GRAMMEHR: Evaluating methods of Grammar Constrained Sampling for SQL Generation on EHR data for Mixture of Experts models

Theo Simon Sorg

Universitätsbachelorarbeit
zur Erlangung des akademischen Grades

Bachelor of Science
(B. Sc.)

im Studiengang
IT Systems Engineering

eingereicht am 09. Juli 2024 am
Fachgebiet Digital Health & Machine Learning
der Digital-Engineering-Fakultät
der Universität Potsdam

Gutachter	Prof. Dr. Christoph Lippert
Betreuer	Noel Danz

Abstract

This thesis explores methods for enhancing Structured Query Language (SQL) generation for Electronical Health Record (EHR) data using Grammar Constrained Decoding (GCD) techniques within language models, more precisely for Mixture of Experts (MoE) models. We address the effectiveness of GCD and propose *expert switching* for improving the accuracy of SQL query generation as a subset of the broad field of code generation. This work is built on the recent advancements in Large Language Models (LLMs) and their application to structured data generation tasks. Expert switching ranks expert combinations of a MoE layer based on their probabilities and iterates through these combinations, stopping if a combination is able to generate a token adhering to the grammar with sufficient probability. Additionally, we provide a corrected version of the *test suite accuracy* [1] metric suite, fixing parsing related bugs and errors in the aggregation counting for difficulty classification of samples.

We evaluate this approach through rigorous benchmarking, particularly focusing on the complex schemata of EHR data. Results indicate that incorporating GCD significantly enhances the generation of valid SQL queries, with notable improvements in execution accuracy and exact set matches for complex queries. However, expert switching does not improve over common GCD techniques when comparing models of different sizes and more or less constraining grammars. Nonetheless, our tests should not serve as a final result on this matter due to the limited hyperparameter space searched as well as the focus on the last MoE layer for computational efficiency, allowing for additional iterations on the presented methods.

The findings of this work come with implications for the deployment of LLMs for SQL generation in the medical domain by validating a flexible approach applicable to any language model for handling complex, domain-specific SQL generation tasks. There is a multitude of use cases, a prominent one is the provision of EHRs for medical researchers without the need to learn the SQL language. Therefore, this research contributes to the broad field of AI-based data processing, demonstrating the potential of constrained LLM decoding techniques in critical environments.

1 A public implementation of Semantic Evaluation for Text-to-SQL with Distilled Test Suites by Zhong et al. found at https://github.com/taoyds/test-suite-sql-eval

Acknowledgments

We want to express our gratitude to a few organizations and individuals for the support and their contributions to this work.

First and foremost, we would like to thank our supervisors, Prof. Lippert and Noel Danz of the Hasso-Plattner-Institute for the guidance and encouragement we received throughout this project.

Additionally, we are very grateful to the project team from our Bachelor's project BPLW2023, who built AIR.MS Turtle with us. This project provided the starting point for our interest in Structured Query Language (SQL) generation tasks, and we greatly appreciate their dedication and hard work.

A special thanks is attributed to Cornelia Philippson of the Hasso-Plattner-Institute for her assistance with all the necessary paperwork, which enabled a smooth progress of this study.

We are very thankful to the Ev. Studienwerk Villigst e.V. that funded the studies at the Hasso-Plattner-Institute, without which we would have been unable to dedicate the time and effort put into this work.

Furthermore, we extend the appreciation to the Mount Sinai Health System for access to their database that allowed us to explore and play around with their Electronical Health Record (EHR) data. Here, we are very grateful to Manbir Singh and Lewis Lo for organizing resources and permissions for the database that made this research possible in the first place.

Lastly, we would like to thank Katharina Timm for her patience and understanding, especially for enduring the discussions about possible switching techniques for days on end.

Thanks to all of you for your contributions and your support.

Contents

Introduction

In recent years, the advancement of artificial intelligence, particularly in the field of Natural Language Processing (NLP), has been significantly driven by the development of Large Language Models (LLMs). These models, exemplified by architectures such as GPT-3 and its successors, have demonstrated remarkable capabilities in generating coherent and contextually appropriate text across various domains. Among the many applications of LLMs, one area that has gathered substantial interest is automatic code generation from natural language.

While automatic code generation is a challenging task itself, this work focuses on a smaller subset with additional constraints, namely the task of generating Structured Query Language (SQL) queries, also known as text-to-SQL. This involves converting a user's natural language question or command into a syntactically and semantically correct SQL statement that can retrieve the desired information from a database. This problem is particularly challenging due to the need for precise understanding of both the natural language input and the underlying database schema. The complexity of SQL generation increases with the complexity of the database schema and the specificity of the queries.

This thesis explores the application of grammar-constrained sampling methods for SQL generation, particularly in the context of Electronical Health Record (EHR) data. EHRs are pivotal in modern healthcare due to their potential to improve the quality of patient care by providing timely access to patient data and enabling automated, large-scale analysis for faster detection and prevention of various illnesses. However, EHRs are characterized by their complexity, diverse data types, difficult vocabulary and the critical nature of the information contained. Mistakes in querying this data can lead to research errors that might result in incorrect diagnoses or treatment plans, making the generation of accurate and reliable SQL queries essential.

Our approach leverages the concept of Mixture of Experts (MoE) models, which aim to improve the performance and generalization of LLMs by dividing the task among specialized sub-models or *experts*, often on a layer basis. By integrating grammatical constraints into the sampling process, we seek to enhance the syntactic correctness of the generated SQL queries, thereby improving the overall robustness of the system.

The goal of this work is to contribute to the development of more reliable and

efficient systems for automated SQL generation, with a particular focus on applications in healthcare. By improving the accuracy of text-to-SQL systems, we hope to facilitate more effective data retrieval and analysis, ultimately supporting better clinical outcomes.

This work is part of a cooperation of the Hasso-Plattner-Institute with the Mount Sinai Hospital to create the AIR.MS Turtle, a natural language database interface for the EHRs at Mount Sinai.

Part I

Related Work

1 SQL Generation using LLMs

The now infamous "Attention is All you Need" [Vas+17] paper proposed the Transformer as well as Multi-Head Attention, laying the foundations for the current increase in performance of AI-models in many fields. There have been successful attempts to apply this architecture not only to visual or auditory data, but to encoder- or decoder-only models [Liu+18] for texts as well, for example BERT [Dev+18] and the different versions of GPT [Rad+18] [Rad+19].

1.1 The Rise of Large Language Models

While the initial encoder-decoder models remain relevant, decoder-only models tend to outperform different architectures on causal language modeling tasks [Liu+18] [Rad+18]. Due to the increasing amount of parameters in these language models, they have become known as Large Language Models (LLMs).

The output of these LLMs represents a probability distribution assigning a probability to each token of the vocabulary. In order to generate a whole text based on textual input, the same input is fed into the LLM repeatedly in an autoregressive manner [Gra13]. After passing the initial input prompt through the LLM and receiving the probability distribution, one of the tokens is selected [Gra13]. It is appended to the prompt and this combination of prompt and generated token serves as the input for another run of the decoder-only Transformer, again appending the generated token to the last input used before querying the LLM a third time [Gra13]. As this iterative approach does not naturally allow for an end of the autoregressive generation, apart from reaching the end of the context window, a special token is used, which, upon being appended to the last input string, will mark the end of the generation [SVL14].

Many different methods can be used to select the correct token from the probability distribution. Graves developed beam search [Gra12], an algorithm that keeps the W most likely candidates (called the width of the beam) at each step. Each of the candidates is then fed into the model to calculate the probability distribution, and the now W most likely tokens across all of the W candidates serve as the newly selected beam [Gra12]. As an alternative to beam search, "top-k sampling" was proposed by Fan et al. [FLD18], in which random samples are drawn from the

k most probable tokens of the distribution, weighted by the probabilities of the tokens, resulting in a less repetitive output [FLD18]. Instead of a fixed k, "nucleus sampling" samples from the smallest set of tokens whose cumulative probability exceeds a fixed value $p \in (0, 1]$ [Hol+19].

1.2 Mixture of Experts Models

By upscaling the individual layer dimensions as well as the number of layers in a neural network, i.e. the number of parameters in general, the trained models tend to perform better compared to their smaller counterparts, which can be explained by the scaling laws of LLMs [Kap+20]. But a large-scale neural network is often inefficient during training as well as in inference due to its sheer size [Sha+17]. Additionally, the vanishing gradient problem arises for models utilizing recurrent neural networks [BSF94]. Though the latter has been solved as modern architectures are mostly built on top of Transformers, both these issues motivated research into Mixture of Experts (MoE) models. To circumvent these problems, "Conditional Computation" has been proposed as a more guided version of dropout, i.e. not dropping parameters uniformly at random but instead in a learned fashion [Ben13], therefore only utilizing a fraction of the model parameters at a time.

