Graphix-T5: Mixing Pre-Trained Transformers with Graph-Aware Layers for Text-to-SQL Parsing + initial proposal

박현우

202305

Table of Contents

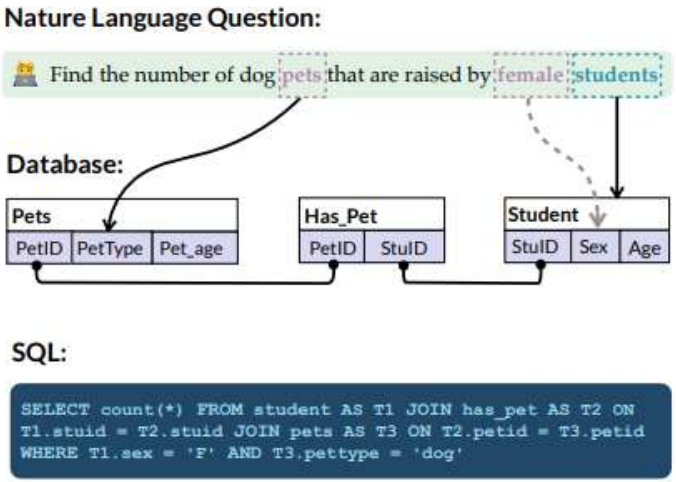
1.1. Introduction

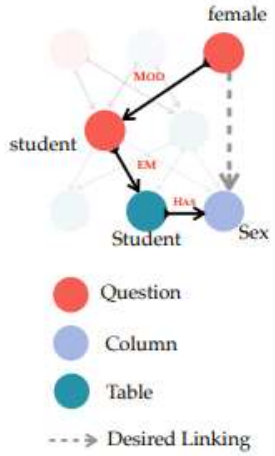
One of the major challenges in Text-to-SQL parsing is domain generalization;
i.e. how to generalize well to unseen databases.

Recently, the pretrained T5 has achieved state-of-the-art performance, though not specialized for text-to-SQL parsing, on T2S domain generalization benchmarks.
e.g. RAT-SQL

= Hoping PLM also generalizes well on other domains
e.g. unseen databases on Text-to-SQL domain

1.1.1. Illustration of cross-domain text-to-SQL challenge





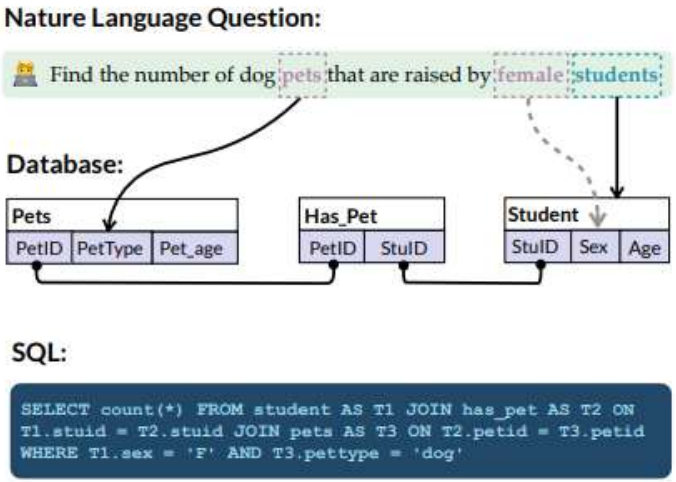
The link is highly desired,
i.e. between

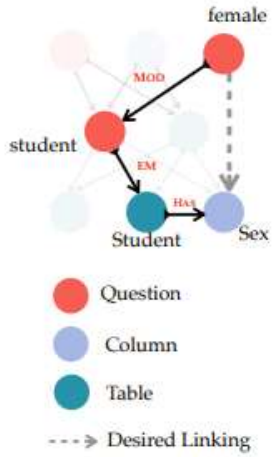
1. the target column sex
2. and the token female

but extremely challenging for the model to capture;

especially when domain-specific data or effective rules is absent.

1.1.1. Illustration of cross-domain text-to-SQL challenge





However, this dilemma can be mitigated

by a multi-hop reasoning path:

(female MOD → student EM → Student HAS → Sex).

= Enable universal generalization of SQL for arbitrary databases.

1.2. Multi-hop Reasoning on T2S task

Specifically, Multi-hop reasoning is the ability to properly contextualize a user question against a given database

by simultaneously considering many

1. explicit relations or Structural representation
   (e.g., table/column relations specified by database schema)
2. implicit relations or Semantic representation
   (e.g., whether a phrase refers to a column or table).

