FastRAT: Fast and Efficient Cross-lingual Text-to-SQL Semantic Parsing

Pavlos Vougiouklis[1†] **Nikos Papasarantopoulos**[2†‡] **Danna Zheng**[3†] **David Tuckey**[4†]
Chenxin Diao[1] **Zhili Shen**[1] **Jeff Z. Pan**[1*]
[1]Huawei Technologies Edinburgh RC, CSI, Edinburgh, United Kingdom
{pavlos.vougiouklis, chenxindiao, zhilishen, jeff.pan}@huawei.com
[2]Priceline, Edinburgh, United Kingdom
nikos.papasa@gmail.com
[3]University of Edinburgh, Edinburgh, United Kingdom
D.Zheng-6@sms.ed.ac.uk
[4]Imperial College London, London, United Kingdom
david.tuckey17@imperial.ac.uk

Abstract

Recent advances of large pre-trained language models have motivated significant breakthroughs in various Text-to-SQL tasks. However, a number of challenges inhibit the deployment of SQL parsers in commercial applications. In this paper, we focus on two such challenges: decoding speed and multilingual input, and introduce FastRAT, a model that includes (i) a decoder-free framework to quickly generate SQL queries from natural language questions based on SQL Semantic Predictions, (ii) a cross-lingual multi-task pre-training scheme, and (iii) a method, based on distant supervision, to extend a semantic parser to new languages.

We apply FastRAT on CSpider and Spider, two challenging zero-shot semantic parsing benchmarks. Our system achieves an average of 10x decoding speedup over a set of competitive baselines based on auto- or semi-auto-regressive decoding. In the cross-lingual CSpider dataset, our approach achieves an exact query match accuracy score of 61.3, outperforming the relevant competition. In the monolingual task, it maintains competitive performance by exhibiting $< 5\%$ accuracy drop compared to disproportionately slower solutions.

1 Introduction

The task of Text-to-SQL semantic parsing is to transform natural language questions into SQL queries. The resulting queries can be executed by the corresponding database instance, in order for appropriate results to be returned to the end-user, who might not be familiar with SQL or the schema of the given database (Zelle and Mooney, 1996; Dong and Lapata, 2016). Given their accessibility benefits, Text-to-SQL applications have become increasingly popular recently, with many corporations developing Business Intelligence platforms.

Despite the market potential, existing solutions are usually based on larger language models, and have notable limitations—they tend to work only for simple queries and data sources, and exhibit relatively slow processing times. Furthermore, while database schemata are often in English, data records might be in other languages, and users might need to query them in languages other than English.

In an effort to alleviate the shortcomings of Text-to-SQL solutions, increasingly difficult datasets and benchmarks have been developed (Zhong et al., 2017; Yu et al., 2018; Min et al., 2019; Dou et al., 2023; Zhang et al., 2023). The efforts to explore the generalisability of such Text-to-SQL systems have recently culminated with the introduction of multiple database datasets, which distinguish between training and evaluation databases. As a result, models are expected to be tested on databases they have not met during training. We refer to this setup as cross-database semantic parsing.

Research efforts for addressing the challenges introduced by this cross-database setting have converged to the general encoder-decoder framework (Dong and Lapata, 2016; Kočiský et al., 2016; Wang et al., 2020a; Rubin and Berant, 2021). In this framework, the encoder (i.e. usually based on a pre-trained language model) processes the input natural language question along with the corresponding database (i.e. database schema with or without its relevant values), whereas the decoder seeks to decode the SQL query (Wang et al., 2020a; Rubin and Berant, 2021; Cao et al., 2021).

A substantial amount of work has focused on the monolingual cross-database setup, where both

[*] Corresponding author.
[†] The authors contributed equally to this work.
[‡] Work done while at Huawei Edinburgh Research Centre.

Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers), pages 564–576
November 1–4, 2023. ©2023 Association for Computational Linguistics

the natural language and the database schema are in English, by (i) increasing the size or complexity of architectures (Wang et al., 2020a; Cao et al., 2021; Shaw et al., 2021) or (ii) introducing series of pre-training stages (Yu et al., 2021). The computational cost of such approaches tends to increase, with a few approaches investigating how improved execution times can facilitate their deployability in production environments (Lukovnikov and Fischer, 2021; Rubin and Berant, 2021).

In this paper, we altogether delete the decoder, which is traditionally responsible for the largest amount of the necessary computational workload (Akoury et al., 2019), and treat the generation of the SQL query as a one-step multi-label classification task. We use the RAT-SQL encoder (Wang et al., 2020a), a popular choice among state-of-the-art models employed for cross-database semantic parsing,[1] and we introduce an SQL generator capable of quickly forming the expected SQL query given a set of semantic predictions on top of the input database schema's elements.

Another factor that prohibits the deployability of state-of-the-art systems is their reliance on the monolingual setting, where both the natural language questions and the database schema are in the same language. While schema information of non-English databases is often available in English, there are several challenges in transferring a monolingual system to the cross-lingual setup, where the natural language question is in a different language than the database. Most existing cross-lingual solutions have focused on shared database semantic parsing setups where evaluation databases are known during training (Sherborne et al., 2020; Xia and Monti, 2021; Sherborne and Lapata, 2022).

In this paper, we address the above challenges, enabling Text-to-SQL semantic parsing in the cross-lingual and cross-database setup in a fast and scalable way, and facilitating the deployment of relevant solutions in business intelligence products. We apply our model (FastRAT) on CSpider and Spider, two challenging zero-shot semantic parsing benchmarks. Our system achieves an average of 10 times decoding speedup over competitive baselines based on auto- or semi-auto-regressive decoding. Interestingly, FastRAT, a 673M-parameter architecture, is able to outperform the multi-billion-parameter ChatGPT system across both benchmarks, while being significantly less computationally expensive than it. In particular, on a cross-lingual dataset, our approach achieves an exact query match accuracy score of 61.3, outperforming the relevant competition.

2 Background

Our method draws inspiration from work on fast decoding and cross-lingual semantic parsing.

