deterministic relationships, cardinalities guide query optimization. When these formal constraints are eliminated, sophisticated models encounter ambiguity and may reason about relationships that no longer exist, causing errors. The 100% loss quantifies this extreme trade-off, validating our statistical finding that it favors smaller models (67% improvement, p = 0.025) while disadvantaging larger models.

5.3.4 Semantic Loss Explains Divergence.

Our analysis confirms 100% loss of enforceable FDs (Table 5.5). **Small models improve** because C_R reduction (2 FKs $\rightarrow$ 0) eliminates multi-hop reasoning requirements, as in the following example: (`Examination.ID → Patient.ID → Patient.Birthday`), and C_A reduction (joins $\rightarrow$ 0) co-locates all required data. **Large models degrade** because 100% FD loss eliminates exploitable semantic structure that being foreign keys that signal join requirements, primary keys indicating deterministic relationships, and cardinalities guiding query optimization. The magnitude of semantic loss (100%) correlates with the magnitude of divergence (+69.9% for smallest vs. -20.2% for largest). This confirms that the trade-off is inherent to denormalization principles.

Model	**# DBs that improved**	**# DBs that worsened**	**# DBs draw**
gemma3:1b	4	4	3
gemma3:4b	7	4	0
granite3.3:2b	8	2	1
llama3.1:8b	7	2	2
llama3.2:1b	7	3	1
llama3.2:3b	6	4	1
mistral:7b	7	2	2
qwen2.5-coder:0.5b	6	4	1
qwen2.5-coder:1.5b	10	1	0
qwen2.5-coder:3b	6	5	1
qwen3:0.6b	5	5	1
qwen3:1.7b	6	5	0
qwen3:4b	2	9	0
qwen3:8b	3	8	0

Table 5.6: Count of DBs that improved/worsened/stayed the same in text-to-SQL translation

Table 5.6 shows us that 10/14 models improved across the majority of 11 databases. These databases span diverse domains (medical records, school performance, business planning) with varying complexity (3-10+ tables). The consistent pattern of small model improvement across this heterogeneity indicates our principles apply broadly, not just to specific database structures.

Implications for Future Work: When we talk about distinct databases that improved, we have very positive results: 10 models having majority of databases with better performance, 2 models with equal improved/worsened counts, and just 2 models with more DBs performing worse. This distribution across majority of databases for majority of models is excellent indication that the method can generalize to varied DB types. This is of extreme importance, as it shows future improvements can benefit broader range of database schemas. Within the dev BIRD benchmark, we encounter significant heterogeneity in database complexity, ranging from minimal tables to over ten tables with extensive columns. This analysis provides substantial confidence that our Distillation methodology demonstrates robust capability in accommodating this considerable diversity in database architectures, suggesting inherent flexibility for real-world database environments. It is important to acknowledge that certain databases do not exhibit performance improvements. This heterogeneous response indicates specific database characteristics may limit current distillation effectiveness. In following iterations we will investigate underlying factors contributing to these discrepancies, focusing on identifying structural, semantic, or complexity-related attributes correlating with suboptimal distillation outcomes.

Critically, these results should be read as proof of concept. Across our experiments we show that schema manipulation can meaningfully improve smaller models (e.g., 67% of models with ≤3B parameters improved, with a mean relative gain of 32.27% among the ones that improved), but we also acknowledge that a substantial gap to flagship systems remains.

5.4 COMPARISON WITH LEADING SOLUTIONS

To contextualize the performance of our Schema Distillation method and establish realistic expectations for this proof-of-concept approach, we compare our best results using distilled schemas against the current state-of-the-art solutions on the BIRD benchmark leaderboard and show it in Table 5.7. This comparison is not intended to claim competitive performance with these systems, but rather to transparently acknowledge the substantial performance gap between our small-model approach and flagship-model solutions, while also illustrating the different use cases and trade-offs (accessibility and cost vs. maximum accuracy), and also motivate the research direction needed to transform Schema Distillation from proof of concept to competitive alternative.