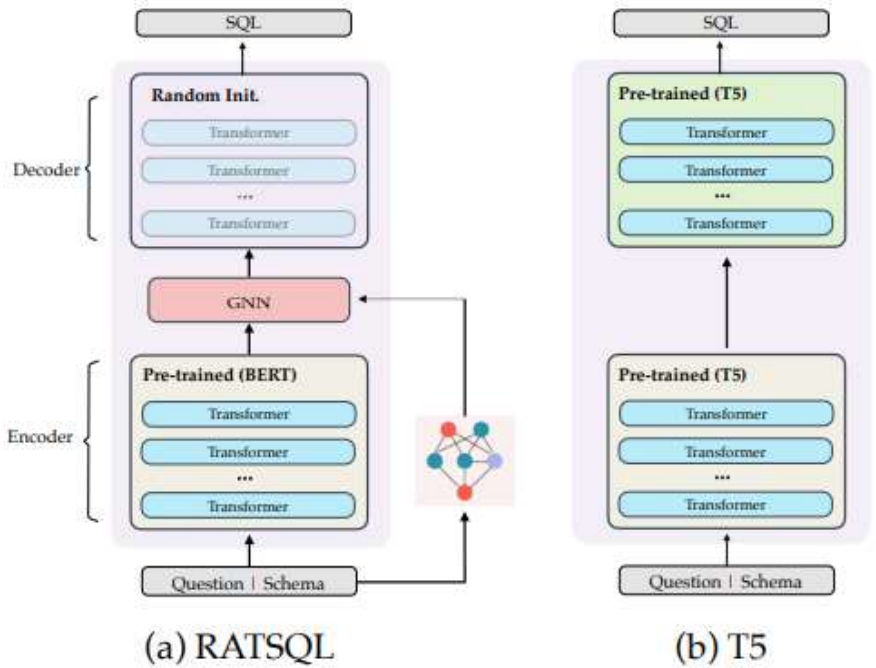
1.3. Previous works

From the modeling perspective, there are two critical dimensions in T2S task:

1. How to effectively imbue relational structures (both explicit and implicit) in the form of graphs into neural networks
   = PICARD (@ later)
2. How to take the most advantage of pre-trained models (e.g. T5)
   = RAT-SQL (@ later)

NOTE : PICARD is non-invasive, thus can be plugin-ed into other models.

1.3.1 RAT-SQL



1. Only utilize pre-trained encoders (e.g., BERT)
2. and explicitly capture desired relations via specialized relation-aware models.
   (i.e. GNN-based module)
3. Use randomly initialized decoder

1.3.2. Deficiencies of previous works and breakthrough

More powerful encoder-decoder based pre-trained models are not exploited;
i.e. randomly initialized decoder

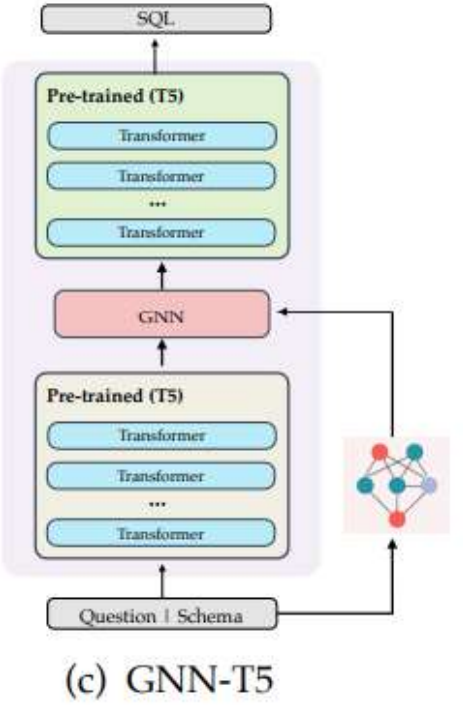
Consequently, relational structures are accommodated at most;
i.e. via GNN-module

Therefore, in this work,

1. the full encoder-decoder based pre-trained models (specifically T5)
2. and relation-aware encodings

are deeply coupled in favor of better domain generalization.

1.4. Warning on combined approach



(c) GNN-T5

GNN-T5 showed that simply adding a relational graph-based module in the middle of full T5 does not work very well on standard benchmarks.

Presumably, the deficiency comes from the middle graph-based modules breaking the original information flow inside T5.

