From BERT to GPT-3 Codex: Harnessing the Potential of Very Large Language Models for Data Management

Immanuel Trummer
Write a tagline for an ice cream shop.
My Background

DB Tuning
- SkinnerMT [VLDB'23]
- UDO [VLDB'22; AAAI'22]
- Dingo [VLDB'22]
- SkinnerDB [TODS'21]

DB Interfaces
- NEAT [CIDR'22]
- DB-BERT [SIGMOD'22]
- CodexDB [VLDB'22]
- BABOONS [VLDB'22]
- MUVE [VLDB'21]
- WebChecker [DEB'21]
- CiceroDB [ICDE'21]

ML

NLP
My Background

**DB Tuning**
- SkinnerMT [VLDB'23]
- UDO [VLDB'22; AAAI'22]
- Dingo [VLDB'22]
- SkinnerDB [TODS'21]

**DB Interfaces**
- NEAT [CIDR'22]
- DB-BERT [SIGMOD'22]
- CodexDB [VLDB'22]
- BABOONS [VLDB'22]
- MUVE [VLDB'21]
- WebChecker [DEB'21]
- CiceroDB [ICDE'21]

*Uses Language Models*
Target Audience

**Database Systems**
- Advanced

**Deep Learning**
- Basic

**Language Models**
- None
Tutorial Goal

Database Systems
- Advanced

Deep Learning
- Basic

Language Models
- Basic
Tutorial Outline

1. **Transformer** Architecture
2. **Transfer** Learning
3. **Libraries** and Interfaces
4. **Applications** in Data Management

https://itrummer.github.io/lm4db/
The Transformer Model
Attention Is All You Need

Ashish Vaswani*  
Google Brain  
avaswani@google.com

Noam Shazeer*  
Google Brain  
nnoam@google.com

Niki Parmar*  
Google Research  
nikkip@google.com

Jakob Uszkoreit*  
Google Research  
usz@google.com

Llion Jones*  
Google Research  
lion@google.com

Aidan N. Gomez* †  
University of Toronto  
aidan@cs.toronto.edu

Łukasz Kaiser*  
Google Brain  
lukaszkaizer@google.com

Illia Polosukhin* ‡  
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less memory. 
Attention is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez†
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Ilia Polosukhin* †
ilia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less computational capacity.
Attention Mechanism vs. Key-Value Stores

• Shared **vocabulary**:  
  • Keys, values, queries

• Similar **semantics**:  
  • Find keys matching query  
  • Retrieve associated values
Attention: Simplifying Intuition

I Love Database Research
Attention: Simplifying Intuition

Key
Value

I
Love
Database
Research
Attention: Simplifying Intuition

Query
- Triangle
- Star
- Circle
- Square

Key
- Star
- Circle
- Square
- Triangle

Value
- Circle
- Triangle
- Square
- Star

I Love Database Research
Attention: Simplifying Intuition

Query

Key

Value

I Love Database Research
Attention: Simplifying Intuition

Query

Key

Value

I Love Database Research
Attention: Simplifying Intuition

Query

Key

Value

I Love Database Research
Attention: Simplifying Intuition

I Love Database Research

Key
Value
Query
Parallel!
Match!
Attention: Simplifying Intuition

Value

Query

Key

I

Love

Database

Research

Match!

Retrieve!

Parallel!
Attention: Simplifying Intuition

Output

Query

Parallel!

Key

Value

I Love Database Research

Match!

Retrieve!
Attention: Simplifying Intuition

Output
- Star
- Circle
- Triangle
- Square

Query
- Triangle
- Star
- Circle
- Square

Key
- Star
- Circle
- Triangle
- Square

Value
- Circle
- Triangle
- Square
- Star

I Love Database Research
Attention vs. Simplifying Intuition

- Real-valued **vectors** instead of discrete symbols
- Continuous **similarity** between queries and keys
- Output value is **sum** of values, weighted by similarity
- Several **normalization** steps
Attention
Attention
Attention

![Diagram showing linear layers and attention process](image)
Attention

Linear Layer

Q

K

V

I ❤️ DB

Slides by Immanuel Trummer, Cornell University
Attention

\[ Q \times K^T \]

Linear Layer

I ❤ DB

I ❤ DB

I ❤ DB

I ❤ DB

I ❤ DB

I ❤ DB
Attention

\[
\mathbf{Q} \times \mathbf{K}^T + \mathbf{M}
\]

Linear Layer

Linear Layer

Linear Layer
Attention

Scale(\text{Linear Layer})
Attention

Scale(\text{Linear Layer} \times \text{Linear Layer} + \text{Linear Layer}) \times \text{Linear Layer}
Attention

\[
\text{Scale}(Q, K^T, M, V) = Q K^T M V
\]
Demo: Visualizing Attention

https://colab.research.google.com/drive/1DG2h6uakCsSVmU0Vem0E5qUGWeADTqe5
Recurrent Neural Network

I Love Database Research
Recurrent Neural Network

Sequential!!