Our method achieves significantly lower absolute performance compared to leading solutions, llama3.1:8b with distilled schema reached 19.29% EX, representing approximately 26-28% of top-ranking systems' performance. However, this requires critical contextualization. First, leading solutions predominantly employ flagship models with substantially larger parameter counts and computational requirements, leveraging closed-source models like GPT-4 and Gemini that require significant resources. In stark contrast, our approach deliberately targets smaller

Model/System	Organization	Dev (%)
Agentar-Scale-SQL	Ant Group	74.90
AskData + GPT-4o	AT&T CDO - DSAIR	76.14
LongData-SQL	LongShine AI Research	74.32
CHASE-SQL + Gemini	Google Cloud	74.90
JoyDataAgent-SQL	JD:CHO-JDT-JDL	74.25
llama3.1:8b + Schema Distillation	**Our Method**	**19.29**
llama3.1:8b	**Original model**	**18.31**
qwen3:0.6b + Schema Distillation	**Our Method**	**5.54**
qwen3:0.6b	**Original model**	**3.71**
llama3.2:1b + Schema Distillation	**Our Method**	**5.08**
llama3.2:1b	**Original model**	**2.99**

Table 5.7: Top 5 performing solutions on BIRD benchmark (Dev set) compared to our best distilled results and original models

open-source models (1B to 8B parameters) deployable locally on consumer-grade hardware, addressing fundamentally different use cases.

Second, our method successfully achieves its primary research objective: demonstrating that schema simplification meaningfully improves smaller model performance. Approximately 70% of evaluated models showed improvement with the distilled schema, with gains reaching 69.9% for the smallest models (llama3.2:1b), validating our core hypothesis.

Third, the economic implications become particularly compelling. While systems like Agentar-Scale-SQL achieve 74% accuracy, they require substantial infrastructure and significant API costs. Our Schema Distillation requires only a single call to a large model for schema simplification, followed by inference on a substantially cheaper, locally deployable model. For applications where cost-performance trade-off favors accessibility and data privacy over maximum accuracy, our approach presents a pragmatic alternative.

From Proof of Concept to Competitive Performance. The current results establish Schema Distillation as a validated proof of concept but substantial advancement is required to transform this proof of concept into a competitive alternative. The critical research direction forward is understanding how to systematically bridge this performance gap while preserving the economic and accessibility advantages. The ultimate goal is to develop Schema Distillation into a technique enabling mid-sized models (7B-13B parameters) to approach or exceed current flagship model solutions' performance democratizing access to high-quality text-to-SQL systems and enable organizations with data privacy requirements or budget constraints to deploy effective solutions locally. The current work establishes the foundation and validates the core premise; the challenge ahead is systematic refinement to close the performance gap.

6 CONCLUSION

This research sought to understand how database schema characteristics affect text-to-SQL translation with open-source language models. We specifically investigated whether schema simplification could make smaller models more competitive. The BIRD benchmark leaderboard is dominated by closed-source LLMs like GPT-4 and Gemini. This creates financial barriers for reproduction. We focused on open-source models (0.5B to 14B parameters) widely accessible to the research community.

The problem had two phases. First, understanding what makes schemas hard for LLMs: poor naming, structural complexity, or both. Second, testing whether simplifying schemas could help smaller models compensate for limited capacity. This led to Wrecking SQL, which systematically degraded schemas to isolate naming effects. It also led to Schema Distillation, which simplified schemas through denormalization.

Initially, we analyzed schema nomenclature impacts across different variants. Throughout our investigation, we noticed that column names matter significantly more than table names. Table name degradation caused modest performance drops (10-20% relative decrease). Some models even improved slightly. This suggests LLMs can navigate databases reasonably well with suboptimal table naming.

But column naming is a completely different story. Performance crashed by 60-75% when column names became meaningless. Llama 3.1's accuracy on simple questions fell from 30% to 6%. That's an 80% relative decline. This pattern held across all models and difficulty levels, even the simplest queries. The practical implication is clear: if deploying text-to-SQL on legacy databases, focus schema improvement efforts on column names first.

Given column name criticality, we tested automatic inference from table content using two approaches: simple inference (column content only) and complete inference (column content plus row context). Most models improved by 2-4 percentage points with inferred names. Complete inference generally outperformed simple inference. However, even with our best approach, performance remained far below default schemas. It recovered only 25-30% of lost performance. The problem is domain-specific terminology. Inference struggles when column names encode niche domain knowledge.

Concerning Schema Distillation, we tested our main contribution by systematically simplifying schemas through denormalization. The approach eliminates foreign keys, flattens join paths, and creates specialized views that co-locate related data. When restricting analysis to models $\leq$3B parameters (N=9), Schema Distillation produces statistically significant improvements: $p = 0.025$, Cohen's $d = 0.97$, with 67% showing performance gains and mean relative improvements of 32%. The smallest models benefit most dramatically: llama3.2:1b improved 69.9%, qwen3:0.6b gained 49.3%, granite3.3:2b improved 26.16%.

