Efficient Use of LLMs for Data Preparation

Effy Xue LI

Date: 11/05/2026

NHR4CES Workshop

Who am I?

- PostDoctoral Researcher in TRL Lab, CWI Amsterdam, working on Agentic Data Science.
- I had background in Knowledge Graph Construction from textual data, and Data Wrangling with LLMs.
- Now I focus on how can we (effectively) integrate knowledge into Data Science Agents?

Efficient Use of LLMs for Data Preparation

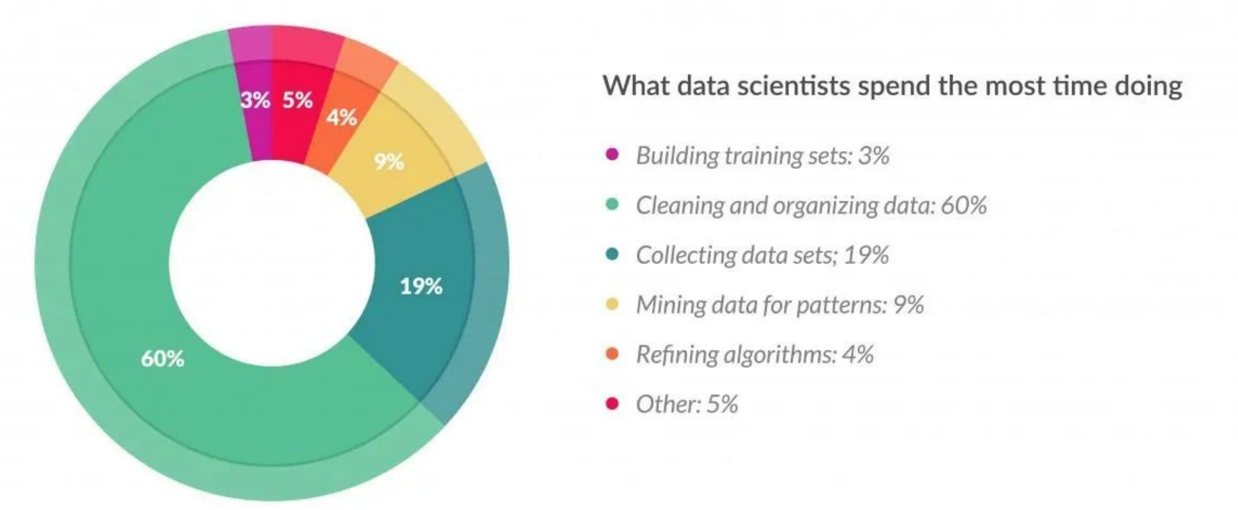
Effy Xue LI

Date: 11/05/2026

NHR4CES Workshop

Data Preparation is Time-consuming

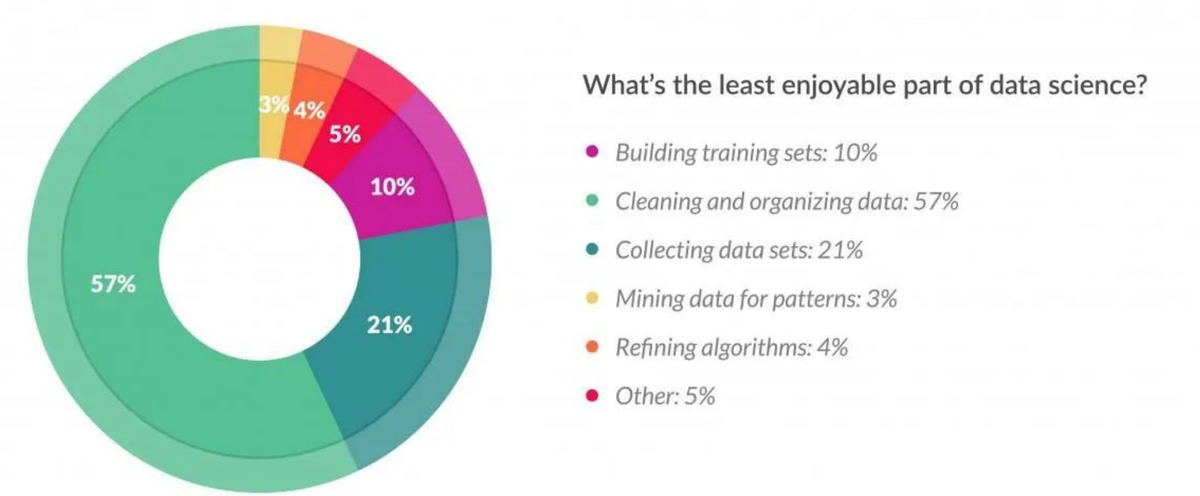
- **Data preparation accounts for about 80% of the work for data scientists.**