2. T2S task definition

A natural language question $\mathcal{Q} = \{q_1, ..., q_{|\mathcal{Q}|}\}$

with its corresponding database schemas $\mathcal{D} = \{\mathcal{C}, \mathcal{T}\}$,

where $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$ $\mathcal{T} = \{t_1, ..., t_{|\mathcal{T}|}\}$ represent columns and tables,

($|\mathcal{C}|$ and $|\mathcal{T}|$ refer to the number of columns and tables in each database respectively.)

Goal of T2S : Given a natural language question Q, to generate the corresponding SQL query y.

2.1. T5 input for T2S

The most canonical and effective format of inputs to T5 for T2S task is by PeteShaw (T5, Shaw et al. 2021) :

Unify

1. natural language questions Q
2. and database schema D

as a joint sequence as shown:

$$x = [q_1, ..., q_{|Q|} \mid \mathcal{D}_{name} \mid t_1 : c_1^{t_1}, ..., c_{|C|}^{t_1} \mid ... \mid t_{|T|} : c_1^{t_{|T|}}, ..., c_{|C|}^{t_{|T|}} \mid *],$$

2.1. T5 input for T2S

$$x = [q_1, ..., q_{|Q|} \mid \mathcal{D}_{name} \mid t_1 : c_1^{t_1}, ..., c_{|C|}^{t_1} \mid ... \mid t_{|T|} : c_1^{t_{|T|}}, ..., c_{|C|}^{t_{|T|}} \mid *],$$

q_i : i-th token in the question

t_j : represents j-th table in the D

$c_k^{t_j}$: refers to the k-th column in the j-th table.

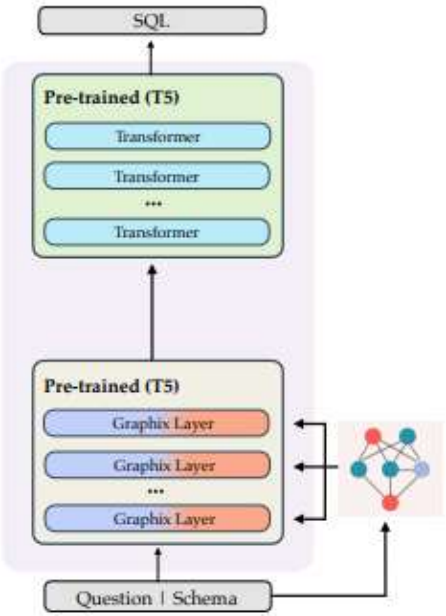
$|*],$: the special column token in the database (for relation-aware encodings)

$\mathcal{D}_{name}$: the name of each database.

3. GRAPHIX-T5

1. Basic idea
2. GRAPHIX input
3. GRAPHIX layer
    a. Semantic Representation
    b. Structural Representation
4. GRAPHIX-T5 model

3.1. Basic idea



(d) Graphix-T5

GRAPHIX constructs a new encoder by replacing the original T5 encoder with stacks of GRAPHIX layers.

In each GRAPHIX layer, the parameters of the semantic block are still initialized by T5, maintaining the contextualized encoding power of the pretraining.

c.f. randomly initialized encoder of GNN-T5

3.1. Basic idea

GRAPHIX layer simultaneously encodes a mixture of semantic and structural information.

1. Hidden states of inputs (composed by questions and databases) are modelled by contextualized semantic encoding
2. Structural representation is injected in each transformer layer using a relational GNN block.
   This GNN block enhances multi-hop reasoning through message passing, aiding capture explicit and implicit relations
   i.e. introduce structural inductive bias.

3.2. Input for GRAPHIX

Input for GRAPHIX is preprocessed through following procedures:

1. Contextual encoding
2. Graph construction
3. Bridge mode node matching

3.2.1. Contextual encoding

Same as T5 for T2S : create a joint sequence by unifying

1. natural language questions Q
2. and database schema D

$$x = [q_1, ..., q_{|Q|} \mid \mathcal{D}_{name} \mid t_1 : c_1^{t_1}, ..., c_{|C|}^{t_1} \mid ... \mid t_{|T|} : c_1^{t_{|T|}}, ..., c_{|C|}^{t_{|T|}} \mid *],$$

3.2.2. Joint input as Graph

The joint input questions and schemas can be displayed as a heterogeneous graph $\mathcal{G} = \{\mathcal{V}, \mathcal{R}\}$

which consists of

1. three types of nodes $\mathcal{V} = \mathcal{Q} \cup \mathcal{C} \cup \mathcal{T}$
2. and multiple types of relations $\mathcal{R} = r_1, ..., r_{|\mathcal{R}|}$

