mathematics

MDPI

Article

Beyond Standard Losses: Redefining Text-to-SQL with Task-Specific Optimization

Iker Azurmendi [1,2], Ekaitz Zulueta [1], Gustavo García [2], Nekane Uriarte-Arrazola [1,2] and Jose Manuel Lopez-Guede [1,*]

[1] Department of Systems and Automatic Control, Faculty of Engineering of Vitoria-Gasteiz, University of the Basque Country (EHU), Nieves Cano, 01006 Vitoria-Gasteiz, Spain; iker.azurmendi@ehu.eus or iazurmendi@mc3.es (I.A.); ekaitz.zulueta@ehu.eus (E.Z.); nuriarte024@ikasle.ehu.eus or nuriarte@mc3.es (N.U.-A.)
[2] MC3 Mondragon Componentes Competence Center, Avda. Álava 3, 20550 Aretxabaleta, Spain; ggarcia@mc3.es
* Correspondence: jm.lopez@ehu.eus

Abstract

In recent years, large language models (LLMs) have shown an impressive ability in translating text to SQL queries. However, in real-world applications, standard loss functions frequently fail to capture the complexity of queries adequately. Therefore, in this study, a dynamic loss function is proposed, which assigns different weights to specific groups of tokens, such as SQL keywords or table names. The objective is to guide the model during training to facilitate the mastery of more fundamental concepts within the SQL. Our custom loss function is composed of four components: cross-entropy with sequence matching loss, focal loss, F-beta loss, and contrastive sequence loss. During the training process, the weights of each component of the loss function are dynamically adjusted to prioritize different aspects of query generation at the appropriate stage. This approach avoids computationally expensive approaches such as SQL validation or detokenization, which improves the efficiency of the learning process compared to alternative methods. We empirically tested this method on several open source LLMs with less than 2 billion parameters, using a customized real vehicle diagnostic dataset. The findings demonstrate that the employment of our dynamic loss function can enhance SQL execution accuracy by up to 20% in comparison with standard cross-entropy loss. It has been demonstrated that customized loss functions for specific tasks can improve the efficiency of LLMs without extending the model or acquiring additional labelled data. The proposed technique is also scalable and adaptable to new domains or more complex weighting schemes, highlighting the importance of custom design of loss functions in real world applications.

Keywords: text-to-SQL; natural language processing; large language models; database querying; custom loss; dynamic weighting

MSC: 68T50

Academic Editor: Chengjie Sun

Received: 4 June 2025
Revised: 7 July 2025
Accepted: 17 July 2025
Published: 20 July 2025

Citation: Azurmendi, I.; Zulueta, E.; García, G.; Uriarte-Arrazola, N.; Lopez-Guede, J.M. Beyond Standard Losses: Redefining Text-to-SQL with Task-Specific Optimization. *Mathematics* **2025**, *13*, 2315. https://doi.org/10.3390/math13142315

1. Introduction

In the domain of natural language processing (NLP), text-to-SQL is defined as the process of translating natural language queries into Structured Query Language (SQL) commands. This facilitates the interaction with databases in the preferred language of the user, avoiding the need to manually enter SQL queries. As Mohammadjafari et al. [1]

discuss, the objective is to eliminate the necessity of SQL proficiency to access databases. This task is crucial in the democratization of data access, as it facilitates the translation of human language into machine-readable query formats [2,3]. The workflow of a typical text-to-SQL conversion system consists of three main steps (see Figure 1). Initially, a user asks a question in natural language regarding the contents of a database. Secondly, an artificial intelligence (AI) model is employed to translate this question into an SQL query, taking into account the database schema. Finally, the SQL query that has been generated is executed on the database to obtain the desired result. Moreover, the implementation of multi-conversation memory and error-correction techniques has the potential to enhance this process [4–6].

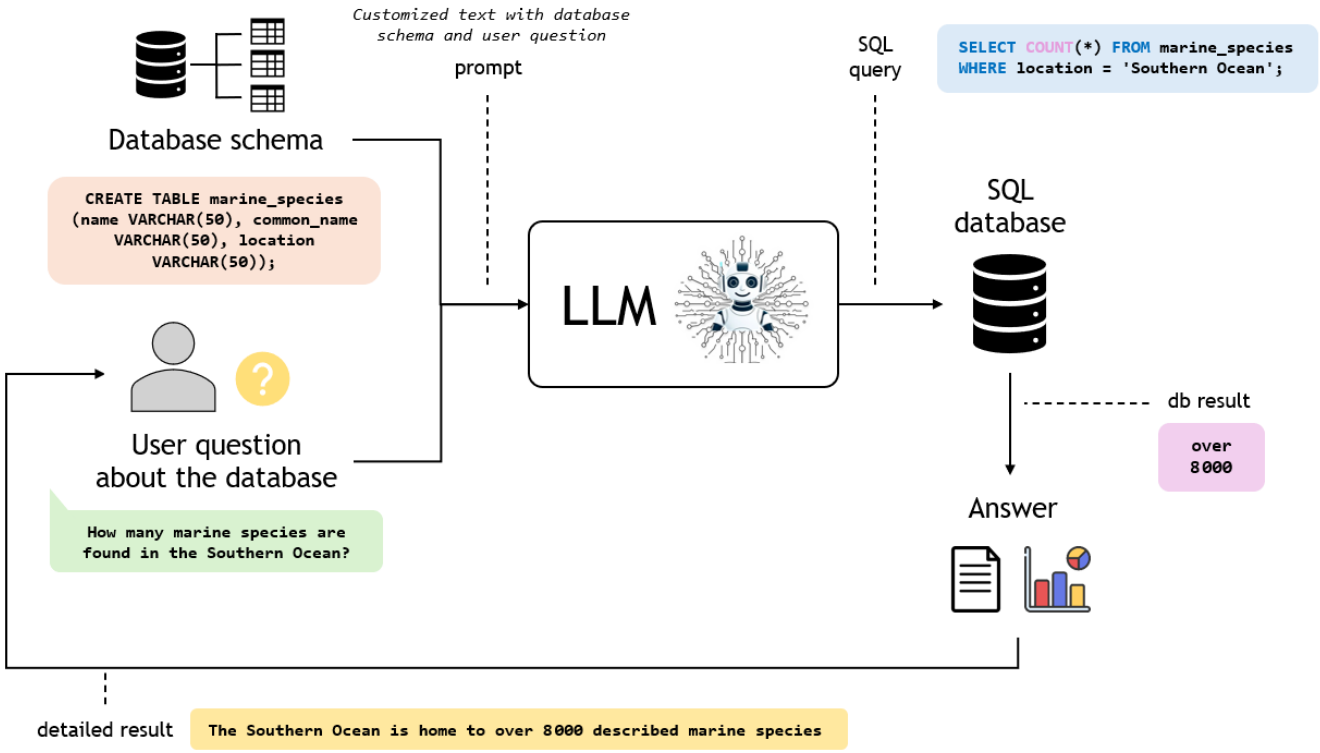

Figure 1. Simple text-to-SQL workflow with general example of synthetic text-to-SQL dataset [7].

The way text-to-SQL systems have developed has changed a lot. They started with simple rule-based methods and now use modern Deep Learning (DL) techniques. As Xiaohu Zhu et al. [8] explain, the history of converting text into SQL goes back to 1973. That year, Kaplar and Webber [9] developed a system called LUNAR. This was primarily used to answer questions about rocks brought back from the Moon. Afterwards, these models were based on grammatical rules that were hand-crafted and specific to a given area. This meant that they could not be used more generally in other types of applications [8]. More recently, DL models have become much better at handling large amounts of data, using neural network structures and Large Language Models (LLMs), making them really interesting for translating text to SQL queries [10]. Table 1 shows a review of text-to-SQL techniques over the years.

Table 1. Evolution of text-to-SQL techniques: from template-based to large language models.

Technique	Description	Refs
Template-based	Earlier systems often relied on hand-crafted rules, but these struggled with complexity and domain shifts	[11]
Sequence-to-sequence Models	Neural architectures treat SQL statements as a sequence to be generated from the user's question; however, they may overlook the structured nature of the database or the formal constraints of SQL	[12–14]
Grammar-based decoding	These methods factorize SQL into syntax components (SELECT, WHERE, GROUP BY, etc.) and predict each subcomponent separately, enforcing SQL consistency	[15,16]
Graph Encoding and Schema Linking	To handle the complexity of multi-table schemas and to better align question tokens with columns/tables, graph neural networks and linking mechanisms have been adopted	[17,18]
Large Language Models	Recent work uses large language models such as GPT or instruction-tuned transformers, relying on in-context learning or fine-tuning to parse complex queries with minimal additional supervision	[8]

Despite the advances summarized in Table 1, LLM-based text-to-SQL conversion systems continue to show poor performance in the most complicated cases [19]. This is mainly due to the complexity of SQL syntax, the need for task-specific schemas and the need for detailed management of logical operations (e.g., joins or aggregations). Recent advances in LLMs have demonstrated the potential to address some of the most important challenges; however, performance shortcomings persist, especially in the context of long queries or the prioritization of critical tokens during SQL execution.

For example, as Hong et al. [20] point out, pre-trained language models with a small number of parameters often generate incorrect SQL for complex queries. Moreover, LLMs have been observed to experience hallucinations when dealing with schema names and may also omit necessary joins or violate logical constraints when faced with extended queries involving multiple joins or domain-specific vocabulary [21].

The main problem is that traditional model tuning objectives (e.g., token-level cross-entropy) do not directly guarantee logical consistency, schema alignment, or domain-specific correctness. As Hong et al. [20] observe, LLM performance on text-to-SQL tasks is inherently limited by the reasoning capabilities of the underlying models. Simply put, current training programmes are ineffective in transferring knowledge from SQL generation to execution, especially in specialized domains.

In order to address this issue, several strategies for improving LLMs in the text-to-SQL task have been explored in the literature:

- Length-based training: Long queries cause problems because of error propagation in autoregressive generation. So, recent studies have tried to solve this problem by adding curriculum learning [22]. The plan is to start by teaching models how to predict short and simple questions. Then, as training continues, the model learns how to answer more complex and longer samples. In this paper, the RASAT sequence-to-sequence model is used, which is built on the T5 architecture, enhanced with relation-aware self-attention to capture structural relationships in SQL queries and database schemas.