The general idea of using a system composed of many smaller networks, the so called experts, has been proposed more than 30 years ago at the Connectionist Summer School in 1988 [Jac+91]. Initially, the output of the system was built by combining the output of all experts as a weighted sum by using a gating network that itself would adjust its weights during training [JJ90] to obtain the weights for each of the experts. While old versions focus on the whole model consisting of multiple experts, applying the MoE concepts on a layer basis was proven to yield better results by Shazeer et al. [Sha+17]. Nowadays, MoE models utilize only one or k (Shazeer et al. [Sha+17]) of their n experts, an adaption of the original linear combination proposed by Jacobs et al. in 1991 [Jac+91]. Since MoE is not dependent on the underlying architecture of the networks, newer MoE models are often built on top of Transformers by replacing some or all of the Feed Forward Networks with MoE layers [Jia+24], on which this work will focus on. Currently, models only using one expert per layer as well as k experts per layer exist and perform equally well for language modelling [FZS22].

One of the key problems faced in developing MoE models is ensuring that each of the experts receives the same amount of queries during training. If one of the experts performs significantly better than the others due to a better initialization of its weights, the gate network would otherwise learn to activate the best expert for

every training sample which would in turn increase the performance of the expert. Since a single expert is not meant and unable to capture all the complexities of the data, this leads to worse results than balanced experts [ERS13]. To prevent this, a regularization term was introduced by Bengio et al. [Ben+15] and Gaussian noise might be added before the softmax of the gating network [Sha+17].

1.3 LLMs in SQL Generation

The task of program synthesis, often referred to as code generation, in general dates back to the 1940s and 50s [Aus+21], with a focus on restricted languages [Gul11]. Since then, there have been successful attempts to generate programs in specific representations designed for this particular use case, consequently losing the ability to be easily interpretable by humans [OS20].

In recent years, LLMs have been utilized for program synthesis due to their ability to learn from unlabeled data, thus learning to generate code given enough training samples [Aus+21]. Codex [Che+21], the Large Language Model behind "GitHub Copilot", displays this capability of LLMs by finetuning GPT-3 on publicly available code from GitHub.

The HumanEval benchmark is a famous choice to evaluate models on code generation tasks. While LLMs achieved state-of-the-art results at the time [Che+21], the highest scores in the "pass@1" category are currently achieved by "AgentCoder" [Hua+23] and "LDB" [ZWS24]. Both of these approaches try to incorporate feedback into the code generation by executing the code and iteratively correcting mistakes, an attempt to overcome the limitations of LLMs for code generation. These limitations have been famously described by the Codex developers as "we find that Codex can recommend syntactically incorrect or undefined code, and can invoke functions, variables, and attributes that are undefined or outside the scope of the codebase." [Che+21].

Structured Query Language (SQL) generation is a field of code generation and faces the same problems with additional challenges. The underlying database schema has to be followed which varies on a per database basis, thus limiting generalization of the generation on unknown databases. Furthermore, the performance of LLMs decreases with an increase in complexity of the database schema as well as the complexity of the natural language query, resulting in very low scores for complex cross-domain applications [Yu+18]. Consequently, LLMs have been unable to beat smaller finetuned models until recently by utilizing few-shot learning as well as dividing the task of generating a SQL query into smaller subtasks and solving them individually. Even though state-of-the-art results were achieved using this,

there were still errors in the generated SQL requiring correction through another query to the LLMs [PR24]. As of today, there is still room for improvement in the field, with the current state of the art achieving a weighted execution accuracy of 81.3/100 *pts* on an Electronical Health Record (EHR) benchmark named EHRSQL [AI24].

1.4 Evaluating LLMs for SQL on EHR data

In order to evaluate the state of SQL generation using LLMs in general as well as on EHR data specifically, common benchmarks can be used. One of the drawbacks of these benchmarks is the lack of a standardized metric suite. The WikiSQL benchmark [ZXS17] uses Exact Matching as well as Execution Accuracy to compute a match. Exact Matching is a string comparison between the original and the generated answer, while Execution Accuracy compares the results of the generated and the original SQL on a per-tuple basis. The well known SPIDER benchmark [Yu+18] uses Component Matching, Exact Matching and Execution Accuracy, though results are mostly reported using Exact Matching. The Exact Matching used here differs from WikiSQL as SPIDER ensures that the ordering is ignored whenever SQL allows different orderings with the same semantic. To differentiate this approach from a string comparison as done by WikiSQL, the term *exact set matching* is used by Yu et al. Component Matching extracts different SQL components, for example the SELECT part of a query, and checks whether they match individually, potentially ignoring the ordering within a component, and computes the average exact set match per component [Yu+18]. Zhong et al. improved the metrics used for the SPIDER dataset by proposing Test Suite Accuracy, fixing numerous issues with the SPIDER approach [ZYK20]. The UNITE benchmark [Lan+23] utilizes a form of Execution Accuracy with filtering of empty execution results. The BIRD benchmark [Li+24] introduces another metric to measure the efficiency of valid SQL, the Valid Efficiency Score. It is computed as the average of the square root of the coefficient of the execution time of the original and the predicted SQL.

Though there are quite a few SQL generation benchmarks, most of them are domain-specific in areas other than medicine or do not feature complex queries. To overcome this and allow for applications in the medical sector, EHRSQL has been proposed [Lee+22], operating on a modified version of the MIMIC-III database schema. EHRSQL uses a specific version of Execution Accuracy, in which the system must first conclude whether the query is answerable and only if it is the Execution Accuracy is computed. To reflect the accuracy regarding answerable and unanswerable queries, a F1 score is computed.

2 State of Grammar Constrained Decoding for Mixture of Experts

There have been different approaches to overcome the instability of code generation using causal language models at runtime by enforcing specific tokens in the generation. A softer approach is controlled text generation which works by biasing the probability distribution of a language model based on an attribute model [Dat+20]. As this only modifies probabilities of tokens but still allows invalid tokens (just with lower probability), this approach is impractical for hard constraints on the syntactic structure of the output. Lexically constrained decoding ensures specific constraints are met in the output using grid beam search [HL17] or constrained beam search [And+17]. Grid beam search represents beams in a grid of shape $C \times T$, where T is the maximum number of generation steps and C the number of constraints, each constraint representing a word or phrase that has to be present in the output. Therefore, each column in the grid identifies one generation step while each row c indicates that c constraints have been completed. With each of the beams of fixed width W, for a new generation step the grid elements of the same row in the adjacent left column and of the row below in the adjacent left column are considered to generate the new beam elements, leading to $C + 1$ beams that need to be kept. [HL17]

On the contrary, constrained beam search can be applied to any finite state automaton by representing each state with a beam of fixed width W. At each generation step every beam is updated by keeping the W most likely sequences that satisfy the condition represented by the corresponding state but do not satisfy any further conditions, i.e. are not a part of another beam with more advanced conditions. [And+17]

While lexically constrained decoding yields promising results, the constraints have to be expressible in a regular expression, thus only regular languages can be generated [And+17]. A more flexible approach that allows context free languages is Grammar Constrained Decoding (GCD). While most programming languages are irregular, they can be expressed using a context free grammar. As the same holds true for Structured Query Language (SQL), we will focus on GCD constrained decoding for this work as this can also be applied to SQL. Furthermore, GCD even allows for limited generation of context sensitive grammars [Mel+24]. [SSB21]

2.1 Grammars in Language Modelling

Grammars have been used in Language Modelling tasks in various ways. Apart from grammars to constrain the decoding of a language model, models were used to generate Abstract Syntax Trees (ASTs) in encoder-decoder settings [Zhu+24] [RSK17]. Additionally, models were trained to predict production rules instead of tokens [YN17]. As these methods all require a custom architecture and new pretraining, they can not be applied easily on current foundational models pretrained by large corporations with great amounts of computational power. Language models are also able to predict grammars on their own. Grammar prompting serves as a way to predict minimal grammars tailored to the specific input prompt before generating the output and constraining said output to the specified grammar. [Wan+23]

While grammars are mostly used for ensuring syntactical structures in the generation output, Peyrard et al. argues that approaches like Chain-of-Thought try to enforce a specified grammar on the semantics of the output.

Unlike methods involving ASTs, GCD provides a non-invasive way to enforce syntactically correct output for language models and allows the use of pretrained models. GCD prunes the probability distribution to a subset of all vocabulary tokens that are considered *allowed tokens* by an incremental parser [Gen+23b]. For the generation of SQL, an allowed token could be a FROM after SELECT and an attribute name have already been generated, even though more attribute names, their prefixes, and prefixes of FROM would also be allowed tokens. This process can be combined with top-k sampling, nucleus sampling and beam search and works well with few-shotting [Gen+23b]. GCD performs well on different structured prediction tasks [Gen+23b], including SQL generation [SSB21].