LSTM

I

LSTM

Love

LSTM

Database

LSTM

Research
## Attention versus Recurrence

$d$: Vector dimension  
$n$: Sequence length

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Complexity per Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention</td>
<td>$O(n^2 d)$</td>
</tr>
<tr>
<td>Recurrent</td>
<td>$O(n d^2)$</td>
</tr>
</tbody>
</table>
Attention versus Recurrence

\( d \): Vector dimension  
\( n \): Sequence length

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Complexity per Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention</td>
<td>( O(n^2 \cdot d) )</td>
</tr>
<tr>
<td>Recurrent</td>
<td>( O(n \cdot d^2) )</td>
</tr>
</tbody>
</table>

Faster if \( d > n \)
## Attention versus Recurrence

\( d \): Vector dimension  
\( n \): Sequence length

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Complexity per Layer</th>
<th>Sequential Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention</td>
<td>( O(n^2d) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>Recurrent</td>
<td>( O(n^2d^2) )</td>
<td>( O(n) )</td>
</tr>
</tbody>
</table>

Self-attention is ... Faster if \( d > n \)
## Attention versus Recurrence

$d$: Vector dimension  
$n$: Sequence length

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Complexity per Layer</th>
<th>Sequential Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention</td>
<td>$O(n^2d)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Recurrent</td>
<td>$O(n^2d^2)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Self-attention is faster if $d > n$ and more parallelizable.
### Attention versus Recurrence

$d$: Vector dimension  
$n$: Sequence length

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Complexity per Layer</th>
<th>Sequential Operations</th>
<th>Maximum Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention</td>
<td>$O(n^2d)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Recurrent</td>
<td>$O(n^2d^2)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Self-attention is faster if $d > n$  
More parallelizable
### Attention versus Recurrence

$d$: Vector dimension  
$n$: Sequence length

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Complexity per Layer</th>
<th>Sequential Operations</th>
<th>Maximum Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention</td>
<td>$O(n^2d)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Recurrent</td>
<td>$O(n^4d^2)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Self-attention is ...  
Faster if $d > n$  
More parallelizable  
Easier to learn

$d$: Vector dimension  
$n$: Sequence length
The Transformer

(Details omitted: skip connections, layer normalization, masking)
Multi-Head, Multi-Layer Attention Visualization
The Transformer

(Details omitted: skip connections, layer normalization, masking)
The Transformer

(Details omitted: skip connections, layer normalization, masking)
The Transformer

(rect details omitted: skip connections, layer normalization, masking)
Transfer Learning
Transfer Learning: Idea

Untrained
Transfer Learning: Idea

Untrained
Transfer Learning: Idea

Untrained

Not Enough Lessons!
Transfer Learning: Idea

Untrained

Not Enough Lessons!
Transfer Learning: Idea

Untrained

Not Enough Lessons!
Transfer Learning: Idea

Untrained

Not Enough Samples!

Text-to-SQL
Pre-Training
Pre-Training

Lorem ... – Ipsum

... Lorem – Ipsum
Pre-Training

Marco ... Polo
Pre-Training
Pre-Training

Big ... 

... Data
Pre-Training
Fine-Tuning

How many customers?

SELECT Count(*) FROM Customer
Transfer Learning: Idea

Untrained

Not Enough Samples!

Text-to-SQL

Text Completion
# Pre-Training Objectives

<table>
<thead>
<tr>
<th>Objective</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Masked Language Modeling</td>
<td>Predict obfuscated words</td>
<td>BERT</td>
</tr>
<tr>
<td>Causal Language Modeling</td>
<td>Predict next word</td>
<td>GPT</td>
</tr>
<tr>
<td>Denoising Objective</td>
<td>Correct text with noise</td>
<td>BART</td>
</tr>
</tbody>
</table>
Quantifying Advantages

Universal Language Model Fine-tuning for Text Classification

Jeremy Howard*
fast.ai
University of San Francisco
j@fast.ai

Sebastian Ruder*
Insight Centre, NUI Galway
Aylien Ltd., Dublin
sebastian@ruder.io

Abstract
Inductive transfer learning has greatly impacted computer vision, but existing approaches in NLP still require task-specific modifications and training from scratch. While Deep Learning models have achieved state-of-the-art on many NLP tasks, these models are trained from scratch, requiring large datasets, and days to converge. Research in NLP focused mostly on transductive transfer (Blitzer et al.,
Quantifying Advantages

... with only 100 labeled examples, it matches the performance of training from scratch on 100x more data

Universal Language Model Fine-tuning for Text Classification

Jeremy Howard*
fast.ai
University of San Francisco
j@fast.ai

Sebastian Ruder*
Insight Centre, NUI Galway
Aylien Ltd., Dublin
sebastian@ruder.io

Abstract
Inductive transfer learning has greatly impacted computer vision, but existing approaches in NLP still require task-specific modifications and training from scratch.

While Deep Learning models have achieved state-of-the-art on many NLP tasks, these models are trained from scratch, requiring large datasets, and days to converge. Research in NLP focused mostly on transductive transfer (Blitzer et al.,...
Evolution of Language Models

Year

2018 2019 2020 2021 2022 2023

# Parameters

10000 1000 100 1

BERT GPT-2 Megatron T5 Turing-NLG Switch Transformer Wu-Dao 2 Megatron-Turing PaLM

Switch Transformer Wu-Dao 2 Megatron-Turing PaLM

Megatron-Turing

Turing-NLG

GPT-3

T5

Megatron

BERT

GPT-2
Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*
Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam
Girish Sastry Amanda Askell Sandhini Agarwal Ariel Herbert-Voss
Gretchen Krueger Tom Henighan Rewon Child Aditya Ramesh
Daniel M. Ziegler Jeffrey Wu Clemens Winter
Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray
Benjamin Chess Jack Clark Christopher Berner
Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

Abstract
We demonstrate that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even becoming competitive with prior state-of-
Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*
Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam
Girish Sastry Amanda Askell Sandhini Agarwal Ariel Herbert-Voss
Gretchen Krueger Tom Henighan Rewon Child Aditya Ramesh
Daniel M. Ziegler Jeffrey Wu Clemens Winter
Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray
Benjamin Chess Jack Clark Christopher Berner
Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

Abstract
We demonstrate that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even becoming competitive with prior state-of-
Prompting

- Describe task as text input
- **Zero-shot** learning
  - No samples are provided in input
- **Few-shot** learning
  - Few (typically up to ten) samples
Prompting

