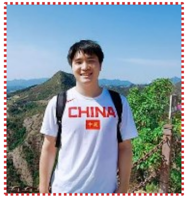
TAble Pre-training via Learning a Neural SQL Executor








Qian Liu[1] Bei Chen[2] Jiaqi Guo[3] Morteza Ziyadi[2] Zeqi Lin[2] Weizhu Chen[2] Jian-Guang Lou[2]

[1] 北京航空航天大學 BEIHANG UNIVERSITY [2] Microsoft [3] 西安交通大學 XI'AN JIAOTONG UNIVERSITY

ICLR

10th International Conference on
Learning Representations (ICLR 2022)

Background: Table-based Question Answering

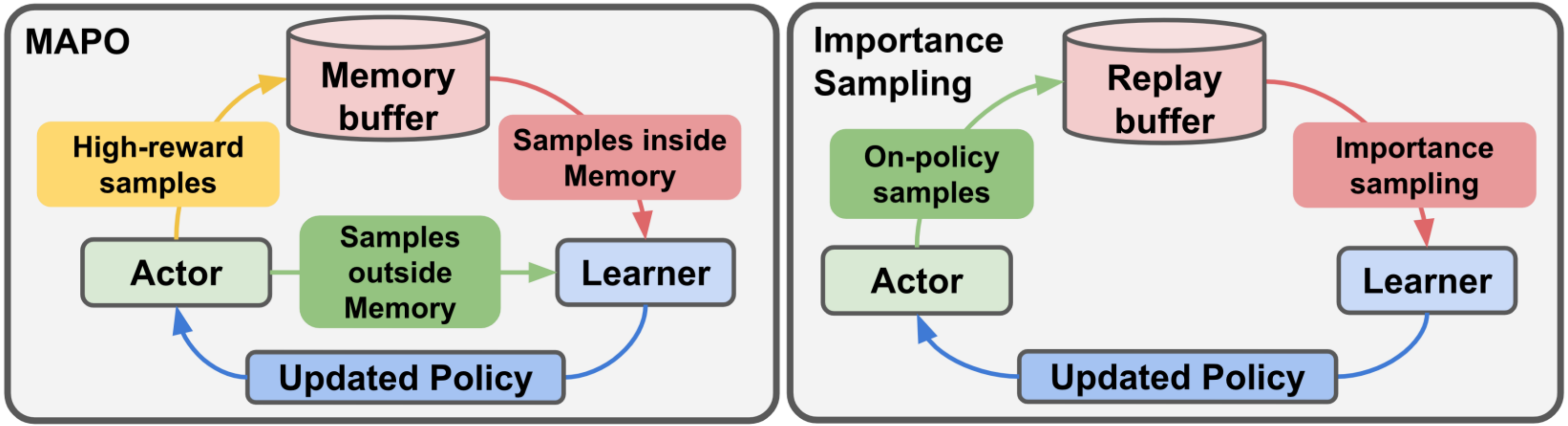
Year	City	Country	Nations
1896	Athens	Greece	14
1900	Paris	France	24
1904	St. Louis	USA	12
…	…	…	…
2004	Athens	Greece	201
2008	Beijing	China	204

Greece held its last Summer Olympics in which year?

2004

Previous Work: Reinforcement Learning

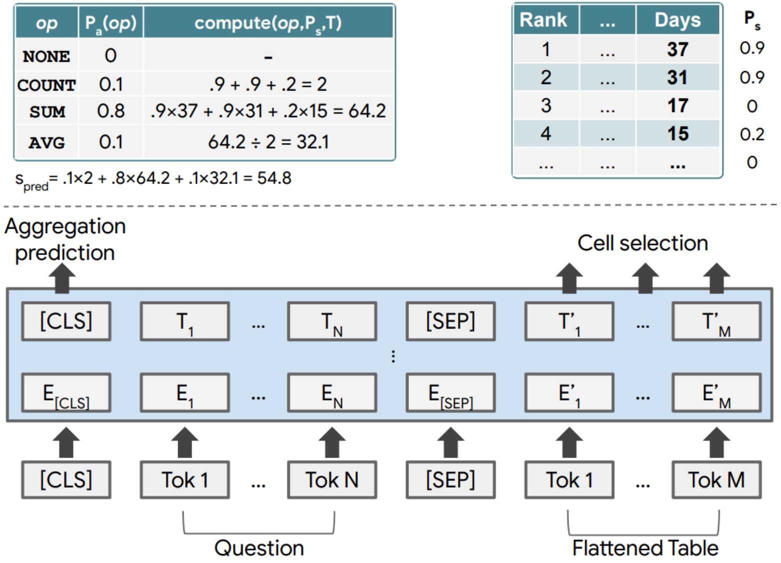
Obtain rewards by comparing execution results of sampled SQL queries with golden answers to train a text-to-SQL semantic parser. Hard to scale to complex scenarios.



