LARGE LANGUAGE MODEL FOR ENTITY RESOLUTION PROBLEMS

Davood Sheibani

Abstract

Foundation Models (FMs), such as GPT-3.5, have demonstrated remarkable capabilities in natural language understanding and image processing, often achieving state-of-the-art results across a wide range of tasks without task-specific fine-tuning. While the potential of these models in language and image tasks is well-established, their application in classical data tasks, like data cleaning and integration, remains an underexplored area. This thesis aims to investigate the utilization of large FMs for entity matching, a critical component of data cleaning and integration.

In this proof-of-concept study, we frame five data cleaning and integration tasks as prompting tasks and assess the performance of FMs in these domains. Surprisingly, we discover that large FMs exhibit exceptional generalization abilities, achieving state-of-the-art performance on data cleaning and integration tasks, despite not being explicitly trained for these specific tasks. This finding suggests that these models hold immense potential for enhancing data management processes by automating and improving entity matching tasks.

We further identify several research challenges and opportunities that arise from the integration of FMs into data management systems. Challenges include dealing with private and domain-specific data, ensuring data privacy, and addressing the need for model explainability and interpretability. On the other hand, opportunities include democratizing data management by making it more accessible to non-experts through user-friendly interfaces and automated data cleaning and integration pipelines.

This research sheds light on the transformative potential of Foundation Models in the realm of entity matching and data management. By harnessing the generalization capabilities of these models, organizations can streamline their data cleaning and integration processes, leading to improved data quality and more informed decision-making. Moreover, this work paves the way for future innovations in data management systems, making them more adaptable, efficient, and user-friendly, ultimately benefiting a wide range of industries that rely on accurate and integrated data.

Acknowledgements

I would like to express my deepest gratitude to my esteemed professor, Paolo Merialdo , for their unwavering support and invaluable guidance throughout the completion of this thesis. Their mentorship and expertise have been instrumental in shaping my research and academic development.

Davood sheibani

September 2023

Contents

1 INTRODUCTION

Large Language Models (LLMs) represent a remarkable advancement in natural language processing and artificial intelligence research [1]. These models have significantly enhanced the capabilities of machines to understand and generate human-like language [2]. By utilizing deep learning techniques and vast datasets, LLMs have demonstrated their proficiency in various language-related tasks, including text generation, translation, summarization, question answering, and sentiment analysis. The history of LLMs can be traced back to the early development of language models and neural networks. Early efforts to build language models were based on statistical methods and n-gram models [3]. However, these models had limitations in capturing long-range dependencies and context in language. With the advent of neural networks and the availability of larger datasets, researchers began to explore more sophisticated approaches. One significant milestone was the development of the Recurrent Neural Network (RNN) [1], [4], which allowed for modeling sequential data, including language. However, RNNs also faced challenges with vanishing gradients and long-term dependencies, limiting their effectiveness.

The breakthrough in LLMs came with the introduction of the Transformer architecture in the seminal work ”Attention is All You Need” by Vaswani et al. in 2017 [5]. The Transformer model, based on the self-attention mechanism, enabled parallelization and efficient handling of long-range dependencies. It laid the foundation for models like OpenAI’s GPT (Generative Pre-trained Transformer) series and BERT (Bidirectional Encoder Representations from Transformers)[6] by Google, which achieved groundbreaking results in a wide range of language tasks.

Since then, LLMs have undergone several developmental stages, with models increasing in size and complexity. The GPT series, starting with GPT-1 and continuing with GPT-2 and GPT-3, has successively grown in the number of parameters, allowing for more sophisticated language understanding and generation capabilities. Likewise, BERT-inspired models have seen advancements in pre-training strategies, such as ALBERT [7] (A Lite BERT) and RoBERTa [8], which further improved performance and efficiency.

Furthermore, advancements in LLMs have extended to more specific domains, with models designed for specialized tasks like medical language processing, scientific research, and code generation. Moreover, efforts have been made to address ethical concerns, interpretability, and reducing biases in LLMs to ensure responsible and equitable use. The evolution of Large Language Models has revolutionized natural language processing and AI research, leading to remarkable achievements in various language-related tasks. The development stages of these models have witnessed a constant quest for larger models, improved pre-training strategies, and specialized domain adaptations. As research continues, the potential applications and impact of LLMs on various fields, including education, healthcare, and human-computer interaction, continue to expand, inspiring further innovations and advancements.

TABLE I: List of Acronyms and corresponding definitions.