• Describe task as text input

• **Zero-shot** learning
  • No samples are provided in input

• **Few-shot** learning
  • Few (typically up to ten) samples

*Prompt Formulation Matters!*
Libraries and Interfaces
Hugging Face 🙈 Transformers

https://huggingface.co/
GPT-3 by OpenAI

https://openai.com/api/
Applications to Data Management
Natural language interfaces to databases – an introduction

I. ANDROUTSOPOULOS, G.D. RITCHIE
Department of Artificial Intelligence, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, Scotland, UK
e-mail: ion@aisb.ed.ac.uk, G.D.Ritchie@ed.ac.uk

P. THANISCH
Department of Computer Science, University of Edinburgh
King’s Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland, UK
e-mail: pt@dcs.ed.ac.uk

(Received 25 June 1994; revised 10 December 1994)
NaLIR: An Interactive Natural Language Interface for Querying Relational Databases

Fei Li
Univ. of Michigan, Ann Arbor
lifei@umich.edu

H. V. Jagadish
Univ. of Michigan, Ann Arbor
jag@umich.edu

ABSTRACT

In this demo, we present NaLIR, a generic interactive natural language interface for querying relational databases. NaLIR can accept a logically complex English language sentence as query input. This query is first translated into a SQL query, which may include aggregation, nesting, and various types of joins, among other things, and then evaluated against an RDBMS. In this demonstration, we show that NaLIR, while far from being able to pass the Turing test, is perfectly usable in practice, and able to handle even quite complex queries in a variety of application domains. In addition, we also demonstrate how carefully designed interactive communication can avoid misinterpretation with minimum user burden.

Categories and Subject Descriptors

H.5.2 [Information Interfaces and Presentation]: User/Machine Systems—Interfaces—Human factors

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

1. INTRODUCTION

Keywords

Structured query approach, while expressive and powerful, is not easy for naive users. The keyword-based approach is quite complex queries in a variety of application domains.

Traditionally, research work in querying data from relational databases often follows one of two paths: the structured query approach and the keyword-based approach. Both have advantages to a large extent: even naive users are able to express very friendly to use, but cannot express accurately. In contrast, natural language has both advantages to a large extent: even naive users are able to express very friendly to use, but cannot express accurately. In contrast, natural language queries is often regarded as the ultimate goal for a database query interface.

We believe that an ideal natural language interface should believe we have removed the greatest barrier in natural language querying of databases.

In this demo, we describe NaLIR, a generic interactive natural language interface for querying relational databases. In NaLIR, an arbitrary English language sentence, which can be quite complex in logic, is taken as query input. This query is first translated into a SQL query, which may contain aggregation, nesting, and various types of joins, among other things. Then, an RDBMS is used to evaluate the translated SQL query and return the results to the user. For exam-
<table>
<thead>
<tr>
<th>Rank</th>
<th>Model</th>
<th>Test</th>
</tr>
</thead>
</table>
| 1    | SHiP+PICARD (DB content used)  
*Anonymous* | 76.6 |
| 2    | RASAT+PICARD (DB content used)  
*Anonymous* | 75.5 |
| 3    | T5-SR (DB content used)  
*Anonymous* | 75.2 |
| 4    | RESDSQL+T5-1.1-lm100k-xl (DB content used)  
*Anonymous* | 75.1 |
| 4    | T5-3B+PICARD (DB content used)  
*Element AI, a ServiceNow company*  
(Scholak et al., EMNLP21) code | 75.1 |
| 6    | RESDSQL+T5-1.1-lm100k-large (DB content used)  
*Anonymous* | 74.8 |
| 7    | SeaD + SP (DB content used)  
*Anonymous* | 74.1 |
| 8    | RATSQL+GAP+NaïSQL (DB content used)  
*Queen Mary University of London*  
(Gan et al., EMNLP Findings'21) code | 73.3 |
| 9    | SmBoP + GraPPa (DB content used)  
*Tel-Aviv University & Allen Institute for AI*  
(Rubin and Berant, NAACL'21) code | 71.1 |
| 10   | RaSaP + ELECTRA (DB content used)  
*Ant Group, ZhiXiaoBao & Ada*  
(Huang et al.,'21) | 70.0 |

Leaderboard of SPIDER benchmark
PI2: End-to-end Interactive Visualization Interface Generation from Queries

Yiru Chen
cyru.chen@columbia.edu
Columbia University
New York, NY, USA

Eugene Wu
eyu@cs.columbia.edu
Columbia University
New York, NY, USA

ABSTRACT
Interactive visualization interfaces are critical in data analysis. Yet creating new interfaces is challenging, as the developer must understand the queries needed for the desired analysis task, and then design the appropriate interface. Existing task models are too abstract to be used to automatically generate interfaces, and visualization recommenders do not take the queries nor interactions into account. PI2 is the first system to generate fully functional interactive visualization interfaces from a representative sequence of task queries. PI2 analyzes queries syntactically and proposes a novel DiffTree representation that encodes the systematic variations between query abstract syntax trees. PI2 then poses interface generation as a schema mapping problem from each DiffTree to a visualization that renders its results, and the variations encoded in each DiffTree to interactions in the interface. Interface generation further takes the layout and screen size into account. Our user studies show that PI2 interfaces are comparable to or better than those designed by developers, and that PI2 can generate novel visualization tasks.

Figure 1: (a) Google's Covid-19 visualization. Using queries in Listing 1, interfaces generated by (b) this paper (PI2) and (c) prior work (PI).