[Chen et al. 2018]

Previous Work: Table Parsing

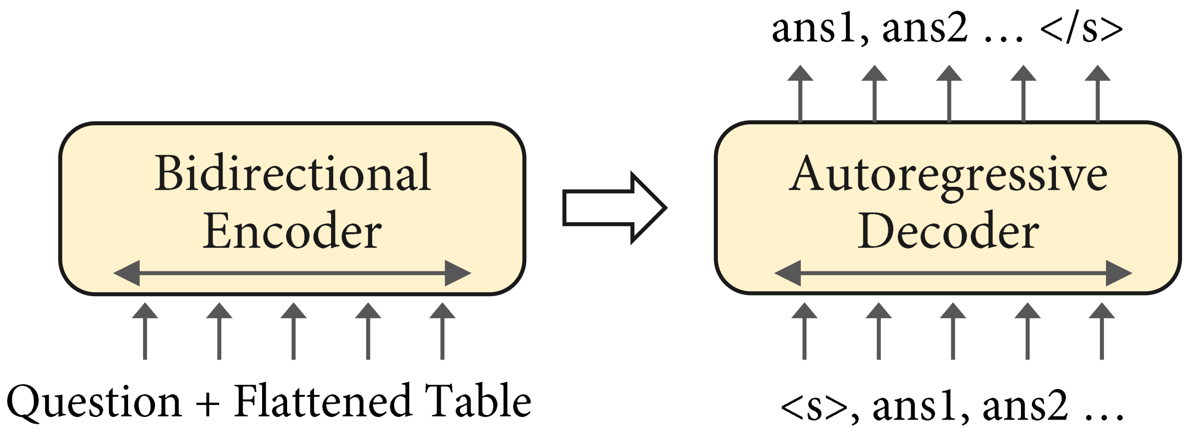
Predict answer by selecting table cell values and optionally applying an aggregation operator to the selected region. Flexibility is limited.



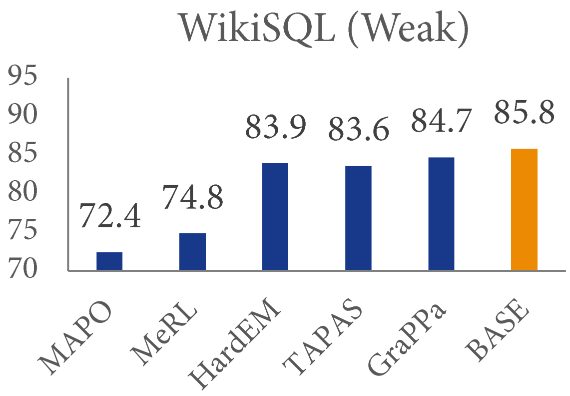
[Herzig et al. 2020]

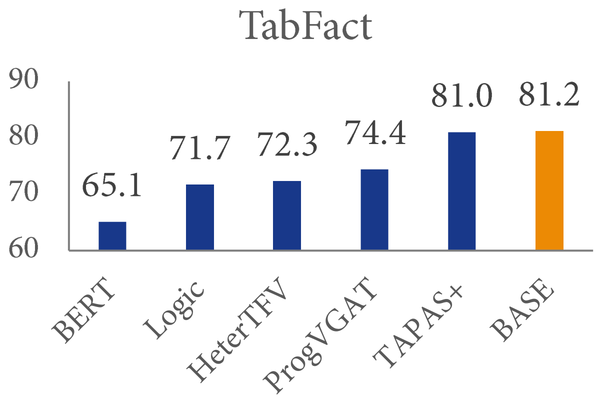
Our Proposal: Generative Language Model

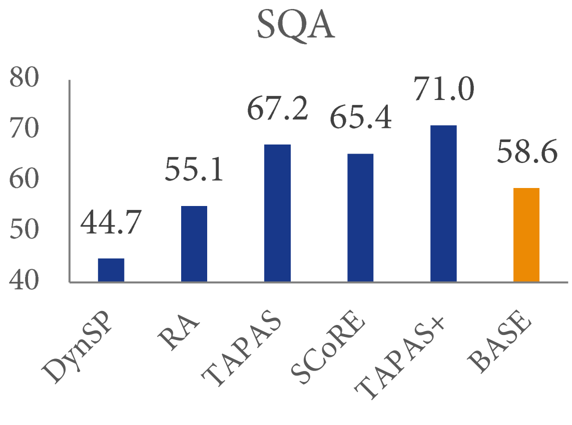
We formulate the task of table-based question answering as answer generation, and leverages generative language models (e.g., BART) to output autoregressively.