- Hybrid architectures: Some researchers, such as Berdnyk and Colley [23] and Nguyen et al. [24], use reinforcement learning (RL) [25,26] to improve LLMs' ability to convert text into SQL. To achieve this, they give the model rewards to help it make correct SQL queries that work properly on the database execution. On the one hand, Berdnyk and Collery use flan-t5-base [27] as the primary LLM for SQL generation and LLaMa-3-405B-Instruct LLM to reward function design. On the other hand, Nguyen et al. use T5-small and T5-base [28], allowing them to train and deploy on standard user hardware rather than requiring specialized cloud infrastructure.
- Workflow modification: Yuanzhen Xie and his team [29] concentrate on the workflow paradigm, which aims to improve how well and how widely LLMs solve problems through decomposition. They use OpenAI ChatGPT-3.5 and GPT-4 as their base models to carry out the training sessions. This technique uses the information determination module to get rid of unnecessary information. It also uses a new prompt structure based on problem classification, which improves how the model focuses. Also, there are self-correction and active learning modules. The idea is to make LLM problem solving more extensive.
- Two-stage learning: Ling Xiao et al. [30] present a method that divides training into two phases. In the first phase, the system understands the schema. In the second phase, the system generates the SQL query. According to the authors this has allowed them to make the model much better by making sure that the way the data is organized is correct before they start working on the questions. This approach also tries to reduce errors in complex queries involving joins or aggregations.

On the other hand, besides changing the architecture of the model or the nature of the training loop, there is a rapidly developing research field that attempts to modify the loss function to change the learning objectives of the model. This approach has resulted in improved LLMs in many NLP tasks, although it has mostly been studied in areas unrelated to text-to-SQL conversion. For example, consistency-based linguistic models (CLLMs) [31] combine the standard cost function with a new loss term. This new term forces the model to reach a consensus state at different decoding orders [8]. As the authors explain, this means combining the usual autoregressive loss (cross-entropy) with a consistency loss that reduces the variation in the model representations, helping to avoid making contradictory predictions. In other studies, dynamic weighting has been suggested to make examples with high loss values more important during training, helping the model to focus on the most difficult cases [20,32].

In addition, Wang et al. [33] came up with the MinorSFT loss for supervised fine-tuning (SFT). This loss reduces the differences in how a pre-trained LLM behaves during training. Furthermore, the FLAT (Forget data only Loss AjustmenT) method by Wang et al. [34] introduces a novel way to adjust the loss by trying to make the best use of the forgotten data. The idea is to allow the unlearning of data without losing the ability of the model to generalize. Finally, recent studies have also added focal loss [35] in LLMs to help with the calibration of weights. This loss was designed to help with detecting objects in computer vision tasks. However, in large linguistic models, it tries to avoid making one type of information more important than another and enhances the system when dealing with different types of tasks [36].

These examples and techniques demonstrate that modifying the loss function can improve the training of models and optimize the quality of their output in specific applications. In summary, the idea is that the modifications to the loss functions provide a simple way to directly integrate structured and semantic criteria into the optimization objective of the text-to-SQL translation task. The goal is to guide the model beyond basic

token prediction. Even so, these techniques remain relatively underexplored in text-to-SQL conversion tasks, suggesting an opportunity for innovation in this area.

For this reason, this paper proposes a custom loss function that automatically updates the weights during training to better guide the generation of SQL queries by LLMs. In contrast to previous methodologies, which rely on modifying the input data, adding auxiliary training steps or modifying model architectures, our approach preserves the standard training process while guiding the model for the optimization objective task. Thus, a dynamic, multi-component loss is proposed that weighs different aspects of SQL correctness (e.g., schema alignment, logical consistency, clause coverage, tables/columns, etc.) according to user criteria.

This approach aims to concentrate model learning on the most informative or error-prone parts of the output. As we demonstrate, a personalized loss not only complements existing techniques (e.g., reinforcement learning or multitask training) but also provides a simple, straightforward, domain-independent training method that can improve performance accuracy and reduce semantic errors in model predictions. Building upon these insights, our work introduces a novel dynamic loss function tailored specifically for text-to-SQL tasks involving LLMs.

Our proposed loss function combines four losses that are dynamically adjusted during training:

- Sequence-Matching Cross-Entropy Loss: Extends standard cross-entropy by weighting important token sequences (e.g., table names, error codes), allowing flexibility in their positions.
- Focal Loss: Addresses class imbalance by focusing on hard-to-predict tokens.
- F-beta Loss: Optimizes accuracy and recovery for critical token sequences, emphasizing recovery as training progresses.
- Contrastive Sequence Loss: Ensures correct relative distances between token sequences, preserving SQL structure.

By dynamically updating these components throughout training, our method progressively prioritizes different query generation scenarios: for example, it initially targets fundamental correctness of SQL queries, before progressively emphasizing more complex structures as training progresses. As discussed above, this method is compatible with the standard training process and avoids the computational overhead associated with methods such as reinforcement learning. It is also particularly suitable for specific tasks, such as the vehicle diagnostics project examined in this research. Our contributions are as follows:

- We propose a new dynamic loss function specifically adapted to text-to-SQL conversion tasks. On the one hand, this loss combines standardized components in LLM training with novel token sequence-level targets (as opposed to the standard token-to-token target) with innovative components adapted from other AI tasks to emphasize structural and semantic correctness. On the other hand, custom weights are introduced in various aspects: static weights for important groups of words to emphasize aspects of our database that are considered most important; dynamic weights for the individual components of the cost function to focus on different parts of SQL as the tuning progresses.
- Our approach introduces schema alignment loss and logical consistency terms that adaptively focus on the most error-prone aspects of SQL summarization. Unlike other methods such as curriculum learning, which requires the stepwise preparation of data, our method automatically incorporates length-aware training. Moreover, in contrast to reinforcement learning approaches, it stays within the standard supervised learning process, which simplifies its implementation and deployment.

- Our proposed method is evaluated on multiple open-access models of less than 2B parameters, achieving more than 20% more improvement in some of the cases.

The rest of this paper is structured as follows: Section 2 describes the used dataset, the standard cross-entropy loss, and our proposed dynamic custom loss; Section 3 presents experimental results; Section 4 discusses the results in Section 3; and finally, Section 5 concludes this paper and explores future directions.

2. Materials and Methods

2.1. Dataset

The data used for this study was based on a customized set of actual diagnostic and telemetry information from a fleet of connected vehicles stored in an SQL database. Then, multiple pairs of natural language Spanish queries were generated with their corresponding SQL Server queries, which were executed on a real database to extract the requested information. In total, the data used for this study has 250 training and validation samples and 50 samples for the final evaluation on a real server. The augmentation of the training set was also implemented to enhance the robustness of the model. In summary, the dataset under consideration contains 300 raw samples that were manually labelled. Some examples of the custom text-to-SQL database are provided in Table 2.

Table 2. Examples of user questions and their corresponding SQL Server queries in the custom text-to-SQL database.

User Question	SQL Query
Give me the top 10 vehicles with the most 'UH0043' failures in 2021	SELECT TOP 10 Vehicle_Id, COUNT(*) AS FailureCount FROM Failure_Codes WHERE Error_Code = 'UH0043' AND YEAR(Datetime) = 2021 GROUP BY Vehicle_Id ORDER BY FailureCount DESC;
Could you tell me the top 5 vehicles that have been to the car workshop?	SELECT TOP 5 Vehicle_Id, COUNT(*) AS TotalEvents FROM Vehicle_Events GROUP BY Vehicle_Id ORDER BY TotalEvents DESC;
Which vehicle has been driven the longest distance?	SELECT TOP 1 Vehicle_Id, Distance FROM Fleet ORDER BY Distance DESC;
Show me what percentage of failures occur in each month of 2024	SELECT MONTH(Datetime) AS Month, COUNT(*) AS Quantity FROM Failure_Codes WHERE YEAR(Datetime) = 2024 GROUP BY MONTH(Datetime);

The dataset focuses on real-world queries related to vehicle performance, error codes, workshop visits and sensor readings (speed, position, consumption). The database has queries across 10 tables, more than 100 columns that are interconnected between tables, more than 1000 vehicles, and more than 100 types of faults. The SQL queries were designed to extract information such as vehicle failures, braking frequency, and other diagnostic insights. The dataset includes both simple and complex queries, involving operations such as aggregation (COUNT, MAX, MIN), filtering (WHERE conditions), grouping (GROUP BY), ordering (ORDER BY), and ranking (TOP N).

In addition, augmentation techniques were applied to increase data diversity during training, including paraphrasing user queries, varying SQL query structures while preserving semantics, and introducing slight modifications in numerical constraints (e.g., different

year filters or ranking limits). These augmentations aim to improve generalization and adaptability to new, unseen queries. Figure 2 shows the data augmentation process carried out for the database examples. As can be seen, the raw examples have placeholders that can take various contextual synonyms when augmenting the examples. Thus, placeholders (e.g., $failure$, $year$) are variables that are used to generate various domain-specific natural language queries by replacing them with relevant database values. For example, the placeholder $failure$ can correspond to any of the more than 100 types of failures available in the database.

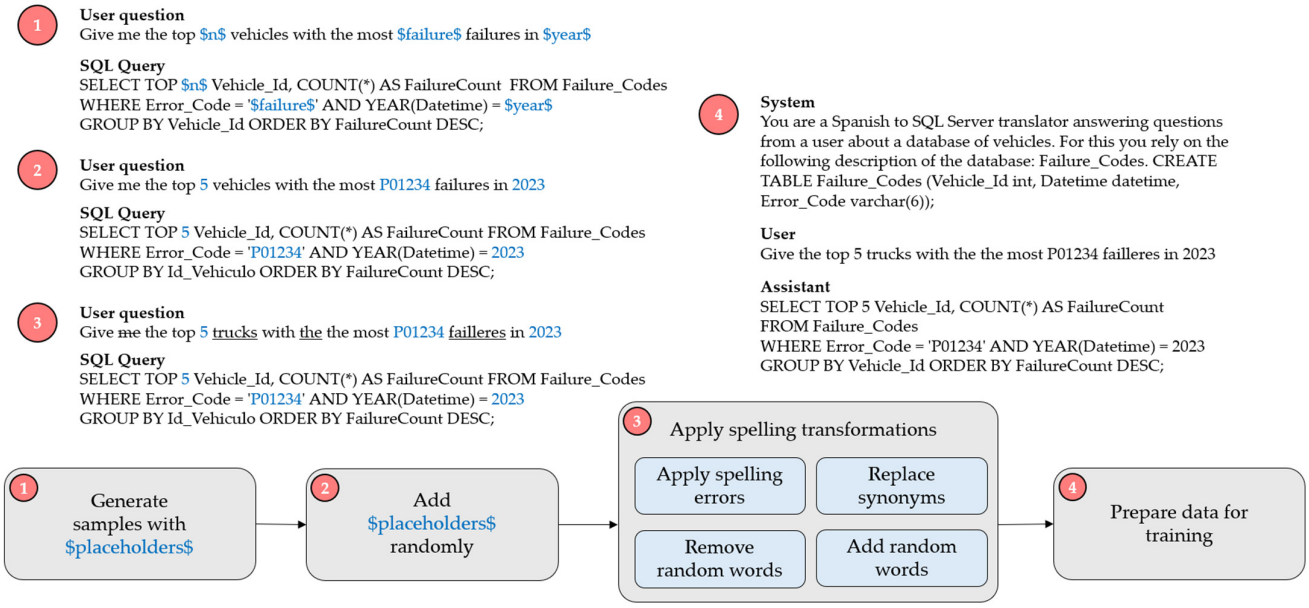

Figure 2. Data augmentation pipeline: (1) User question and its corresponding SQL query are manually generated with placeholders. (2) Placeholders are placed to generate custom database-specific samples. (3) Linguistic variations are introduced by applying grammatical errors, replacing words with synonyms, removing words, or adding extra words to simulate noisy user input (in the example, the crossed-out words have been removed from the new example, while the underlined words correspond to new words, words with grammatical errors or contextual synonyms). (4) Samples are prepared to fit the model in training.