1 INTRODUCTION

Interactive visualization interfaces are critical in data analysis. Yet creating new interfaces is challenging, as the developer must understand the queries needed for the desired analysis task, and then design the appropriate interface. Existing task models are too abstract to be used to automatically generate interfaces, and visualization recommenders do not take the queries nor interactions into account. PI2 is the first system to generate fully functional interactive visualization interfaces from a representative sequence of task queries. PI2 analyzes queries syntactically and proposes a novel DiffTree representation that encodes the systematic variations between query abstract syntax trees. PI2 then poses interface generation as a schema mapping problem from each DiffTree to a visualization that renders its results, and the variations encoded in each DiffTree to interactions in the interface. Interface generation further takes the layout and screen size into account. Our user studies show that PI2 interfaces are comparable to or better than those designed by developers, and that PI2 can generate novel visualization tasks.
we fine-tuned a recent NL-to-SQL language model ... using 50 manually annotated ... examples

PI2: End-to-end Interactive Visualization Interface Generation from Queries

Yiru Chen
yiru.chen@columbia.edu
Columbia University
New York, NY, USA

Eugene Wu
ewu@cs.columbia.edu
Columbia University
New York, NY, USA

ABSTRACT
Interactive visualization interfaces are critical in data analysis. Yet creating new interfaces is challenging, as the developer must understand the queries needed for the desired analysis task, and then design the appropriate interface. Existing task models are too abstract to be used to automatically generate interfaces, and visualization recommenders do not take the queries nor interactions into account. PI2 is the first system to generate fully functional interactive visualization interfaces from a representative sequence of task queries. PI2 analyzes queries syntactically and proposes a novel DiffTree representation that encodes the systematic variations between query abstract syntax trees. PI2 then poses interface generation as a schema mapping problem from each DiffTree to a visualization that renders its results, and the variations encoded in each DiffTree to interactions in the interface. Interface generation further takes the layout and screen size into account. Our user studies show that PI2 interfaces are comparable to or better than those designed by developers, and that PI2 can generate ex-

Introduction

Figure 1: (a) Google's Covid-19 visualization. Using queries in Listing 1, interfaces generated by (b) this paper (PI2) and (c) prior work (PI).

(c) PI
Annotating Columns with Pre-trained Language Models

Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang
Megagon Labs
{yoshi,jinfeng,yuliang,dan_z}@megagon.ai

Çağatay Demiralp
Sigma Computing
cagatay@sigmacomputing.com

Chen Chen
Megagon Labs
chen@megagon.ai

Wang-Chiew Tan
Meta AI
wanchie@fb.com

ABSTRACT
Inferring meta information about tables, such as column headers or relationships between columns, is an active research topic in data management as we find many tables are missing some of this information. In this paper, we study the problem of annotating table columns (i.e., predicting column types and the relationships between columns) using only information from the table itself. We develop a multi-task learning framework (called Doneto) based on pre-trained language models, which takes the entire table as input and predicts column types/relations using a single model. Experimental results show that Doneto establishes new state-of-the-art performance on two benchmarks for the column type prediction and column relation prediction tasks with up to 4.0% and 11.9% improvements, respectively. We report that Doneto can already...
Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks

Riccardo Cappuzzo
cappuzzo@eurecom.fr
EURECOM

Paolo Papotti
papotti@eurecom.fr
EURECOM

Saravanan Thirumuruganathan
sthirumuruganathan@hbku.edu.
qa
QCRI, HBKU

ABSTRACT
Deep learning based techniques have been recently used with promising results for data integration problems. Some methods directly use pre-trained embeddings that were trained on a large corpus such as Wikipedia. However, they may not always be an appropriate choice for enterprise datasets with custom vocabulary. Other methods adapt techniques from natural language processing to obtain embeddings for the enterprise’s relational data. However, this approach blindly treats a tuple as a sentence, thus losing a large amount of contextual information present in the tuple.

We propose algorithms for obtaining local embeddings that are effective for data integration tasks on relational databases. We make four major contributions. First, we de-
Valentine: Evaluating Matching Techniques for Dataset Discovery

Christos Koutras1 George Siachamis1,2 Andra Ionescu1 Kyriakos Psarakis1 Jerry Brons2
Marios Fragkoulis1 Christoph Lofi1 Angela Bonifati3 Asterios Katsifodimos1

1Delft University of Technology 2ING Bank Netherlands 3Lyon 1 University

Abstract—Data scientists today search large data lakes to discover and integrate datasets. In order to bring together disparate data sources, dataset discovery methods rely on some form of schema matching: the process of establishing correspondences between datasets. Traditionally, schema matching has been used to find matching pairs of columns between a source and a target schema. However, the use of schema matching in dataset discovery methods differs from its original use. Nowadays schema matching serves as a building block for indicating and ranking inter-dataset relationships. Surprisingly, although a discovery method’s success relies highly on the quality of the underlying matching algorithms, the latest discovery methods employ existing schema matching algorithms in an ad-hoc fashion due to the lack of openly-available datasets with ground truth, reference method implementations, and evaluation metrics.

In this paper, we aim to rectify the problem of evaluating the effectiveness and efficiency of schema matching methods for discovering matched columns between two datasets. We present a unified, open-source schema matching experiment suite for comparing the state-of-the-art schema matching techniques. We also introduce a novel dataset discovery method that can serve as a guide for employing schema matching techniques, that can serve as a guide for employing schema matching techniques, that can serve as a guide for employing schema matching techniques in future dataset discovery methods.

The majority of these methods are based on a common, very critical component: schema matching, i.e., capturing relationships between elements of different schemata. In the case of tabular data, dataset discovery methods typically use schema matching techniques to automatically determine whether two columns (or even entire tables) are joinable or unionable. Since dataset discovery methods exploit relatedness information about a given set of datasets, the underlying matching technique of any data discovery method greatly affects its performance.