2.2 Drawbacks of Grammar Constrained Decoding

While GCD appears to be a promising way to enforce syntax for code generation, the quality strongly depends on the implementation of the incremental parser. Beurer-Kellner et al. [BFV24] introduced the concept of "Minimally invasive" decoding methods which are defined as: "We consider a constrained decoding method minimally invasive, if every valid output that can be generated by an unconstrained model (for any given prompt), is also generated by the constrained model, given the same prompt". They showed that most GCD methods are not minimally invasive, often forbidding so called bridge tokens that span multiple terminals in the underlying grammar leading to high perplexity tokens (terminals of the grammar)

being chosen instead [BFV24]. Interestingly, most modern methods perform worse than unconstrained generation according to Beurer-Kellner et al.

Part II

Methodology

3 Mixture of Expert Models

We evaluated different state of the art Mixture of Experts (MoE) models for switching experts decoding. For this, we selected models using the top-k routing on expert level as proposed by Shazeer et al. [Sha+17]. We allowed for $k \geq 1$ in contrast to the $k > 1$ by Shazeer et al. as Fedus et al. [FZS22] found that $k > 1$ and $k = 1$ achieve equal scores on various text generation tasks.

The notation used is similar to the notation by Fedus et al. For a layer with N experts $E_1, \ldots, E_N$ and an input representation x, the gating network has a trainable variable W_r which is used to compute the logits vector $h(x) = W_r \cdot x$. The logits were normalized using softmax to obtain a gate-value for each expert i computed as

$$p_i(x) = \frac{e^{h(x)_i}}{\sum_{j=1}^{N} e^{h(x)_j}}$$

Given $k \geq 1$, which is usually the same for all layers [Sha+17], we selected the top-k experts before applying softmax, similar to Shazeer et al. [Sha+17]. Let T now be the selected k experts. We compute the output y of a MoE layer as

$$y = \sum_{i \in T} p_i(x) \cdot E_i(x)$$

We allowed for activation functions to be applied to x before the computation of y, for example SwiGLU in Mixtral [Jia+24].

3.1 Selected Models

We implemented our approach for decoder-only models as most Large Language Models (LLMs) are decoder-only. Models meeting the above criteria selected for our experiments were Mixtral 8x7b Instruct v0.1 [Jia+24], Mixtral 8x22b Instruct v0.1 [Jia+24], the latter will be referred to as Mixtral from now on, which both use $N = 8, k = 2$, and Helion 4x34b [Wey24], which runs $N = 4, k = 2$. This selection represents models of different sizes, ranging from 46.7 billion parameters for Mixtral 8x7b Instruct v0.1 [Jia+24] over 136 billion for Helion 4x34b [Wey24] to 141 billion for Mixtral [Tea24]. Helion 4x34b was included as it was not trained as

a MoE model but is a merge of four independent LLMs that were trained without MoE and then combined with *mergekit* by Goddard et al. [God+24].

4 Grammar Constrained Decoding

We applied Grammar Constrained Decoding (GCD) on the resulting probability distribution of the selected Large Language Models (LLMs). We selected *gcd* [Gen+23b] as a framework to build upon as it is minimally invasive according to Beurer-Kellner et al. [BFV24]. Since Geng et al. offer a newer implementation compatible with transformers (linked in the original README [Gen+23a]), we used this *transformers-cfg* [Gen+24] instead.

The distribution was pruned so that only valid tokens were included. We greedily selected the most likely token. We refrained from using beam search, top-k or nucleus sampling as well as adjusting the distribution with a temperature parameter [AHS85] to reduce the amount of hyperparameters for which our experiments had to be conducted and to ensure reproducibility of our results by sampling deterministically.

4.1 Grammars

While Structured Query Language (SQL) is a context-free language and can therefore be expressed in Backus-Naur form [Ron17], Text-To-SQL generation focuses mainly on generating SQL queries to retrieve data, so many capabilities of SQL are not utilised. As a consequence, we evaluated two custom made grammars for this work that do not support all valid SQL statements but a reasonable subset used for querying a database.

The first grammar restricts the generation to SELECT statements with a limited instruction set while allowing arbitrary table and attribute names. The other grammar is the more restrictive one, generating only table and attribute names present in the schema. We did not use a mapping between attributes and tables referenced so far (as done by Arcadinho et al. [Arc+22]) since this would limit the generalization of the results of this work for other structured prediction tasks. Refer to the appendix for the custom grammars in extended Backus-Naur-Form A.

4.2 Termination and Repetition

Upon generating the corresponding EOS-token of the respective model, we ended the generation. Additionally, to avoid stalling forever, we set a hard limit at 200 newly generated tokens, ensuring that the autoregressive behavior will come to an end. As LLMs tend to enter repetition loops [Li+16] [Rad+19], we added a repetition penalty of $\theta = 1.2$ as proposed by Keskar et al. [Kes+19].

5 Expert Switching

Instead of using the output of the gating network to run the top-k experts and then sum their weighted distributions and apply Grammar Constrained Decoding (GCD) on this final distribution, one could reevaluate the choices performed by the gating network. This could involve assigning a higher weight to experts with higher probability tokens that adhere to the grammar or exchanging the k chosen experts if the final distribution does not conform with the grammar. The latter option, which we refer to as expert switching, will be the focus of this work. We chose expert switching as it does not require the computation of all experts for the subsequent decision on which experts to use based on their distributions and thus does not lead to the same performance overhead.

5.1 Switching Layers

Since most Mixture of Experts (MoE) models are layer based, we focus on switching for MoE layers. Especially, we introduce *Switching Layers* as an extension of MoE layers. Additionally to MoE layers, *Switching Layers* allow for another parameter called "expert mask", a binary mask with $\binom{N}{k}$ entries, which is utilized during inference. At first, a noise is applied as described in [Sha+17]. Instead of selecting the top-k experts, softmaxing and running them, a score is computed for each combination of k experts. These scores are then masked using the expert mask and the maximum remaining score is selected. The corresponding combination is chosen and the probability distribution of the gating network is pruned to these k. Now softmax is applied and the final output is computed similarly to the procedure described in section 3. The structure of a switching layer is visualized in Figure 5.1.

5.2 Decode Switching

To utilize the switching layers during inference, we used GCD and rejected the current experts if specific conditions were met. For this, we defined different hyperparameters indicating whether or not switching should occur. Switching should occur whenever there is no allowed token (i.e. no token matching the grammar) within the top-k tokens. We refer to this hyperparameter k as *top-k-tokens*.

Applying the idea of nucleus sampling [Hol+19], switching is also mandatory whenever there is no allowed token within the smallest set of tokens with cumulative probability p, which is referred to as *top-p-tokens*.

Still, switching experts might not result in meeting the token criteria mentioned above. While we can repeatedly switch and mask more and more combinations, there is an upper bound given by the $\binom{N}{k}$ combinations. If we tried all combinations, we fallback to the most likely expert combination, i.e. reset the expert mask and take the most probable token after pruning the distribution. Additionally, we evaluated earlier criteria to end expert switching for a token generation. The more expert combinations tried, the lower the probabilities that the gating network assigns to these experts. Consequently, they might not provide a better solution but still cause performance overhead. Thus, we introduce *top-k-experts*, indicating that we fallback after having tried k combinations. The respective counterpart based on nucleus sampling [Hol+19], named *top-p-experts*, ensures we stop switching after the cumulative probability of the tried expert combinations reaches p. To compute the probability of a combination, we summed the gating network output per combination and applied softmax to the resulting vector of combination logits. Note that this differs from the distribution obtained in 5.1 as the distribution used here is not pruned to k entries before softmaxing. For all tested models, we exchanged the last sparse expert layer with a switching layer. If we switch, we therefore only switch experts in the last decoder layer. We computed the expert mask based on the experts used for previous generations of the same token and provided this as additional input for our switching experts layer. Lastly, we noticed that Large Language Models (LLMs) often produce free text before the structured output, e.g. “The answer is”, “The SQL query is” or similar, leading to frequent fallbacks with the first generated token. To solve this, we incorporated instructions in the prompt asking the language model to refrain from using any boilerplate text. Please find the exact prompt in the appendix B.1.