Acronym	Definition
AI	Artificial Intelligence
AGI	Artificial General Intelligence
BERT	Bidirectional Encoder Representations from Transformers
CV	Computer Vision
CTRL	Conditional Transformer Language Model
FFF	Fused Filament Fabrication
GANs	Generative Adversarial Networks
GNMT	Google Neural Machine Translation
GPT	Generative Pre-Trained transformers
GPT-3	Generative Pre-trained Transformer 3
GPT-4	Generative Pre-trained Transformer 4
GPUs	Graphical Processing Units
GRUs	Gated Recurrent Units
LLaMA	Large Language Model Meta AI
LLM	Large Language Models
LM	Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MLM	Masked Language Modeling
NSP	next-sentence prediction
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PLMs	Pre-trained Language Models
RNN	Recurrent neural networks
RNNLM	Recurrent neural network language model
SLMs	Statistical Language Models
T5	Text-to-Text Transfer Transformer
TPUs	Tensor Processing Units
USMLE	United States Medical Licensing Exam
VL-PTMs	Vision-Language Pre-trained Models
XLNet	eXtreme Language Understanding Network

In summary and as can be seen from Fig 1, language modeling (LM) research has received widespread attention and has undergone four significant development stages including: statistical language models, neural language models, pre- trained language models and large language models[9]. this research mainly focus on LLMs. In Table II, i present sources of data for pre-training GPT 3.5. Modern language model called ChatGPT was created by OpenAI. It is based on the GPT-3.5 architecture and wastrained using a sizable amount of internet-sourced text data, including books, articles, wikis and websites .

ChatGPT is exceptional at producing human-like responses and having conversations with users. In computer vision (CV), researchers are actively engaged in the development of vision-language models inspired by the capabilities of ChatGPT. These models are specifically designed to enhance multimodal dialogues, where both visual and textual information are important [10]. Moreover, the advancements in the field have led to the introduction of GPT-4 [11], which has further expanded the capabilities of language models by seamlessly integrating visual information as part of the input. This integration of visual data empowers the model to effectively understand and generate responses that incorporate both textual and visual cues, enabling more contextually rich and nuanced conversations in multimodal settings.