CrowdFlower. Data Science Report; CrowdFlower: San Francisco, CA, USA, 2016

Data Preparation is Unsatisfying 😪

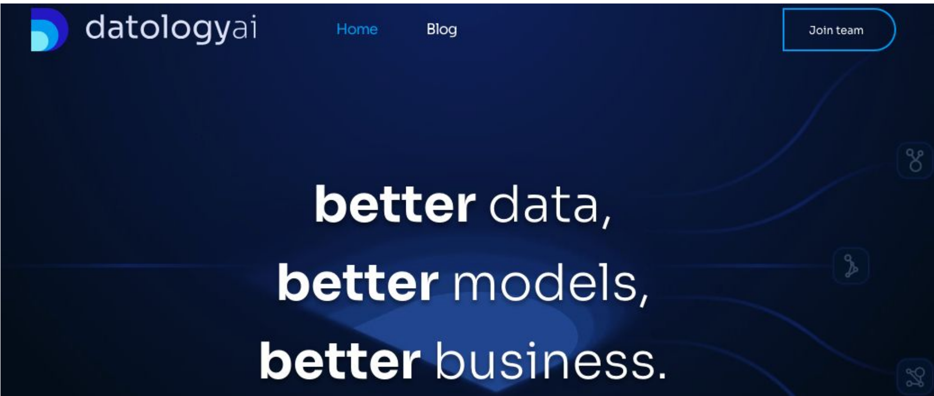
- 76% of data scientists view data preparation as their least enjoyable part of work.



CrowdFlower. Data Science Report; CrowdFlower: San Francisco, CA, USA, 2016

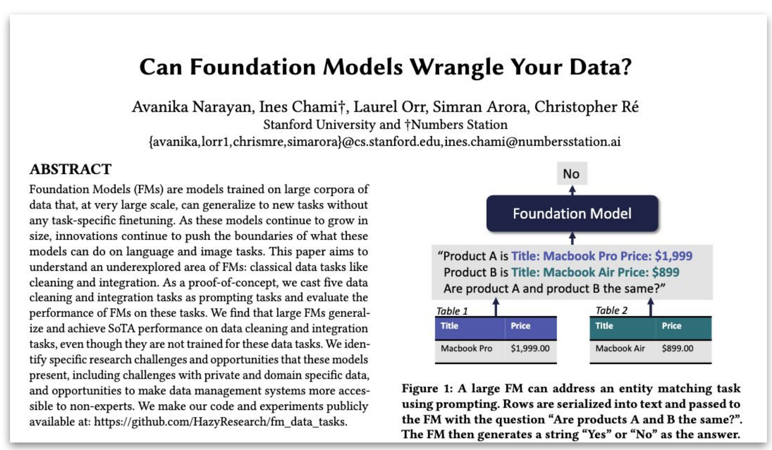
Many Companies are Building Platforms 🚀

Unstructured.io



datologyai

Many researchers are working on using LLMs for Data Prep



Can Foundation Models Wrangle Your Data?

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

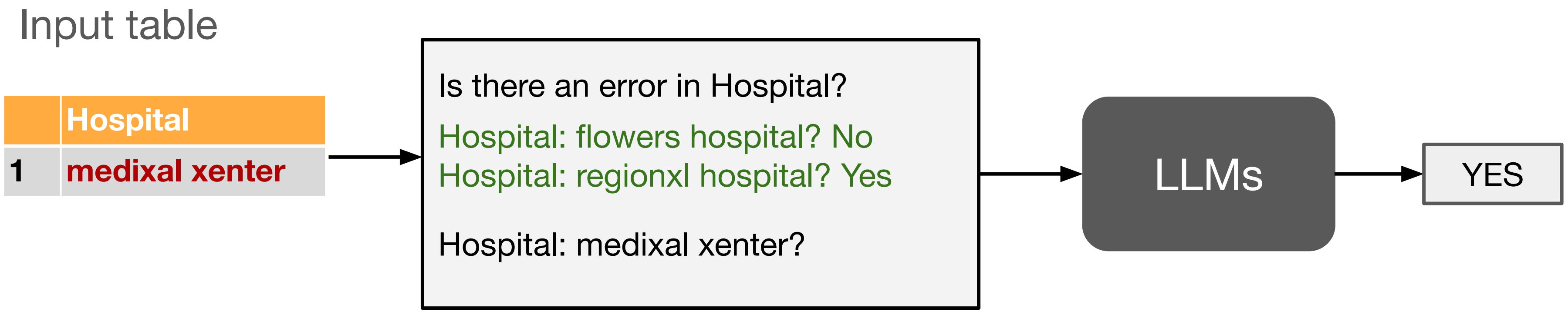
ABSTRACT

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 five 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 domain specific data, and opportunities to make data management systems more accessible to non-experts. We make our code and experiments publicly available at: https://github.com/HazyResearch/fm_data_tasks.

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.

- Prompting based data wrangling.

Data cleaning pipelines powered by LLMs



It performs LLM on a **per-row** basis (LLMPR).

Can Foundation Models Help Wrangle Data? Narayan et al. VLDB 23.