2.1 Fast Decoding

Text-to-SQL semantic parsing problems are quite often solved with architectures that follow the sequence-to-sequence paradigm. Such models include an encoder, which reads and encodes the input, and a decoder, which predicts the corresponding output, often sequentially. Depending on the size and complexity of the models, the encoding and decoding process can be long; however, admittedly, decoding is the most time-consuming part.

The main reason for the time inefficiency of decoders is the fact that decoding is usually performed in an auto-regressive manner: one token at a time, in a specified direction. Auto-regressive models are based on the idea that, in order to predict the k-th element of a sequence, a model will need to have first predicted and consulted the previous $k-1$ elements. Interestingly, due to the intuitive information flow, auto-regressive models generally achieve better performance compared to their non-auto-regressive counterparts (Akoury et al., 2019).

A more efficient decoder would be one that is able to predict more than one token or element per decoding timestep t. Zhu et al. (2020); Lukovnikov and Fischer (2021) reduce the number of decoding timesteps by using insertion-based tree decoding. The most recent work following this paradigm is the work by Rubin and Berant (2021). The authors propose a semi-auto-regressive semantic parser, SmBoP, that works in a bottom-up fashion by constructing, for each timestep t, the top-scoring subtrees of height less than or equal to t. Since SmBoP operates in a bottom-up fashion, all sub-trees of a certain height can be computed in parallel, thus reducing the computational complexity of the decoding task. While SmBoP substantially improves decoding times over auto-regressive models, its average runtime is still quite high to meet deployment standards. For reference, in a well-known Text-to-SQL dataset (Yu et al., 2018), SmBoP needs an average of 9 timesteps to decode SQL queries.

[1] https://yale-lily.github.io/spider and https://taolusi.github.io/CSpider-explorer/

Taking the semi-auto-regressive idea to its extreme, one can have a completely non-auto-regressive decoder that generates the whole output in one timestep. Inspired by recent work (Yu et al., 2021), instead of predicting SQL queries as sequences of tokens, we predict sets of SQL Semantic Prediction (SSP) labels associated with each table and column name of the input database. Subsequently, using a simple algorithm, we translate the set of SSP labels into an SQL query.

2.2 Cross-lingual Semantic Parsing

The efforts of the scientific community to build systems for cross-lingual semantic parsing have led to the development of a series of relevant benchmark datasets (Min et al., 2019; Dou et al., 2023; Zhang et al., 2023). Most recent approaches for cross-lingual semantic parsing have sought to localise parsers to new target languages using back-translations (Sherborne et al., 2020) or machine translation (Xia and Monti, 2021; Shi et al., 2022). Such solutions precondition access to high-quality machine translation or source-target language alignment systems. However, under realistic circumstances, the resulting data diverges from actual test cases, leading to poor generalisation. More recently, Sherborne and Lapata (2022) proposed a multi-task encoder-decoder model to transfer parsing knowledge to additional languages using only English logical form paired data and in-domain natural language corpora in the target languages.

While this approach has shown promising results for shared-database semantic parsing, challenges associated with linking column mentions to unseen databases are not addressed in the cross-lingual setup. In this work, we introduce a unified framework, based on distant supervision, for generating high-quality, aligned logical form queries in SQL and questions in a target language. This framework does not assume the existence of any machine translation systems, thus preserving the resulting data from any relevant inefficacies. Furthermore, inspired by Sherborne and Lapata (2022), we propose a new multi-task training scheme which offers tighter interaction between the input language and the schema representations.

3 Our Approach

Let $\mathbf{q} = q_1, q_2, \ldots, q_Q$ be the sequence of tokens of a natural language question, $\mathbf{t} = t_1, t_2, \ldots, t_T$ tables, and $\mathbf{c} = c^1_1, c^1_2, \ldots, c^1_{C_1}, \ldots, c^T_{C_T}$ columns of a database $\mathbb{D}$ related to $\mathbf{q}$, where $C_1, \ldots, C_T \in \mathbb{N}$ are the indices of the last column of table t_1 and t_T respectively.

3.1 Query Decoding with SSP Labels

We follow the formulation of the auxiliary SSP task, introduced by Yu et al. (2021). In this task, given a database schema and a natural language question, the goal is to compute the SQL operation in which each column of the input schema would participate in the expected SQL query. An idealised example of the task of assigning SSP labels to the columns of an input schema and of their relevance to the expected SQL query is presented in Table 7 of Appendix A.

Our architecture consists of a pre-trained language model which takes as input the concatenation of the input natural language question with the column and table names of a database schema of interest. In contrast to Yu et al. (2021), we extend the SSP task across the entire schema, and compute relevant SQL operations across both column and table names. Furthermore, we treat learning this version of the task as the final training step required by our system, and not as an additional pre-training phase. Consequently, we minimise any potential task-relevance gaps that can be introduced by pre-training and fine-tuning architectures in tasks of different natures. Our system is capable of efficiently decoding the SQL query that answers the input question $\mathbf{q}$, by using the set of SSP labels that are computed for each element of the input schema.

3.1.1 From SQL to SSP Labels

Columns and tables (schema elements) appear in SQL queries in specific contexts: each occurrence of a schema element appears in a specific SQL keyword or clause, along with an aggregate (e.g. `COUNT` or `MAX`) or an operator (e.g. `=` or `!=`), possibly within a sub-query. Each schema element occurrence can be translated into a string or snippet, that contains the basic information about where this element appears in the SQL query. Such information is encoded in SSP labels by design: each label is the concatenation of the snippets corresponding to instances of a particular column or table name in the SQL query.

Occurrence snippets are composed of three elements: (i) the *nest* which indicates in which sub-query the schema element appears (empty for the main query), (ii) the *SQL keyword/clause* that is involved (`SELECT`, `FROM`, `WHERE` etc), and (iii) the various arguments providing additional information, such as aggregate or comparison operators. For instance, in Figure 1, column furniture_id of table clients appears in the sub-query indicated by the "OP_SEL" nest. This column appears in the WHERE statement of the SQL query, and its comparison operator is "=". Consequently, the occurrence snippet for the furniture_id column is "OP_SEL WHERE =". The SSP label for furniture_id is then the concatenation of all its occurrence strings; in this case only "OP_SEL WHERE =".