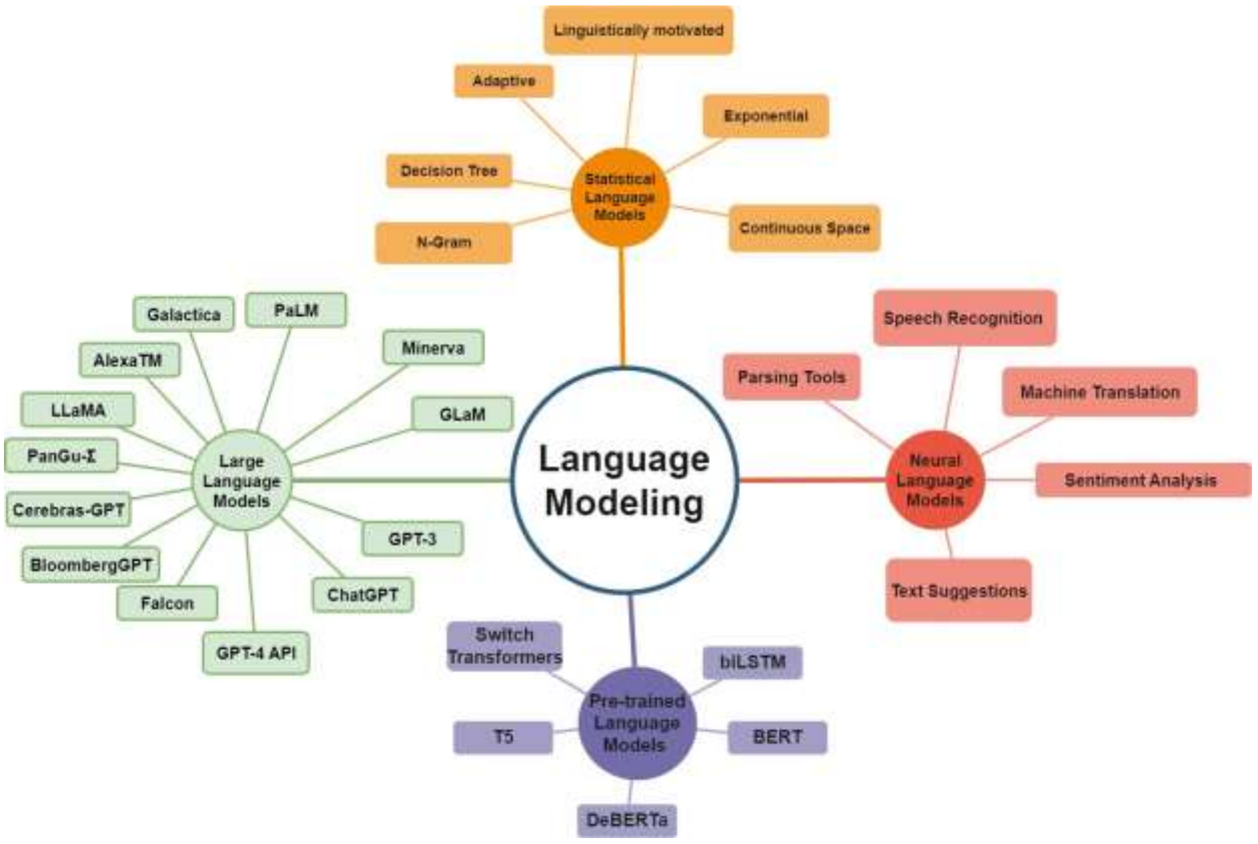


Figure I: Types of language modeling. The division of LLMs is categorized into four major blocks

TABLE II: Data Mixtures for Pre-training GPT-3.5 (Percentage of Total)

Data Source	Description	Percentage of Total
Common Crawl	Web text collected from publicly available websites and pages.	56.05%
Books1	A large corpus of books and literature texts.	3.93%
Books2	Additional books and literary texts.	2.95%
Wikipedia	Text extracted from various language editions of Wikipedia.	1.47%
News Websites	News articles from diverse online news sources.	0.98%
Scientific Papers	Academic papers and research articles.	0.49%
Social Media Texts	Textual data from social media platforms.	0.79%
Conversational Data	Chat logs and conversations from the internet.	0.69%
Encyclopedias	Content from online encyclopedias and reference sources.	0.30%
Total		**100%**

Most used LLMs

Despite the recent attention and excitement surrounding Large Language Models, their development is not new in the field of AI. OpenAI introduced the first GPT-model in 2018, followed by GPT-2 in 2019. Even back in 2019, ML6 was already working on leveraging LLMs for business use. For example, we fine tuned the pre-trained English GPT-2 model to Dutch. Thanks to the exponential growth of LLM capabilities, LLMs have recently gained significant momentum, leading to the emergence of numerous new models, with sometimes weekly releases.