In this way, this dataset serves as the foundation for evaluating the proposed task-specific optimization approach, ensuring that the model is tested on realistic, domain-relevant queries that reflect practical use cases in vehicle diagnostics.

2.2. Loss Function

In the field of Machine Learning (ML), a loss function constitutes a method of evaluating the performance of a specific model. This is achieved by calculating the discrepancy between the predicted and actual outputs of the model [37].

Within the LLM paradigm, the loss function quantifies the discrepancy between the probability distribution of the next word in a sentence predicted by the model and the actual distribution observed in the training data. The quality of the model predictions depends on the minimization of the loss function [38].

Large language models are trained to predict the next word in a sequence, a task that is closely related to linguistic modelling. The process involves the generation of a probability distribution for a given vocabulary. The standard loss function employed in this context is known as cross-entropy loss. This loss is a metric that quantifies the discrepancy between model predictions and the actual posterior token, making it optimal for this probabilistic

task. Cross-entropy loss is a specific type of loss function that is particularly well suited to the comparison of probability distributions, which is the reason for its use in LLMs [39].

In mathematical terms, cross-entropy is defined for two discrete probability distributions, P (the true distribution of the data) and Q (the distribution predicted by the model), as follows [37,40]:

$$L_{CE}(P, Q) = -\sum_{i=1}^{n} P(i) log Q(i) \quad (1)$$

where n is the number of classes (vocabulary size in LLMs), $P(i)$ is true distribution of the data, and $Q(i)$ is the predicted probability for class i. In language modelling, this translates to predicting the next word w_t, given the history $w_1, w_2, \ldots, w_{t-1}$.

In the context of LLMs, $P(i)$ is the actual probability of a word occurring next in a sentence, and $Q(i)$ is the model's predicted probability. The log function was used to calculate the information content of each prediction. These values were then multiplied by the actual probabilities and added together to give the overall cross-entropy loss. The cross-entropy loss makes the model less likely to give low probabilities to the right tokens. This makes the model learn to create representations that closely match the target distribution [38,41].

In short, cross-entropy loss is the standard way to train large language models. It is an effective and robust way to optimize tasks that involve generating sequences. Its alignment with maximum likelihood estimation, combined with its scalability and versatility, has established it as an essential tool in developing state-of-the-art LLMs.

2.3. Dynamic Custom Loss Function

In order to enhance the performance of large language models on domain-specific text-to-SQL tasks, a dynamic custom loss function was proposed. This function integrates multiple loss components, each of which was designed to address unique challenges in generating accurate and structurally valid SQL queries. This approach helps the model to focus on important token sequences (e.g., SQL keywords, table names, or specific conditions) while still working well with different query structures. The custom loss function has four parts: a sequence-matching cross-entropy loss, a focal loss, an F-beta loss, and a contrastive sequence loss. The components were weighted dynamically based on training progress, with the idea of balancing learning objectives over time.

Furthermore, custom weightings were used to highlight important token sequences, regardless of where they are in the predicted SQL query. Unlike a cross-entropy with token-level weights [42], our work extrapolates it to the level of token sequences. This approach leverages the commutative nature of certain SQL constructs, such as logical conjunctions or condition order, ensuring that semantically equivalent but syntactically varied queries are not unfairly penalized. The following 9 groups were weighed during the custom training of the LLM for our custom text-to-SQL task: tables, columns, vehicle identifiers, errors, workshops, vehicle type, vehicle group, vehicle model and vehicle body. The weight attributed to each group was determined by the importance it is considered to have by the user. The pipeline used to detect important sequences is illustrated in Figure 3.

The aim is to make sure the model is accurate at the token level and that the generated SQL queries are structurally and semantically correct. This means focusing on the presence and placement of important sequences, such as error codes, vehicle IDs, or workshop names. The following section details the components of the loss function and their implementation basis.

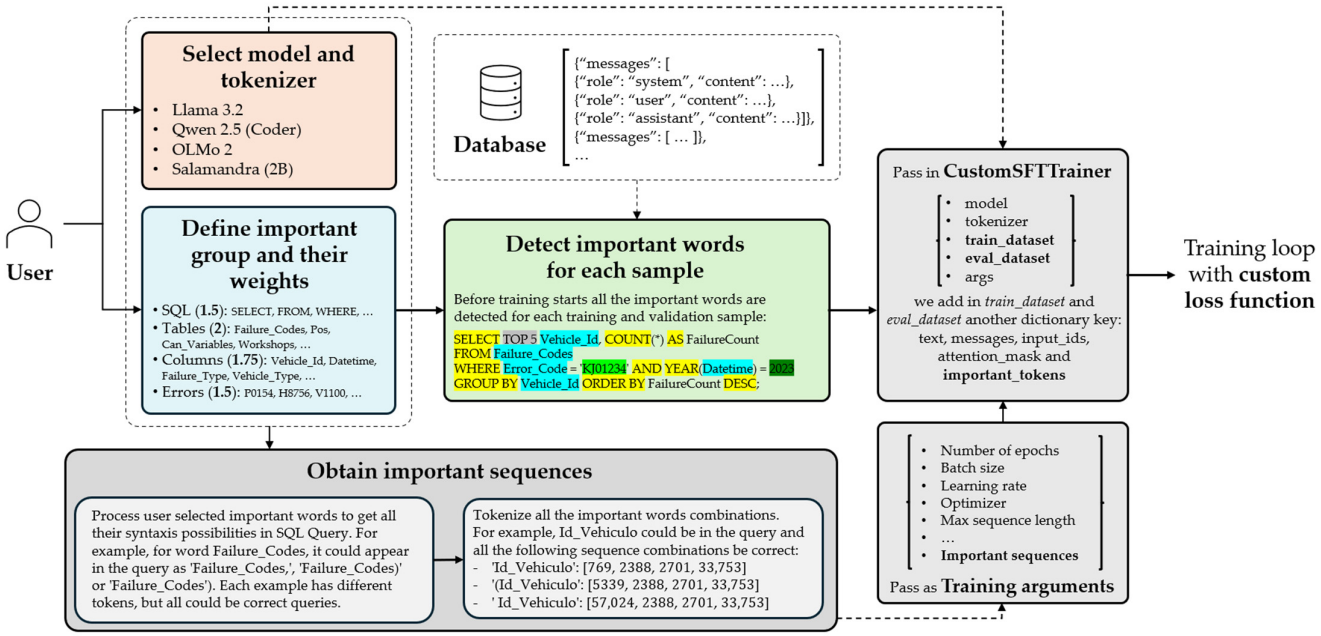

Figure 3. Custom important sequence identification pipeline. First, the user selects the model to be trained and defines the important groups with their associated weights to the database. Then, the important user-defined groups are processed to obtain all possible combinations of tokens. Additionally, all user-defined words and word groups in the training and validation data are detected (the different color highlights indicate different groups of words). Finally, the model is trained to optimize the cost function with respect to the important token sequences selected by the user.

2.3.1. Sequence-Matching Cross-Entropy Loss

The sequence-matching cross-entropy loss is an extension of the standard cross-entropy loss (Equation (1)), incorporating weights that emphasize the presence of important token sequences, even if they appear in different positions than in the target SQL query. This is a particularly useful function for SQL generation, where the order of certain clauses (e.g., WHERE conditions) may vary while preserving semantic correctness. The sequence-matching cross-entropy loss is defined as follows:

$$L_{SMCE} = \frac{1}{N}\sum_{b=1}^{B}\sum_{t=1}^{T-1}[w_{b,t}\cdot L_{CE}(\hat{y}_{b,t}, y_{b,t})\cdot \updownarrow_b] \quad (2)$$

where B is the batch size, T is the sequence length, $\hat{y}_{b,t}$ and $y_{b,t}$ are the predicted and true tokens at position t in batch b, $w_{b,t}$ is a weight based on the presence of important sequences, $\updownarrow_b$ is a length-based weighting factor, and N is the number of valid tokens. The weights $w_{b,t}$ were increased for positions corresponding to important tokens (e.g., table names or error codes) if their sequences were absent or underrepresented in the predictions, and they were computed by iterating over important token sequences provided in the input *important_tokens* (Figure 3) and checking their presence in the predicted tokens.

Additionally, the length weight $\updownarrow_b = 1 + \left[log\left(1 + \frac{L_b}{L_{max}}\right) - 1\right]\cdot\lambda$ penalized longer sequences to encourage concise SQL queries, where L_b is the query length, L_{max} is the maximum sequence length, and $\lambda = 1 + progress$ increases with training. This term was applied per batch to scale the loss based on query length, promoting generalization to varying query complexities. In short, $\updownarrow_b$ penalizes longer sequences to a greater extent as training progresses.

2.3.2. Focal Loss

Focal loss was originally presented in 2017 to address class imbalance during training in tasks like object detection [35]. Focal loss applies a modulating term to the cross-entropy loss to focus learning on hard misclassified examples [43]. It is a dynamically scaled cross-entropy loss, where the scaling factor decays to zero as confidence in the correct class increases. Intuitively, this scaling factor can automatically down-weigh the contribution of easy examples during training and rapidly focus the model on hard examples [43]. In this way, the standard cross-entropy loss treats all classes (or tokens) equally, which may lead to insufficient adjustment of rare classes. Focal loss modulates the cross-entropy loss by focusing on hard-to-predict classes. The mathematical formulation of the focal loss is based on the cross-entropy loss and is defined according to Equation (3) [44]:

$$\text{Focal loss} = -\alpha \cdot (1 - p_{b,t})^{\gamma} \cdot \log(p_{b,t}) \tag{3}$$

where $p_{b,t}$ represents the estimated model probability for each class, α is a balancing factor that adjusts the importance of positive/negative examples, and γ is the focusing parameter that controls the rate at which easy examples are displayed weighted downward [44]. When $\gamma = 0$, the focal loss becomes equivalent to the cross-entropy loss. As γ increases, the effect of the focusing mechanism becomes more pronounced, allowing the model to concentrate on difficult examples and decreasing the contribution of the easy ones γ [44].