At the moment of writing, dataset discovery methods typically implement their own matcher, by combining or customizing existing schema matching methods. However, due to the lack of openly-available datasets with ground truth, reference method implementations, and evaluation metrics, the majority of these methods are based on a common, very critical component: schema matching, i.e., capturing relationships between elements of different schemata. In the case of tabular data, dataset discovery methods typically use schema matching techniques to automatically determine whether two columns (or even entire tables) are joinable or unionable. Since dataset discovery methods exploit relatedness information about a given set of datasets, the underlying matching technique of any data discovery method greatly affects its performance.
Deep Entity Matching with Pre-Trained Language Models

Yuliang Li, Jinfeng Li, Yoshihiko Suhara
Megagon Labs
{yuliang,jinfeng,yoshi}@megagon.ai

AnHai Doan
University of Wisconsin Madison
anhai@cs.wisc.edu

Wang-Chiew Tan
Megagon Labs
wangchiew@megagon.ai

ABSTRACT

We present DITTO, a novel entity matching system based on pre-trained Transformer-based language models. We fine-tune and cast EM as a sequence-pair classification problem to leverage such models with a simple architecture. Our experiments show that a straightforward application of language models such as BERT, DistilBERT, or RoBERTa pre-trained on large text corpora already significantly improves the matching quality and outperforms previous state-of-the-art (SOTA), by up to 29% of F1 score on benchmark datasets. We also developed three optimization techniques to further improve DITTO’s matching capability. DITTO allows domain knowledge to be injected by highlighting important pieces of input information that may be of interest when making matching decisions. DITTO also summarizes strings that are too long so that only the essential information is retained and used for EM. Finally, DITTO adapts a SOTA technique on data augmentation for text to EM to augment the training data with (difficult) examples. This way, DITTO is forced to learn “harder” to improve the model’s matching capability. The optimizations we developed further boost the performance. If the datasets are large, it can be expensive to determine the pairs of matching entries. For this reason, EM is typically accompanied by a pre-processing step, called blocking, to prune pairs of entries that are unlikely matches to reduce the number of candidate pairs to consider. As we will illustrate, correctly matching the candidate pairs requires substantial language understanding and domain-specific knowledge. Hence, entity matching remains a challenging task even for the most advanced EM solutions.

We present DITTO, a novel EM solution based on pre-trained Transformer-based language models (or pre-trained language models in short). We cast EM as a sequence-pair classification problem to leverage such models, which have been shown to generate highly contextualized embeddings that capture better language understanding compared to traditional word embeddings. DITTO further improves its matching capability through three optimizations: (1) It allows domain knowledge to be added by highlighting important pieces of the input that may be useful for matching decisions. (2) It summarizes long strings so that only the most essential information is retained and used for EM. (3) It augments training data with (difficult) examples, which challenges DITTO to learn “harder” and
Can Foundation Models Wrangle Your Data?

Avanika Narayan, Ines Chami†, Laurel Orr, Christopher Ré
Stanford University and †Numbers Station
{avanika,lorr1,chrismre}@cs.stanford.edu,ines.chami@numbersstation.ai

ABSTRACT
Foundation Models (FMs) are models trained on large corpora of data that, at very large scale, can generalize to new tasks without any task-specific finetuning. As these models continue to grow in size, innovations continue to push the boundaries of what these models can do on language and image tasks. This paper aims to understand an underexplored area of FMs: classical data tasks like cleaning and integration. As a proof-of-concept, we cast three data cleaning and integration tasks as prompting tasks and evaluate the performance of FMs on these tasks. We find that large FMs generalize and achieve SoTA performance on data cleaning and integration tasks, even though they are not trained for these data tasks. We identify specific research challenges and opportunities that these models present, including challenges with private and temporal data, and opportunities to make data driven systems more accessible to non-experts. We make our code and experiments publicly available at: https://github.com/HazyResearch/FM_prompting.

1 INTRODUCTION
Foundation Models (FMs) [17] are models trained on broad data that can be adapted to a wide array of tasks. The models trained on large corpora of text display several useful properties that make them an appealing choice compared to traditional approaches: a large amount of labeled data and external knowledge bases are not required; they do not require significant changes and can be adapted to many domains through prompt generation; they generalize to new tasks without any task-specific finetuning. As these models continue to grow in size, innovations continue to push the boundaries of what these models can do on language and image tasks. This paper aims to understand an underexplored area of FMs: classical data tasks like cleaning and integration. As a proof-of-concept, we cast three data cleaning and integration tasks as prompting tasks and evaluate the performance of FMs on these tasks. We find that large FMs generalize and achieve SoTA performance on data cleaning and integration tasks, even though they are not trained for these data tasks. We identify specific research challenges and opportunities that these models present, including challenges with private and temporal data, and opportunities to make data driven systems more accessible to non-experts. We make our code and experiments publicly available at: https://github.com/HazyResearch/FM_prompting.

Figure 1: A large FM can address an entity matching task using prompting. Rows are serialized into text and passed to the FM with the question “Are products A and B the same?”. The FM then generates a string “Yes” or “No” as the answer.