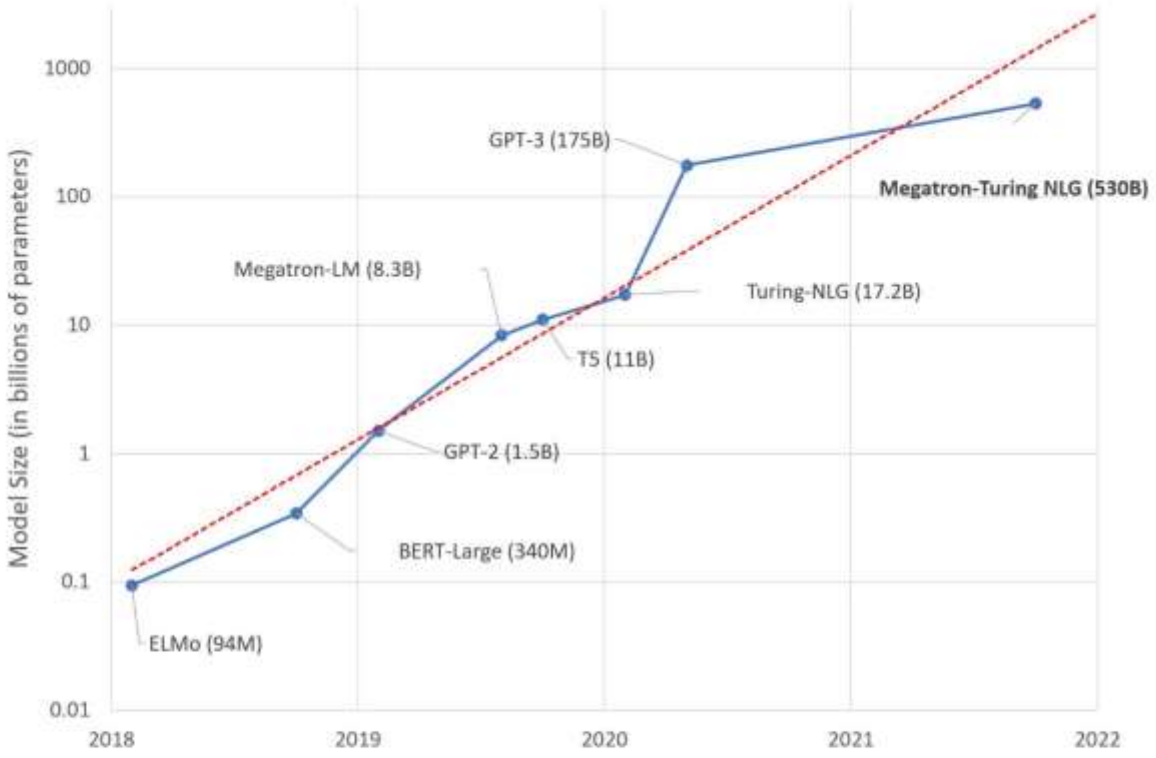


Figure II : Most used LLMs

Some of the most widely used LLMs that are currently making waves in the field of AI, include:

GPT-models

A family of LLMs introduced by OpenAI. With the recent releases of ChatGPT and GPT-4, GPT-models have drawn a lot of interest from the AI community. Those models have a proprietary licence, which is a non-open source licence that requires users to pay a fee and impose some usage restrictions.

LLaMA

An open-source collection of LLMs developed by Meta. LLaMA is designed to help researchers in advancing their work in the subfield of LLMs. It is available in multiple sizes, ranging from 7 to 65 billion parameters, and aims to democratise the access to LLMs by requiring less computing power and resources. LLaMA can only be used for research.

PaLM-2

Next-generation Large Language Model developed by Google. PaLM-2 is built upon Google's previous research in AI and has been announced during their annual I/O keynote in May 2023. Google also released a paper introducing Med-PaLM-2, a fine-tuned version on medical data.

These large FMs are often autoregressive language models (e.g. GPT-3 and PaLM) that are trained to predict the next word in large text corpora and can be adapted to new tasks given a simple natural language description of the task .These breakthrough capabilities have led to a race for building bigger and better models, and innovations continue to push the boundaries of what large FMs can do on a variety of hard language tasks

1.1 Survey Motivation

The revolutionary ChatGPT has captivated the attentionof the community, sparking a wealth of fascinating reviews and discussions on the advancements of LLMs and artificial intelligence [9], [12], [13], [14].A natural question that arises is whether these advances can benefit hard classical data tasks (e.g. data cleaning and integration). While it is clear that FMs benefit text-intensive tasks, it is not clear whether these models can be applied to data tasks over structured data. The symbols commonly found in structured data (e.g. dates, numbers, alphanumeric codes) are less frequent in natural language text so it is unclear that FMs possess the ability to reason over them. Moreover, since FMs are trained to predict the next word, it is nonobvious that they can work out-of-the-box on complex data tasks. This paper explores the aforementioned question and introduces a new research vision for leveraging FMs for data management, focusing on data cleaning and integration tasks—two keys steps in data-driven enterprise pipelines. Recently, a large body of research has applied machine learning (ML) [49] and deep learning (DL) [57, 74] methods—namely pre-trained language models (PLMs) like BERT [32]—to semantically complex data tasks. However, these approaches still require a significant amount of engineering effort as they rely on:

Task-specific architectures: Data cleaning and integration encapsulate many different tasks such as entity matching [77], schema matching [92], and error detection [41]. Existing approaches, whether they are rule-, ML- or DL-based, vary greatly from one task to the other, often with complex, task-specific architectures. For instance, adapting BERT to data tasks requires architectural changes and finetuning the entire model for each task. This leads to siloed and hard-to-maintain systems.

Hard-coded knowledge: Data tasks often rely on domain knowledge (e.g. understanding the relationship between a city and its zip code for data cleaning constraints) and commonsense reasoning. These are usually hard-coded with human-engineered rules or external knowledge bases [24, 84]. Consequently, systems can be brittle and fail to generalize to a diverse set of domains.

Labeled data: ML- and DL-based solutions require copious amounts of hand-labeled data [9]. For instance, PLMs that have achieved state-of-the-art (SoTA) results on data tasks (e.g. Ditto [38]) require a significant amount of task-specific labeled data and finetuning to achieve good performance. Labeling data for each task is engineering intensive and adds to the difficulty of maintaining data cleaning and integration systems. Excitingly, FMs display several useful properties that make them an appealing choice compared to traditional approaches:

Task-agnostic architecture: As a result of their natural language interface, FMs can be applied to a wide-range of tasks. For instance, Figure 1 shows how an entity matching task—which requires identifying whether two table entries refer to the same entity—can be cast as a prompting task. This unifying interface eliminates the need for siloed architectures, in contrast to existing learned approaches where architectures need to be carefully crafted for each task (e.g. task-specific classification layer).