In our loss function, the focal loss addresses class imbalance by focusing on hard-to-predict tokens, such as rare SQL keywords or specific conditions. It is defined as follows:

$$L_{Focal} = \frac{1}{N} \sum_{b=1}^{B} \sum_{t=1}^{T-1} \left[-\alpha \cdot (1 - p_{b,t})^{\gamma} \cdot log(p_{b,t}) \cdot w_{b,t} \cdot m_{b,t} \right] \tag{4}$$

where $p_{b,t}$ is the predicted probability for the true token, $\alpha = 0.25$ and $\gamma = 2.0$ are hyperparameters, $w_{b,t}$ is the sequence-matching weight, and $m_{b,t}$ is a mask for valid tokens. Specifically, $m_{b,t}$ is a binary mask that indicates which tokens should be included in the loss calculation: value of 1 for valid tokens that should contribute to the loss and value of 0 for tokens that should be ignored (padding tokens, special tokens, etc.). The mask ensures that only users selected important tokens are considered when computing loss function. The term $(1 - p_{b,t})^{\gamma}$ reduces the loss contribution from easy examples, allowing the model to focus on challenging tokens. The sequence matching weights ensure that critical tokens (e.g., GROUP BY or SELECT) are prioritized.

2.3.3. F-Beta Loss

The F-beta loss incorporates precision and recall metrics to optimize both correctness and completeness of important token sequences [45]. This was considered fundamental by the authors in text-to-SQL conversion tasks, as omitting a key token or generating incorrect tokens can invalidate a query completely. For the definition of our F-beta loss, we started from F-Beta score, which is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst value at 0. It is defined as follows:

$$F_{\beta}(t) = \frac{(1 + \beta^2) \cdot precision_t \cdot recall_t}{\beta^2 \cdot precision_t + recall_t + \varepsilon} \tag{5}$$

where $precision_t = \frac{TP_t}{TP_t + FP_t}$, $recall_t = \frac{TP_t}{TP_t + FN_t}$, and TP_t, FP_t and FN_t are true positives, false positives, and false negatives for token sequences at position t. The β parameter represents the ratio of recall importance to precision importance. $\beta > 1$ gives more weight to recall,

while $\beta < 1$ favours precision [46]. For this work, we modify the loss function presented by Lee et al. [47] to fit in LLM training. It is defined as follows:

$$L_{F\beta} = \frac{1}{N}\sum_{b=1}^{B}\sum_{t=1}^{T-1}\left[\left(\delta_1 \cdot L_{CE}(\hat{y}_{b,t}, y_{b,t}) + \delta_2 \cdot \left(1 - F_\beta(t)\right)\cdot w_{b,t}\right)\cdot m_{b,t}\right] \quad (6)$$

We define $\beta = 1.0 + progress$ that increases the emphasis on recall as training progresses, ensuring that all critical tokens are generated, and $\epsilon = 10^{-10}$ for numerical stability. Precision and recall were computed for important token sequences using vectorized matching within a maximum distance of 5 tokens. The weights $w_{b,t}$ prioritize critical sequential tokens, ensuring that the model captures essential SQL components. The loss combined the standard cross-entropy loss δ_1 (weighted at 0.6) with the F-beta loss term δ_2 (weighted at 0.4), where $1 - F_\beta(t)$ converted the F-beta score into a loss.

2.3.4. Contrastive Loss

The standard contrastive loss takes the output of the network for a positive example and calculates its distance to an example of the same class and contrasts that with the distance to negative examples [48–50]. The contrastive loss term is inspired by contrastive learning, which minimizes the difference between predicted and true representations:

$$Contrastive\ term = \max\left(0, \left|d_{pred} - d_{true}\right| - margin\right) \quad (7)$$

where d_{pred} and d_{true} are the predicted and true distances between pairs of important token sequences, and *margin* defines a tolerance threshold such that the model incurs a penalty only when the absolute difference $\left|d_{pred} - d_{true}\right|$ exceeds this value.

We modified it, so our contrastive sequence loss encourages the model to maintain correct relative distances between important token sequences in the predicted and true SQL queries [48–50]. The adaptation we present in this work is defined as follows:

$$L_{Contrastive} = \frac{1}{N}\sum_{b=1}^{B}\sum_{t=1}^{T-1}\left[\left(L_{CE}(\hat{y}_{b,t}, y_{b,t}) + \delta_3 \cdot max\left(0, \left|d_{pred} - d_{true}\right| - margin\right)\right)\cdot m_{b,t}\right] \quad (8)$$

We define $margin = 0.5$, allowing for small variations in distances, and factor $\delta_3 = 0.3$ to balance the contrastive term with the cross-entropy loss. This loss ensures that the model preserves the structural relationships between SQL components (e.g., the distance between SELECT and FROM).

2.3.5. Dynamic Custom Loss

The final loss combines the four components with dynamic weights that evolve with training progress:

$$L = \omega_{ce} \cdot L_{SMCE} + \omega_{focal} \cdot L_{Focal} + \omega_{beta} \cdot L_{F\beta} + \omega_{contrastive} \cdot L_{Contrastive} \quad (9)$$

where ω_{ce}, ω_{focal}, ω_{beta}, and $\omega_{contrastive}$ are the weights given to each component of the custom loss function and are dynamically updated during training. By combining these components with dynamic weighting, our custom loss function bridges the gap between standard language modelling objectives and the structured nature of SQL generation, enabling the model to produce both syntactically correct and semantically meaningful queries.

3. Results

This section presents a comparative analysis of the standard loss function and our proposed dynamic custom loss function across multiple open-source models for the text-to-

SQL task. The evaluation focuses on SQL execution accuracy using a vehicle diagnostics dataset, demonstrating the effectiveness of our approach in real-world, domain-specific applications.

The evaluated models, all open access with less than 2B parameters, include the following: Llama 3.2 1B [51], Qwen 2.5 0.5B and 1.5B [52], Qwen 2.5 Coder 0.5B and 1.5B [53], Qwen 3 0.6B and 1.7B [54], Salamandra 2B [55], EuroLLM 1.7B [56], and OLMo 2 1B [57]. We used the HuggingFace repository to download all the open-source models used in this study, and we employed Python 3.10.16 as the programming language to carry out the experiments. All models were initialized with pre-trained weights from their original papers and fine-tuned for 100 epochs under identical conditions on an A100 GPU of the ARINA UPV-EHU cluster.

Additionally, our implementation uses the hyperparameters detailed in Table 3. Hyperparameters are configuration variables that must be set before training an ML model, influencing how the model learns and performs [58]. As commented by Honghe Jin [59], hyperparameters play an essential role in the fitting of supervised machine learning algorithms. However, as it is computationally expensive to tune them all, we have selected them after multiple training courses to achieve the best result.

Table 3. Training hyperparameters that are used for paper results.

Hyperparameter	**Value**
Number of epochs	100
Training and evaluation batch size	8
Gradient accumulation steps	8
Neftune noise alpha	3
Learning rate	4×10^{-5}
Learning rate scheduler	Cosine
Optimizer	Paged AdamW 8bit
Lora α	128
Lora dropout	0.1
Lora r	64
Training and validation samples	2125 | 375
Test samples (server)	383

On the other hand, to improve the text-to-SQL model's performance on the vehicle diagnostic dataset, a series of tests were conducted to determine the weights of the sequence groups presented in Table 4, which were used to train the models presented in this paper. To obtain these values, an iterative evaluation process was performed in which different configurations were tested on a smaller validation set of the vehicle diagnostic dataset, emphasizing the importance of certain database aspects, such as vehicle IDs, their associated errors, and table/column references. These aspects were considered critical by database users for query accuracy. The final weights in Table 4 were therefore chosen because they provided better results than the other configurations, achieving a balance between token-level accuracy and executable SQL query generation. For example, groups such as Errors, Tables, Columns, and Vehicles were assigned a higher weight (5.0) than Vehicle Body, Vehicle Type, Vehicle Group, and Workshop (2.0–2.5), reflecting the importance of accurate error codes (e.g., 'HNN89' or 'P1772HJ') and schema elements in achieving accurate query results. Errors in these fields could result in significantly different database results, which

users may not notice since SQL queries interact directly with the database if intermediate validation is not performed correctly.

Table 4. Experiment group weights that are used for the paper results on the custom vehicle text-to-SQL dataset.

Group	Weights
SQL	3.0
Tables	5.0
Columns	5.0
Vehicles	5.0
Errors	5.0
Vehicle body	2.0
Vehicle type	2.5
Vehicle group	2.5
Workshop	2.0

As described in Section 2, the proposed loss function integrates four components: sequence-matching cross-entropy loss (L_{SMCE}), focal loss (L_{Focal}), F-beta loss ($L_{F\beta}$), and contrastive sequence loss ($L_{Contrastive}$) according to Equation (10):

$$L = \omega_{ce} \cdot L_{SMCE} + \omega_{focal} \cdot L_{Focal} + \omega_{beta} \cdot L_{F\beta} + \omega_{contrastive} \cdot L_{Contrastive} \quad (10)$$

Each component is dynamically weighted based on training progress:

- $\omega_{ce} = 1.0 - 0.7 \cdot$progress: Decreases from 1.0 to 0.3, reducing emphasis on sequence matching as training progresses to allow other components to refine complex structures.
- $\omega_{focal} = 0.8 \cdot$progress: Increases from 0.0 to 0.8, prioritizing hard-to-predict tokens (e.g., rare error codes) in later epochs.
- $\omega_{beta} = 1.0 \cdot$progress: Increases from 0.0 to 1.0, enhancing focus on precision and recall for critical token sequences.
- $\omega_{contrastive} = 0.5$: Remains constant, ensuring the consistent enforcement of structural integrity throughout training.

where *progress* is defined as the epoch divided by the total number of epochs and is represented in Equation (11). Then, this variable takes values between 0 and 1 in a linear way.

$$progress = \frac{current\ epoch}{total\ epochs} \quad (11)$$

This strategy provides a progressive learning trajectory during the training process. In the early epochs, the dominant cross-entropy weight emphasizes sequence-level accuracy, enabling the model to learn the basic syntax of SQL and the token-level mappings (in a similar way to how regular training is performed). As training progresses, the focal loss weight and the F-beta loss weight increase, resulting in an emphasis on refining precision and recall for domain-specific entities, as well as resolving ambiguous or challenging token sequences. The contrastive loss weight is designed to remain constant: this is to enforce structural consistency across all training stages. The objective is to guarantee the integrity of the syntax, avoiding errors such as mismatched parentheses or invalid join conditions. In short, the approach adopted in this study involves staged emphasis to ensure robust SQL generation: first, learning foundational syntax; then, domain-specific refinement; and

maintaining structural validity throughout. The combined loss is then normalized by the number of valid tokens to stabilize the training dynamics. Figure 4 illustrates the evolution of the total loss over global training steps, comparing the standard loss (cross-entropy) with the proposed custom loss for both training and evaluation phases for the Llama-3.2-1B-Instruct model. The figure also breaks down the individual contributions of each custom loss component (sequence-matching cross-entropy loss, focal loss, F-beta loss, and contrastive loss) across training and evaluation. These plots highlight the convergence behaviour and stability of the proposed approach, suggesting that the dynamic weighting of loss components does not introduce instability or convergence delays during training.