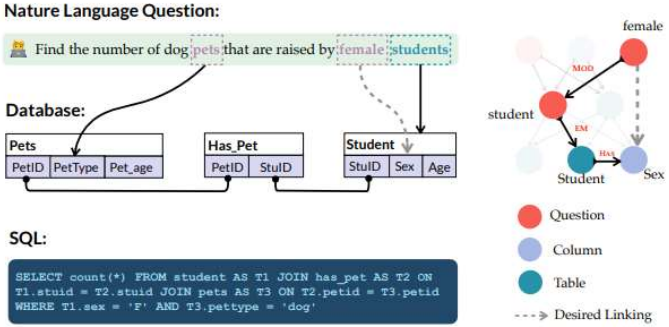
3.2.2. Joint input as Graph

Specifically, for multiple types of relations $\mathcal{R} = r_1, ..., r_{|\mathcal{R}|}$,

1. each r_i refers to a one-hop relation between nodes
2. and a multi-hop relation r^k is defined as a composition of one-hop relations:

$$r^k = r_1 \circ r_2 \cdots \circ r_I$$

as shown in the Figure 1 (where I refers to the length of each r^k (= |r^k|)



3.2.2.1. Categories of predefined relations

Inspired by previous works (RAT-SQL, S^2 SQL), we enumerated a list of pre-defined relations to connect nodes.

The relation sets can be divided into three main categories:

1. Schema relations
2. Schema linking relations
3. Question relations

3.2.2.1. Categories of predefined relations

Question relations

1. MODIFIER
2. and ARGUMENT

Question relations are implicit dependency relations between tokens in a question.

(@ Need to check the code for details.)

3.2.2.1. Categories of predefined relations

Schema relations

1. FOREIGN-KEY,
2. PRIMARY-KEY,
3. SAME-TABLE

Schema relations pertain to the particular explicit schema relations that the original T5 cannot obtain from linear inputs.

3.2.2.1. Categories of predefined relations

Schema linking relations

1. EXACT-MATCH,
2. PARTIALMATCH,
3. and VALUE-MATCH

Schema linking relations are implicit linking relations between question and schema nodes.

A new type of relation BRIDGE is introduced. (@ yet reading)

4. GRAPHIX layer

GRAPHIX layer plays a role as an Encoder part of the Encoder-Decoder architecture, replacing the original encoder of the pretrained T5.

The GRAPHIX layer is designed to integrate these informations:

1. Semantic information
   obtained from each transformer block
2. Structural information
   obtained from relational graph neural network (GNN) block.

4. GRAPHIX layer

Encoder part of the GRAPHIX-T5 can be formalized as $h = \mathbf{Enc}_{\Theta,\Psi}(x, \mathcal{G})$,

theta : Parameters of Transformer blocks for semantic information

psi : Parameters of GNN blocks for structural information

The core idea of GRAPHIX is to migrate parameters theta from original T5 encoder as the initial parameters of semantic transformer block of the GRAPHIX layer,

in order to preserve the pre-trained semantic knowledge of the original T5.

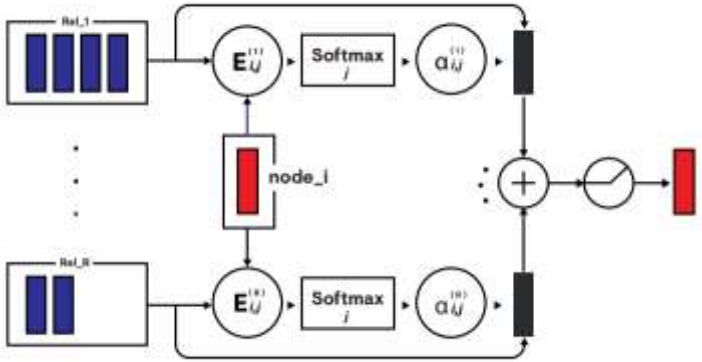
4.1. GNN blocks

In each GRAPHIX Layer, structural representations are produced through the relational graph attention network, RGAT (Wang et al. 2020b) over the pre-defined question-schema heterogeneous graph. $\mathcal{G} = \{\mathcal{V}, \mathcal{R}\}$

For RGAT (Relational Graph Attention Transformer), various node embeddings initialization strategies could be implemented;

GRAPHIX initialized with their semantic representations.