SQL

```
SELECT * , T1.tables , count(T2.chairs)
FROM room as T1 JOIN furniture as T2 ON
T1.place = T2.place WHERE T2.owner IN (
SELECT owner FROM clients WHERE
furniture_id = 5 ) ORDER BY T1.tables DESC
```

SSP labels

```
__all__ : SELECT none
room : FROM
 .chairs : SELECT count
 .owner : WHERE IN
furniture : FROM
 .tables : SELECT none ORDER BY none DESC
clients : OP_SEL FROM
 .owner : OP_SEL SELECT
 .furniture_id : OP_SEL WHERE =
```

Figure 1: Example of the SSP labels that describe a particular SQL query.

We model the various possible sub-queries configurations that can appear in an SQL query using the following keywords in the nest: (i) OP_SEL for sub-query, (ii) INTERSECT for an occurrence in a query after an intersect, (iii) UNION for an occurrence in a query after a union and (iv) EXCEPT for an occurrence in a query after an except. Keywords can be chained together when needed. For example, an occurrence of a column in a sub-query after an intersect operation would result in the inclusion of the following nest snippet in its corresponding occurrence string: "INTERSECT OP_SEL".

Once the nest of an occurrence string is determined, we add the keyword/clause (i.e. SELECT, FROM, WHERE, HAVING, GROUP_BY and ORDER_BY) in which it appears. Each operator might be followed by extra keywords:

- For SELECT, we add the aggregation operator that is applied to the column (e.g., SELECT count for the room.chairs column in Figure 1). In case no aggregator is applied, we add the "none" keyword.
- FROM does not require extra information. Note that FROM can only appear in the SSP label of a table name.
- For HAVING, we add the relevant aggregate and comparison operator (e.g., "=", "<", etc).
- For WHERE, we add the comparison operator, and an "OR" keyword if there is an "OR" condition next to the occurrence in the SQL query.
- For GROUP_BY, we add the aggregator.
- For ORDER_BY, we add the relevant aggregator, the order type ("ASC" or "DESC") and the keyword "LIMIT", if applicable.

3.1.2 From SSP Labels to SQL

Given a set of SSP labels, it is straightforward to uncover the basic structure of the corresponding SQL query, and find the appropriate places of the column and table mentions.

We start by the nests appearing in the SSP labels, as they indicate the overall structure of the SQL query. For each sub-query identified, we extract from the SSP labels the columns that appear in it, and place them in the right position (using operators SELECT, WHERE, etc), respecting the SQL format.

Since SSP labels do not provide details about the position of sub-queries, we construct all possible SQL queries by placing each sub-query in all appropriate HAVING and WHERE statements. This process however does not build JOIN statements, since SSP labels do not contain column-joining operations.

3.1.3 Table Joining Algorithm

The most challenging part of decoding is finding the correct joins for the tables involved. We have developed an algorithm capable of generating complex join statements and finding tables that are required to construct SQL queries in which tables might not necessarily appear in any other places apart from the JOIN operators.

The algorithm (i.e. Algorithm 1) starts by adding all tables that should appear in the FROM statement into a list L_t. It then builds a set of tables L_s, of which the join statements are known: it begins by adding tables such that each table can be directly joined with another one in L_s using a foreign key (lines 5–8). Then, it attempts to complete the JOIN statement as follows: for each table t_i left to join, it finds a table t_j in the database schema that can be joined to t_i and t_j can be joined to a table in L_s (lines 10–13). A more detailed description of the involved steps is provided in Appendix B.

Algorithm 1: Joining algorithm for finding the appropriate join statement given a set of tables.

```
Step 1: Initialise $L_t$ with all the tables appearing the the query;
Step 2: Find the direct joining statements;
    Initialise $L_s$ = [$t_i$] with a random table $t_i$ from $L_t$;
    Delete $t_i$ from $L_t$;
    while $t_b \in L_t$ s.t $T_a \rightleftharpoons T_b$ for $t_a \in L_s$ do
        Add $t_b$ to $L_s$;
        Delete $t_b$ from $L_t$;
    end
Step 3: Find the join statements that require additional tables;
    while $t_b \in L_t$ s.t $T_a \Longleftrightarrow T_b$ for $t_a \in L_s$ do
        Add $t_b$ and all intermediate tables to $L_s$;
        Delete $t_b$ from $L_t$;
    end
Return $L_s$
```

3.2 Pre-training Data Construction

Most Text-to-SQL models are trained and tested on a single language. However, the commercial need to deploy similar models for more than one language is very common. While architectures can be reused, adapting an existing model to a new language requires large amount of data in that language, which can be cumbersome to find in multiple languages. As an example, a popular pre-training dataset for Text-to-SQL (Yu et al., 2021) has 400k examples; obtaining a good translation of it would take a substantial budget and some months of annotation work. Our approach to efficiently create a dataset in a new language works as follows:

- Starting from a dataset containing tuples of the form (natural language question, SQL query) we obtain templates for both elements, resulting in a small dataset of tuples of the form (natural language template, SQL template).
- We translate the natural language templates to the new language, resulting in a dataset of (natural language template in new language, SQL template) tuples.
- We sample databases, table and column names from data in the new language, and populate multiple instances of each template.

Table 1 shows this process for a single example.

en NL question	How many department heads have an age over 40?
SQL	`SELECT COUNT(*) FROM head WHERE age > 40`
en NL template	`COUNT(*) T0 have C0 OP0 VAL0`
SQL template	`SELECT COUNT(*) FROM T0 WHERE C0 OP0 VAL0`
zh NL template	`VAL0 C0 OP0 的 T0 有多少?`
SQL template	`SELECT COUNT(*) FROM T0 WHERE C0 OP0 VAL0`
zh NL question	40 岁 以上 的 部门主管 有多少?
SQL	`SELECT COUNT(*) FROM head WHERE age > 40`

Table 1: Example of how data instances consisting of natural language question (in en) and SQL query can be translated into a new language (i.e. zh) using our approach. C0, T0, OP0, VAL0 are placeholders for columns, tables, operators, and values respectively.

Abstracting away from the templates and considering the grammar that can create natural language and SQL templates, one can think of this approach as a synchronous context-free grammar that has rules to produce text and corresponding SQL in two languages. An example of such grammar is shown in Table 8 of the Appendix.