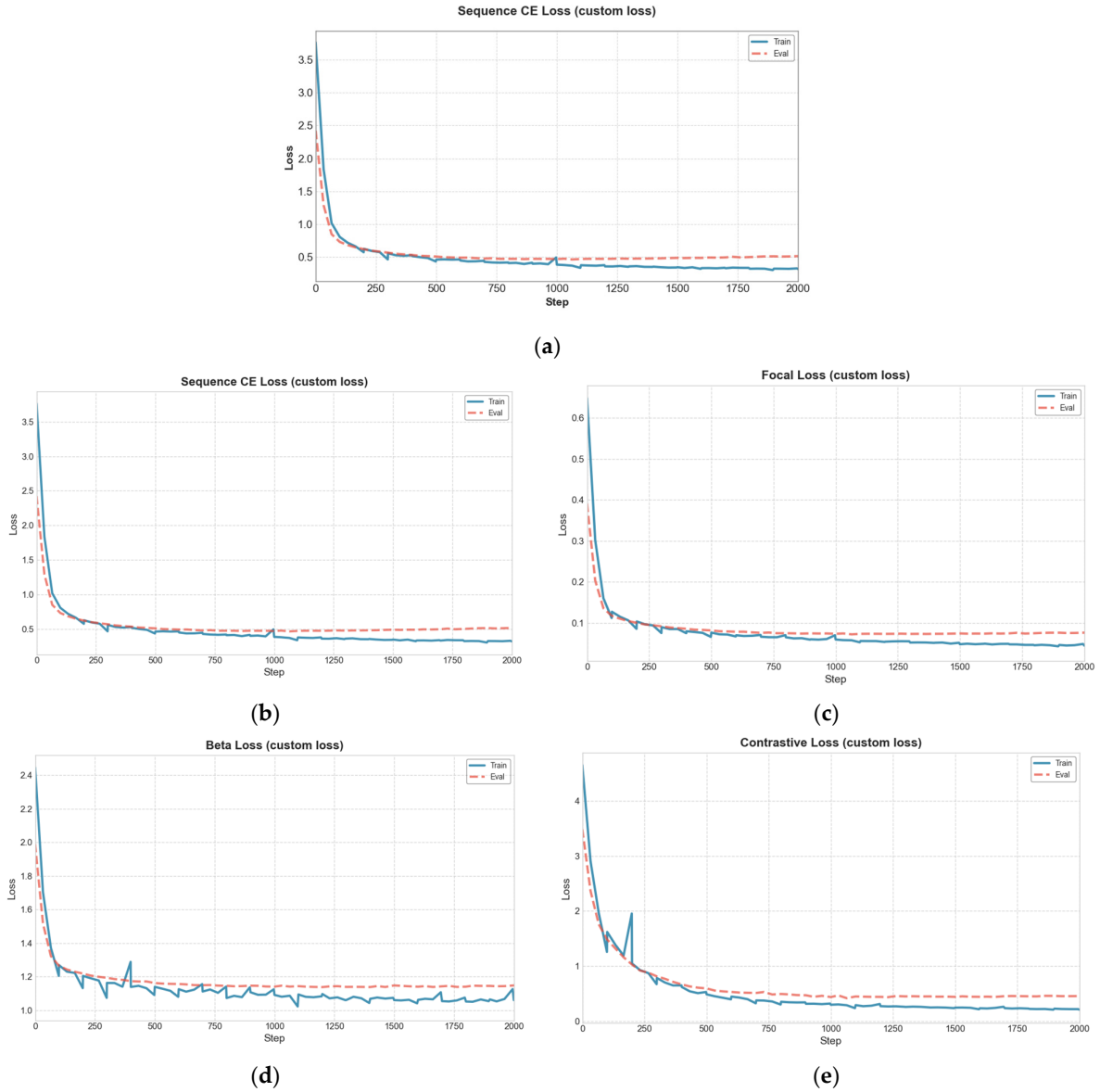

Figure 4. Loss evolution during training and evaluation with Llama-3.2-1B-Instruct model: (**a**) Comparison of total loss using standard cross-entropy loss versus custom loss function for training and evaluation. (**b**) Sequence-matching cross-entropy loss for training and evaluation. (**c**) Focal loss for training and evaluation. (**d**) F-beta loss for training and evaluation. (**e**) Contrastive loss for training and evaluation. All plots show loss values over global training steps.

Additionally, Figure 5 compares the performance of multiple open-source models trained under identical conditions with the standard cross-entropy loss and our custom dynamic loss. We evaluated the results on an SQL Server dataset and found that models

using our custom loss performed much better, especially for complex queries involving joins across multiple tables, aggregations, and specific domain-related entities (like rare error codes like 'P0043'). For example, using custom loss can improve the accuracy of exact matches by up to 25% (i.e., the generated SQL query and its execution output must match the given data result exactly). This improvement was seen in all the models tested.

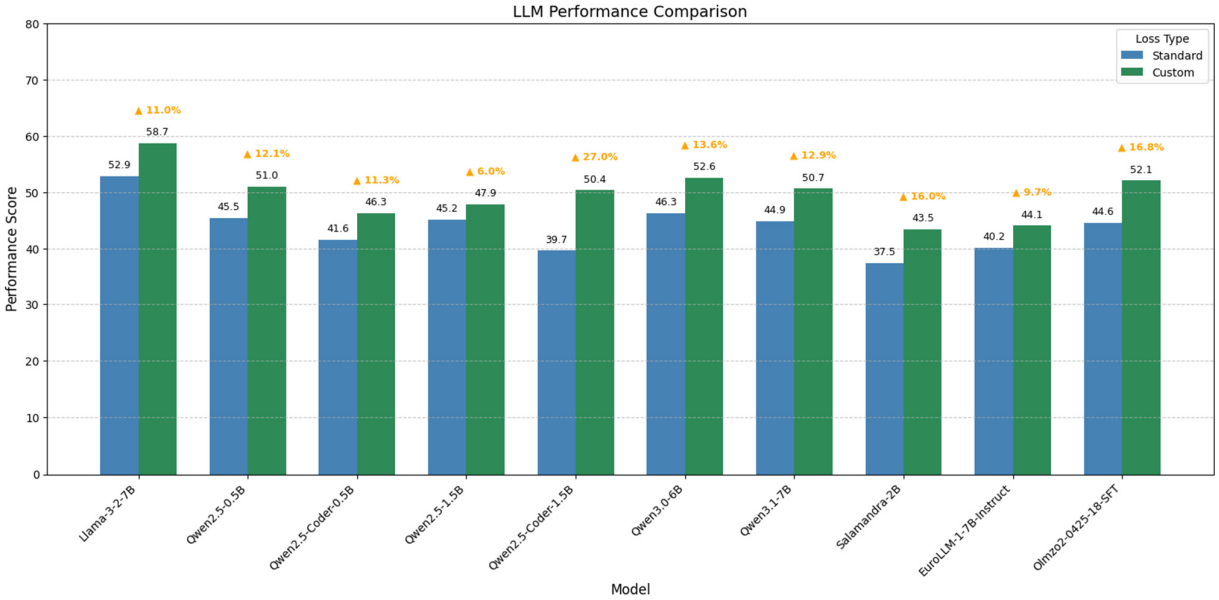

Figure 5. Evaluation results: Performance of different open-source LLMs in standard loss training vs. custom loss training.

As can be seen in Figure 5, the model accuracy increases from 6.0% in the worst case (Qwen 2.5 1.5B) to 27% (Qwen 2.5 Coder 1.5B). This demonstrates the effectiveness, generalizability, and robustness of the proposed method, with an average improvement of 13.64% over the 10 models studied. Furthermore, Figure 6 shows a statistical comparison of the performance of the standard and customized loss methods, averaged across all models evaluated. On the one hand, models trained with the standard loss method achieved a mean score of 43.8% (95% confidence interval (CI): 40.7–46.9%), with a standard deviation of 4.35%, indicating moderate variability in the results. On the other hand, models trained with the customized loss method achieved a mean score of 49.7% (95% CI: 46.5–53.0%), with a standard deviation of 4.51%, showing a significant improvement. A paired t-test performed on the performance scores yielded a t-statistic of 8.6604 and a p-value of 0.000012, indicating a statistically high difference ($p < 0.001$). These results suggest that the observed increases in accuracy are not attributable to random variance but are likely the result of the improved design of the personalized loss function.

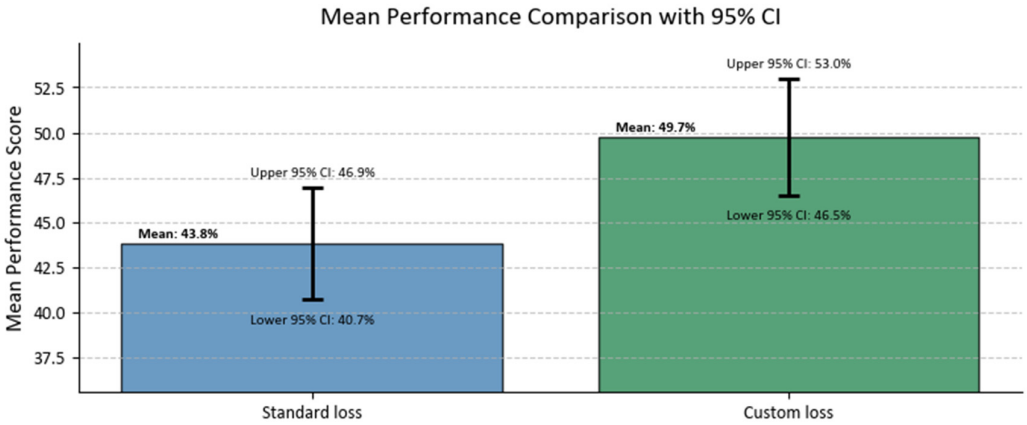

Figure 6. Evaluation results: Statistical comparison of mean performance scores for standard loss and custom loss, including 95% confidence intervals.