4.1. GNN blocks



Formally, given initial node embedding e^{init}_i, for

1. i-th node
2. and its j-th neighbor e^{init}_j,

which are linked by specific types of relations, (pre-defined relations categories)

it can be computed through:

$$\tilde{\alpha}_{ij} = \frac{e_i^{init}\widetilde{\mathbf{W}}_Q \left(e_j^{init}\widetilde{\mathbf{W}}_K + \phi\left(r_{ij}\right)\right)^\top}{\sqrt{d_z}}, \quad (10)$$

$$\alpha_{ij} = \mathbf{softmax}_j\left(\tilde{\alpha}_{ij}\right), \quad (11)$$

$$\hat{e}_i^{init} = \sum_{j \in \widetilde{\mathcal{N}}_i} \alpha_{ij}\left(e_j^{init}\widetilde{\mathbf{W}}_V + \phi(r_{ij})\right), \quad (12)$$

$$\hat{e}_i^{(l)} = \mathbf{LayerNorm}(e_i^{init} + \hat{e}_i^{init}\widetilde{\mathbf{W}}_O), \quad (13)$$

$$\tilde{e}_i^{(l)} = \mathbf{LayerNorm}(\hat{e}_i^{(l)} + \mathbf{FFN}(\hat{e}_i^{(l)})), \quad (14)$$

(@ more in RGAT)

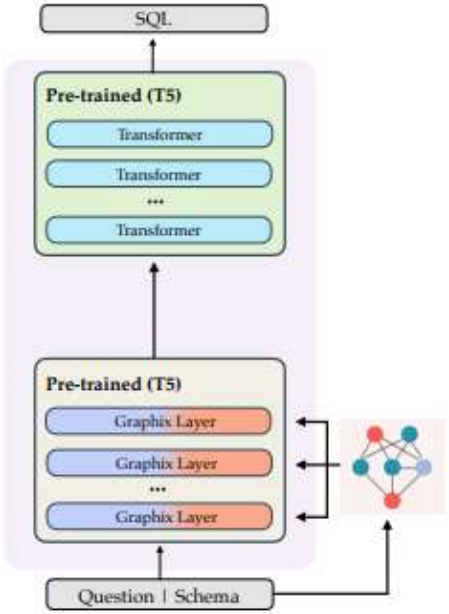
4.2. Joint Representation

After computing representations from both semantic and structural space,

the l-th GRAPHIX Layer employs a mixture of semantic and structural information to enable information integration as following:

$$\tilde{\mathcal{H}}_{\mathcal{M}}^{(l)} = \tilde{\mathcal{H}}_{\mathcal{S}}^{(l)} + \tilde{\mathcal{E}}_{\mathcal{G}}^{(l)},$$

Finally, this joint representation is inputed into decoder part of the pretrained T5.



(d) Graphix-T5

4.3. Ablation

Although GRAPHIX layer can possibly be in both encoder and decoder, adding GRAPHIX layers to the decoder does not lead to any improvements;

Decoder is an autoregressive model, which only considers the history tokens when generating the current token.

However, GRAPHIXT5 may disrupt this characteristic, as it can forecast the information of future tokens by global linking (i.e. message passing)

Therefore, currently the best tactic is to only incorporate GRAPHIX layers into the encoder.

4.4. Training

Similar to original T5, we also follow a common fine-tuning strategy:

The whole training framework is to optimize the following log-likelihood.

$$\max_{\Theta,\Upsilon,\Psi} \log p_{\Theta,\Upsilon,\Psi}(y \mid x) = \sum_{i=1}^{|y|} \log p_{\Theta,\Upsilon,\Psi}\left(y_i \mid y_{1:i-1}, x, \mathcal{G}\right). \tag{17}$$

4.4. Training

We evaluate our effectiveness of GRAPHIXT5 across two main versions:

1. T5-Large with approximately 800M parameters
2. and T5-3B, with more than 3 Billion parameters literally.

All experiments are conducted on one NVIDIA Tesla A100.

5. Evaluation metrics

In T2S task, these are the two standard metrics:

1. Exact Match (EM)
2. Execution Accuracy (EX)

EM can evaluate how much a generated SQL is comparable to the gold SQL.

EX can reflect whether a predicted SQL

1. is (syntactically) valid
2. and returns the exact result as desired by users.