Encoded knowledge: Because FMs are trained on large, generic corpora of data, they contain knowledge about an extensive set of common entities, and thus do not rely on human-engineered rules to acquire knowledge [82].

Limited to no labeled data: FMs can be applied to a breadth of tasks with little to no labeled data (e.g. few-shot and zero-shot). When a FM needs to be fine-tuned, it typically needs dramatically less labeled data to achieve competitive results [46].

Our goal is to better understand if large FMs can be applied to data integration and cleaning tasks. We study the behavior of GPT-3—an early and promising FM. While GPT-3 is already a high quality model, we expect the significant investment in FMs from both academia and industry to lead to more performant and scalable FMs over time. Like many other communities, the data management community stands to benefit from these trends. As such, we aim to understand the advantages and limitations of FMs on data tasks, by focusing on three key questions

How well do large FMs transfer to data tasks? To answer this, we cast several data tasks as natural language generation tasks (Section 3) and explore whether a single FM can generalize well to these tasks. In Section 4.2, we quantify the zero- and few-shot performance of FMs on five enterprise data tasks: entity matching, error detection, schema matching, data transformation, and data imputation. We find that the largest GPT-3 variant (175B parameters) outperforms SoTA ML-and DL-based approaches on these tasks with few examples. This is particularly surprising since prior approaches are fully-finetuned on task-specific labeled data for these tasks, while GPT-3-175B is simply pretrained to generate text.

What are the caveats in applying FMs to data tasks? In Section 4.3, we unpack the few-shot "prompt tuning" process—serializing tabular data to text, casting data tasks as text generation tasks and constructing demonstrative task examples—for applying FMs to data tasks. We quantify the effects of prompt formatting variations on performance and the differences between manually and randomly selecting task examples. We find that FMs are brittle to differences in prompt formatting and that performance improves when prompts are manually selected versus randomly selected.

What opportunities do FMs present for data tasks and what are the relevant research challenges? Finally, in Section 5, we discuss the potential challenges and related research questions with using FMs in data management pipelines. We discuss the forthcoming shift in how ML systems are built, challenges around updating FM knowledge, and opportunities and considerations pertaining to private, temporal and local data. We hope that our preliminary exploration will encourage the data management community to explore the effectiveness of FMs for other data tasks and develop techniques to overcome the shortcomings of FMs in this setting.

Despite the growing number of studies on LLMs, there remains a scarcity of research focusing on their technical intricacies and effective utilization. In this review and tutorial article, our primary objective is to explore, learn, and evaluatelanguage models across various domains. We delve into the working principles of language models, analyze different architectures of the GPT family, and discuss strategies for theiroptimal utilization. Furthermore, we provide detailed insights into writing prompts. Our comprehensive examination also encompasses adiscussion on the limitations associated with LLMs, including considerations related to security, ethics, economy, and the environment. In addition, we present a set of guidelines to steer future research and development in the effective use of LLMs. We hope that this research will contribute to a better understanding and utilization of LLMs. A list of commonly

1.2 Contributions

This paper makes several significant contributions to the field of Generative AI and the application of Large Language Models (LLMs) to data-related tasks. The key contributions are as follows:

1. **Comprehensive Exploration of Foundation Models:** This thesis provides an in-depth exploration of foundation models, particularly focusing on their history, training processes, and mechanisms. It aims to demystify the workings of LLMs, making them more accessible to researchers and practitioners.

2. Integration of LLMs with Data Tasks: We propose a novel approach for integrating LLMs into data tasks by treating data-related challenges as natural language tasks. This innovative framework expands the applicability of LLMs to a wide range of data manipulation, transformation, and integration tasks.

3. Task Demonstrations: In the "Foundation Models for Data Tasks" section, we introduce the concept of task demonstrations as a means to help LLMs learn and perform data-related tasks effectively. This approach enhances the adaptability and usability of LLMs in practical data scenarios.

4. Experimental Insights: The "Experiments" section presents a rigorous evaluation of LLMs in the context of data tasks. We provide insights into their performance, strengths, and limitations, aiding researchers and practitioners in making informed decisions when applying LLMs to real-world data challenges.

5. Research Agenda for LLMs: In the "Research Agenda" section, we outline opportunities, practical considerations, and technical challenges associated with the use of LLMs. This forward-looking perspective guides future research directions and encourages the development of data orchestration workbenches centered around LLMs.