In our experiments, we manually translate the rules of the grammar released by Yu et al. (2021) to Chinese. In our translated grammar version, there are 328 natural language production rules, corresponding to 88 SQL templates; each SQL template is paired with 1–18 natural language templates. We generate the final version of the dataset by adding terminal rules with names from new databases.

This approach, inspired by distant supervision (since it creates a weakly labelled dataset from existing data), requires substantially lower effort and resources than translating the whole dataset.

3.3 Pre-training Strategy

The cross-lingual capabilities of our model are powered by a multi-task pre-training process. Starting off with an already pre-trained model (such as, XML-RoBERTa; Conneau and Lample 2019), we continue pre-training with a compound loss consisting of two terms: one for Semantic SQL Prediction (SSP) and one for Language Prediction (LP). A schematic representation of our pre-training framework can be seen in Figure 2 of the Appendix.

Semantic SQL Prediction (SSP) This objective encourages the model to implicitly learn associations between table and column names and their mentions in the text. Specifically, we prepend all columns with a delimiter $\langle/s\rangle$ in the input, and apply a sequence of {Linear, GELU, and LayerNorm} layers on top of the encoder representations for each $\langle/s\rangle$. The loss term for this task, $\mathcal{L}_{\text{SSP}}$, is the cross-entropy loss of the SSP label predictions.

Language Prediction (LP) Aiming to reduce the distance between the distributions of different languages in the encoder representations, we include a loss term for language prediction of the natural language question. Specifically, we classify between the available languages using the representation generated for the first token $\langle s\rangle$ of the input. The loss term for this task, $\mathcal{L}_{\text{LP}}$, is the cross-entropy loss of the language prediction.[2]

We combine the two above-mentioned loss terms to get the loss which we use to pre-train our model,

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{SSP}} - \mathcal{L}_{\text{LP}}. \qquad (1)$$

During pre-training, our model tries to minimise the above cost function. During fine-tuning, $\mathcal{L}_{\text{LP}}$ is omitted, and the system is optimised using $\mathcal{L}_{\text{SSP}}$.

4 Experiments

We experiment on two SQL semantic parsing datasets, seeking to explore the effectiveness of our approach on both a monolingual and a cross-lingual setup. Specifically, we report experiments on CSpider (Min et al., 2019) and Spider (Yu et al., 2018), which contain database schema information and examples in Chinese and English respectively. Since CSpider is a translated version of the Spider dataset, the characteristics of the two with respect to structure and number of examples are identical.

Both datasets contain $8,659$ examples of questions and SQL queries along with their relevant SQL schemata (i.e. 146 unique databases). Since the test splits are only available through the evaluation servers associated with the datasets, we focus our evaluation on the development set, which is used as a test set in our experiments. This split consists of $1,034$ examples of questions on 20 unique databases that are not seen during training. Consistently with the CSpider[3] and Spider[4] leaderboards, we report results using exact match accuracy.[5]

4.1 Experimental Setup

FastRAT provides a strong decoding subsystem, but has no specific implementation for a text and schema encoder. As such, in order to instantiate an end-to-end system, we mount FastRAT to the RAT-SQL encoder.[6] We couple this encoder with XLM-RoBERTa (i.e. XLM-RoBERTa-large) to create a cross-lingual semantic parser, and with BERT$_{\text{LARGE}}$ to create a monolingual semantic parser. The selection of BERT$_{\text{LARGE}}$ and XLM-RoBERTa-large is in line with the pre-trained language models that were used by the RAT-SQL baseline (Wang et al., 2020a), and the GraPPa pre-training framework (Yu et al., 2021). For the experiments on CSpider, we simply opt for an equivalent multilingual version, and XLM-RoBERTa appears to achieve better performance for natural language understanding tasks than multilingual BERT (Conneau and Lample, 2019; Conneau et al., 2019).

Hyper-parameters used for pre-training and fine-tuning are listed in Appendix E.

Pre-training We pre-train our model using the pre-training setup described in Section 3.3, on a dataset generated as described in Section 3.2. The resulting data is constructed using the pairs of translated NL and SQL templates, which we populate with data (column and table names, and other relevant values) from DuSQL (Wang et al., 2020b) databases, since their schemata are in Chinese. The mono-lingual variants (tested on Spider) are pre-trained using the official GraPPa pre-trained model provided by Yu et al. (2021). In all scenarios, fine-tuning is performed exclusively on the target dataset: CSpider and Spider for the cross-lingual and mono-lingual variants respectively.

Baselines We compare the performance of our system to that of the following architectures:

- **RAT-SQL** (Relation-Aware Transformer; Wang et al. 2020a) is a strong baseline for SQL semantic parsing. The relation-aware self-attention mechanism of RAT-SQL has been used in the architecture of several subsequent parsers, including our own.

[2] Similarly to Sherborne et al. (2020), we reverse the gradient of the LP network in the backward pass, to encourage our model to learn language invariant representations.

[3] `https://taolusi.github.io/CSpider-explorer/`

[4] `https://yale-lily.github.io/spider`

[5] Exact match accuracy scores are computed using the framework provided by: `https://github.com/taoyds/test-suite-sql-eval`.

[6] `https://github.com/microsoft/rat-sql`.

- **SmBoP** (Semi-autoregressive Bottom-up Semantic Parsing; Rubin and Berant 2021) is a semi-auto-regressive semantic parser, which constructs SQL bottom-up, parallelising the calculation of sub-trees in the same height.

4.2 Performance on Semantic Parsing

The performance of FastRAT and baseline systems on the cross-lingual setup, using the CSpider dataset, can be seen in Table 2. Our best model variant, FastRAT+SSP+LP outperforms all variants of the baseline systems. Although the vanilla version of FastRAT (without our pre-training step) is not the strongest method, the contribution of our pre-training scheme is verified, since FastRAT+SSP and FastRAT+SSP+LP outperform RAT-SQL and SmBoP variants with the same pre-training.