Challenges

- Expensive when doing back and forth wrangling.
    - Both time and money

- Transparency, privacy, reproducibility …

Many researchers are working on using LLMs for Data Prep

Can Foundation Models Wrangle Your Data?

Avanika Narayan, Ines Chami†, Laurel Orr, Simran Arora, Christopher Ré
Stanford University and †Numbers Station
{avanika,lorr1,chrismre,simarora}@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 five 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 domain specific data, and opportunities to make data management systems more accessible to non-experts. We make our code and experiments publicly available at: https://github.com/HazyResearch/fm_data_tasks.

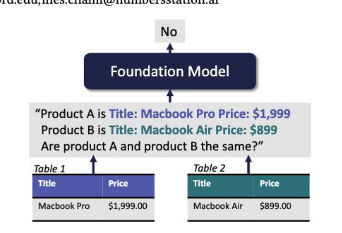


Figure 1: A large FM can address an entity matching task using prompting. Rows are serialized into text and passed to the FM with the question "Are products A and B the same?". The FM then generates a string "Yes" or "No" as the answer.

- Prompting based data wrangling.

Towards Parameter-Efficient Automation of Data Wrangling Tasks with Prefix-Tuning

David Vos
University of Amsterdam
d.j.a.vos@uva.nl

Till Döhmen
University of Amsterdam
t.r.dohmen@uva.nl

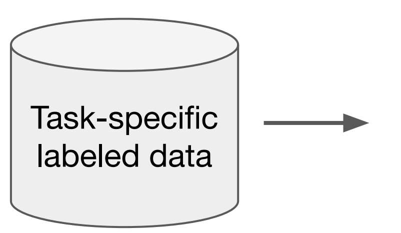
Sebastian Schelter
University of Amsterdam
s.schelter@uva.nl

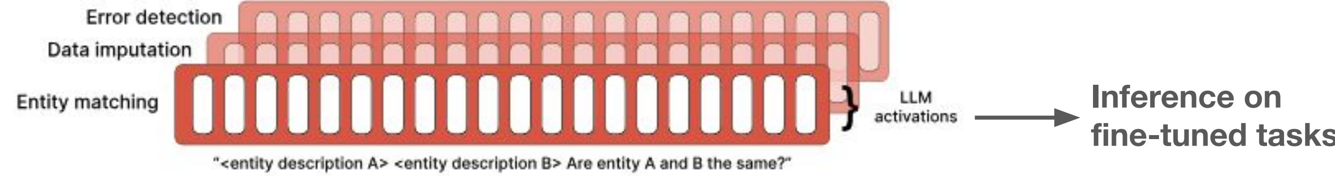
Abstract

Data wrangling tasks for data integration and cleaning arise in virtually every data-driven application scenario nowadays. Recent research indicated the astounding potential of Large Language Models (LLMs) for such tasks. However, the automation of data wrangling with LLMs poses additional challenges, as hand-tuning task- and data-specific prompts for LLMs requires high expertise and manual effort. On the other hand, finetuning a whole LLM is more amenable to automation, but incurs high storage costs, as a copy of the LLM has to be maintained. In this work, we explore the potential of a lightweight alternative to finetuning an LLM, which automatically learns a continuous prompt. This approach called prefix-tuning does not require updating the original LLM parameters, and can therefore re-use a single LLM instance across tasks. At the same time, it is amenable to automation, as continuous prompts can be automatically learned with standard techniques. We evaluate prefix-tuning on common data wrangling tasks for tabular data such as entity matching, error detection, and data imputation, with promising results. We find that in five out of ten cases, prefix-tuning is within 2.3% of the performance of finetuning, even though it leverages only 0.39% of the parameter updates required for finetuning the full model. These results highlight the potential of prefix-tuning as a parameter-efficient alternative to finetuning for data integration and data cleaning with LLMs.

- Fine-tuning based data wrangling.

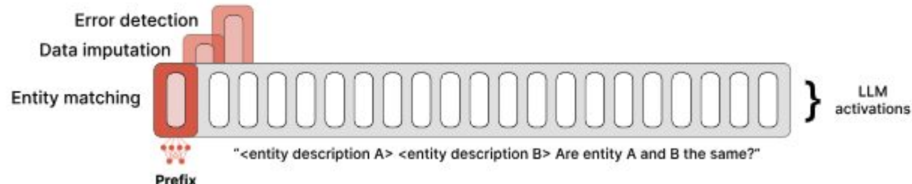
Fine-tuning based data wrangling





Inference on fine-tuned tasks

or



Towards Parameter-Efficient Automation of Data Wrangling Tasks with Prefix-Tuning. Vos et al. TRL'22.

Challenges

- Disadvantages of automatable alternatives such as fully fine-tuning a model per customer
  - High storage costs (for copies of model parameters)
  - High computational costs (for model training)

Methods Space

High zero-shot ability

(One model for many tasks)

Low costs

(Smaller models / less API calls)

High costs

(Bigger models / more API calls)

Low zero-shot ability

(Re-train for new task / domain)

Methods Space

Parameter-efficient / low compute methods

High zero-shot ability

LLMPR

Low costs

High costs

Fine-tuned models

Low zero-shot ability

New directions of efficient use of LLMs for Data Prep

In today's talk:

- TableSwift: Using LLM to generate code for data wrangling [1].
- AnyMatch: Fine-tuned small models preserving generalizability [2].