5.3 Performance Improvements

In the naive approach described above, the experts are recalculated unnecessarily when switching to another combination if the same expert was already included in a previous combination. Since the combinations are chosen by their score and the underlying distribution of the gating network before masking is the same for all combinations, it is guaranteed that the expert with the highest probability on the first run will also be part of the second combination, leading to at least one unnecessary computation when top-k-experts > 1. The same holds true for top-k-

experts > 2 and the highest or second highest expert respectively. Strictly speaking, there is at least one computation per switch that has already been performed. To prevent repeated computation, we introduce *expert caching*. Provided a mask that indicates which experts have already been computed and the cached outputs of the experts, we only calculated the previously unused experts and added them to the mask and cache. It is important to note that this approach can be combined with other forms of caching as it only alters caching behavior for generating a distribution given the same input. This setting is not required in traditional caching algorithms as autoregressive models usually generate a distribution for the same input only once. Furthermore, the cache for later layers has to be invalidated if a switch occurs on a previous layer, as the hidden states and therefore the inputs for later layers change. Since we only evaluated expert switching on the last layer, this invalidation was not mandatory for us.

We utilized KV caching [Pop+23] for faster inference, enabled for the first generation of each position. Upon switching experts, a different approach comes into play.

Similarly to expert caching, we introduce *hidden state caching* that is used while expert switching. Instead of generating all hidden states again with a different expert mask, we kept all hidden states until we reached the first layer that is masked differently. Therefore, we only had to compute the hidden states for the following layers starting at the layer with the switched experts and used the cached versions for all layers prior to this.

5.4 Performance Analysis

Since we only switched in the last layer, we had to perform the computation in the last layer again when utilizing hidden state caching and only for experts that had not been computed so far when additionally using expert caching, leading to up to $N - k$ experts computed beyond the required amount for generation. To evaluate the speed of our approach, we measured the time it took to run the hyperparameter tuning and tokens-per-second speeds using Mixtral 8x7b Instruct v0.1.

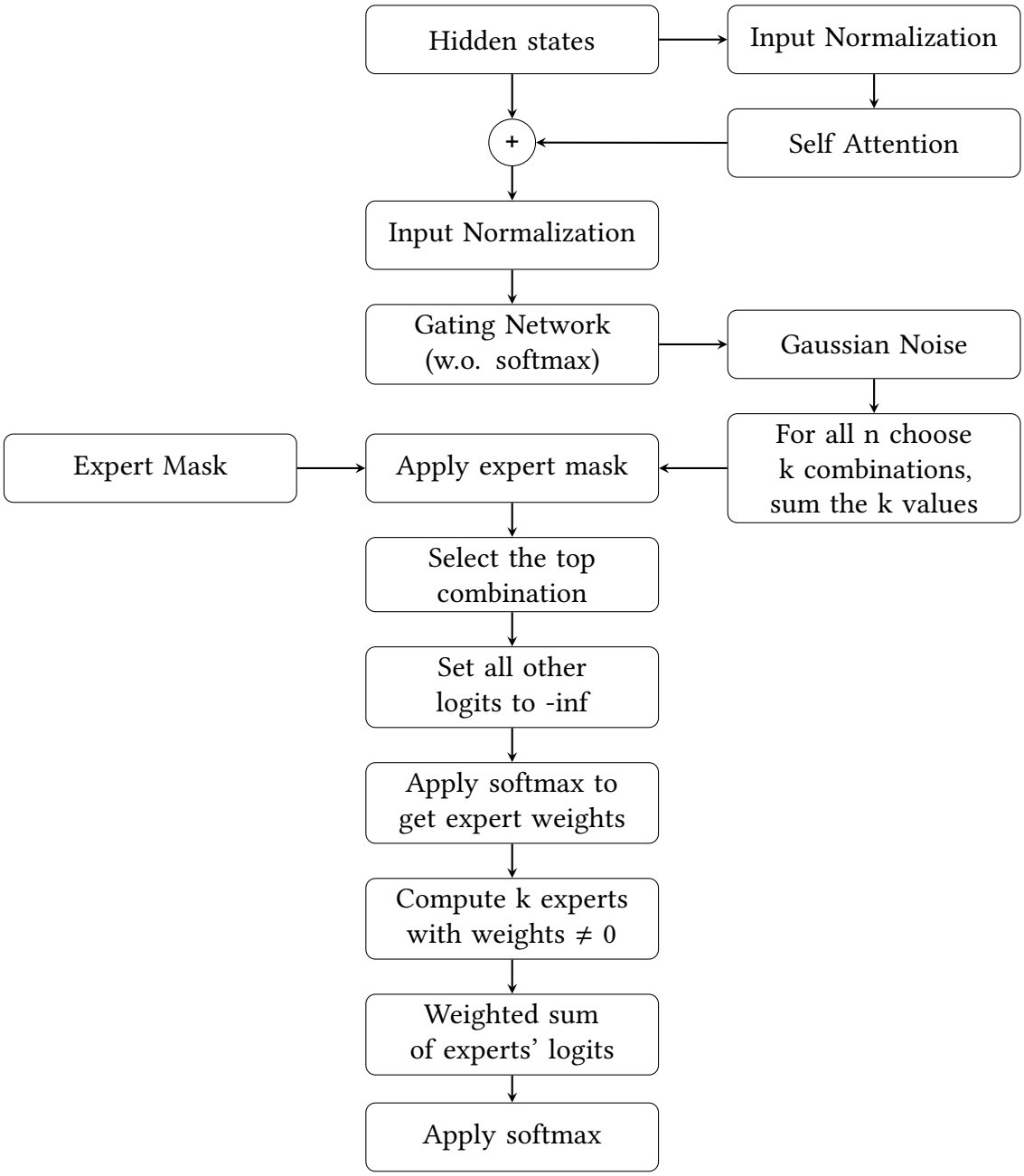

Figure 5.1: Workflow in a single switching layer, an extended version of a MoE layer

Part III

Evaluation

6 Benchmarks

We evaluated our approach on the popular Structured Query Language (SQL) benchmark SPIDER [Yu+18]. Since this work is based on a collaboration with the Mount Sinai Health System, we additionally evaluated the generation of SQL on Electronical Health Records (EHRs) using EHRSQL, a dataset for SQL generation on EHRs [Lee+22].

6.1 Hyperparameters

We tuned the hyperparameters for the decoding with switching (see section 5.2) using a grid search. We searched top-k-tokens and top-k-experts in the range of $[0, 25]$, top-p-tokens and top-p-experts within the interval $[0, 1]$. We did not try all values within the specific ranges (as p is continuous this is impossible) but only a limited, reasonable subset. The exact values tested are included in the Appendix C.

The search was based on the SPIDER validation split as it is a high quality task with rather simple schemata. Therefore, even smaller models perform well enough to measure differences between our approach and the baselines. We selected 100 indices from the entire validation dataset and used them for the grid search. Please note that the indices remained the same throughout the search.

6.2 Metrics

We used a modified version of *test-suite-accuracy* [ZYK20]. Test-suite-accuracy classifies SQL queries into four levels of difficulty *easy*, *medium*, *hard*, *extra* and provides execution accuracy as well as exact set match scores for each of them and their union.

For our experiments, the combined execution accuracy served as the primary metric to optimize, but exact set match is reported as well. We detected numerous bugs in the official implementation related to SQL parsing which is done by tokenizing the query and then selecting components based on SQL keywords. For evaluating EHRSQL, we required *strftime* and *is not null* checks which are not supported in the original implementation. Since tokenization and keyword-grouping does not reflect the complexity of SQL – for example subqueries can not be represented in

this way – we used the SQL parser *sqlglot* [2] instead to obtain a dialect agnostic representation and reimplemented the metrics. We additionally reimplemented the difficulty classification. To ensure replicability, we compared our difficulty results on the validation set of SPIDER with the results of the official implementation and found 21 differences in difficulty classification. Since we only used the combined version featuring all four difficulty levels, our results remain comparable. [3]

To formally express execution accuracy, we used a notation similar to Qin et al. [Qin+22]. Let N be the samples in total, and for one of the samples i let $\hat{V}_i$ be the predicted and V_i the groundtruth query. We then define

$$score(\hat{V}_i, V_i) = \begin{cases} 1 & \hat{V}_i = V_i \\ 0 & \hat{V}_i \neq V_i \end{cases} \tag{6.1}$$

The execution accuracy EX can then be computed as

$$EX = \frac{\sum_{n=1}^{N} score(\hat{V}_n, V_n)}{N} \tag{6.2}$$

To measure the impact of expert switching, we counted the amount of expert switches, the amount of fallbacks and the amount of expert switches that did not result in a fallback, i.e. an expert combination was found after switching that met one of the token-criteria.

We performed our grid search using Mixtral 8x7b Instruct v0.1 [Jia+24] and applied the obtained parameters on the other selected models (see section 3.1). Our limited instruction set grammar with free choice of table names and columns was used for the hyperparameter tuning.