The exact match accuracy (EM) compared in Figure 5 is given by Equation (12).

$$EM\ (\%) = \frac{exact\ query + exact\ result + same\ result}{total\ samples} \tag{12}$$

This metric measures how well the trained model performs in generating correct and semantically equivalent SQL queries when evaluated against the reference SQL queries and their corresponding database results. This is because, in our case, the most interesting aspect to evaluate was that the database requests of the predicted SQL query and the GT query gave an equivalent result.

1. Exact Query: This counts the number of cases where the LLM-generated SQL query exactly matches the ground truth (GT) SQL query. That includes matching keywords, table names, column names, conditions, and order of operations. Obviously, if the predicted query and its label are the same, the result in the database is the same. An example of an exact query is shown below:

   Generated query
   ```
   SELECT TOP 10 Vehicle_Id, COUNT(*) AS FailureCount
   FROM Failure_Codes
   WHERE YEAR(Datetime) = 2021 AND Error_Code = 'UH0043'
   GROUP BY Vehicle_Id
   ORDER BY FailureCount DESC;
   ```
   GT query
   ```
   SELECT TOP 10 Vehicle_Id, COUNT(*) AS FailureCount
   FROM Failure_Codes
   WHERE YEAR(Datetime) = 2021 AND Error_Code = 'UH0043'
   GROUP BY Vehicle_Id
   ORDER BY FailureCount DESC;
   ```

2. Exact Result: This counts the number of cases where the result returned by executing the LLM-generated SQL query on the database exactly matches the result of the ground truth SQL query. This includes matching values, column names, and the order of rows and columns in the resulting table. An example of an exact result is shown below:

Generated query
```
SELECT TOP 10 Vehicle_Id, COUNT(*) AS FailureCount
FROM Failure_Codes
WHERE
    Datetime >= '2021-01-01'
    AND Datetime < '2022-01-01'
    AND Error_Code = 'UH0043'
GROUP BY Vehicle_Id
ORDER BY FailureCount DESC;
```
GT query
```
SELECT TOP 10 Vehicle_Id, COUNT(*) AS FailureCount
FROM Failure_Codes
WHERE YEAR(Datetime) = 2021 AND Error_Code = 'UH0043'
GROUP BY Vehicle_Id
ORDER BY FailureCount DESC;
```

3. Same Result: This counts the number of cases where the query result is logically the same, even if the column names differ or the column order is different in the results table. An example of the same result is shown below:

Generated query
```
SELECT TOP 10 Vehicle_Id, COUNT(*) AS Num_of_failures
FROM Failure_Codes
WHERE Error_Code = 'UH0043' AND YEAR(Datetime) = 2021
GROUP BY Vehicle_Id
ORDER BY Num_of_failures DESC;
```
GT query
```
SELECT TOP 10 Vehicle_Id, COUNT(*) AS FailureCount
FROM Failure_Codes
WHERE YEAR(Datetime) = 2021 AND Error_Code = 'UH0043'
GROUP BY Vehicle_Id
ORDER BY FailureCount DESC;
```

Additionally, Figure 7 shows the training time of different models from 0.5B to 2B parameters. Thus, it can be observed that the training time increases differently depending on the trained model. For example, the training time increases up to 10 times for the model with 0.5B parameters, while the Salamandra model, with 2B parameters, increases only by 3.

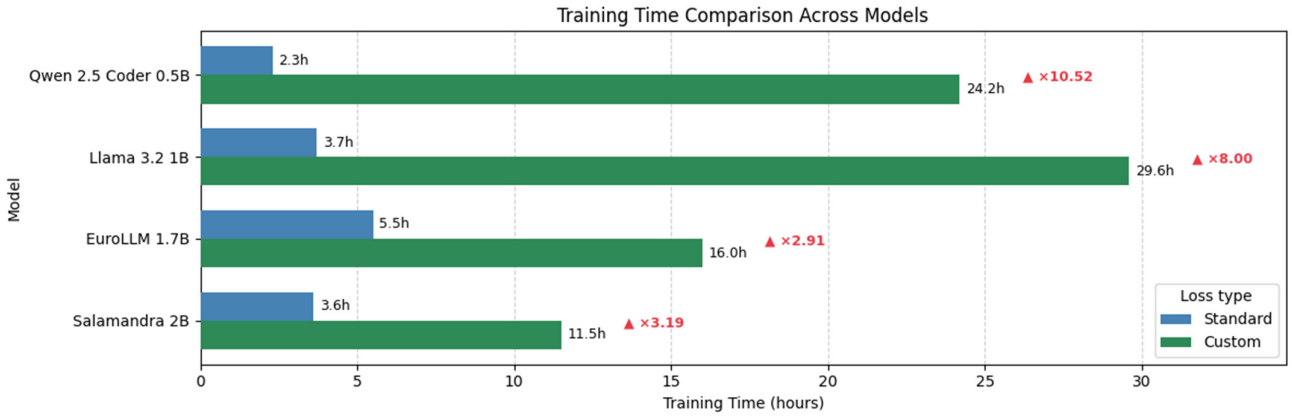

Figure 7. Evaluation results: Training time across different trained models.

As can be seen in Figure 7, the custom loss introduces additional computational overhead, increasing the training speed compared to standard cross-entropy. This is due to the dynamic weighting and multi-component loss calculations, which are required. However, these additional calculations are justified by improved accuracy, particularly in domain-specific applications where precision is critical. As demonstrated in Figure 4, the efficacy of

the method in managing rare tokens and intricate query structures, which are frequently underrepresented in conventional general-purpose datasets, is further emphasized. This increase in the performance of the models is important for real-world applications, such as vehicle diagnostics.

On the one hand, the dynamic weighting strategy is a key reason for the observed performance improvements. In the early epochs, the high ω_{ce} helps models learn basic SQL syntax, which reduces errors in basic query structures. As training continues and ω_{focal} increases, models start to look at rare tokens, like specific error codes or time limits, to improve how they respond to less common query patterns. Additionally, the progressive increase in ω_{beta} enhances the balance of precision and recall, ensuring that critical token sequences (e.g., Vehicle_Id, Failure_Codes) are accurately generated. The constant $\omega_{contrastive}$ weight enforces syntactic validity during training, minimizing errors like mismatched parentheses or incorrect join conditions.

On the other hand, high group weights for critical domain-specific components, including tables, columns, vehicles, and error codes (see Table 4), ensure that the model prioritizes these elements during training. This contributes to the observed 25% average improvement in execution accuracy on real-world SQL Server queries. This performance enhancement is especially notable in complicated scenarios involving joins across multiple tables, aggregations, and specific constraints based on domains (for example, filters based on time, like YEAR(Datetime) = 2024). Our custom loss function is designed to focus on a specific sequence of tokens (for example, Vehicle_Id, Failure_Codes), which helps to reduce errors in query syntax and database entity alignment.

The method further demonstrates strong generalization to rare and challenging cases, as evidenced by improved accuracy on queries containing infrequent error codes (e.g., JH1013') and intricate temporal logic (e.g., 'what are the most common fleet errors between March 3, 2023, and April 4, 2024'). The findings emphasize the efficacy of integrating domain-informed token weighting with dynamic loss prioritization to achieve robust, high-accuracy SQL generation in specialized applications.

Furthermore, to conclude the Results Section, Figure 8 presents the accuracy results obtained by comparing the cross-entropy function with the proposed dynamic cost function on the Spider public dataset [60]. This dataset has been selected for its popularity, diversity of examples, and ease of implementation and validation. The idea is to demonstrate the generalizability and robustness of the loss function proposed in this work. Accordingly, Figure 8 shows a comparison of two implementations of the cost function proposed in this work: on the one hand, a static customization of the proposed loss function (Equation (13)) and, on the other hand, the same dynamic cost function implemented for the vehicle diagnostic dataset (Equation (14)).

$$CL_1 = 0.6 \cdot L_{SMCE} + 1.0 \cdot L_{Focal} + 0.6 \cdot L_{F\beta} + 0.4 \cdot L_{Contrastive} \tag{13}$$

$$CL_2 = (1 - 0.7) \cdot p \cdot L_{SMCE} + 0.8 \cdot p \cdot L_{Focal} + p \cdot L_{F\beta} + 0.4 \cdot L_{Contrastive} \tag{14}$$

where p is the training progress and is dynamically updated as question 8. Additionally, all custom important sequence weights have been initialized to 1 since the dataset is multithematic, and there are no categories that want to be prioritized. The hyperparameters used for this test are the same as Table 3.

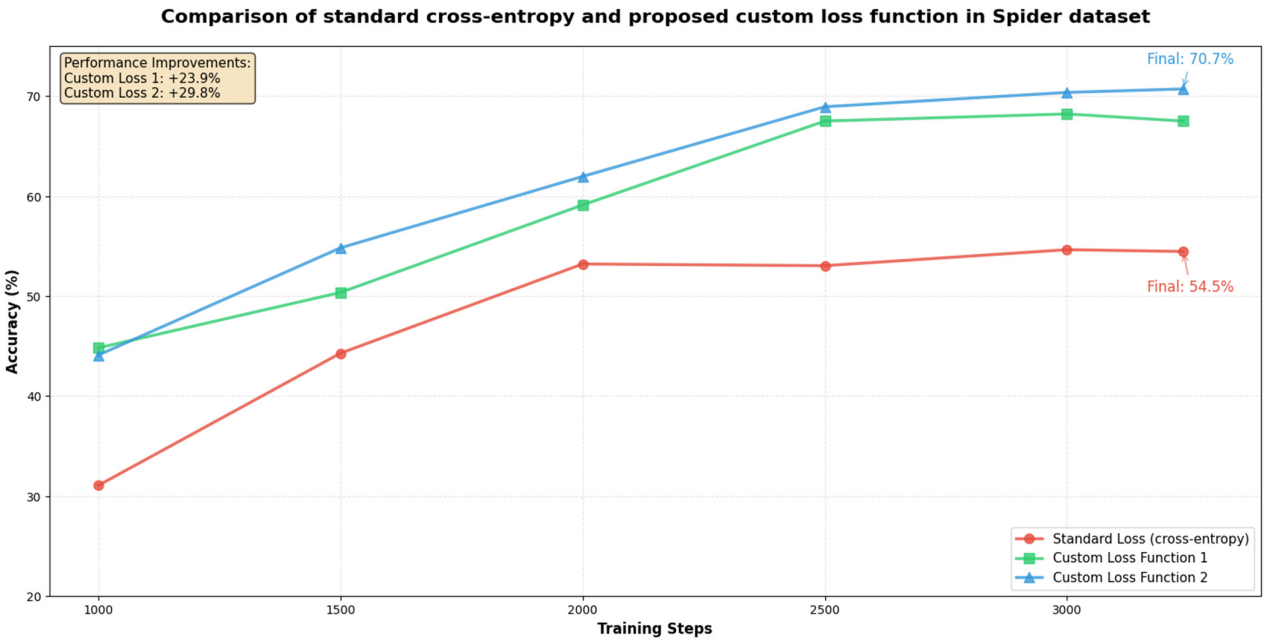

Figure 8. Evaluation results on public Spider dataset: Evaluation SQL query accuracy across full training session.