These contributions collectively advance the understanding and application of Large Language Models in the realm of data manipulation and integration, paving the way for more efficient and innovative solutions to data-related challenges.

2 BACKGROUND

The paper first gives some background on the different data tasks considered in this paper and then provide a brief review of FMs.

2.1 Problem Setup

The paper focus on entity matching (EM), error detection (ED), and data imputation (DI) and describe the setup for these tasks. We denote D, a structured dataset with n entries, such that each entry is represented by a collection of$m$ attribute value pairs: for entry $ei \in D$ we have

ei = {ei,j }1≤j≤m where for attribute j, ei,j = {attr$j$, val$j$ }.

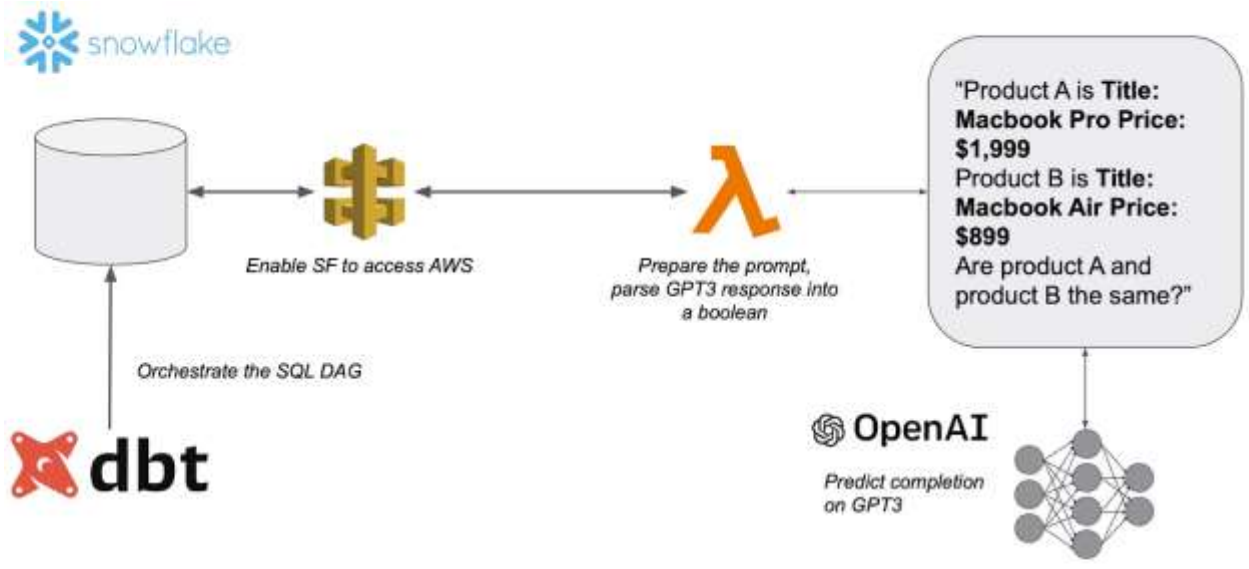

snowflake
Enable SF to access AWS
Prepare the prompt, parse GPT3 response into a boolean
"Product A is Title: Macbook Pro Price: $1,999
Product B is Title: Macbook Air Price: $899
Are product A and product B the same?"
Orchestrate the SQL DAG
dbt
OpenAI
Predict completion on GPT3

Figure III : The functional workflow for the proposed implementation: the MDS on the left, and the intelligence supplied by GPT-3 through an API call

Entity Matching (EM):

Introduction: Entity Matching, also known as Entity Resolution or Record Linkage, is a fundamental task in data management and integration. Its primary objective is to identify and match entities, which represent real-world objects like individuals, companies, products, or locations, across different datasets. EM plays a pivotal role in ensuring data quality, consistency, and accuracy in various applications, including data warehousing, master data management, customer data integration, and beyond.

Problem Statement: In EM, the central challenge is to determine whether pairs of entries from two structured datasets, often denoted as D and D', represent the same real-world entity. Formally, for each pair of entries $(e, e') \in D \times D'$, the goal is to predict whether these entries correspond to the same entity or not. This problem is typically approached as a binary classification task, where the objective is to classify pairs as matches (representing the same entity) or non-matches (representing different entities).