For testing, we run test suite accuracy with execution accuracy and exact set match on the results of the two other selected models using the three different grammars. We used the test split of EHRSQL with the MIMIC-III version since the MIMIC-III schema is more complex than the eICU, displaying the ability of the model to generate complex queries. We report the performance on the validation split for SPIDER, though one should be aware that our hyperparameters might fit the dataset, therefore providing non-meaningful results. We removed the groundtruth samples from EHRSQL that resulted in an error on the MIMIC-III database as

2 A common SQL parser written in python, we used the version v25.2.0 found on GitHub https://github.com/tobymao/sqlglot

3 Most differences can be attributed to a bug in the amount of aggregations of a SQL query. You can find a more detailed explanation and the implementation at https://github.com/sorgfresser/test-suite-sql-eval/

execution accuracy is undefined if the groundtruth does not exist[4]. Furthermore, we only considered samples that where flagged as possible in the EHRSQL dataset.

6.3 Baselines

As a reference, we computed the results for both selected models (see section 3.1) other than Mixtral 8x7b Instruct v0.1 without the use of expert switching but with the use of Grammar Constrained Decoding (GCD) using *transformers-CFG* [Gen+24] for the two different grammars. Furthermore, we evaluated these models without the use of GCD entirely, resulting in 10 runs for SPIDER and EHRSQL respectively.

4 While there should not be any gold SQL statements resulting in errors that are marked as possible, there actually were four of them in the test split of EHRSQL. Consequently, we had to remove these samples from the test set.

7 Evaluation

Interestingly, there was no single set of parameters that achieved the best execution accuracy. Out of the 900 parameter combinations tested, 756 achieved an execution accuracy of 0.1 while the remaining 144 resulted in 0.09. The 144 combinations with lower accuracy all failed on one medium sample of SPIDER [Yu+18] that the other 756 combinations generated correctly.

7.1 Many Fallbacks

Additionally, most switches resulted in fallbacks. The highest number of switches without fallbacking on the token is achieved with top-k-experts and top-p-experts set to 0, i.e. disabled. In this case, all the $\binom{8}{2}$ expert combinations have to be tried before fallbacking, therefore fallbacks become less likely and the overall amount of switches as well as switches without fallbacks increases. The best ratio of switches without fallbacks to switches in general has been achieved with top-k-experts disabled, top-p-experts disabled and top-p-tokens disabled and top-k-tokens set to 5 with 7245 switches and 1602 switches without fallback. Out of the five switches with highest switches without fallback ratios, three had top-k-experts, top-p-experts and top-p-tokens disabled. The other two had high top-k-expert values, 16 to be precise, and disabled top-p-experts and top-p-tokens. Since even a small value for top-k-tokens was enough, this indicates that there were tokens adhering to the grammar within the top k tokens but with low probabilities, as otherwise there would be top-p-token values present. This is confirmed by the finding that the first top-p-token value was 0.8, but only occurred after several runs with top-k-experts = 16.

On average, there were 0.0086 switches without fallback per switch, meaning only 0.86% of switches did not lead to a fallback in the end. The median value is 0.0 as only 384 of the 900 combinations had any switches without fallback. It is therefore safe to say that most switching attempts were unsuccessful as they did not lead to the selection of a different expert combination. Since the sweep was done using a small model and a reduced version of SPIDER, the generalization of these results for larger models and different datasets was not necessarily possible. We tried to find a combination that allowed for many working switches which

could be used to test our approach on larger models and more samples. Therefore, we selected a parameter pair with the execution accuracy of 0.1, with all four parameters set to nonzero-values, a switches without fallback to switches ratio of at least 0.015 and an overall number of at least 3000 switches. This resulted in four candidate combinations left of which we chose top-k-experts = 16, top-p-experts = 0.8, top-k-tokens = 20 and top-p-tokens = 0.8 as this combination had the highest switches without fallback to total switches ratio.

7.2 Transferring the Parameters

Applying the same parameters to the other selected models results in execution accuracy values described in table 7.1 and exact set match as reported in table 7.2. There is an improvement in execution accuracy when comparing the different grammars, for example the 0.090 compared to 0.208 of Mixtral on SPIDER [Yu+18]. This demonstrates the positive effect of a more constrained grammar on Structured Query Language (SQL) generation and consequently the use of Grammar Constrained Decoding (GCD).

Nonetheless, the results reported when utilizing expert switching do not differ from the ones without, indicating that expert switching in the last layer has no measurable effect regardless of the grammar. This aligns with the amount of fallbacks and most importantly the very low amount of switches without fallback in particular, for example 338 for Mixtral on SPIDER using the restrictive SQL grammar. Since this is the amount of switches without fallback, not the amount of times a fallback did not occur – there can be multiple switches within one token generation that is not leading to a fallback – the times a new token was generated without fallbacking is even lower. Subsequently, expert switching was used rarely and did not improve performance when it was used in generation. For both grammars tested, there were no switches without fallback at all for Helion 4x34b.

The low amount of fallbacks could have various meanings. On the one hand, a low amount of fallbacks might indicate that the decisions by the gating network were correct and there was no need to switch as it would only worsen the results. On the other hand, expert switching was only tested on the last Mixture of Experts (MoE) layer. At this point in time, the output by the model might be set to a large degree, implying that the final output can only be altered to a certain degree by the last layer. Switching experts on a different layer might yield more promising results.

Interestingly, the results without GCD for SPIDER, the dataset with less com-

Grammar			SPIDER	EHRSQL
Selected Instruction Set	**Switching**	Mixtral	0.090	0.002
		Helion 4x34b	0.021	0.009
	No switching	Mixtral	0.090	0.002
		Helion 4x34b	0.021	0.009
Restrictive SQL	**Switching**	Mixtral	0.208	0.099
		Helion 4x34b	0.056	0.181
	No switching	Mixtral	0.208	0.099
		Helion 4x34b	0.056	0.181
Without GCD		Mixtral	0.698	0.060
		Helion 4x34b	0.535	0.062

Table 7.1: Execution accuracy results for both selected models on the SPIDER and EHRSQL dataset. Both grammars are tested for switching and no switching. Additionally, results without GCD are provided.

Grammar			SPIDER	EHRSQL
Selected Instruction Set	**Switching**	Mixtral	0.044	0.002
		Helion 4x34b	0.000	0.000
	No switching	Mixtral	0.044	0.002
		Helion 4x34b	0.000	0.000
Restrictive SQL	**Switching**	Mixtral	0.082	0.010
		Helion 4x34b	0.000	0.001
	No switching	Mixtral	0.082	0.010
		Helion 4x34b	0.000	0.001
Without GCD		Mixtral	0.354	0.010
		Helion 4x34b	0.409	0.008

Table 7.2: Exact set match results for the two selected models on the SPIDER and EHRSQL dataset. Both grammars are tested for switching and no switching. Additionally, results without GCD are provided.

plex queries, were significantly better than results with GCD. This aligns with the findings of Beurer-Kellner et al. [BFV24], showing that GCD techniques do not always outperform methods without decoding constraints. While we chose *transformers-cfg* because of it being "minimally invasive", an improvement or on par-performance is still not guaranteed. More specifically, we have found that Large Language Models (LLMs) often generate explanatory text before code generation, which is not allowed in GCD. Therefore, generation with GCD leads to higher perplexity[5] that could have a negative impact on the resulting SQL statements. Please note that this behavior occurred even though we incorporated instructions into the prompt to immediately start generating SQL (see section 5.2).

7.3 Performance

Our sweep was executed on three different GPU types. We measured the runtime of each run and compared this to the amount of switches in total as each switch implies that a token generation is repeating, therefore adding performance overhead. Since the speeds of the underlying GPUs varied, we only compared results on the same GPU type. The scatter-plot 7.1 indicates a linear relationship between switches and runtime. To validate this, we fit linear regressions for each GPU type. These regressions strongly support the linear relationship with their respective r^2 scores. We achieved r^2 scores of 0.914, 0.911 and 0.839 for *NVIDIA A100-SXM4-80GB*, *NVIDIA H100 80GB HBM3* and *NVIDIA A100-SXM4-40GB* respectively. Since our linear regressions have coefficients of 46.24, 67.12 and 42.76, each switch adds less than $\frac{1}{40}$ seconds to the total runtime on average.

To evaluate the impact of our performance improvements, we compare the generation speed of Mixtral 8x7b Instruct v0.1 [Jia+24] using expert switching with our performance improvements as well as without improvements. We also compare the two with the speed of generation without expert switching as a third option. We compute *tokens-per-second* scores for all three versions on the validation set of SPIDER [Yu+18] on the same machine to eliminate the influence of hardware specifications on our results. To calculate the tokens-per-second, we count the number of generated tokens and measure the runtime without the initial loading of the model. Our performance improvements lead to a tokens-per-second score of 25.08 compared to 5.88 without improvements, a relative increase of 326.5%. The version without switching performed best with 31.96 tokens-per-second, a

5 A common metric to evaluate language models, e to the average of the negative log-likelihood of each token of a sequence. The larger the perplexity, the less likely the sequence is according to the model.