[1] Li et al., "TableSwift: Efficient Data Wrangling with Large Language Models using Code Generation", *under review*.

[2]Zhang et al., "AnyMatch -- Efficient Zero-Shot Entity Matching with a Small Language Model ", arxiv 2024

New directions of efficient use of LLMs for Data Prep

In today's talk:

- **TableSwift: Using LLM to generate code for data wrangling [1].**
- AnyMatch: Fine-tuned small models preserving generalizability [2].

[1] Xue Li, Till Doehmen, Jan-Christoph Kalo, Paul Groth, , "TableSwift: Efficient Data Wrangling with Large Language Models using Code Generation", ***under review*****.**

[2]Zhang et al., "AnyMatch -- Efficient Zero-Shot Entity Matching with a Small Language Model ", arxiv 2024

Data Wrangling tasks

	Phone	City
1	212/582-7200	

	Hospital	Error?
1	medixal xenter	

	Celsius	Fahrenheit
1	7	

	Beer_Name	BV
1	IJwit IPA	6.50 %

	Beer_Name	BV
1	IJwit IPA	0.50 %

same?

	Phone	City
1	212/582-7200	NYC

	Hospital	Error?
1	medixal xenter	Yes

	Celsius	Fahrenheit
1	7	44.6

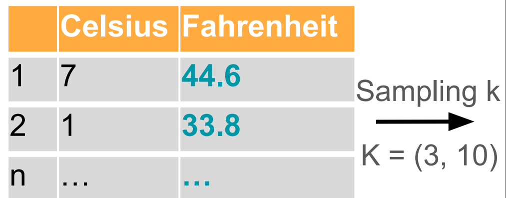
No

Data Imputation **Error Detection** **Data Transformation** **Entity Matching**

LLMPR is expensive, how can we utilize LLMs more strategically?

→ we use LLMs to generate code to wrangle your data.

Code Generation Workflow



Labelled Data

Sampling k

K = (3, 10)

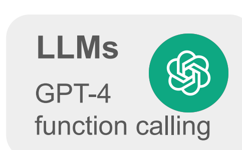
Generate code that's called "transform(input_str)" given instruction and input-output examples. Reason first and then generate.

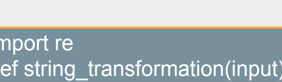
Instruction: Convert celsius to fahrenheit.

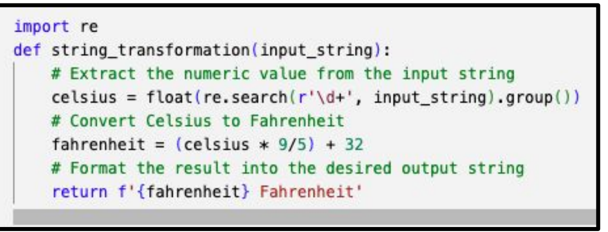
Examples: Input:7 Out: 44.6

Prompt Formulation

In-context Learning

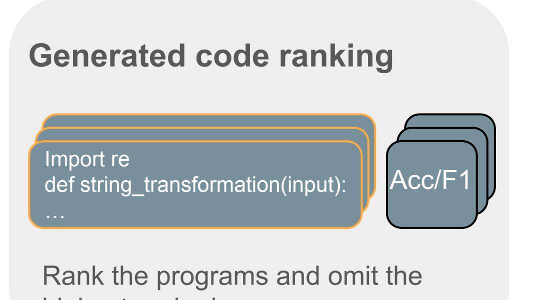






Generated code

N trials



Code

Code validation

Import re
def string_transformation(input):
…

1. Is_Executable?
2. Acc/F1 score > threshold on demonstration?

Code Generation Results

Entity Matching, Data Imputation, Error Detection

Task	Dataset	LLMPR[9]	Code Generation (Ours)
EM	Fodors-Zagats	100	95.5
EM	Beer	100	75.0
EM	DBLP-ACM	96.6	19.7
EM	DBLP-GoogleScholar	83.8	69.7
EM	Amazon-Google	63.5	42.1
EM	iTunes-Amazon	98.2	70.0
EM	Walmart-Amazon	87.0	25.5
DI	Buy	98.5	84.6
DI	Restaurant	88.4	50
ED	Hospital	97.8	23.5
ED	Adult	99.1	100*

76.9 ↓

- Evaluate based on rows.
- A single generated program can only solve a part of the dataset.