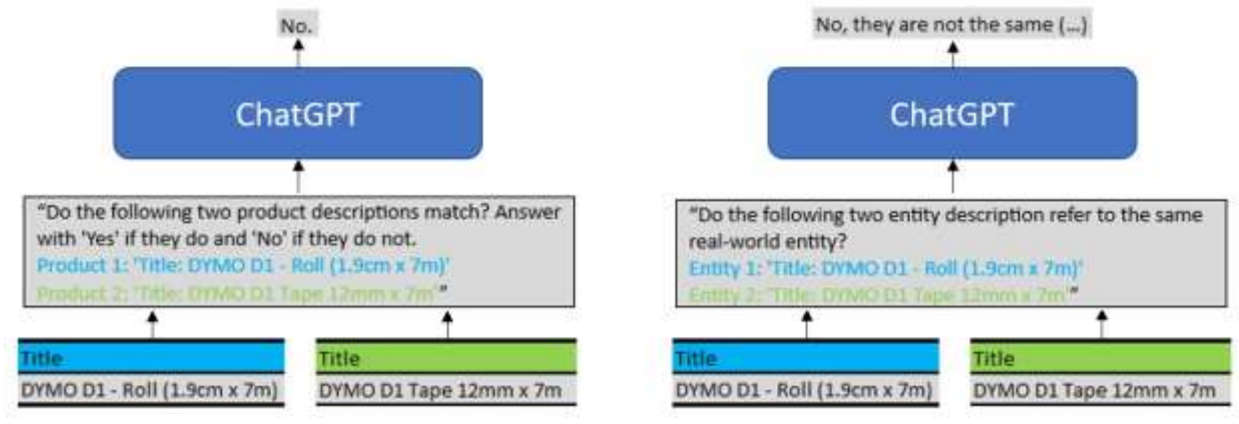

No.
ChatGPT
"Do the following two product descriptions match? Answer with 'Yes' if they do and 'No' if they do not.
Product 1: 'Title: DYMO D1 - Roll (1.9cm x 7m)'
Product 2: 'Title: DYMO D1 Tape 12mm x 7m'"
Title
DYMO D1 - Roll (1.9cm x 7m)
Title
DYMO D1 Tape 12mm x 7m
No, they are not the same (...)
ChatGPT
"Do the following two entity description refer to the same real-world entity?
Entity 1: 'Title: DYMO D1 - Roll (1.9cm x 7m)'
Entity 2: 'Title: DYMO D1 Tape 12mm x 7m'"
Title
DYMO D1 - Roll (1.9cm x 7m)
Title
DYMO D1 Tape 12mm x 7m

Figure IV : 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.

Key Components:

Structured Datasets (D, D'): EM typically involves two or more structured datasets that may vary in schema, format, and data quality. These datasets can originate from diverse sources, such as databases, spreadsheets, documents, or web sources.

Example EM Dataset of Amazon: a simplified example of what an EM dataset for Amazon could look like:

Blocking Heuristics: To handle the scalability challenge, real-world EM systems often employ blocking heuristics. Blocking reduces the number of pairwise comparisons by grouping potentially matching records together. Obvious non-matching pairs are excluded from further consideration, improving efficiency.

Matching Criteria: Defining matching criteria is essential in EM. These criteria determine whether two records are similar enough to be considered a match. Matching criteria can include exact matching, fuzzy matching, semantic matching, or context-based matching.

Categories of EM Methods: EM methods can be categorized into three main categories:

Rule-Based Methods: These methods rely on manually crafted rules and heuristics to define matching conditions. Rules may consider attributes like names, addresses, or identifiers to determine matches.

Crowd-Based Methods: Crowd-sourced workers or domain experts participate in making matching decisions. Human judgment helps resolve complex matching scenarios and can be integrated into the matching process.

ML/DL-Based Methods: Machine Learning (ML) and Deep Learning (DL) approaches have gained prominence in EM. These methods use algorithms to automatically learn patterns and features from the data to make matching decisions. Recent advancements, such as methods relying on Pre-trained Language Models (PLMs), have achieved state-of-the-art (SoTA) results for EM tasks.

State-of-the-Art with Pre-trained Language Models (PLMs): In recent years, methods relying on Pre-trained Language Models (PLMs), such as GPT-3, have achieved State-of-the-Art (SoTA) performance in EM tasks. PLMs are adept at understanding textual data, capturing semantic relationships, and generating context-aware matching criteria. Their ability to handle unstructured and semi-structured textual data makes them versatile tools for entity matching.

Research Landscape: Entity Matching has witnessed extensive research and development over the past decade, with numerous approaches, techniques, and algorithms proposed. Surveys, like the one referenced as [77], provide comprehensive overviews of the field, highlighting key advancements and trends in EM.