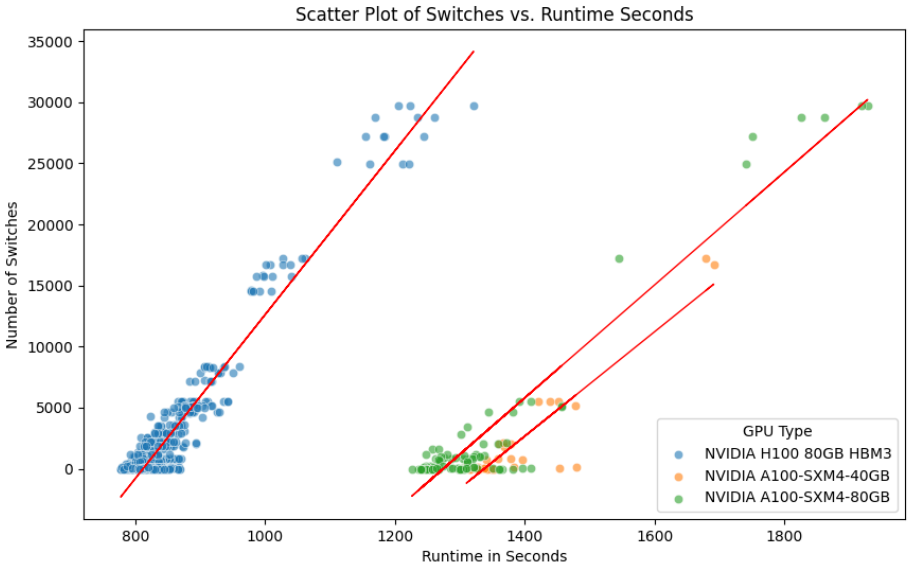

Figure 7.1: Scatter-plot showing switches in total in comparison to the runtime in seconds. We group the data based on GPU types and fit linear regressions for each of them.

moderate increase in the number of tokens per second compared to expert-switching with performance improvements, confirming the effectiveness of our strategies in section 5.4. We visualized the improvements in Figure 7.2 for easy comparison.

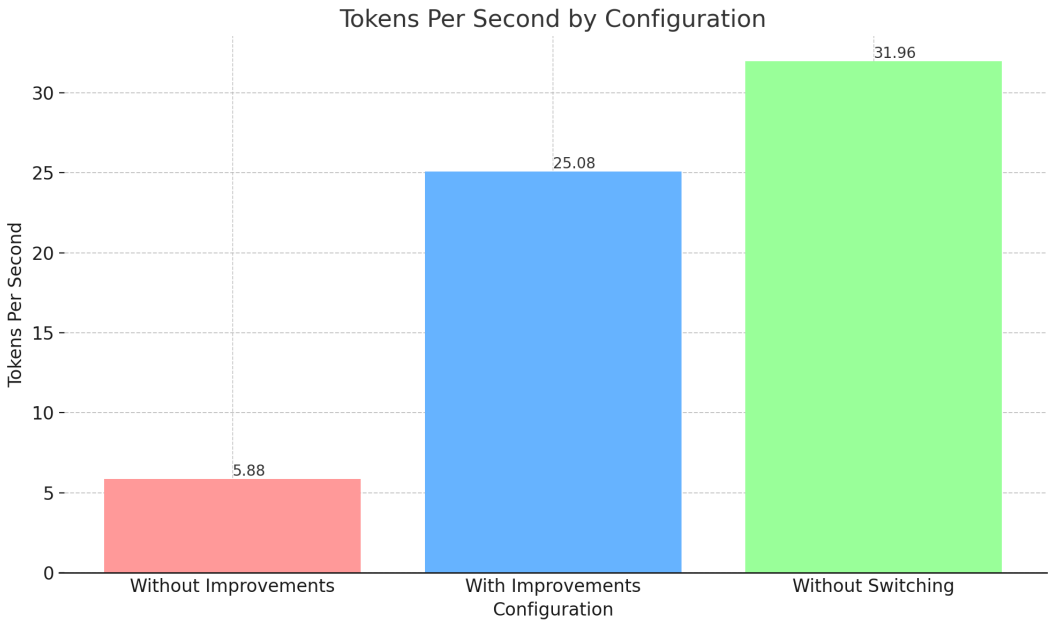

Figure 7.2: Bar graph displaying tokens-per-second scores for token generation using expert switching without performance improvements and with performance improvements as well as token generation without expert switching

Part IV

Applying to EHR

8 Incorporating Learnings into the AIR.MS Turtle

The AIR.MS Turtle – Turtle is short for *Technology-Empowered Utility for Research Tranquility and Language-Driven Exploration of Electronic Health Records* is a novel project created as part of a collaboration between the Hasso-Plattner-Institute and the Mount Sinai Health System. It was created to simplify and enhance the querying and analyzing of Electronical Health Records (EHRs) stored in a HANA database at Mount Sinai.

The so called AIR.MS database is a comprehensive collection of EHRs in the Observational Medical Outcomes Partnership format. This database is widely used by researchers conducting studies based on well-defined cohorts. Accessing this information involved a labor-intensive process. At first, researchers needed to find the required medical concepts that describe diseases, medication or patient's attributes. Afterwards, Structured Query Language (SQL) queries had to be built manually. As these queries can get very complex easily, this task is often both time-consuming and error-prone.

The AIR.MS Turtle addresses these challenges through a multi-step process similar to the traditional process by researchers that allows researchers to generate SQL from natural language. At first, users submit their query in natural language. Afterwards, Named Entity Recognition and Entity Linking is applied to identify and extract potentially relevant medical concepts. These concepts have to be validated by Turtle's users. A Large Language Model (LLM) is leveraged to generate a SQL query – the text-to-SQL task that laid the foundation for this work. Finally, the SQL query is executed on the HANA database. The extracted data can be exported as CSV, but can also be visualized as an image or described as a text using LLMs.

The results obtained from the SQL generation task often do not adhere to the SQL syntax and spell table or attribute names incorrectly. To fix this, this work has been conducted to evaluate the effects of Grammar Constrained Decoding (GCD) on EHR data. Leveraging the quality and speed improvements of Mixture of Experts (MoE) models, we developed *expert switching* as a potential trade-off for better accuracy with only slightly worse computational performance. Since expert switching does not provide a measurable benefit but GCD does, we built a restrictive SQL grammar for the version 5.3 of the OMOP Common Data Model [Gro09]. We use the same subset grammar tested in this work and only accept table and attribute names present in the database schema. After this, we added the

corresponding generation mechanism to the AIR.MS Turtle. Since GCD can only be applied when the underlying distribution is known, this mechanism does not offer improvements for the use of Turtle with external APIs. Therefore, the version of Turtle applying GCD does not use the OpenAI-API unlike the regular Turtle, but relies on available OpenSource-Models instead, leading to an overall decrease in generation capabilities. However, with the surge in model quality of available OpenSource models and the general applicability of GCD on any autoregressive generation with known output distributions, this gap might be closed sooner than later.

Conclusions & Outlook

In this thesis, we have explored expert switching for Mixture of Experts (MoE) layers and its influence on Structured Query Language (SQL) generation of Large Language Models (LLMs). While expert switching does not impact performance significantly, we confirm Grammar Constrained Decoding (GCD) provides a measurable enhancement for SQL generation for complex queries through autoregressive generation in language models, as proven by our extensive benchmarks on different models, namely Mixtral 8x7b Instruct v0.1, Mixtral 8x22b Instruct v0.1 and Helion 4x34b, across two SQL datasets. The application of GCD to a restrictive SQL grammar, hereby also restricting the table names and attributes, resulted in a particularly substantial improvement in execution accuracy.

There is a multitude of possible reasons for why expert switching might not have performed as expected. Our experiments restricted expert switching to the last MoE layer. While this provides the smallest computational overhead, the application on earlier layers might yield different results. Furthermore, the hyperparameter tuning was not exhaustive, leaving many potential combinations untested. Since the model used for the grid search was quite small and the accuracy very low, a particular combination would have had to have a relative improvement of 10% to be detectable. Using a larger model might result in specific combinations emerging as the best instead of having to choose from 756 candidate combinations. Additionally, expert switching was not used in most cases, evident from the small number of switches that did not lead to a fallback. This indicates that gating networks in MoE layers assign high probabilities to very few experts, usually exactly the k experts that are used per layer – we further verify this assumption by comparing the sweep runs with maximal switches without fallbacks for top-p-experts values of 0 and 0.8. The former resulted in 2216 switches without fallback out of 24950 while the latter had a total of 86 switches without fallback out of 4613 switches. Noting that the ratio of switches without fallback decreased significantly between the two as well as the amount of switches in total, there appear to be only a few expert combinations that can be tried before their sum exceeds the probability of 0.8. We therefore suggest evaluating expert switching without fallback mechanisms for probabilities of experts in place, i.e. without top-p-experts, as switching often results in early fallbacks otherwise.