Data Transformation

Dataset	PBE [4]	LLMPR [9]	Code Generation (Ours)
BingQL-semantics	32.0	54.0	**91.6**
BingQL-Unit	**96.0**	N/A	95.0
Stack-overflow	63.0	65.3	**87.4**
FF-GR-Trifacta	**91.0**	N/A	83.7
Head cases	**82.0**	N/A	74.6
Average	72.8	N/A	**86.46**

37.6 ↑

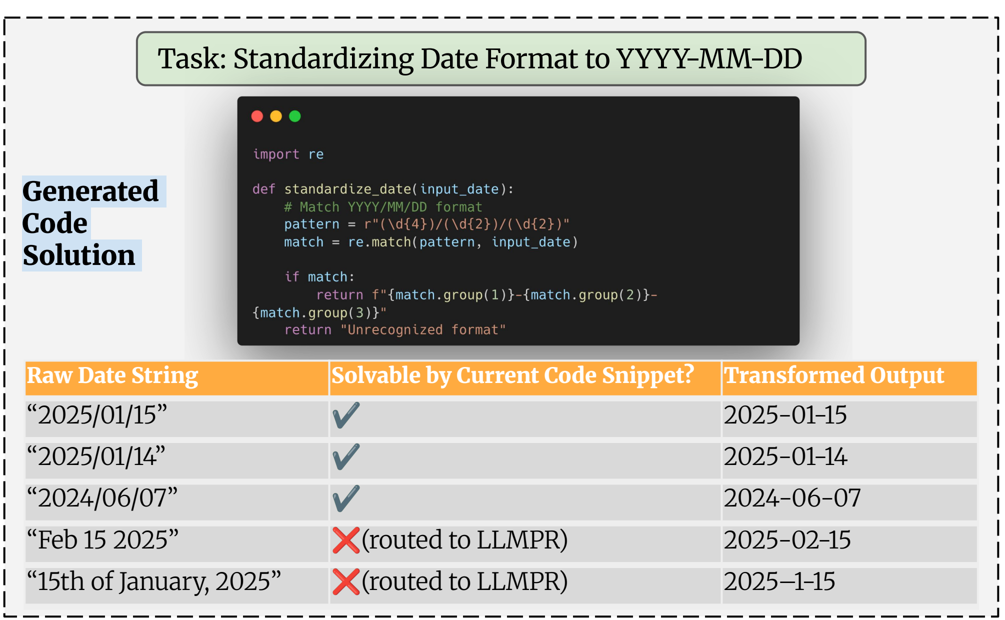
- Evaluate based on tasks.
- Improved performance for some datasets compare to the previous SOTA.

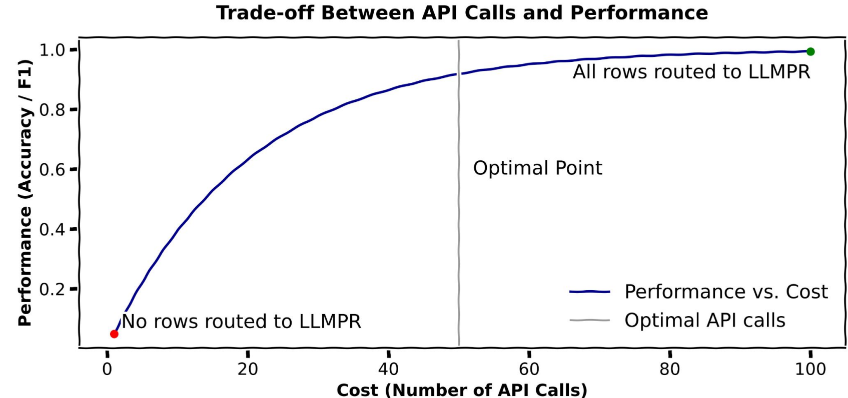
Code Generation Results with Different Models

Code	Methods	BingQL-semantics	BingQL-Unit	Stack-overflow	FF-GR-Trifacta	Head cases
N/A	PBE	32.0	**96.0**	63.0	**91.0**	**82.0**
	LLMPR	54.0	65.3	N/A	N/A	N/A
DuckDB-SQL	Llama3.2-3b	0.0	4.0	3.0	2.3	3.1
	Qwen2.5-coder-32b	67.3	**96.0**	61.0	59.4	50.4
	Llama3.1-405b	64.3	93.0	60.6	62.1	55.3
	GPT-4	73.3	93.0	63.2	69.8	66.7
Python	Llama3.2-3b	28.0	51.0	26.3	10.7	13.6
	Qwen2.5-coder-32b	88.0	92.0	73.0	67.8	50.1
	Llama3.1-405b	82.3	94.0	86.1	79.7	77.2
	GPT-4	**91.6**	95.0	**87.4**	83.7	74.6

- Bigger models have better results,
- but the relationship is not linear.
- Python code generation works better.

What is failing?





What is failing?