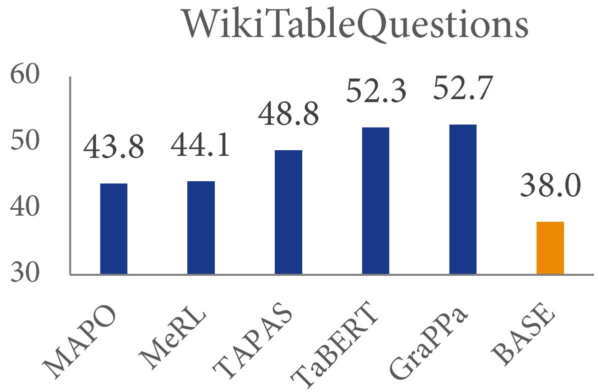


Preliminary Result: Models Are Data-Hungry









Motivation: Program as Proxy

Programming Context : Database

Year	City	Country	Nations
1896	Athens	Greece	14
1900	Paris	France	24
1904	St. Louis	USA	12
…	…	…	…
2004	Athens	Greece	201
2008	Beijing	China	204

Program:

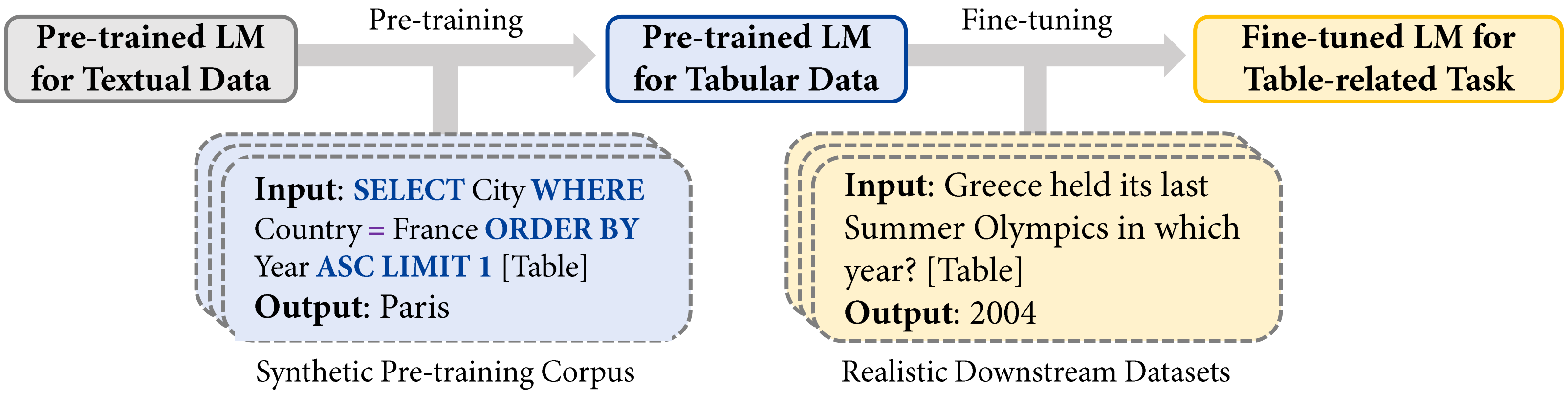
SELECT City **WHERE** Country = France **ORDER BY** Year **ASC LIMIT** 1

Executor

Paris

Our Proposal: SQL Execution Pre-training

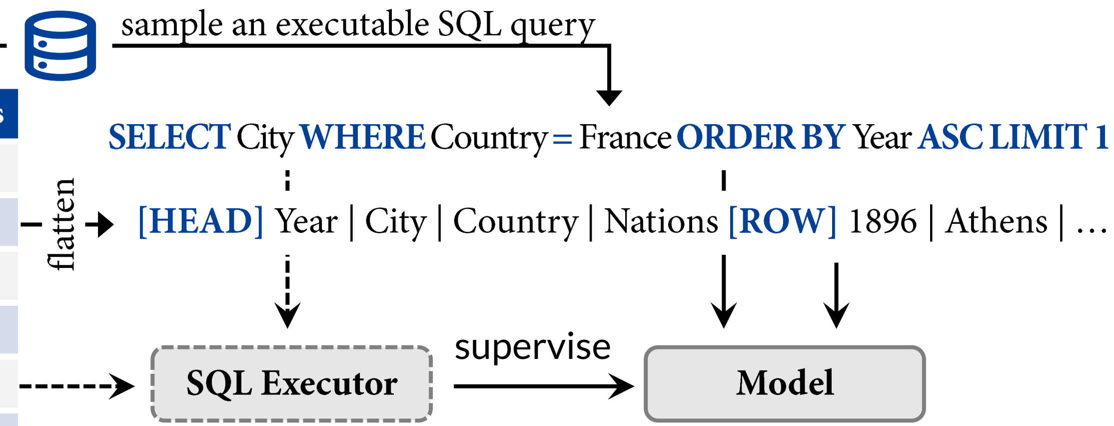
Pre-training a model to mimic the behavior of a symbolic execution engine.