As can be seen in Figure 8, the SQL query accuracy results improve by almost 30% over the standard cross-entropy function. Also, the increase in accuracy can be observed throughout the training, demonstrating the increased performance of the text-to-SQL models thanks to the proposed cost function.

In summary, the proposed dynamic custom loss function significantly outperforms the standard loss function across multiple open-source models for the text-to-SQL task. By dynamically adjusting loss weights, the model effectively learns to prioritize critical aspects of SQL query generation, resulting in higher execution accuracy and robustness, particularly for complex and domain-specific queries. These findings underscore the potential of tailored loss functions in enhancing LLM performance for structured prediction tasks.

4. Discussion

In this work, we introduce a dynamic custom loss function designed to improve the performance of language models in the structured task of text-to-SQL translation. Unlike traditional static loss functions that apply uniform weighting across all tokens, our approach dynamically assigns importance to different types of token sequences based on their semantic and syntactic roles in SQL queries. This is achieved through a group-based weight assignment mechanism, where token groups such as SQL keywords, table names, column identifiers, and domain-specific entities are assigned distinct weights during training.

One of the core innovations of our method is dynamic weight assignment, which allows the model to focus on multiple aspects of SQL query generation at different stages of training. For example, early in training, higher emphasis is placed on basic SQL syntax and token-level mappings using weighted cross-entropy loss, helping the model learn correct representations. As training progresses, the weights shift toward components like focal loss to prioritize learning rare or challenging tokens and F-beta loss to balance precision and recall, particularly for specific domain characteristics such as vehicle IDs or diagnostic codes. Throughout training, contrastive loss ensures structural consistency, preventing syntax errors and ensuring valid SQL outputs.

This flexible architecture enables fine-grained control over learning priorities by simply modifying a group weight dictionary. This means that practitioners can easily adjust which parts of the SQL output should be emphasized more strongly, depending on the application domain or dataset characteristics. For instance, in vehicle diagnostics, prioritizing fault code tokens significantly improved accuracy on queries involving rare conditions. Furthermore, the design is inherently scalable. The group-based structure allows for straightforward extensions, such as adding new token categories or introducing more complex weighting schemes based on contextual features or attention patterns.

The present experiments, conducted across a range of open-source language models with up to 2 billion parameters, including Llama 3.2, Qwen variants and Salamandra, have demonstrated consistent enhancements in execution accuracy. These enhancements have been observed to reach up to 20% improvements in comparison to standard cross-entropy loss metrics. These improvements were most pronounced in handling complex queries involving multi-table joins, aggregations, and domain-specific temporal constraints. The dynamic nature of the loss also contributed to better generalization on rare or infrequent tokens, which are often problematic in specialized domains. Finally, our method is also compared across the Spider open-source dataset, reaching nearly 30% of improvement using the dynamic custom loss function.

5. Conclusions

The present work aimed to enhance the performance of large language models in the text-to-SQL task by introducing a dynamic custom loss function that adaptively prioritizes different token groups and learning objectives throughout training. Our approach uses a variety of loss functions (such as cross-entropy, focal loss, F-beta loss, and contrastive sequence loss) and group-based token weighting. This helps models gradually move from basic SQL syntax to more rare, specialized terms and making sure the structure is consistent.

The results demonstrate that this strategy significantly improves SQL execution accuracy across multiple open-source language models under 2B parameters. Evaluated on a SQL Server dataset of vehicle diagnostics queries, our method achieves up to a 20% increase in exact match accuracy (i.e., both syntactically correct SQL queries and identical execution outputs) compared to standard cross-entropy loss. Improvements are most pronounced for complex queries involving multi-table joins, aggregations, and temporal constraints, highlighting the method's robustness in handling domain-specific challenges.

While the custom loss function introduces an increase in training time due to its multi-component and dynamic nature, the important accuracy improvements and enhanced robustness justify this trade-off for applications where precision is crucial. The flexibility and scalability of the group-based weighting scheme further allow for straightforward adaptation to new domains or the inclusion of additional token categories, making this approach widely applicable to other structured prediction tasks.

Future work will focus on improving the dynamic adjustment of the weights that take part in the personalized cost function: on the one hand, the weights associated with the important groups and, on the other hand, the weights associated with the four components of the cost function. The idea is to intelligently modify these weights, depending on the results they are giving in training. In addition, we will study the extension of this approach using reinforcement learning techniques, with the aim of finding an efficient solution that evaluates the responses during training on the database server. Finally, deploying the method on low-cost hardware for real-time applications or combining it with advanced architectures, such as attention-based or graph-based neural networks, could open new possibilities for structured prediction tasks.

Author Contributions: Conceptualization, I.A. and N.U.-A.; methodology, I.A.; software, I.A. and N.U.-A.; validation, I.A., N.U.-A., J.M.L.-G. and G.G.; formal analysis, E.Z.; investigation, I.A.; resources, I.A. and N.U.-A.; data curation, I.A. and N.U.-A.; writing—original draft preparation, I.A.; writing—review and editing, J.M.L.-G. and G.G.; visualization, I.A., J.M.L.-G. and G.G.; supervision, J.M.L.-G., G.G. and E.Z.; project administration, G.G.; funding acquisition, J.M.L.-G., G.G. and E.Z. All authors have read and agreed to the published version of the manuscript.

Funding: The research of Iker Azurmendi was funded by BIKAINTEK, grant number 016-B2/2022. The current study was sponsored by the Government of the Basque Country, through the ELKARTEK programme, and the grant number is KK-2025/00012 ("Creación de algoritmia de aprendizaje profundo de despliegue rápido en la industria que dote de técnicas de razonamiento a las máquinas").

Data Availability Statement: The data presented in this study is available upon request from the corresponding author.

Acknowledgments: The authors are grateful to Fagor Electronica Smart Data Services, and especially to Rosa María Martínez, for all the support provided during this project. The authors would also like to thank the technical and human support provided by SGIker (UPV/EHU/ERDF, EU).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

NLP	Natural Language Processing
SQL	Structured Query Language
AI	Artificial Intelligence
DL	Deep Learning
LLM	Large Language Model
RL	Reinforcement Learning
CLLM	Consistency-Driven Language Models
SFT	Supervised Fine-Tuning
FLAT	Forget data only Loss AjustmenT
ML	Machine Learning
CI	Confidence Interval

References

1. Mohammadjafari, A.; Maida, A.S.; Gottumukkala, R. From Natural Language to SQL: Review of LLM-Based Text-to-SQL Systems. *arXiv* **2024**, arXiv:2410.01066.
2. Gan, Y.; Purver, M.; Woodward, J.R. A Review of Cross-Domain Text-to-SQL Models. In Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop, Suzhou, China, 4–7 December 2020.
3. Baig, M.S.; Imran, A.; Yasin, A.; Butt, A.H.; Khan, M.I. Natural Language to SQL Queries: A Review. *Int. J. Innov. Sci. Technol.* **2022**, *4*, 147–162. [CrossRef]
4. Fu, Y.; Ou, W.; Yu, Z.; Lin, Y. MIGA: A Unified Multi-Task Generation Framework for Conversational Text-to-SQL. In Proceedings of the AAAI Conference on Artificial Intelligence, Washington, DC, USA, 7–14 February 2023.
5. Askari, A.; Poelitz, C.; Tang, X. MAGIC: Generating Self-Correction Guideline for In-Context Text-to-SQL. In Proceedings of the AAAI Conference on Artificial Intelligence, Philadelphia, PA, USA, 27 February–2 March 2025.
6. Chen, Z.; Chen, S.; White, M.; Mooney, R.; Payani, A.; Srinivasa, J.; Su, Y.; Sun, H. Text-to-SQL Error Correction with Language Models of Code. *arXiv* **2023**, arXiv:2305.13073.
7. Meyer, Y.; Emadi, M.; Nathawani, D.; Ramaswamy, L.; Boyd, K.; Van Segbroeck, M.; Grossman, M.; Mlocek, P.; Newberry, D. Synthetic-Text-To-SQL: A Synthetic Dataset for Training Language Models to Generate SQL Queries from Natural Language Prompts 2024. Available online: https://huggingface.co/datasets/gretelai/synthetic_text_to_sql (accessed on 14 March 2025).
8. Zhu, X.; Li, Q.; Cui, L.; Liu, Y. Large Language Model Enhanced Text-to-SQL Generation: A Survey. *arXiv* **2024**, arXiv:2410.06011.