<table>
<thead>
<tr>
<th>Title</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macbook Pro</td>
<td>$1,999.00</td>
</tr>
<tr>
<td>Macbook Air</td>
<td>$899.00</td>
</tr>
</tbody>
</table>

Table 1

<table>
<thead>
<tr>
<th>Title</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macbook Pro</td>
<td>$1,999.00</td>
</tr>
<tr>
<td>Macbook Air</td>
<td>$899.00</td>
</tr>
</tbody>
</table>

Table 2
RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation

Nan Tang
QCRI, HBKU, Qatar
ntang@hbku.edu.qa

Ju Fan
Renmin University, China
fanj@ruc.edu.cn

Fangyi Li
Renmin University, China
fangyili@ruc.edu.cn

Jianhong Tu
Renmin University, China
tujh@ruc.edu.cn

Xiaoyong Du
Renmin University, China
duyong@ruc.edu.cn

Guoliang Li
Tsinghua University, China
liguoliang@tsinghua.edu.cn

Sam Madden
CSAIL, MIT, USA
madden@csail.mit.edu

Mourad Ouzzani
QCRI, HBKU, Qatar
mouzzani@hbku.edu.qa

ABSTRACT

Can AI help automate human-easy but computer-hard data preparation tasks that burden data scientists, practitioners, and crowd workers? We answer this question by presenting RPT, a denoising autoencoder for tuple-to-X models ("X" could be tuple, token, label, JSON, and so on). RPT is pre-trained for a tuple-to-tuple model by corrupting the input tuple and then learning a model to reconstruct the original tuple. It adopts a Transformer-based neural translation architecture that consists of a bidirectional encoder (similar to BERT) and a left-to-right autoregressive decoder (similar to GPT), leading to a generalization of both BERT and GPT. The pre-trained RPT can already support several common data preparation tasks such as data cleaning, auto-completion and schema matching. Better still, RPT can be fine-tuned on a wide range of data preparation tasks, such as value normalization, data transformation, data annotation, etc.

(a) Sample Tasks for Value Filling (IM: value to fill)

<table>
<thead>
<tr>
<th>type</th>
<th>company</th>
<th>year</th>
<th>memory</th>
<th>screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>iPhone 10</td>
<td>Apple</td>
<td>2017</td>
<td>64GB</td>
</tr>
<tr>
<td>e2</td>
<td>iPhone X</td>
<td>Apple Inc</td>
<td>2017</td>
<td>256GB</td>
</tr>
<tr>
<td>e3</td>
<td>iPhone 11</td>
<td>AAPL</td>
<td>2019</td>
<td>128GB</td>
</tr>
</tbody>
</table>

(b) A Sample Entity Resolution Task

<table>
<thead>
<tr>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>notebook</td>
</tr>
</tbody>
</table>
DB-BERT: A Database Tuning Tool that “Reads the Manual”

Immanuel Trummer
Cornell University
Ithaca, New York
itrummer@cornell.edu

ABSTRACT

DB-BERT is a database tuning tool that exploits information gained via natural language analysis of manuals and other relevant text documents. It uses text to identify database system parameters to tune as well as recommended parameter values. DB-BERT applies large, pre-trained language models (specifically, the BERT model) for text analysis. During an initial training phase, it fine-tunes model weights in order to translate natural language hints into recommended settings. At run time, DB-BERT learns to aggregate, adapt, and prioritize hints to achieve optimal performance for a specific database system and benchmark. Both phases are iterative and use reinforcement learning to guide the selection of tuning settings to evaluate (penalizing settings that the database system rejects while rewarding settings that improve performance). In our experiments, we leverage hundreds of text documents about database tuning as input for DB-BERT. We compare DB-BERT against various baselines, considering different benchmarks (TPC-C and TPC-H), metrics (throughput and run time), as well as database systems (Postgres and MySQL). In all cases, DB-BERT finds the best parameter settings among all compared methods. The code of experiments, we leverage hundreds of text documents about data-

Table 1: Example tuning hints with extractions.

<table>
<thead>
<tr>
<th>Text Snippet</th>
<th>Extraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>The default value of shared_buffer is very low ... The recommended value is 25% of your total machine RAM. [23]</td>
<td>shared_buffers = 0.25 · RAM</td>
</tr>
<tr>
<td>I changed 'random_page_cost' to 1 and retried the query. This time, PostgreSQL used a Nested Loop and the query finished 50x faster. [21]</td>
<td>random_page_cost = 1</td>
</tr>
<tr>
<td>On a dedicated database server, you might set the buffer pool size to 80% of the machine's physical memory size. [19]</td>
<td>innodb_buffer_pool_size = 0.8 · RAM</td>
</tr>
</tbody>
</table>

Manuals are useful. For instance, before starting to tune a database, a human database administrator might set the `innodb_buffer_pool_size` to 80% of the machine’s physical memory size. [19]
EMBER: No-Code Context Enrichment via Similarity-Based Keyless Joins

Sahaana Suri
Stanford University
sahaana@stanford.edu

Christopher Ré
Stanford University
chrismre@cs.stanford.edu

Ihab F. Ilyas
University of Waterloo
ilyas@uwaterloo.ca

Theodoros Rekatsinas
UW-Madison
thodrek@cs.wisc.edu

ABSTRACT
Structured data, or data that adheres to a pre-defined schema, can suffer from fragmented context: information describing a single entity can be scattered across multiple datasets or tables tailored for specific business needs, with no explicit linking keys. Context enrichment, or rebuilding fragmented context, using keyless joins is an implicit or explicit step in machine learning (ML) pipelines over structured data sources. This process is tedious, domain-specific, and lacks support in now-prevalent no-code ML systems that let users create ML pipelines using just input data and high-level configuration files. In response, we propose EMBER, a system that abstracts and automates keyless joins to generalize context enrichment. Our key insight is that EMBER can enable a general keyless join operator by constructing an index populated with task-specific embeddings. EMBER learns these embeddings by leveraging Transformer-based representation learning techniques. We describe our architectural configuration change.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s).