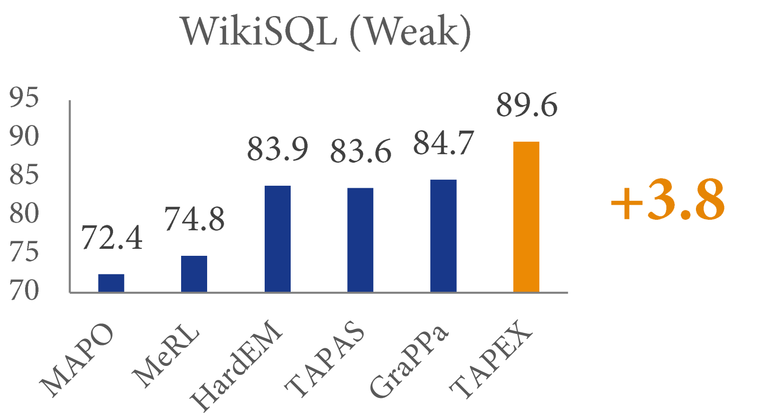
Our Proposal: SQL Execution Pre-training

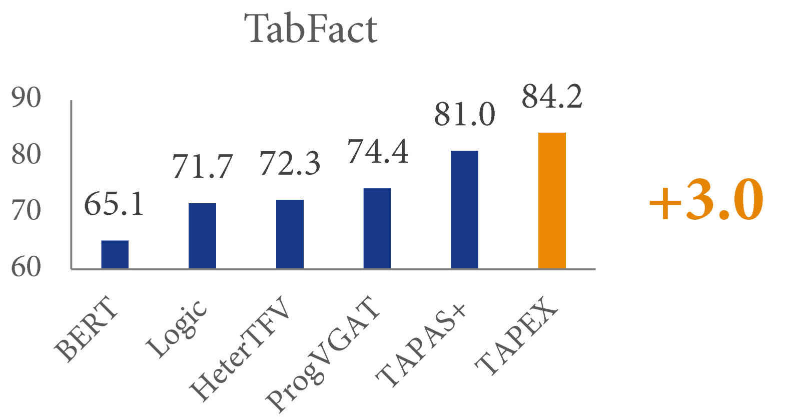
If we train a model to mimic the SQL query execution procedure over databases, we believe it learns latent programmatic operations from the execution engine.

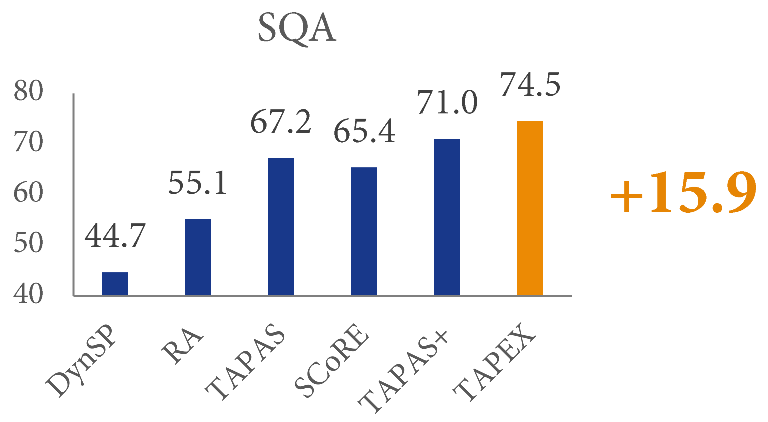
Year	City	Country	Nations
1896	Athens	Greece	14
1900	Paris	France	24
1904	St. Louis	USA	12
…	…	…	…
2004	Athens	Greece	201
2008	Beijing	China	204