Conversely, the monolingual results in Table 3 show that RAT-SQL and SmBoP, both specifically designed as monolingual models, outperform FastRAT, although the performance of the latter is comparable with the baselines. The RAT-SQL encoder relies on matching mentions of tables and columns in the natural language question to the accompanying schema, a task which is simpler in the monolingual setup, where both question and schema are in the same language. Nonetheless, in the cross-lingual setup, the effect of schema linking becomes much more sparse. We believe that this explains the performance difference of RAT-SQL across CSpider and Spider, and how its performance relates to our scores. Interestingly, SSP pre-training[7] results in improvements for all three models.

4.2.1 Ablation

The last columns of Tables 2 and 3 show the contribution of each of the components of FastRAT. It can be seen that adding SSP pre-training increases the exact match scores by more than 5 points in both the monolingual and the cross-lingual setup. Moreover, in the cross-lingual setup, language prediction pre-training seems to benefit all models and, when used with FastRAT, results in a system which outperforms all baselines.

4.2.2 Evaluation on Known Databases

While the original data splits of Spider and CSpider seek to assess the models' capability to generalise to unseen databases, this is not the only possible scenario in industrial applications. A realistic deployment scenario can involve users or developers of a semantic parser, who are able to provide the system with a number of training examples for a particular database, which can be used to further refine the model to the schema of interest.

System	**Variant**		
	XLM	+SSP	+SSP+LP
RAT-SQL	47.8	55.1	56.4
SmBoP	**56.4**	57.4	57.7
FastRAT	54.3	**59.9**	**61.3**
RoBERTa$_{\text{Seq2SQL}}$	66.2		

Table 2: Exact match accuracy for the cross-lingual setup, on the development split of CSpider. "XLM" (i.e. XLM-RoBERTa-large) refers to the vanilla variant of each system, "+SSP" includes SSP pre-training, and "+SSP+LP" includes SSP and LP pre-training. The last row is the state-of-the-art system according to CSpider leaderboard (accessed 19 Sep. 2023).

System	**Variant**	
	BERT$_{\text{LARGE}}$	+SSP
RAT-SQL	**69.7**	**73.6**
SmBoP	63.4	72.1
FastRAT	63.2	69.1
CatSQL + GraPPa	78.6	

Table 3: Exact match accuracy for the monolingual setup, on the development split of Spider. "BERT" column refers to the vanilla variant of each system, while "+SSP" to adding SSP pre-training to each model.The last row is the state-of-the-art system according to the Spider leaderboard (accessed 19 Sep. 2023).

In order to test our approach in such a setup, in line with previous research on a compositional generalisation of semantic parsers (Shaw et al., 2021), we experiment on a number of different data splits of the Spider dataset: (i) based on source length (Len), (ii) based on Target Maximum Compound Divergence (TMCD), (iii) based on templates generated by anonymising integers and quoted strings (Template), and (iv) a random split (Random) . In all four splits, databases are shared between the train and development sets. Results can be seen in Table 4, which also includes the performance of two other models, (T5-Base, and NQG-T5-Base) for reference. It can be seen that FastRAT's perfor-

[7] We use the GraPPa model provided by Yu et al. (2021), which performs pre-training with both SSP and Masked Language Modelling (`https://github.com/taoyds/grappa`).

mance follows from the difficulty of the data splits (lowest scores for Len split, higher for Random, and better performance on TMCD than Template).

System	Rand.	Templ.	Len	TMCD
T5-Base	**82.0**	59.3	49.0	60.9
NQG-T5	81.8	59.2	49.0	60.8
FastRAT	80.4	**62.1**	**50.0**	**65.4**

Table 4: Exact query match accuracy reported on different splits of the Spider dataset. "T5-Base" is a T5 sequence-to-sequence semantic parser, and "NQG-T5" (base) a model that uses a flexible quasi-synchronous grammar; data splits and results for T5 and NQG-T5 obtained from previous work (Shaw et al., 2021).

4.2.3 Comparing against ChatGPT

We include a comparison of our system against Liu et al. (2023), that explores a way of leveraging ChatGPT for zero-shot semantic parsing. Table 5 summarises the results. Since larger pre-trained language models tend to under-perform in the exact match accuracy setting (Liu et al., 2023), we also include the relevant execution accuracy scores. We can see that the exact-match-accuracy performance of FastRAT is superior across all settings. Interestingly, FastRAT (which has only 673M parameters compared to multi-billion-parameter, GPT models), outperforms ChatGPT even with respect to execution accuracy.

System	Exact Match	Execution
CSpider		
ChatGPT	32.6	65.1
FastRAT	**61.3**	**67.7**
Spider		
ChatGPT	37.9	70.1
FastRAT	**69.1**	**73.2**

Table 5: Exact match and execution accuracy scores, on the development splits of CSpider (top) and Spider (bottom). The FastRAT version that has been employed for Spider has been subjected to SSP pre-training whereas the version for CSpider has been subjected to both SSP and LP pre-training. Exact match accuracy scores for ChatGPT were obtained using: `https://github.com/THU-BPM/chatgpt-sql` by Liu et al. (2023).

4.3 Runtime Performance

The strongest feature of FastRAT is its speed; while it may not outperform state-of-the-art models in all setups and datasets as shown in the previous section, its decoder-free architecture allows for very fast decoding times compared to baseline models.

Table 6 shows the runtimes of our system and baseline models, including end-to-end times and decoding times separately, on a CPU (2x AMD EPYC 7763 64-Core 1.8GHz) and a GPU (single A100-SXM-80GB). It can be seen that FastRAT decoding speed is an order of magnitude smaller than that of the baselines, with 2ms average decoding time on a GPU. Interestingly, decoding with FastRAT is 4 times faster than decoding with SmBoP,[8] which in turn has a lower decoding time than the original RAT-SQL implementation. End-to-end prediction results paint a similar picture: the average FastRAT prediction time is less than 1s on CPU, and 33ms on GPU, with the next fastest being RAT-SQL (1.4s and 257ms respectively).

The SQL queries in the datasets that we used consist of an average of 17.5 tokens (assuming word-level tokenisation), with a maximum length of 62 tokens. We believe that these numbers are on par with the length of the output of systems for other language generation tasks. It is important to note that industrial text-to-SQL applications commonly involve very long SQL queries, spanning several lines of code. The value of our efficient decoding paradigm becomes even more apparent in such cases, since with conventional, auto-regressive approaches, decoding time would increase linearly with respect to the expected query's length.