PVLDB Artifact Availability:
doi:10.14778/3494124.3494149

No-Code Context Enrichment via Similarity-Based Keyless Joins. PVLDB, Sahaana Suri, Ihab F. Ilyas, Christopher Ré, Theodoros Rekatsinas.
PVLDB Reference Format:

PVLDB Artifact Availability:
doi:10.14778/3494124.3494149

1 INTRODUCTION
Structured data, or data that adheres to a pre-defined schema, can

unstructured data have revolutionized domains spanning computer

Machine learning (ML) systems that extract semantic context from

applied these systems to structured and semi-structured datasets

leveraging Transformer-based representation learning techniques. We describe our architectural configuration change.

up to 39% recall, with as little as a single line con

mendation and question answering, and can exceed alternatives by

domains, including search, recom-

enrichment tailored to their task, such as similarity-based blocking

and dataset heterogeneity. Engineers develop solutions for context

context enrichment—yet is a heavily manual endeavor due to task

aspects to fragmented context: information describing a single

entity can be scattered across multiple datasets or tables tailored

due to data heterogeneity—we aim to automate it (bottom).

Predicting the rating of and recommending a new product

(A) Goal: Downstream ML

ITEM is a joinable key,

(B) Challenge: Info scattered across three catalogs

(C) Existing Approach: Manual Joins

(D) EMBER: replace & automate

Domain-specific analyses to join A, B, C

Enriched Data

Normalized Ratings Table

Proprietary Catalog A

ShoeMart, Ext Catalog C

ShoeShop, Ext Catalog B

 normalized schema is challenging as their context is often
CodexDB: Synthesizing Code for Query Processing from Natural Language Instructions using GPT-3 Codex

Immanuel Trummer
Cornell University
Ithaca, NY
itrummer@cornell.edu

ABSTRACT

CodexDB enables users to customize SQL query processing via natural language instructions. CodexDB is based on OpenAI’s GPT-3 Codex model which translates text into code. It is a framework on top of GPT-3 Codex that decomposes complex SQL queries into a series of simple processing steps, described in natural language. Processing steps are enriched with user-provided instructions and descriptions of database properties. Codex translates the resulting text into query processing code. An early prototype of CodexDB is able to generate correct code for up to 81% of queries for the WikiSQL benchmark and for up to 62% on the SPIDER benchmark.

PVLDB Reference Format:
doi:10.14778/3551793.3551841

PVLDB Artifact Availability:
The remainder of this paper is organized as follows. Section 2 discusses recent progress in natural language processing and compares CodexDB to prior work. Section 3 describes the architecture of the first prototype. Section 4 reports first experimental results, based on an early prototype of CodexDB. The paper discusses contributions are the following: each workload query, obtaining performance results for a subset can guide future development efforts. Also, generated code can be manually validated and reused in case of recurrent queries.

Example 1.2. A user needs help "debugging" a complex SQL query. To that purpose, the user wants to print intermediate results during query processing. CodexDB allows users formulating natural language instructions that are executed after each processing step. Instructing CodexDB to "Print intermediate results" has the desired effect and helps with query debugging.

CodexDB accepts queries, together with natural language instructions, as input. These instructions customizes the way in which queries are executed. CodexDB generates code to process queries while complying with additional instructions. A first option is to submit queries and instructions directly to GPT-3 for code generation. We will see in Section 4 that this approach does not work. Instead, CodexDB adapts techniques from classical query planning. It decomposes complex SQL queries into sequences of simple language instructions, inspired by the use cases outlined before. Processing steps are enriched with user-provided instructions and descriptions of database properties. Codex translates the resulting text into query processing code. An early prototype of CodexDB is able to generate correct code for up to 81% of queries for the WikiSQL benchmark and for up to 62% on the SPIDER benchmark.

The range of applications is vast. To name just a few, consider examples like debugging query plans, and formulating plans. A developer wants to benchmark different data processing frameworks (e.g., Pandas and Vaex in Python or Tableau in Excel) for their performance and convergence. This system is OpenAI’s GPT-3 Codex model. Codex is a large neural network, currently available via a private beta test, that translates text into code. This paper presents an early prototype of CodexDB. Section 3 describes the architecture of this system. Section 4 presents first experimental results in a series of simple processing steps, described in natural language. Processing steps are enriched with user-provided instructions and descriptions of database properties. Codex translates the resulting text into query processing code. An early prototype of CodexDB is able to generate correct code for up to 81% of queries for the WikiSQL benchmark and for up to 62% on the SPIDER benchmark.

PVLDB Reference Format:
doi:10.14778/3551793.3551841

PVLDB Artifact Availability:
The remainder of this paper is organized as follows. Section 2 discusses recent progress in natural language processing and compares CodexDB to prior work. Section 3 describes the architecture of the first prototype. Section 4 reports first experimental results, based on an early prototype of CodexDB. The paper discusses contributions are the following: each workload query, obtaining performance results for a subset can guide future development efforts. Also, generated code can be manually validated and reused in case of recurrent queries.

Example 1.2. A user needs help "debugging" a complex SQL query. To that purpose, the user wants to print intermediate results during query processing. CodexDB allows users formulating natural language instructions that are executed after each processing step. Instructing CodexDB to "Print intermediate results" has the desired effect and helps with query debugging.

CodexDB accepts queries, together with natural language instructions, as input. These instructions customizes the way in which queries are executed. CodexDB generates code to process queries while complying with additional instructions. A first option is to submit queries and instructions directly to GPT-3 for code generation. We will see in Section 4 that this approach does not work. Instead, CodexDB adapts techniques from classical query planning. It decomposes complex SQL queries into sequences of simple processing steps. In contrast to prior work, those steps are formulized in natural language using corresponding text templates. Processing steps are enriched with user-provided instructions and descriptions of database properties. Codex translates the resulting text into query processing code. An early prototype of CodexDB is able to generate correct code for up to 81% of queries for the WikiSQL benchmark and for up to 62% on the SPIDER benchmark.