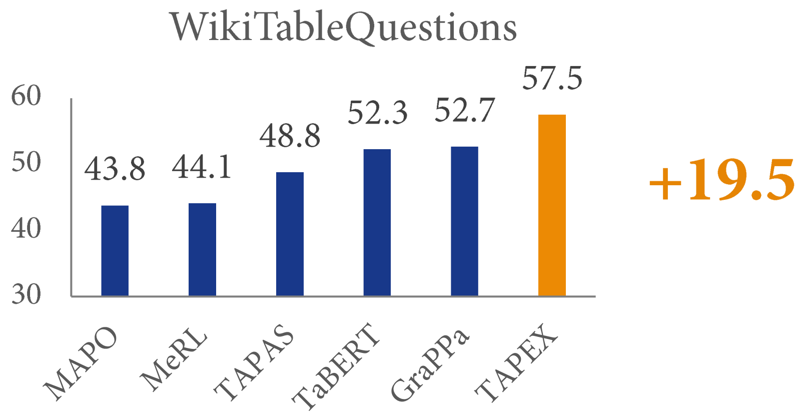


Experimental Result: SOTA Across Benchmarks



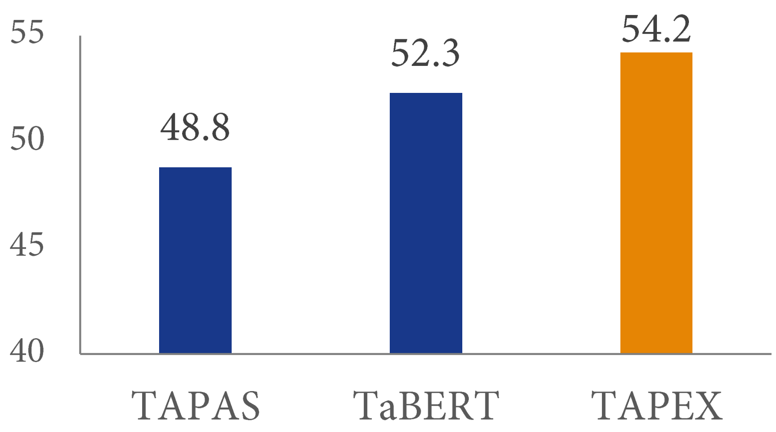




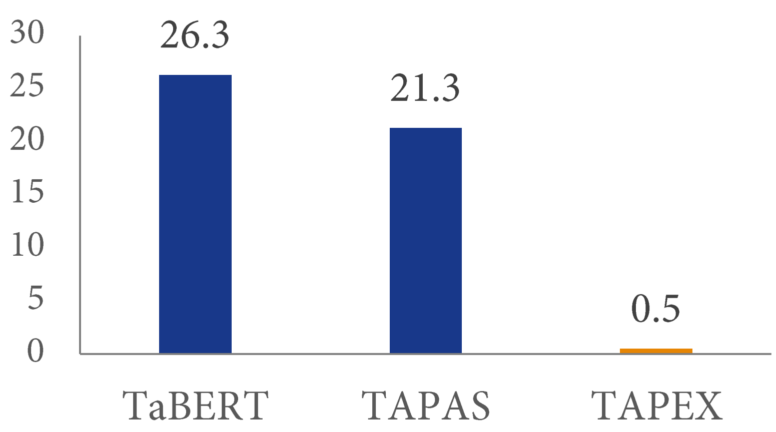


Experimental Result: Cost-Effective Pre-training

Fine-tuning Performance



Pre-training Corpus (Million)



Compared with TaBERT, **2%** of corpus yields **2%** improvement!

Take Away: Pre-training without Real Data

When performing continual pre-training, instead of mining a large noisy web corpus, we can also try to synthesize an accurate and small corpus.

Take Away: Pre-training without Language Modeling

When performing continual pre-training, instead of performing the general-purpose language modeling, we can also try to simulate the specialized skill.

Take Away: Pre-training without Natural Language

When performing continual pre-training for natural language tasks, instead of natural language, we can also try to leverage programs.

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

```
def start_link(name, opts \\ []) do
  GenServer.start_link(__MODULE__, {name}, opts)
end

def init({name}) do
  require Logger
  Logger.log :debug, "Started channel #{name}!"
  :pg2.join(:channels, self)
  :ets.insert(:channels, {name, self})
  users = :ets.new(:users, [:set, :protected])
  {:ok, {name, users, []}}
end

def handle_call({:send, message}, _from, {name, _users, _buffer} = state
  Kaguya.Util.sendPM(name, message)
  {:reply, :ok, state}
end

def handle_call({:rename_user, {old_nick, new_nick}}, _from, {_name, use
  case :ets.lookup(users, old_nick) do
    [{^old_nick, user}] ->
      new_user = %{user | nick: new_nick}
      :ets.delete(users, old_nick)
      :ets.insert(users, {new_nick, new_user})
    [] -> :ok
  end
  {:reply, :ok, state}
end
```