System	Decoding		End-to-End	
	CPU	GPU	CPU	GPU
RAT-SQL	0.412	0.120	1.441	0.257
SmBoP	0.558	0.037	7.338	0.065
FastRAT	**0.010**	**0.002**	**0.993**	**0.033**

Table 6: Average elapsed times (sec), CPU and GPU.

5 Discussion

FastRAT is proposed as a very fast Text-to-SQL semantic parser (substantially faster than its coun-

[8] In our tests on a CPU, SmBoP was slower than RAT-SQL. We hypothesise that this is due to the SmBoP implementation using some GPU-specific parallelisation options.

terparts); as such, it navigates a trade-off between fast decoding times and parsing performance. Our view is that FastRAT balances this tradeoff effectively, since it achieves 10 times decoding speedup with a small performance hit (3 EM accuracy points compared to SmBoP) in Spider, and increased performance against the baseline systems on CSpider.

5.1 Expressive Power of the Decoder

FastRAT uses a deterministic decoder mechanism on top of the SSP label predictions. This design signifies a shift from commonly used decoding algorithms, which decode on a token, token chunk, or sub-tree basis.

It is important to verify that the decoding capability of FastRAT is expressive enough to fully construct SQL queries. While our algorithm is not designed to fully cover the entirety of the SQL syntax, we can quantify its expressive power using publicly available benchmarks. To this end, we run an oracle experiment: we encode SQL queries from Spider (Yu et al., 2018), DuSQL (Wang et al., 2020b) and NL2SQL (Sun et al., 2020) to SSP and then translate the SSP labels back to SQL using the approach described in Section 3.1. Exact match accuracy scores on the generated SQL queries provide an upper-bound performance for our system. The scores on Spider are 95.2%, on DuSQL 89.04%, and on NL2SQL 96.31%. We conclude that our decoding algorithm is capable of successfully constructing most queries in the public parts of the datasets, and leaves enough room for improvement in the model development process.

The SQL queries that cannot be fully reconstructed belong to one of the following types: (i) they contain multiple `HAVING/WHERE` clauses with sub-queries, and our algorithm fails to correctly determine the antecedent of the sub-query, (ii) they contain sub-queries in the `SELECT` or `FROM` clauses of the SQL query which are not currently supported by our algorithm, or (iii) they include multiple elements in `ORDER BY` or `GROUP BY` clauses, which our algorithm predicts with a different order than the original.

5.2 Error Analysis

Conversely to other decoding mechanisms, FastRAT is trained on the SSP label classification task. In other words, the model is explicitly trained to identify correct SSP labels, not to generate correct SQL queries. As such, errors in SSP predictions result directly in errors during SQL generation.

Throughout our experiments, we observed low generalisation in a subset of the development set, which we attribute to the characteristics of the dataset. Specifically, there are 11 SSP labels in the development set that do not appear in the training split, and which account for 30 development examples. Naturally, it is challenging for the model to correctly output predictions for those labels.

Moreover, is worth noting that there is a number of patterns of SSP labels[9] in the development split, which are underrepresented or absent from the training data. In total, there are $1,084$ unique SSP patterns in the training set and 245 in the development set. The development set includes 73 patterns which do not appear in the training set and which account for 127 examples. Unsurprisingly, our model performs worse on those unseen SSP patterns (correct output in only 18% of the examples). This disparity in the data splits is an inherent limitation of the dataset, which affects many semantic parsers. We observed similar low performance (32%) on this subset of examples in LGESQL (Cao et al., 2021), a system which has been used in different variants by many state-of-the-art models. However, given FastRAT's heavy reliance on SSP semantics, it is disproportionately affected by this disparity.

6 Conclusion

In this paper, we propose FastRAT, an efficient cross-lingual Text-to-SQL semantic parser. FastRAT includes a deterministic decoding mechanism, which makes it 10 times faster than the fastest previously available Text-to-SQL semantic parser.

Furthermore, we propose a method to efficiently construct pre-training datasets in new languages given a dataset in a source language. Finally, we introduce a pre-training setup, designed for the cross-lingual Text-to-SQL parsing setup; a setup in which other systems do not perform favourably. Our approach can be used to port Text-to-SQL semantic parsers to new languages quickly, and our experiments show that FastRAT outperforms strong baselines in the cross-lingual setup, while being significantly faster than them.

As for future work, we plan to further improve the expressive power of the decoder. Furthermore, we will look into coupling FastRAT with other decoding setups based on larger language models.

[9] A pattern of SSP labels is the set of SSP labels of a specific example, after discarding all the empty ones.

Limitations

The design of the deterministic FastRAT decoder does not fully cover the entirety of SQL syntax. Consequently, a subset of SQL queries cannot be decoded using our approach. While Section 5 provides a detailed analysis of the expressive power of our decoder, it is important to note FastRAT is able to effectively and efficiently decode most SQL queries of general-purpose datasets (92% of Spider), and that challenging queries (see also Section 5 for a qualitative analysis) are usually part of the hard or extra hard difficulty splits of the datasets, which are rarely observed in everyday deployment scenarios.

Additionally, the analysis presented in this paper includes a small number of baselines: in most experiments, we compare FastRAT with RAT-SQL and SmBoP. This choice of baselines experimentally highlights how FastRAT navigates the trade-off between fast decoding times and parsing performance. Specifically, RAT-SQL is a very strong text-to-SQL baseline (all Spider leader-board submissions compare against the original RAT-SQL baseline or its variants; also, a large number of submissions use RAT-SQL or parts thereof in their architecture). On the other hand, SmBoP is the fastest parser to date. FastRAT balances this trade-off effectively, since it achieves 10x decoding speedup over SmBoP with a small performance loss.

Ethics Statement

Research described in this paper is done in accordance to the ACL code of ethics. Our work does not make use of private, proprietary, or sensitive data. The proposed models are trained on publicly available community datasets, building on top of publicly available pre-trained models. Due to the nature of the pre-training/fine-tuning scheme, it is conceivable that our system might perpetuate possible biases included in the datasets and/or the pre-trained models.