However, larger models (>3B parameters) showed mixed results. Several degraded significantly. Qwen3:8b dropped 20.2%, qwen3:4b fell 14.0%. This wasn't a bug. It's a fundamental trade-off. Our functional dependency analysis revealed that distillation eliminates 100% of enforceable functional dependencies. Smaller models benefit from cognitive simplification, while larger models lose exploitable semantic structure. It is notable that generalization across databases was encouraging. Ten models showed improvement across the majority of 11 tested databases spanning diverse domains.

Regarding comparison with state-of-the-art, our best result (llama3.1:8b with Schema Distillation, 19.29%) represents roughly 26% of top-ranking systems' performance. That's a massive gap. Leading solutions use flagship models achieving 74-77% accuracy with substantial infrastructure and API costs. We're targeting 1B-8B parameter models deployable locally. These aren't competing products. They're solutions for fundamentally different use cases.

Finally, the experimental results demonstrate both promise and limitations. Schema Distillation successfully shows that schema simplification meaningfully improves smaller model performance. This achieves our research objective. Our results showed that each optimization technique tested had positive and negative points. Combined use obtained the best results. We demonstrated three main contributions: (1) column naming is non-negotiable for text-to-SQL translation, degrading performance by 60-75% when poor; (2) schema simplification helps small models ($p = 0.025$, $d = 0.97$) but hurts large models through fundamental trade-offs; and (3) the approach generalizes across diverse database types.

This is a proof of concept, not a finished product. The critical research direction forward is understanding how to systematically bridge the performance gap while preserving economic and accessibility advantages. The ultimate goal is developing techniques enabling mid-sized models (7B-13B parameters) to approach flagship solutions. This would democratize access to high-quality text-to-SQL systems for organizations with data privacy requirements or budget constraints. The challenge ahead is transforming Schema Distillation from validated proof of concept into competitive technique. We've established that the principle works. Now comes the engineering.

REFERENCES

Allen, J. F. (1995). *Natural Language Understanding*. Benjamin/Cummings Publishing Company.

Androutsopoulos, I., Ritchie, G. D., and Thanisch, P. (1995). Natural language interfaces to databases – an introduction. *Natural Language Engineering*, 1(1):29–81.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Calero, C., Piattini, M., and Genero, M. (2001). Database complexity metrics. In *Proceedings of QuaTIC 2001*, pages 79–86.

Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 2nd edition.

Cvijetić, B. and Radivojević, Z. (2023). Restoration of data structures using machine learning techniques. *IEEE Access*, 11:1–14.

Dani, C., Jahangiri, S., and Hütter, T. (2024). Introducing schema inference as a scalable SQL function [extended version]. *arXiv preprint arXiv:2404.10810*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

Docini, R., de Almeida, E. C., and Oliveira, L. E. S. (2025). The Wrecking SQL incremental validation methodology. In *Database and Expert Systems Applications*, pages 376–382, Cham. Springer Nature Switzerland.

Frozza, A., Defreyn, E., and Mello, R. (2020). A process for inference of columnar NoSQL database schemas. In *Anais do XXXV Simpósio Brasileiro de Bancos de Dados*, pages 175–180, Evento Online.

Gao, Y., Liu, Y., Li, X., Shi, X., Zhu, Y., Wang, Y., Li, S., Li, W., Hong, Y., Luo, Z., Gao, J., Mou, L., and Li, Y. (2025). A preview of XiYan-SQL: A multi-generator ensemble framework for text-to-SQL. *arXiv preprint arXiv:2411.08599*.

Hariharan, N. and Reeshma, K. (2015). Challenges of core banking systems. *Mediterranean Journal of Social Sciences*, 6(5):24.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2021). LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall.

Klisura, D. and Rios, A. (2024). Unmasking database vulnerabilities: Zero-knowledge schema inference attacks in text-to-SQL systems. *arXiv preprint arXiv:2406.14545*.

Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., et al. (2024). Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. *Advances in Neural Information Processing Systems*, 36.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Luoma, K. and Kumar, A. (2025). SNAILS: Schema naming assessments for improved LLM-based SQL inference. *Proceedings of the ACM on Management of Data*, 3(1).

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Shin, S. K. and Sanders, G. L. (2006). Denormalization strategies for data retrieval from data warehouses. *Decision Support Systems*, 42(1):267–282.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2):257–285.

Trabelsi, M., Cao, J., and Heflin, J. (2021). SeLaB: Semantic labeling with BERT. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. (2022). Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Xu, X., Li, M., Tao, C., Shen, T., Cheng, R., Li, J., Xu, C., Tao, D., and Zhou, T. (2024). A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. (2019). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.