Task: Standardizing Date Format to YYYY-MM-DD

Generated Code Solution

```
import re

def standardize_date(input_date):
    # Match YYYY/MM/DD format
    pattern = r"(\d{4})/(\d{2})/(\d{2})"
    match = re.match(pattern, input_date)

    if match:
        return f"{match.group(1)}-{match.group(2)}-{match.group(3)}"
    return "Unrecognized format"
```

Raw Date String	Solvable by Current Code Snippet	Transformed Output
"2025/01/15"	✓	2025-01-15
"2025/01/14"	✓	2025-01-14
"2024/06/07"	✓	2024-06-07
"Feb 15 2025"	✗ (routed to LLMPR)	2025-02-15
"15th of January, 2025"	✗ (routed to LLMPR)	2025–1-15

Trade-off Between API Calls and Performance

A hybrid method that includes a dynamic routing mechanism will help to balance cost and accuracy.

Overview of TableSwift

Overview of TableSwift

Task

Code Generation Framework

Data

Eval data
with labels

Data
Router

Solvable
Rows

Task solving
Code

Transformed
Data

Not solvable
Rows

Fallback/LLMPR

Transformed
Data

How does TableSwift perform?

Data Transformation

Methods	BingQL-semantics	BingQL-Unit	Stack-overflow	FF-GR-Trifacta	Head cases	Average
PBE	32.0	96.0	63.0	**91.0**	82.0	72.8
LLMPR	54.0	65.3	N/A	N/A	N/A	59.65
CG-DDBSQL	73.3	93.0	63.2	69.8	66.7	73.2
TS-DDBSQL	72.0	93.0	73.8	64.0	67.0	73.96
# Rows Routed	5/102	2/99	598/710	24/83	26/90	-
CG-Python	**91.6**	95.0	87.4	83.7	74.6	86.46
TS-Python	90.3	**96.0**	**89.1**	83.6	**85.2**	**88.84**
# Rows Routed	2/102	4/99	584/710	16/83	19/90	-

- With the data router, TableSwift have achieved the highest average performance, by ~16 points.
- Data router improves performance in general.

How does TableSwift perform?

ED, DI and EM

	Error Detection		Data Imputation		Entity Matching						
Methods	Adult	Hospital	Buy	Restaurant	Amazon-Google	Beer	DBLP-ACM	DBLP-Google	Fodors-Zagats	iTunes-Amazon	Walmart-Amazon
GPT-3	99.1	**97.8**	98.5	88.4	63.5	**100**	96.6	83.8	**100**	98.2	87.0
GPT-3.5	92.0	90.7	98.5	94.2	66.5	96.3	94.9	76.1	**100**	96.4	86.2
GPT-4	92.0	90.7	**100**	**97.7**	**74.2**	**100**	**97.4**	**91.9**	**100**	**100**	**90.3**
GPT-4o	83.6	44.8	**100**	90.7	70.9	90.3	95.9	90.4	93.6	98.2	79.2
CG - Python	**100***	23.5	84.6	50	42.1	75.0	19.7	69.7	95.5	70	25.5
TableSwift - Python	**100***	28.6	87.7	72.1	29.6	90.3	91.3	71.6	95.7	85.1	64.6

- Data router improves the performance on almost all datasets.
- Still space for improvements on these tasks, comparing to LLMPR.

How does TableSwift perform in terms of costs?

LLMPR-DI
LLMPR-ED
TableSwift-DT
LLMPR-EM
TableSwift-DI
TableSwift-EM
TableSwift-ED
LLMPR-DT

Average Performance (acc or F1)

100 90 80 70 60 50

10^1 10^2 10^3 10^4 10^5 10^6

Average Costs (Number of API calls, log scale)

Empirical costs plot.

- Most of the time, TableSwift uses only a fraction of costs of LLMPR.
- For some tasks, TS hurts accuracy (but save costs).
- Presenting signals for task selection.

Conclusion

- TableSwift strategically utilizes LLMs, achieving comparable or even better performance while saving costs.
- On data transformation tasks, TableSwift achieves the new state-of-the-art.
- Our work presents exciting trade-off between efficiency and accuracy on tasks like Entity Matching.

Methods Space

High zero-shot ability

TableSwift

LLMPR

(Less API calls)

TableSwift saves costs by having less API calls, but it still uses big models. Is it possible to use smaller language models but still have high zero-shot ability?

Low costs

High costs

Fine-tuned models

Low zero-shot ability

New directions of efficient use of LLMs for Data Prep

In today's talk:

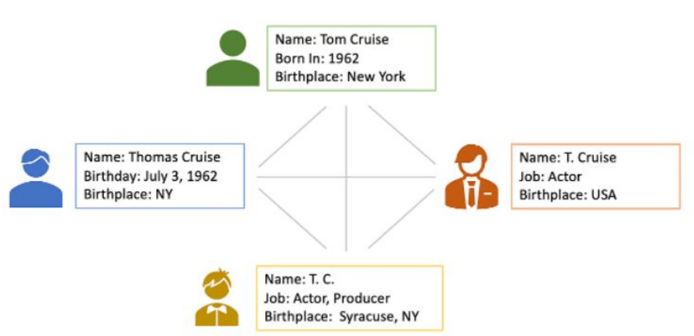
- TableSwift: Using LLM to generate code for data wrangling [1].
- ➛ **AnyMatch: Fine-tuned small models preserving generalizability [2].**

[1] Li et al., "TableSwift: Efficient Data Wrangling with Large Language Models using Code Generation", *under review*.