References

Nader Akoury, Kalpesh Krishna, and Mohit Iyyer. 2019. Syntactically supervised transformers for faster neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1269–1281, Florence, Italy. Association for Computational Linguistics.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2541–2555, Online. Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 7059–7069. Curran Associates, Inc.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Longxu Dou, Yan Gao, Mingyang Pan, Dingzirui Wang, Wanxiang Che, Dechen Zhan, and Jian-Guang Lou. 2023. Multispider: Towards benchmarking multilingual text-to-sql semantic parsing. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Tomáš Kočiský, Gábor Melis, Edward Grefenstette, Chris Dyer, Wang Ling, Phil Blunsom, and Karl Moritz Hermann. 2016. Semantic parsing with semi-supervised sequential autoencoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1078–1087, Austin, Texas. Association for Computational Linguistics.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability.

Denis Lukovnikov and Asja Fischer. 2021. Insertion-based tree decoding. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3201–3213, Online. Association for Computational Linguistics.

Qingkai Min, Yuefeng Shi, and Yue Zhang. 2019. A pilot study for Chinese SQL semantic parsing. In *Proceedings of the 2019 Conference on Empirical*

Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3652–3658, Hong Kong, China. Association for Computational Linguistics.

Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 311–324, Online. Association for Computational Linguistics.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 922–938, Online. Association for Computational Linguistics.

Tom Sherborne and Mirella Lapata. 2022. Zero-shot cross-lingual semantic parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4134–4153, Dublin, Ireland. Association for Computational Linguistics.

Tom Sherborne, Yumo Xu, and Mirella Lapata. 2020. Bootstrapping a crosslingual semantic parser. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 499–517, Online. Association for Computational Linguistics.

Peng Shi, Linfeng Song, Lifeng Jin, Haitao Mi, He Bai, Jimmy Lin, and Dong Yu. 2022. Cross-lingual text-to-SQL semantic parsing with representation mixup. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5296–5306, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Ningyuan Sun, Xuefeng Yang, and Yunfeng Liu. 2020. TableQA: a large-scale chinese text-to-sql dataset for table-aware sql generation. *ArXiv*, abs/2006.06434.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020a. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Lijie Wang, Ao Zhang, Kun Wu, Ke Sun, Zhenghua Li, Hua Wu, Min Zhang, and Haifeng Wang. 2020b. DuSQL: A large-scale and pragmatic Chinese text-to-SQL dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6923–6935, Online. Association for Computational Linguistics.

Menglin Xia and Emilio Monti. 2021. Multilingual neural semantic parsing for low-resourced languages. In *Proceedings of *SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, pages 185–194, Online. Association for Computational Linguistics.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2021. GraPPa: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*.

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

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1050–1055. AAAI Press.

Yusen Zhang, Jun Wang, Zhiguo Wang, and Rui Zhang. 2023. XSemPLR: Cross-lingual semantic parsing in multiple natural languages and meaning representations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15918–15947, Toronto, Canada. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Qile Zhu, Haidar Khan, Saleh Soltan, Stephen Rawls, and Wael Hamza. 2020. Don't parse, insert: Multilingual semantic parsing with insertion based decoding. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 496–506, Online. Association for Computational Linguistics.

A From SQL to SSP Example

An idealised example of the task of assigning SSP labels to the columns of an input schema and of their relevance to the expected SQL query is presented in Table 7.

B Table Joining Algorithm

A detailed description of the steps that are involved for the table joining algorithm of Section 3.1.3 is presented below:

- **Step 1**

question	how many car models come from Germany	

SQL	SELECT COUNT(*) FROM car_models
	WHERE country = “Germany”	

		SSP Label
Schema	__all__.*	SELECT count
	$t_1 =$ cars	
	$t_1.c^1_1 =$ cars.model	
	$t_1.c^1_2 =$ cars.brand	
	$t_1.c^1_3 =$ cars.country	WHERE =

Table 7: An idealised example of the alignment of SSP labels (in underlined font) with a particular SQL query given a single-table schema consisting of three columns

- Go through all the columns that appear in the query, and add their respective tables to the list of tables to join L_t.
- Go through all the tables which have a FROM in their labels, and add them to L_t.

• **Step 2**: Starting with a random table t_i, attempt to create an initial FROM statement by first adding direct joining. Assuming that $t_a \in L_s$, and that we are considering table t_b, we evaluate whether t_b can be directly joined to t_a (noted $t_a \rightleftharpoons t_b$) by checking if any of the following hold:

- **Case 1**: t_a has a column $c^a_{a'}$ which references or is referenced by $c^b_{b'}$ in t_b (noted $c^a_{a'} \rightarrow c^b_{b'}$). We add t_b to the FROM statement and join on $t_a.c^a_{a'} = t_b.c^b_{b'}$.
- **Case 2**: t_a has a column $c^a_{a'}$ which references $c^b_{b'}$ in t_b via a *foreign key chain* (noted $c^a_{a'} \rightarrow^* c^b_{b'}$). A foreign key chain $(c^{t_i}_{l_i})_{i\in[1,k]}$ is a sequence such that: $\forall i \in [1, k-1]$, t_{t_i} is a table and $c^{t_i}_{l_i}$ is a column of table t_{t_i} st. $c^{t_i}_{l_i} \rightarrow c^{t_{i+1}}_{l_{i+1}}$. We define that $c^a_{a'} \rightarrow^* c^b_{b'}$ iif there exist a foreign key chain $(c^{t_i}_{l_i})_{i\in[1,k]}$ st. $c^a_{a'} = c^{t_1}_{l_1}$ and $c^b_{b'} = c^{t_k}_{l_k}$. In this case, we add t_b to the FROM statement and join on $t_a.c^a_{a'} = t_b.c^b_{b'}$. Note: We do not add all the intermediate tables t_i.
- **Case 3**: t_a has a column $c^a_{a'}$ which references $c^b_{b'}$ in t_b via a *common reference chain*. $c^a_{a'}$ references $c^b_{b'}$ via a common reference chain if there exists a column $c^d_{d'}$ in a table t_d st. $c^a_{a'} \rightarrow^* c^d_{d'}$ and $c^b_{b'} \rightarrow^* c^d_{d'}$. In this case, add t_b to the FROM statement and join on $t_a.c^a_{a'} = t_b.c^b_{b'}$. Note that t_d does not need to be in the set of tables to join.