For future work, there are multiple promising areas that should be explored

for further enhancement of SQL generation models. GCD is mostly applied to general purpose models, but not to task specific ones. It would be interesting to investigate the results of GCD on finetuned models as well, for example using dynamically generated SQL grammars also enforcing valid table and attribute names. Furthermore, GCD does not always outperform generation without constraints. The higher perplexity of generations starting immediately with SQL generation indicates that approaches combining free-text generation followed by GCD techniques upon starting the SQL part of the generation could provide a significant accuracy improvement by combining low perplexity with valid syntax. With the rapid improvement of LLMs, the application in different fields becomes more important. In order to utilize the capabilities of LLMs as individual components in a larger process and to enable efficient postprocessing, the demand for structured output increases. Therefore, though not always more accurate than unconstrained generation, the application of GCD on more diverse tasks and domains should be explored to further the applicability of language models.

Bibliography

[AHS85] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. **A learning algorithm for Boltzmann machines**. *Cognitive science* 9:1 (1985), 147–169 (see page 17).

[AI24] EdLab @ KAIST AI. *EHRSQL Codabench Competition*. https://www.codabench.org/competitions/1889 [Accessed: (2024-06-13)]. 2024 (see page 8).

[And+17] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. **Guided Open Vocabulary Image Captioning with Constrained Beam Search**. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Ed. by Martha Palmer, Rebecca Hwa, and Sebastian Riedel. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, 936–945. DOI: 10.18653/v1/D17-1098. URL: https://aclanthology.org/D17-1098 (see page 9).

[Arc+22] Samuel Arcadinho, David Aparício, Hugo Veiga, and António Alegria. **T5QL: Taming language models for SQL generation**. *arXiv preprint arXiv:2209.10254* (2022) (see page 17).

[Aus+21] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. **Program synthesis with large language models**. *arXiv preprint arXiv:2108.07732* (2021) (see page 7).

[Ben+15] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. **Conditional computation in neural networks for faster models**. *arXiv preprint arXiv:1511.06297* (2015) (see page 7).

[Ben13] Yoshua Bengio. **Deep learning of representations: Looking forward**. In: *International conference on statistical language and speech processing*. Springer. 2013, 1–37 (see page 6).

[BFV24] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. **Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation**. *arXiv preprint arXiv:2403.06988* (2024) (see pages 10, 11, 17, 32).

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. **Learning long-term dependencies with gradient descent is difficult**. *IEEE transactions on neural networks* 5:2 (1994), 157–166 (see page 6).

[Che+21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. **Evaluating large language models trained on code**. *arXiv preprint arXiv:2107.03374* (2021) (see page 7).

[Dat+20] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. **Plug and Play Language Models: A Simple Approach to Controlled Text Generation**. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=H1edEyBKDS (see page 9).

[Dev+18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. **Bert: Pre-training of deep bidirectional transformers for language understanding**. *arXiv preprint arXiv:1810.04805* (2018) (see page 5).

[ERS13] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. **Learning factored representations in a deep mixture of experts**. *arXiv preprint arXiv:1312.4314* (2013) (see page 7).

[FDZ22] William Fedus, Jeff Dean, and Barret Zoph. **A review of sparse expert models in deep learning**. *arXiv preprint arXiv:2209.01667* (2022) (see page 15).

[FLD18] Angela Fan, Mike Lewis, and Yann Dauphin. **Hierarchical neural story generation**. *arXiv preprint arXiv:1805.04833* (2018) (see pages 5, 6).

[FZS22] William Fedus, Barret Zoph, and Noam Shazeer. **Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity**. *Journal of Machine Learning Research* 23:120 (2022), 1–39 (see pages 6, 15).

[Gen+23a] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. *GCD Repository*. https://github.com/epfl-dlab/GCD [Accessed: (2024-05-19)]. 2023 (see page 17).

[Gen+23b] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. **Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning**. In: *The 2023 Conference on Empirical Methods in Natural Language Processing*. 2023. URL: https://openreview.net/forum?id=KkHY1WGDII (see pages 10, 17).

[Gen+24] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. *transformers-CFG*. https://github.com/epfl-dlab/transformers-CFG [Accessed: (2024-05-19)]. 2024 (see pages 17, 27).

[God+24] Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vlad Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. **Arcee's MergeKit: A Toolkit for Merging Large Language Models**. *arXiv preprint arXiv:2403.13257* (2024) (see page 16).

[Gra12] Alex Graves. **Sequence transduction with recurrent neural networks**. *arXiv preprint arXiv:1211.3711* (2012) (see page 5).

[Gra13] Alex Graves. **Generating sequences with recurrent neural networks**. *arXiv preprint arXiv:1308.0850* (2013) (see page 5).

[Gro09] OHDSI CDM Working Group. *OMOP Common Data Model*. https://ohdsi.github.io/CommonDataModel/index.html [Accessed: (2024-06-13)]. 2009 (see page 37).

[Gul11] Sumit Gulwani. **Automating string processing in spreadsheets using input-output examples**. *ACM Sigplan Notices* 46:1 (2011), 317–330 (see page 7).

[HL17] Chris Hokamp and Qun Liu. **Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search**. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Regina Barzilay and Min-Yen Kan. Vancouver, Canada: Association for Computational Linguistics, July 2017, 1535–1546. DOI: 10.18653/v1/P17-1141. URL: https://aclanthology.org/P17-1141 (see page 9).

[Hol+19] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. **The curious case of neural text degeneration**. *arXiv preprint arXiv:1904.09751* (2019) (see pages 6, 20).

[Hua+23] Dong Huang, Qingwen Bu, Jie M Zhang, Michael Luck, and Heming Cui. **AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation**. *arXiv preprint arXiv:2312.13010* (2023) (see page 7).

[Jac+91] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. **Adaptive mixtures of local experts**. *Neural computation* 3:1 (1991), 79–87 (see page 6).

[Jia+24] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. **Mixtral of experts**. *arXiv preprint arXiv:2401.04088* (2024) (see pages 6, 15, 26, 32).

[JJ90] Robert Jacobs and Michael Jordan. **A competitive modular connectionist architecture**. *Advances in neural information processing systems* 3 (1990) (see page 6).

[Kap+20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. **Scaling laws for neural language models**. *arXiv preprint arXiv:2001.08361* (2020) (see page 6).

[Kes+19] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. **Ctrl: A conditional transformer language model for controllable generation**. *arXiv preprint arXiv:1909.05858* (2019) (see page 18).

[Lan+23] Wuwei Lan, Zhiguo Wang, Anuj Chauhan, Henghui Zhu, Alexander Li, Jiang Guo, Sheng Zhang, Chung-Wei Hang, Joseph Lilien, Yiqun Hu, et al. **Unite: A unified benchmark for text-to-sql evaluation**. *arXiv preprint arXiv:2305.16265* (2023) (see page 8).

[Lee+22] Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. **Ehrsql: A practical text-to-sql benchmark for electronic health records**. *Advances in Neural Information Processing Systems* 35 (2022), 15589–15601 (see pages 8, 25).

[Li+16] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. **Deep Reinforcement Learning for Dialogue Generation**. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Ed. by Jian Su, Kevin Duh, and Xavier Carreras. Austin, Texas: Association for Computational Linguistics, Nov. 2016, 1192–1202. DOI: 10.18653/v1/D16-1127. URL: https://aclanthology.org/D16-1127 (see page 18).

[Li+24] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. **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 (2024) (see page 8).

[Liu+18] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. **Generating wikipedia by summarizing long sequences**. *arXiv preprint arXiv:1801.10198* (2018) (see page 5).

[Mel+24] Daniel Melcer, Nathan Fulton, Sanjay Krishna Gouda, and Haifeng Qian. **Constrained Decoding for Code Language Models via Efficient Left and Right Quotienting of Context-Sensitive Grammars**. *arXiv preprint arXiv:2402.17988* (2024) (see page 9).

[OS20] Augustus Odena and Charles Sutton. **Learning to represent programs with property signatures**. *arXiv preprint arXiv:2002.09030* (2020) (see page 7).

[PJW24] Maxime Peyrard, Martin Josifoski, and Robert West. **The Era of Semantic Decoding**. *arXiv preprint arXiv:2403.14562* (2024) (see page 10).