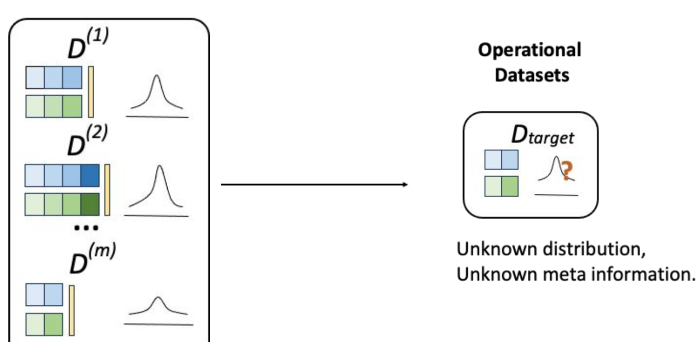
[2] Zeyu Zhang, Paul Groth, Iacer Calixto, Sebastian Schelter, "AnyMatch -- Efficient Zero-Shot Entity Matching with a Small Language Model ", arxiv 2024

Zero-Shot Entity Matching

- Entity Matching (EM) is a long-standing problem with many challenges:
  - Data heterogeneity
  - Ambiguity and context
  - Scalability issue



- Zero-Shot EM is more common in the wild but often neglected:
  - No meta-data provided
  - No prior knowledge on the target datasets
  - e.g. in the cloud setting



Zero-Shot Entity Matching with Large and Small Language Models

Labeled Datasets for Transfer Learning

$D^{(1)}_{transfer}$

$D^{(2)}_{transfer}$

…

$D^{(m)}_{transfer}$

AutoML Filter for record-level examples

Control of **label imbalance**

pos

neg

Misclassifications

Random Samples

Serialisation

Record A is <p>COL …, …, COL …</p>. Record B is <p>COL …, …, COL …</p>. Given the attributes of the two records, are they the same?

Data Augmentation with attribute-level examples

Serialisation

Record A is <p>COL …</p>. Record B is <p>COL …</p>. Given the attributes of the two records, are they the same?

Fine-Tuning

Sequence Classification

Feed Forward

Masked Self-attention

Text & Position Embed

12x

AnyMatch (GPT-2)

$\hat{y} \in \{0,1\}$

Sequence Classification

Feed Forward

Masked Self-attention

Text & Position Embed

12x

AnyMatch (GPT-2)

Inference on **Unseen Data**

Unseen target data

D_{target}

?

Record A is <p>COL …, …, COL …</p>. Record B is <p>COL …, …, COL …</p>. Given the attributes of the two records, are they the same?

Serialisation

Legend: r_l r_r Record pair from left and right input; y Ground truth label for record pair; $r_l[a]$ $r_r[a]$ Attribute pair generated from left and right record; y' Weak label for attribute pair

To achieve high zero-shot ability, diversifying data samples are more important than having more data samples for small LMs.

Zhang et al., “AnyMatch -- Efficient Zero-Shot Entity Matching with a Small Language Model ”, arxiv 2024

Evaluation Framework

- Nine Tabular Datasets across two open Benchmarks included
  - Various domains
  - Different number of attributes
  - Dataset size ranging from hundreds to tens of thousands

	Dataset	Domain	#Attr.	Size
ABT	Abt-Buy	web products	3	8,865
AMGO	Amazon-Google	software	3	11,460
BEER	Beer	food	4	450
DBAC	DBLP-ACM	citation	4	12,363
DBGO	DBLP-Google	citation	4	28,707
FOZA	Fodors-Zagats	food	6	946
ITAM	iTunes-Amazon	music	8	539
WAAM	Walmart-Amazon	electronics	5	10,242
WDC	Web Data Commons	web products	3	6,239

- Measure both the predictive quality and inference cost
  - Predictive quality -- F1 score on the test set, test size down-sampled due to the high cost of prompting large language models
  - Inference cost – Use the API cost or hardware rent cost as a proxy to evaluate

Zhang et al., “AnyMatch -- Efficient Zero-Shot Entity Matching with a Small Language Model ”, arxiv 2024

Evaluation Framework

- Leave-One-Dataset-Out evaluation

All Datasets

ABT train	AMGO train	...	WAAM train	WDC train
ABT valid	AMGO valid	...	WAAM valid	WDC valid
ABT test	AMGO test	...	WAAM test	WDC test

"Leave ABT Dataset Out"

Evaluate on ABT

ABT train | AMGO train | ... | WAAM train | WDC train

ABT valid | AMGO valid | ... | WAAM valid | WDC valid

Fine-tune PLM / Prompt LLM

ABT test | AMGO test | ... | WAAM test | WDC test

Results

Training-based methods

Prompting-based methods

Model	#params (millions)	Mean
StringSim	-	40.21
ZeroER	-	66.86
Ditto	110	66.05
AnyMatch [GPT-2]	124	81.96
Unicorn	143	77.54
AnyMatch [T5]	220	79.57
AnyMatch [Llama3.2]	1,300	**87.06**
Jellyfish	13,000	(77.83)
MatchGPT [GPT-4o-mini]	8,000	81.01
MatchGPT [Mixtral-8x7B]	56,000	64.26
MatchGPT [SOLAR]	70,000	65.37
MatchGPT [Beluga2]	70,000	69.10
MatchGPT [GPT-3.5-Turbo]	175,000	69.13
MatchGPT [GPT-4]	1,760,000	85.82

Fine-tuned small language models can achieve on-par or even better performance than prompted large language models.