5.1. Result

Model	Em	Ex
RAT-SQL + BERT ♡	69.7	-
RAT-SQL + Grappa ♡	73.9	-
GAZP + BERT	59.1	59.2
BRIDGE v2 + BERT	70.0	68.3
NatSQL + GAP	73.7	75.0
SMBOP + GRAPPA	74.7	75.0
LGESQL + ELECTRA ♡	75.1	-
S^2SQL + ELECTRA ♡	76.4	-
T5-large	67.0	69.3
GRAPHIX-T5-large	72.7 (↑ 5.7)	75.9 (↑ 6.6)
T5-large + PICARD ♣	69.1	72.9
GRAPHIX-T5-large + PICARD ♣	76.6 (↑ 7.5)	80.5 (↑ 7.6)
T5-3B	71.5	74.4
GRAPHIX-T5-3B	75.6 (↑ 4.1)	78.2 (↑ 3.8)
T5-3B + PICARD ♣	75.5	79.3
GRAPHIX-T5-3B + PICARD ♣	**77.1** (↑ 1.6)	**81.0** (↑ 1.7)

Exact match (EM) and execution (EX) accuracy (%) on SPIDER development set.

1. ♥
   means the model does not predict SQL values. (@ maybe verbal/ pseudo code output)
2. ♣
   means the model uses the constrained decoding PICARD.
3. ↑ is an absolute improvement.

5.1. Result

Model	Em	Ex
RAT-SQL + BERT ♡	69.7	-
RAT-SQL + Grappa ♡	73.9	-
GAZP + BERT	59.1	59.2
BRIDGE v2 + BERT	70.0	68.3
NatSQL + GAP	73.7	75.0
SMBOP + GRAPPA	74.7	75.0
LGESQL + ELECTRA ♡	75.1	-
S^2SQL + ELECTRA ♡	76.4	-
T5-large	67.0	69.3
GRAPHIX-T5-large	72.7 (↑ 5.7)	75.9 (↑ 6.6)
T5-large + PICARD ♣	69.1	72.9
GRAPHIX-T5-large + PICARD ♣	76.6 (↑ 7.5)	80.5 (↑ 7.6)
T5-3B	71.5	74.4
GRAPHIX-T5-3B	75.6 (↑ 4.1)	78.2 (↑ 3.8)
T5-3B + PICARD ♣	75.5	79.3
GRAPHIX-T5-3B + PICARD ♣	**77.1** (↑ 1.6)	**81.0** (↑ 1.7)

GRAPHIX-T5-3B with a constrained decoding module PICARD achieves the state-of-the-art.

This indicates that

the structural generalization capability of the GRAPHIX layer is crucial for T5, such a text-to-text PLM,

to perform the text-to-SQL task.

5.2. Zero-shot Results

MODEL	SYN	DK	REALISTIC
GNN	23.6	26.0	-
IRNet	28.4	33.1	-
RAT-SQL	33.6	35.8	-
RAT-SQL + BERT	48.2	40.9	58.1
RAT-SQL + Grappa	49.1	38.5	59.3
LGESQL + ELECTRA	64.6	48.4	69.2
T5-large	53.6	40.0	58.5
GRAPHIX-T5-large	61.1 (↑ 7.5)	48.6 (↑ 8.6)	67.3 (↑ 8.8)
T5-3B	58.0	46.9	62.0
GRAPHIX-T5-3B	**66.9** (↑ 8.9)	**51.2** (↑ 4.3)	**72.4** (↑ 10.4)

Table 2: Exact match (EM) accuracy (%) on SYN, DK and REALISTIC benchmark.

GRAPHIX-T5 showed much superior robustness

when it confronts with more challenging and closer to realistic evaluations

in SYN, DK, REALISTIC benchmarks.