• **Step 3**: Check the remaining tables to join for tables t_b that *references without chaining* a table t_a that has already been added to the FROM statement (noted $t_a \iff t_b$). This means that there exists a sequence of tables $(t_j)_{j\in[1,T]}$ such that $t_1 = t_a$ and $t_T = t_b$ and $\forall j \in [1, T-1]$, $t_j \rightleftharpoons t_{j+1}$. In this case, add each intermediate table t_j to the FROM statement and join on the columns allowed by these direct joins. This last step corresponds to the case where tables can be joined together but only on foreign keys referencing different columns in a third table.

C English-Chinese Context-free Grammar Example

Table 8 shows an example of a synchronous context-free grammar for Chinese and English.

D Pre-training Framework

A schematic representation of our pre-training framework, including an example, can be seen in Figure 2.

E Hyper-parameters

The hidden size for RAT (i.e. Relation-Aware Transformer) and the SSP decoder is 1024. Optimisation is performed using Adam (Kingma and Ba, 2014). We trained for 100 epochs with 5 epochs of linear warmups. After the warmup steps, we use a linear decay down to zero for both learning rates. The following tables include detailed hyper-parameters that have been used for pre-training and fine-tuning for both the monolingual and cross-lingual setup of FastRAT.

Non-terminals (EN)	**Production rules (EN)**
TABLE → t_i COLUMN → c_i VALUE → v_i AGG → ⟨ MAX, MIN, COUNT, AVG, SUM ⟩ OP → ⟨ =, $\leq$, . . ., LIKE, BETWEEN ⟩ SC → ⟨ ASC, DESC ⟩ MAX → ⟨ “maximum number of”, “the largest” . . .⟩ $\leq$→ ⟨ “no more than”, “at most” . . .⟩ . . .	1. ROOT → ⟨ “List all the {COLUMN0} which {OP0} {VALUE0}.”, `SELECT {COLUMN0} {FROM} WHERE {COLUMN0} {OP0} {VALUE0}` ⟩ 2. ROOT → ⟨ “What are the {COLUMN0} in which the {COLUMN1} was between {VALUE0} and {VALUE1}?”, `SELECT {COLUMN0} {FROM} WHERE {COLUMN1} BETWEEN {VALUE0} AND {VALUE1}` ⟩ . . .
Non-terminals (ZH)	**Production rules (ZH)**
TABLE → t_i COLUMN → c_i VALUE → v_i AGG → ⟨ MAX, MIN, COUNT, AVG, SUM ⟩ OP → ⟨ =, $\leq$, . . ., LIKE, BETWEEN ⟩ SC → ⟨ ASC, DESC ⟩ MAX → ⟨ “最多的”, “最大的” . . .⟩ $\leq$→ ⟨ “不超过”, “至多” . . .⟩ . . .	1. ROOT → ⟨ “列出{OP0}{VALUE0}的所有{COLUMN0}.”, `SELECT {COLUMN0} {FROM} WHERE {COLUMN0} {OP0} {VALUE0}` ⟩ 2. ROOT → ⟨ “{COLUMN1} 在{VALUE0}和{VALUE1}之间的{COLUMN0}是什么？ ”, `SELECT {COLUMN0} {FROM} WHERE {COLUMN1} BETWEEN {VALUE0} AND {VALUE1}` ⟩ . . .

Table 8: An excerpt of our Synchronous Context-Free Grammar for Chinese (ZH), translated from the English (EN) version released by Yu et al. (2021). t_i, c_i, v_i refer to table names, column names, and entry values respectively.

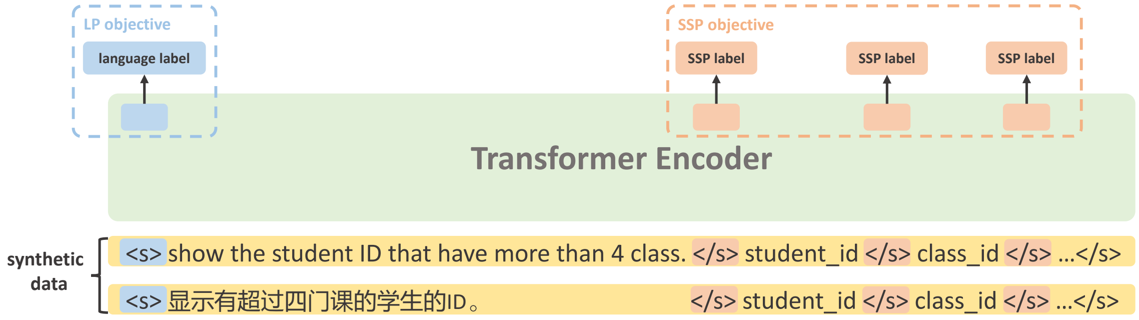
LP objective
language label
SSP objective
SSP label
SSP label
SSP label
Transformer Encoder
synthetic data
<s> show the student ID that have more than 4 class. </s> student_id </s> class_id </s> ...</s>
<s> 显示有超过四门课的学生的ID。 </s> student_id </s> class_id </s> ...</s>

Figure 2: Model Pre-training Framework

Model	num_params	learning_rate	batch_size	num_epochs
xlm-roberta-large	550M	–	–	–
+ cross-lingual SSP	550M	$1e-5$	32	5

Table 9: Pre-training Hyper-parameters

Model	num_params	learning_rate	batch_size	stop_criterion
FastRAT (with `bert-large` or with `https://github.com/taoyds/grappa`)	458M	`lr:` $1e-4$; `lm_lr:` $3e-6$ with polynomial decay (5% warm-up steps)	40	100 epochs
FastRAT (with `xlm-roberta-large` or with `xlm-roberta-large` + cross-lingual SSP)	673M	`lr:` $1e-4$; `lm_lr:` $3e-6$ with polynomial decay (5% warm-up steps)	40	100 epochs

Table 10: Fine-tuning Hyper-parameters