[Pop+23] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. **Efficiently scaling transformer inference**. *Proceedings of Machine Learning and Systems* 5 (2023) (see page 21).

[PR24] Mohammadreza Pourreza and Davood Rafiei. **Din-sql: Decomposed in-context learning of text-to-sql with self-correction**. *Advances in Neural Information Processing Systems* 36 (2024) (see page 8).

[Qin+22] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. **A survey on text-to-sql parsing: Concepts, methods, and future directions**. *arXiv preprint arXiv:2208.13629* (2022) (see page 26).

[Rad+18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. **Improving language understanding by generative pre-training** (2018) (see page 5).

[Rad+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. **Language models are unsupervised multitask learners**. *OpenAI blog* 1:8 (2019), 9 (see pages 5, 18).

[Ron17] Jonathan Leffler Ron Savage. *SQL Grammars in BNF*. https://github.com/ronsavage/SQL [Accessed: (2024-05-19)]. 2017 (see page 17).

[RSK17] Maxim Rabinovich, Mitchell Stern, and Dan Klein. **Abstract syntax networks for code generation and semantic parsing**. *arXiv preprint arXiv:1704.07535* (2017) (see page 10).

[Sha+17] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. **Outrageously large neural networks: The sparsely-gated mixture-of-experts layer**. *arXiv preprint arXiv:1701.06538* (2017) (see pages 6, 7, 15, 19).

[SSB21] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. **PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models**. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Ed. by Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, 9895–9901. DOI: 10.18653/v1/2021.emnlp-main.779. URL: https://aclanthology.org/2021.emnlp-main.779 (see pages 9, 10).

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. **Sequence to sequence learning with neural networks**. *Advances in neural information processing systems* 27 (2014) (see page 5).

[Tea24] Mistral AI Team. *Mixtral 8x22b introduction blog post*. https://mistral.ai/news/mixtral-8x22b/ [Accessed: (2024-06-26)]. 2024 (see page 15).

[Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. **Attention is all you need**. *Advances in neural information processing systems* 30 (2017) (see page 5).

[Wan+23] Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. **Grammar Prompting for Domain-Specific Language Generation with Large Language Models**. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., 2023, 65030–65055. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/cd40d0d65bfebb894ccc9ea822b47fa8-Paper-Conference.pdf (see page 10).

[Wey24] Weyaxi. *Helion-4x34B*. https://huggingface.co/Weyaxi/Helion-4x34B/tree/main [Accessed: (2024-05-18)]. 2024 (see page 15).

[YN17] Pengcheng Yin and Graham Neubig. **A syntactic neural model for general-purpose code generation**. *arXiv preprint arXiv:1704.01696* (2017) (see page 10).

[Yu+18] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. **Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task**. *arXiv preprint arXiv:1809.08887* (2018) (see pages 7, 8, 25, 29, 30, 32).

[Zhu+24] Qihao Zhu, Qingyuan Liang, Zeyu Sun, Yingfei Xiong, Lu Zhang, and Shengyu Cheng. **GrammarT5: Grammar-Integrated Pretrained Encoder-Decoder Neural Model for Code**. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024, 1–13 (see page 10).

[ZWS24] Li Zhong, Zilong Wang, and Jingbo Shang. **LDB: A Large Language Model Debugger via Verifying Runtime Execution Step-by-step**. *arXiv preprint arXiv:2402.16906* (2024) (see page 7).

[ZXS17] Victor Zhong, Caiming Xiong, and Richard Socher. **Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning**. *CoRR* abs/1709.00103 (2017) (see page 8).

[ZYK20] Ruiqi Zhong, Tao Yu, and Dan Klein. **Semantic evaluation for text-to-SQL with distilled test suites**. *arXiv preprint arXiv:2010.02840* (2020) (see pages iii, 8, 25).

Glossary

EHR A digital version of a patient's health records containing demographics, medical history, medications and more. iii, v, 1, 2, 8, 25, 37

GCD Pruning the probability distribution of a Seq2Seq-Model to allowed tokens according to some predefined grammar. iii, 9, 10, 17, 19, 27, 30–32, 37–40

LLM An AI model containing many parameters capable of generating human-like text for many tasks, used in various applications across various domains. Does not perform ideally on structured output. iii, 1, 5–8, 15–18, 20, 32, 37, 39, 40

MoE A technique used in deep learning where a model or layer of a model consists of multiple experts of which only few are activated per input. iii, 1, 6, 15, 16, 19, 22, 30, 37, 39

NLP A branch of computer science that involves giving computers the ability to interpret and produce language. 1

SQL A standardized programming language used for manipulating and managing relational databases for tasks such as querying data and updating or deleting records. iii, v, 1, 2, 7–10, 17, 25–27, 30–32, 37, 39, 40

Part V

Appendix

A Grammars

Our grammar used in EBNF. The entrypoint for this EBNF is defined as root. Terminals are marked with quotes.

```
root ::= query

query ::= select ws attributes ws "FROM" ws tables ws ("WHERE"
ws condition)? ws ("GROUP BY" ws attribute)? ws ("ORDER BY" ws
order)? ws ("LIMIT" ws number)? ws ("OFFSET" ws number)? ws ";"

select ::= "SELECT" | "SELECT" ws "DISTINCT"

attribute_func ::= "COUNT" "(" ("*" | attribute | "DISTINCT" ws
attribute) ")" | "AVG" "(" attribute ")" | "MAX" "(" attribute
")" | "MIN" "(" attribute ")"

single_attribute ::= string

attribute ::= (single_attribute | attribute_func) ("AS" ws string)?

attributes ::= attribute ("," ws attribute)*

table ::= string

join ::= "JOIN" | "LEFT JOIN" | "RIGHT JOIN" | "FULL JOIN" | "INNER
JOIN"

join_condition ::= attribute "=" attribute

tables ::= table (ws join ws table ws "ON" ws join_condition)*

condition ::= single_condition | multiple_conditions

single_condition ::= attribute "=" value | attribute "<" value
| attribute ">" value | attribute "<=" value | attribute ">=" value
```

```
| attribute "!=" value | attribute "LIKE" like_value | attribute
"IS" "NULL" | attribute "IS" "NOT" "NULL" | attribute "IN" "("
value ("," value)* ")"

multiple_conditions ::= "(" condition "AND" condition ")" | "("
condition "OR" condition ")"

order ::= attribute ("ASC" | "DESC")?

numeric_value ::= number

string_value ::= ("'" string_with_ws "'")

like_value ::= "'" ([a-z] | [A-Z] | "%" | "_")* "'"

value ::= numeric_value | string_value

string_with_ws ::= (string ws)* string

string ::= ([a-z] | [A-Z])+

number ::= ("-"? ([0-9] | [1-9] [0-9]*)) ("." [0-9]+)? ([eE] [-+]?
[0-9]+)? ws

ws ::= (" " | "\t" | "\n")
```

For the grammar with restricted names, we dynamically adjust single attribute and table to a union between all possible options for the respective value. Please note that this still allows for attributes to be chosen from tables that do not have the specified attribute as long as there is one table in the schema with the attribute.

B Prompt

The prompt comes with two placeholders, *create_tables* and *problem*. We dynamically substitute *problem* with the natural language query and *create_tables* with the create tables statements extracted from the *tables.json*.

B.1 The prompt

```
You are a natural language to SQL translator.
For the given schema, output the SQL query you need to answer the
problem.

The problem is given below in natural language.
If part of the problem can not be accomplished using SQL queries,
for example visualization requests,
only output the most meaningful sql query that returns the data
required for the problem.
Additionally, here are the CREATE TABLE statements for the schema:
{create_tables}

Do not include the CREATE TABLE statements in the SQL query. Do
not write anything after the SQL query.
Do not write anything other than the SQL query - no comments, no
newlines, no print statements.

Problem: {problem}
```

B.2 Model specific changes

For the Mixtral family, we add [INST] at the start and [\INST] at the end of the instruction. For Helion, we substitute “Problem:” with “USER:” and append “ASSISTANT:” to the prompt to comply with their prompting schema.

C Grid search

We used *wandb* to perform our grid search using their hyperparameter sweep feature. The sweep configuration used was

```
metric:
  goal: minimize
  name: execution_accuracy
name: ehrsqlswitchingsweep
parameters:
  top_k_experts:
    values:
      - 0
      - 1
      - 2
      - 4
      - 8
      - 16
  top_k_tokens:
    values:
      - 0
      - 5
      - 10
      - 20
      - 40
  top_p_experts:
    values:
      - 0
      - 0.1
      - 0.2
      - 0.4
      - 0.6
      - 0.8
  top_p_tokens:
    values:
      - 0
```

- 0.2
- 0.4
- 0.6
- 0.8