5.3. Results on Low-resource Settings

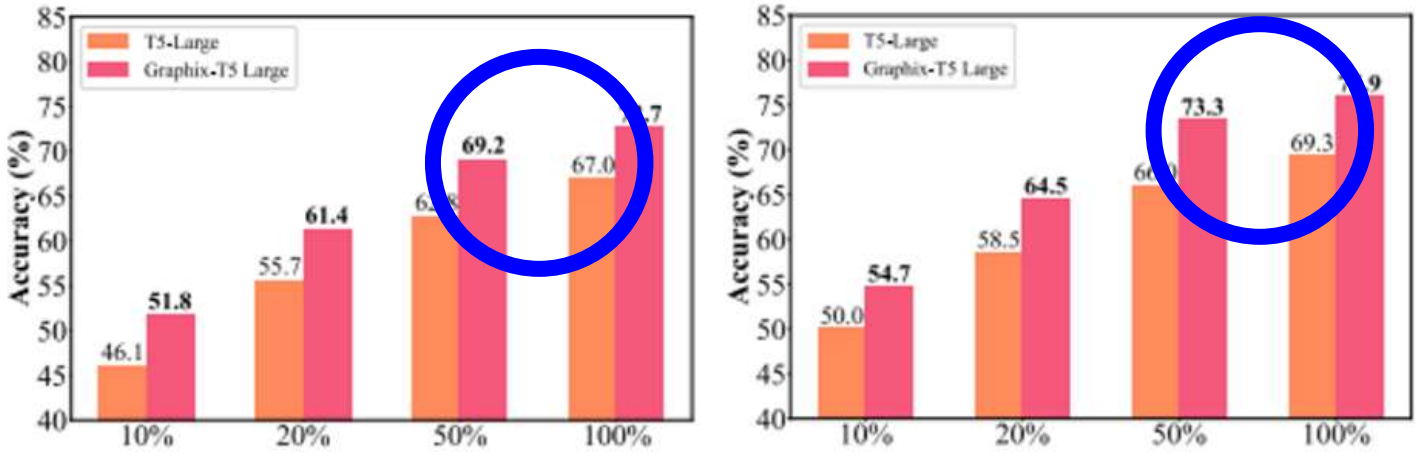


Figure 4: Exact match (EM) (left) and execution (EX) (right) accuracy (%) on SPIDER low-resource setting.

5.4. Further

Though we only focus on text-to-SQL parsing in this work,

we believe that the general methodology of GRAPHIX-T5 can be extended to structured knowledge grounding tasks,

e.g.,

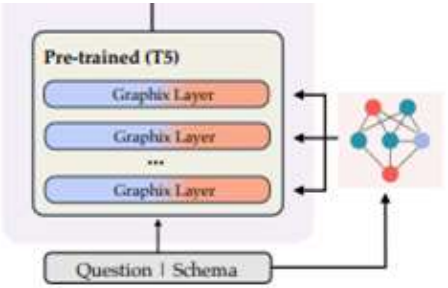
1. TableQA (Pasupat and Liang 2015),
2. Data-to-text (Nan et al. 2021)
3. and KBQA (Talmor and Berant 2018).

6.1. Proposing method 1

Instead of training (fine tuning) the entire model at once,

training on two steps:

1. Training the complete model (just as previous) using Question + SQL
2. Training the GNN module alone using schema only



(d) Graphix-T5

$$h = \mathbf{Enc}_{\Theta,\Psi}\left(x, \mathcal{G}\right), \qquad \tilde{\mathcal{H}}_{\mathcal{M}}^{(l)} = \tilde{\mathcal{H}}_{\mathcal{S}}^{(l)} + \tilde{\mathcal{E}}_{\mathcal{G}}^{(l)},$$

6.1. Proposing method 1

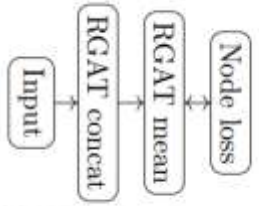
Because amount of training data for SQL is limited,
i.e. collected from real world application or human-generated

the model may not train enough structural information of the database to fine tune on.

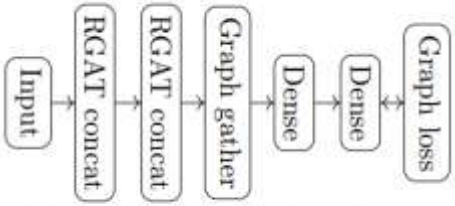
Thus, generate data from database schema as meaning of augmentation and further train the GNN module.

Hopefully, this can reinforce the structural recognition of the model.

6.1. Proposing method 1



(a) Node classification.



(b) Multi-task graph classification.

Specifically:

the GNN module (RGAT) is trained on Node Classification task with question(predefined relation categories)-schema heterogeneous graph. $\mathcal{G} = \{\mathcal{V}, \mathcal{R}\}$

Here, schema info can simply be extracted from given schema from users.

Thus, set relation info first, and make schema info based on that relation info.

Example: For Schema relations of SAME-TABLE, constraint to only extract schema within the same table.

Or use following data synthesis method.