9. Kaplan, R.M.; Webber, B.L. The Lunar Sciences Natural Language Information System. 1972. Available online: https://www.researchgate.net/publication/24285293_The_Lunar_Sciences_Natural_Language_Information_System (accessed on 18 April 2025).
10. Kanburoğlu, A.B.; Tek, F.B. Text-to-SQL: A Methodical Review of Challenges and Models. *Turk. J. Electr. Eng. Comput. Sci.* **2024**, *32*, 403–419. [CrossRef]
11. Lee, D.; Yoon, J.; Song, J.; Lee, S.; Yoon, S. One-Shot Learning for Text-to-SQL Generation. *arXiv* **2019**, arXiv:1905.11499.
12. Iyer, S.; Konstas, I.; Cheung, A.; Krishnamurthy, J.; Zettlemoyer, L. Learning a Neural Semantic Parser from User Feedback. *arXiv* **2017**, arXiv:1704.08760. [CrossRef]
13. Mellah, Y.; Rhouati, A.; Ettifouri, E.H.; Bouchentouf, T.; Belkasmi, M.G. SQL Generation from Natural Language: A Sequence-to-Sequence Model Powered by the Transformers Architecture and Association Rules. *J. Comput. Sci.* **2021**, *17*, 480–489. [CrossRef]
14. Xu, K.; Wu, L.; Wang, Z.; Feng, Y.; Sheinin, V. SQL-to-Text Generation with Graph-to-Sequence Model. *arXiv* **2018**, arXiv:1809.05255.
15. Lin, K.; Bogin, B.; Neumann, M.; Berant, J.; Gardner, M. Grammar-Based Neural Text-to-SQL Generation. *arXiv* **2019**, arXiv:1905.13326.
16. Wu, K.; Wang, L.; Li, Z.; Xiao, X. Faster and Better Grammar-Based Text-to-SQL Parsing via Clause-Level Parallel Decoding and Alignment Loss. In *CCF International Conference on Natural Language Processing and Chinese Computing*; Springer Nature Switzerland: Cham, Switzerland, 2022. [CrossRef]
17. Liu, A.; Hu, X.; Lin, L.; Wen, L. Semantic Enhanced Text-to-SQL Parsing via Iteratively Learning Schema Linking Graph. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, Washington, DC, USA, 14 August 2022; pp. 1021–1030.
18. Zhang, Q.; Dong, J.; Chen, H.; Li, W.; Huang, F.; Huang, X. Structure Guided Large Language Model for SQL Generation. *arXiv* **2024**, arXiv:2402.13284. [CrossRef]
19. Zhang, T.; Chen, C.; Liao, C.; Wang, J.; Zhao, X.; Yu, H.; Wang, J.; Li, J.; Shi, W. SQLfuse: Enhancing Text-to-SQL Performance through Comprehensive LLM Synergy. *arXiv* **2024**, arXiv:2407.14568.
20. Hong, Z.; Yuan, Z.; Zhang, Q.; Chen, H.; Dong, J.; Huang, F.; Huang, X. Next-Generation Database Interfaces: A Survey of LLM-Based Text-to-SQL. *arXiv* **2024**, arXiv:2406.08426.
21. Pourreza, M.; Talaei, S.; Sun, R.; Wan, X.; Li, H.; Mirhoseini, A.; Saberi, A.; Arik, S. Reasoning-SQL: Reinforcement Learning with SQL Tailored Partial Rewards for Reasoning-Enhanced Text-to-SQL. *arXiv* **2025**, arXiv:2503.23157.
22. Zhang, Y.; Zhou, S.; Huang, G. SE-HCL: Schema Enhanced Hybrid Curriculum Learning for Multi-Turn Text-to-SQL. *IEEE Access* **2024**, *12*, 39902–39912. [CrossRef]
23. Berdnyk, M.; Collery, M. LLM-Based SQL Generation with Reinforcement Learning. 2025. Available online: https://openreview.net/forum?id=84M0Jaiapl (accessed on 16 April 2025).
24. Nguyen, X.-B.; Phan, X.-H.; Piccardi, M. Fine-Tuning Text-to-SQL Models with Reinforcement-Learning Training Objectives. *Nat. Lang. Process. J.* **2025**, *10*, 100135. [CrossRef]
25. Pack Kaelbling, L.; Littman, M.L.; Moore, A.W.; Hall, S. Reinforcement Learning: A Survey. *J. Artif. Intell. Res.* **1996**, *4*, 237–285. [CrossRef]
26. Ghasemi, M.; Ebrahimi, D. *Introduction to Reinforcement Learning*; MIT Press: Cambridge, MA, USA, 2024.
27. Chung, H.W.; Hou, L.; Longpre, S.; Zoph, B.; Tay, Y.; Fedus, W.; Li, Y.; Wang, X.; Dehghani, M.; Brahma, S.; et al. Scaling Instruction-Finetuned Language Models. *arXiv* **2022**, arXiv:2210.11416. [CrossRef]
28. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv* **2023**, arXiv:1910.10683.
29. Xie, Y.; Jin, X.; Xie, T.; Lin, M.; Chen, L.; Yu, C.; Cheng, L.; Zhuo, C.; Hu, B.; Li, Z. Decomposition for Enhancing Attention: Improving LLM-Based Text-to-SQL through Workflow Paradigm. *arXiv* **2024**, arXiv:2402.10671.
30. Ling, X.; Liu, J.; Liu, J.; Wu, J.; Liu, J. Finetuning LLMs for Text-to-SQL with Two-Stage Progressive Learning. In *Proceedings of the Natural Language Processing and Chinese Computing*; Wong, D.F., Wei, Z., Yang, M., Eds.; Springer Nature Singapore: Singapore, 2025; pp. 449–461. [CrossRef]
31. Kou, S.; Hu, L.; He, Z.; Deng, Z.; Zhang, H. CLLMs: Consistency Large Language Models. In Proceedings of the Forty-first International Conference on Machine Learning, Vienna, Austria, 21–27 July 2024.
32. Sow, D.; Woisetschläger, H.; Bulusu, S.; Wang, S.; Jacobsen, H.-A.; Liang, Y. Dynamic Loss-Based Sample Reweighting for Improved Large Language Model Pretraining. *arXiv* **2025**, arXiv:2502.06733.
33. Xie, S.; Chen, H.; Yu, F.; Sun, Z.; Wu, X. Minor SFT Loss for LLM Fine-Tune to Increase Performance and Reduce Model Deviation. *arXiv* **2024**, arXiv:2408.10642. [CrossRef]
34. Wang, Y.; Wei, J.; Liu, C.Y.; Pang, J.; Liu, Q.; Shah, A.P.; Bao, Y.; Liu, Y.; Wei, W. LLM Unlearning via Loss Adjustment with Only Forget Data. *arXiv* **2024**, arXiv:2410.11143. [CrossRef]
35. Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal Loss for Dense Object Detection. *arXiv* **2017**, arXiv:1708.02002.

36. Xia, Y.; de Araujo, P.H.L.; Zaporojets, K.; Roth, B. Influences on LLM Calibration: A Study of Response Agreement, Loss Functions, and Prompt Styles. *arXiv* **2025**, arXiv:2501.03991. [CrossRef]
37. ChatGPT Guide What Is Cross-Entropy Loss: LLMs Explained—Chatgptguide. Available online: https://www.chatgptguide.ai/2024/03/03/what-is-cross-entropy-loss-llms-explained/ (accessed on 15 March 2025).
38. Mao, A.; Mohri, M.; Zhong, Y. Cross-Entropy Loss Functions: Theoretical Analysis and Applications. *arXiv* **2023**, arXiv:2304.07288. [CrossRef]
39. Zhang, Z.; Sabuncu, M.R. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. *arXiv* **2018**, arXiv:1805.07836. [CrossRef]
40. Jerald Teo How Do Large Language Models Learn? | by Jerald Teo | Medium. Available online: https://medium.com/@jeraldteokj/visualising-loss-calculation-in-large-language-models-1af410a9d73d (accessed on 15 March 2025).
41. Zhou, Z.; Huang, H.; Fang, B. Application of Weighted Cross-Entropy Loss Function in Intrusion Detection. *J. Comput. Commun.* **2021**, *9*, 1–21. [CrossRef]
42. Fan, Y.; Li, R.; Zhang, G.; Shi, C.; Wang, X. A Weighted Cross-Entropy Loss for Mitigating LLM Hallucinations in Cross-Lingual Continual Pretraining. In Proceedings of the ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing, Hyderabad, India, 6–11 April 2025. [CrossRef]
43. Focal Loss Explained | Papers With Code. Available online: https://paperswithcode.com/method/focal-loss (accessed on 25 May 2025).
44. Qué Es: Pérdida Focal—APRENDE ESTADÍSTICAS FÁCILMENTE. Available online: https://es.statisticseasily.com/glossario/what-is-focal-loss/ (accessed on 2 July 2025).
45. Understanding F-Beta Score: 4 Metrics Explained Fast. Available online: https://www.numberanalytics.com/blog/understanding-fbeta-score-metrics (accessed on 25 May 2025).
46. Fbeta_score—Scikit-Learn 1.7.0 Documentation. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.fbeta_score.html (accessed on 2 July 2025).
47. Lee, N.; Yang, H.; Yoo, H. A Surrogate Loss Function for Optimization of F_β Score in Binary Classification with Imbalanced Data. *arXiv* **2021**, arXiv:2104.01459.
48. Contrastive Loss Explained. Contrastive Loss Has Been Used Recently. . . | by Brian Williams | TDS Archive | Medium. Available online: https://medium.com/data-science/contrastive-loss-explaned-159f2d4a87ec (accessed on 25 May 2025).
49. Wang, F.; Liu, H. Understanding the Behaviour of Contrastive Loss. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2025.
50. Khosla, P.; Teterwak, P.; Wang, C.; Sarna, A.; Tian, Y.; Isola, P.; Maschinot, A.; Liu, C.; Krishnan, D. Supervised Contrastive Learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 18661–18673.
51. Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; et al. The Llama 3 Herd of Models. *arXiv* **2024**, arXiv:2407.21783. [CrossRef]
52. Qwen; Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; et al. Qwen2.5 Technical Report. *arXiv* **2024**, arXiv:2412.15115.
53. Hui, B.; Yang, J.; Cui, Z.; Yang, J.; Liu, D.; Zhang, L.; Liu, T.; Zhang, J.; Yu, B.; Lu, K.; et al. Qwen2.5 Technical Report. *arXiv* **2024**, arXiv:2409.12186. [CrossRef]
54. Team, Q. Qwen3 2025.
55. Gonzalez-Agirre, A.; Pàmies, M.; Llop, J.; Baucells, I.; Da Dalt, S.; Tamayo, D.; Saiz, J.J.; Espuña, F.; Prats, J.; Aula-Blasco, J.; et al. Salamandra Technical Report. *arXiv* **2025**, arXiv:2502.08489. [CrossRef]
56. Martins, P.H.; Fernandes, P.; Alves, J.; Guerreiro, N.M.; Rei, R.; Alves, D.M.; Pombal, J.; Farajian, A.; Faysse, M.; Klimaszewski, M.; et al. EuroLLM: Multilingual Language Models for Europe. *arXiv* **2024**, arXiv:2409.16235. [CrossRef]
57. OLMo, T.; Walsh, P.; Soldaini, L.; Groeneveld, D.; Lo, K.; Arora, S.; Bhagia, A.; Gu, Y.; Huang, S.; Jordan, M.; et al. 2 OLMo 2 Furious. *arXiv* **2024**, arXiv:2501.00656. [CrossRef]
58. Probst, P.; Bischl, B. Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *J. Mach. Learn. Res.* **2019**, *20*, 1–32.
59. Jin, H. Hyperparameter Importance for Machine Learning Algorithms. *arXiv* **2022**, arXiv:2201.05132. [CrossRef]
60. Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; et al. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *arXiv* **2018**, arXiv:1809.08887.