The range of applications is vast. To name just a few, consider examples like debugging query plans, and formulating plans. A developer wants to benchmark different data processing frameworks (e.g., Pandas and Vaex in Python or Tableau in Excel) for their performance and convergence. This system is OpenAI’s GPT-3 Codex model. Codex is a large neural network, currently available via a private beta test, that translates text into code. This paper presents an early prototype of CodexDB. Section 3 describes the architecture of this system. Section 4 presents first experimental results in a series of simple processing steps, described in natural language. Processing steps are enriched with user-provided instructions and descriptions of database properties. Codex translates the resulting text into query processing code. An early prototype of CodexDB is able to generate correct code for up to 81% of queries for the WikiSQL benchmark and for up to 62% on the SPIDER benchmark.

PVLDB Reference Format:
doi:10.14778/3551793.3551841

PVLDB Artifact Availability:
The remainder of this paper is organized as follows. Section 2 discusses recent progress in natural language processing and compares CodexDB to prior work. Section 3 describes the architecture of the first prototype. Section 4 reports first experimental results, based on an early prototype of CodexDB. The paper discusses contributions are the following: each workload query, obtaining performance results for a subset can guide future development efforts. Also, generated code can be manually validated and reused in case of recurrent queries.

Example 1.2. A user needs help "debugging" a complex SQL query. To that purpose, the user wants to print intermediate results during query processing. CodexDB allows users formulating natural language instructions that are executed after each processing step. Instructing CodexDB to "Print intermediate results" has the desired effect and helps with query debugging.

CodexDB accepts queries, together with natural language instructions, as input. These instructions customizes the way in which queries are executed. CodexDB generates code to process queries while complying with additional instructions. A first option is to submit queries and instructions directly to GPT-3 for code generation. We will see in Section 4 that this approach does not work. Instead, CodexDB adapts techniques from classical query planning. It decomposes complex SQL queries into sequences of simple processing steps. In contrast to prior work, those steps are formulized in natural language using corresponding text templates. Processing steps are enriched with user-provided instructions and descriptions of database properties. Codex translates the resulting text into query processing code. An early prototype of CodexDB is able to generate correct code for up to 81% of queries for the WikiSQL benchmark and for up to 62% on the SPIDER benchmark.

The range of applications is vast. To name just a few, consider examples like debugging query plans, and formulating plans. A developer wants to benchmark different data processing frameworks (e.g., Pandas and Vaex in Python or Tableau in Excel) for their performance and convergence. This system is OpenAI’s GPT-3 Codex model. Codex is a large neural network, currently available via a private beta test, that translates text into code. This paper presents an early prototype of CodexDB. Section 3 describes the architecture of this system. Section 4 presents first experimental results in a series of simple processing steps, described in natural language. Processing steps are enriched with user-provided instructions and descriptions of database properties. Codex translates the resulting text into query processing code. An early prototype of CodexDB is able to generate correct code for up to 81% of queries for the WikiSQL benchmark and for up to 62% on the SPIDER benchmark.
From Natural Language Processing to Neural Databases

James Thorne  
University of Cambridge  
Facebook AI  
jt719@cam.ac.uk

Majid Yazdani  
Facebook AI  
myazdani@fb.com

Marzieh Saeidi  
Facebook AI  
marzieh@fb.com

Fabrizio Silvestri  
Facebook AI  
fsilvestri@fb.com

Sebastian Riedel  
Facebook AI  
sriedel@fb.com

Alon Halevy  
Facebook AI  
ayh@fb.com

ABSTRACT
In recent years, neural networks have shown impressive performance gains on long-standing AI problems, such as answering queries from text and machine translation. These advances raise the question of whether neural nets can be used at the core of query processing to derive answers from facts, even when the facts are expressed in natural language. If so, it is conceivable that we could relax the fundamental assumption of database management, namely, that our data is represented as fields of a pre-defined schema. Furthermore, such technology would enable combining information from text, images, and structured data seamlessly.

This paper introduces neural databases, a class of systems that use NLP transformers as localized answer derivation engines. We ground the vision in NEURALDB, a system for querying facts represented as natural language sentences. Researchers have long considered the application of neural nets to data management problems, including learning indices [16], query optimization, data cleaning and entity matching [20, 23, 32]. In applying neural networks to data management, research has so far assumed that the data was modeled by a database schema.

The success of neural networks in processing unstructured data such as natural language and images raises the question of whether their use can be extended to a point where we can relax the fundamental assumption of database management, which is that the data we process is represented as fields of a pre-defined schema. What if, instead, data and queries can be represented as short natural language sentences, and queries can be answered from these sentences? Furthermore, what if relevant data from images can be

1 INTRODUCTION
Researchers have long considered the application of neural nets to data management problems, including learning indices [16], query optimization, data cleaning and entity matching [20, 23, 32]. In applying neural networks to data management, research has so far assumed that the data was modeled by a database schema.

The success of neural networks in processing unstructured data such as natural language and images raises the question of whether their use can be extended to a point where we can relax the fundamental assumption of database management, which is that the data we process is represented as fields of a pre-defined schema. What if, instead, data and queries can be represented as short natural language sentences, and queries can be answered from these sentences? Furthermore, what if relevant data from images can be
Research Opportunities

Language Models

Data Management
Research Opportunities

Language Models → Interfaces → Text Data → Code Specialization → Data Management → Language Models
Research Opportunities

- Interfaces
- Text Data
- Code Specialization

Language Models

LMs as Operators
LMs as Data

Data Management
Conclusions
Conclusions

- Significant **progress** in natural language processing
  - **Transformer** Model
  - **Transfer** Learning
- Various **interfaces** and libraries
- New **applications** in data management