Inference costs

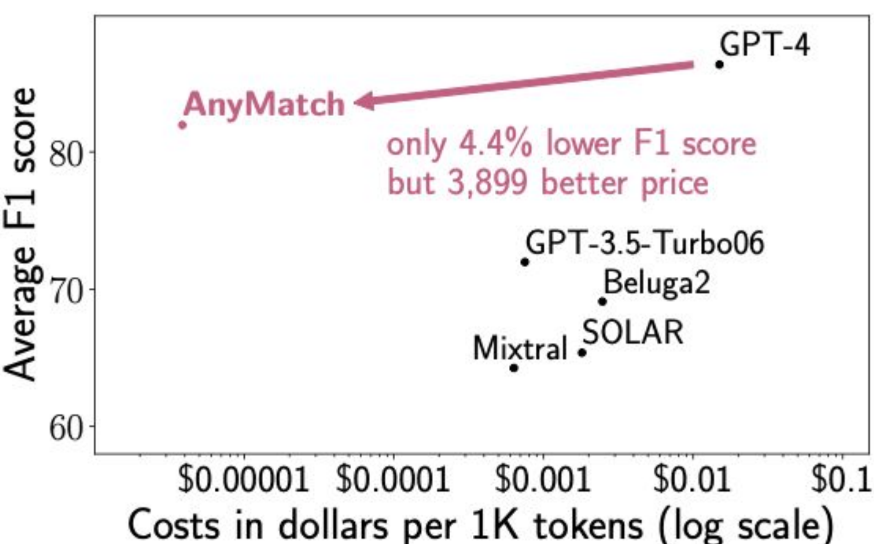
Compute throughput in tokens/s on a machine with 4xA100 (40GB) GPUs for different open-weight LLMs employed by various EM approaches

Model	Used by	#params (millions)	RAM (GB)	batch size	Throughput (tokens/s)
BERT	Ditto	110	0.21	8,192	**862,001**
GPT-2	AnyMatch	124	0.26	8,192	693,999
DeBERTa	Unicorn	143	0.27	4,096	216,396
T5	AnyMatch	220	0.54	8,192	530,656
Llama3.2	AnyMatch	1,300	2.30	4,096	264,952
Llama2-13B	Jellyfish	13,000	24.46	128	26,721
Mixtral-8x7B	MatchGPT	56,000	73.73	32	2,108
Beluga2	MatchGPT	70,000	128.64	32	1,079
SOLAR	MatchGPT	70,000	128.64	64	752

Inference costs

Estimate the deployment cost per 1K tokens for EM with proprietary models and open-sourced models

Method & model	Cost for 1K tokens	Deployment scenario
MatchGPT [GPT-4]	$0.015	OpenAI Batch API
MatchGPT [SOLAR]	$0.0009	Hosting on Together.ai
MatchGPT [Beluga2]	$0.0009	Hosting on Together.ai
MatchGPT [GPT-3.5-turbo-06]	$0.00075	OpenAI Batch API
MatchGPT [Mixtral-8x7B]	$0.00063	4x on p4d.24xlarge
MatchGPT [GPT-4o-mini]	$0.000075	OpenAI Batch API
Jellyfish	$0.000025	8x on p4d.24xlarge
Unicorn[DeBERTa]	$0.000012	8x on p4d.24xlarge
AnyMatch[Llama3.2]	$0.000010	8x on p4d.24xlarge
AnyMatch[T5]	$0.0000050	8x on p4d.24xlarge
AnyMatch[GPT-2]	$0.0000038	8x on p4d.24xlarge
Ditto[Bert]	$0.0000031	8x on p4d.24xlarge



Methods Space

High zero-shot ability

AnyMatch

(Less API calls)

(Less model parameters)

LLMPR

Low costs

High costs

Fine-tuned models

Low zero-shot ability

Conclusions

- Using larger LLMs strategically can help balance cost performance trade-offs (TableSwift) .
- Specialized smaller models fine-tuned with diverse samples can achieve good zero-shot ability on the task (AnyMatch).

Lessons learned and future research

- Knowledge-enriched data preparation.
    - Domain-specific vocabulary.
    - Internal definitions of columns and relations between columns.
    - …
- Better table representations (than just putting them in a sequence as contexts).