6.2. Data synthesis

```
def generate_instruction_following_data(
    output_dir="./",
    seed_tasks_path="./seed_tasks.jsonl",
    num_instructions_to_generate=100,
    model_name="text-davinci-003",
    num_prompt_instructions=3,
    request_batch_size=5,
    temperature=1.0,
    top_p=1.0,
    num_cpus=16,
):
```

LLMs are utilized to generate dataset:

1. Template instruction is given to LLM
2. and LLM generates dataset based on given instruction.

```
You are asked to come up with a set of 20 diverse task instructions. These task instructions will be given to a GPT model and we will evaluate the

Here are the requirements:
1. Try not to repeat the verb for each instruction to maximize diversity.
2. The language used for the instruction also should be diverse. For example, you should combine questions with imperative instrucitons.
3. The type of instructions should be diverse. The list should include diverse types of tasks like open-ended generation, classification, editing,
4. A GPT language model should be able to complete the instruction. For example, do not ask the assistant to create any visual or audio output. Fc
5. The instructions should be in English.
6. The instructions should be 1 to 2 sentences long. Either an imperative sentence or a question is permitted.
7. You should generate an appropriate input to the instruction. The input field should contain a specific example provided for the instruction. It
8. Not all instructions require input. For example, when a instruction asks about some general information, "what is the highest peak in the worlc
9. The output should be an appropriate response to the instruction and the input. Make sure the output is less than 100 words.

List of 20 tasks:
###
```

6.2. Data synthesis

```
List of 20 tasks:
###
1. Instruction: Recommend a movie for me to watch during the weekend and explain the reason.
1. Input:
<noinput>
1. Output:
I would recommend the movie "The Shawshank Redemption" because it is an excellent movie that is both moving and inspiring. It is the story of a ma
###
2. Instruction: Create a monthly budget for a person in the given location.
2. Input:
Seattle, WA
2. Output:
Food: $60 per day, totalling $1800
Rental: $2100 for one-bedroom apartment
Utilities: $150
Transportation: $100 for public transit and taxi, $100 for gasoline
Auto insurance: $150
Medical and dental : $200
Clothes and shopping: $500
Total Expenses: $5100
###
3. Instruction: Detect if there is gender-biased and non-inclusive phrasing in the sentence. If there is, suggest an alternative. Otherwise, outpu
3. Input:
The discovery could change all mankind.
3. Output:
The discovery could change all humankind.
```

6.2. Data synthesis

Generated data :

Variables:

1. Question : Generated or “reverse translated” from LLM (e.g. T5, GPT …)
2. Schema

Response:

1. Schema element (table name, column name, …)

“Reverse translated” as we reversely generate question “from” SQL.

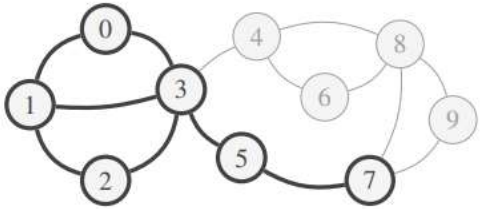
6.2.1. Procedure

1. Randomly generate SQL statements
2. Ask LLM to “translate” generated SQL statements into NL.
3. Make dataset with (NL, schema) - [schema info for SQL].

Example:

1. SELECT count(*) FROM car_list WHERE car_list.manufacturer = “Porsche”
2. “Find the number of car models made by Porsche.”
3. (“Find the number of car models made by Porsche.”, {schema}) - [car_list, car_list.manufacturer]

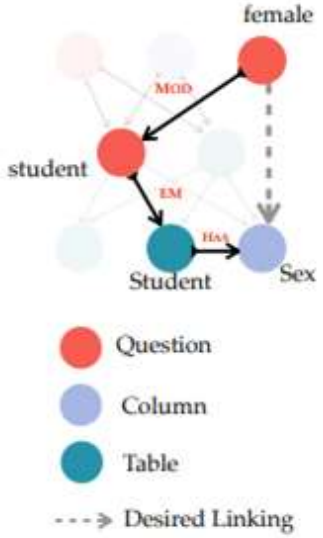
6.3. For large database



$\mathcal{G}_s = \texttt{SAMPLE}(\mathcal{G})$

For large database, the graph for structural information in schema can be too large and complex;

leading to overfitting or neighboring explosion.



To constrain such, apply GraphSaint to GNN-module to restrict the number of neighbors during the multi-hop relation training.

6.4. Expecting benefits

1. Reinforce GNN module.

Contrast to semantic module (T5 encoder), which is already trained enough with vast sized language datasets,
the structural module (GNN) has relatively smaller amount of data (data with Question-SQL pair)

Such data augmentation can help model further learning structural information.

2. Aid end users fine tuning on their target database

Since end users may have little or no data for fine tuning,
generating data from schema automatically can improve user experience.

6.4. Expecting benefits

3. Effectively leverage LLM for data generation

Can leverage powerful LLM in data generation, removing human effort from the process and still gaining good quality of data.

4. Help fine tuning on even large database

Using GraphSaint, prevent too much training cost, yet training well on large complex database schema.