Applications: Entity Matching is integral to numerous applications, including:

Deduplication: Removing duplicate records from databases to ensure data consistency.

Customer Data Integration: Merging customer records from different sources to create a unified view.

Master Data Management: Maintaining high-quality master data records for enterprises.

Fraud Detection: Identifying fraudulent activities by linking related entities.

Conclusion: Entity Matching is a critical component of data integration, ensuring that data from disparate sources can be effectively unified and analyzed. It has evolved significantly over the years, with modern approaches leveraging the power of Pre-trained Language Models to achieve remarkable matching accuracy. As data continues to grow in complexity and volume, EM remains a dynamic and evolving field with exciting opportunities for research and application.

Error Detection (ED):

Introduction:

Error Detection, a fundamental component of data quality assurance, is a critical process in data management and data cleaning pipelines. Its primary goal is to identify and flag data errors or inaccuracies within a dataset, ensuring that the data used for analysis and decision-making is reliable and accurate. ED plays a pivotal role in various domains, including data warehousing, business intelligence, and data science.

Problem Statement:

In the context of Error Detection, the central challenge is to automatically identify attributes within a dataset that contain errors or inconsistencies. Formally, given an entry e in a dataset, the objective is to detect attributes j where the value (val$j$) is erroneous. This task is framed as a classification problem, where the goal is to predict whether val$j$ is correct or incorrect for a given e. Detecting and correcting errors early in the data processing pipeline is crucial for ensuring data accuracy downstream.

Key Components:

Structured Datasets: Error Detection typically operates on structured datasets, such as databases, spreadsheets, or CSV files. These datasets may contain diverse types of data, including numerical, categorical, and textual information.

Types of Errors: Data errors can take various forms, including missing values, outliers, inconsistent formats, and semantic inaccuracies. ED methods need to be capable of detecting a wide range of error types.

Categories of ED Methods:

Error Detection methods can be categorized into several main categories:

Rule-Based Methods: These methods rely on predefined rules and constraints to identify errors. Rules can be based on data constraints, data patterns, or domain-specific knowledge.

Statistical-Based Methods: Statistical approaches leverage mathematical techniques to identify errors. These methods often involve calculating statistics, such as mean, standard deviation, or range, and flagging data points that deviate significantly from expected values.

Machine Learning (ML) and Deep Learning (DL) Methods: Modern ED approaches harness the power of ML and DL algorithms to automatically learn error patterns from labeled training data. These models can generalize and detect errors in diverse datasets.

Commercial and Industrial Applications: In practice, Error Detection is essential across various industries, including finance, healthcare, e-commerce, and more. For example, in the financial sector, accurate detection of erroneous transactions is critical for fraud prevention.

Recent Advances: Recent advancements in Error Detection include the integration of machine learning techniques, such as autoencoders and anomaly detection algorithms, to improve accuracy and automate error detection processes. Additionally, the use of pre-trained language models for error detection in textual data has shown promise.

Challenges and Future Directions: Challenges in Error Detection include handling large and complex datasets, dealing with missing data, and developing robust methods for detecting subtle errors. Future directions may involve exploring automated error correction techniques and enhancing ED methods' scalability and adaptability.

Conclusion: Error Detection is a fundamental step in ensuring data quality and reliability in data-driven decision-making processes. It is a dynamic field with ongoing research and development aimed at addressing emerging challenges in data error detection. As data continues to grow in volume and complexity, ED remains a critical component of data management pipelines, playing a vital role in maintaining data accuracy and integrity.

Data Imputation (DI):

Introduction: Data Imputation is a crucial technique in data preprocessing and data cleaning, primarily focused on addressing missing or incomplete data within datasets. Its primary goal is to estimate or predict missing values, ensuring that datasets are complete and suitable for analysis, modeling, and decision-making. Data imputation is widely used in various domains, including healthcare, finance, machine learning, and social sciences.

Problem Statement: In the context of Data Imputation, the central challenge is to fill in missing or incomplete data points in a dataset accurately. Given a dataset with missing values, the task is to predict or estimate these missing values based on the available data. Data imputation can involve numerical, categorical, or textual data, and it aims to enhance the dataset's completeness while preserving its quality.

Key Components:

- **Missing Data Types:** Data can be missing for various reasons, such as data entry errors, sensor failures, or non-responses in surveys. Data imputation methods need to consider different types of missing data, including missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR).
- **Imputation Techniques:** Data imputation employs a range of techniques, including statistical methods, machine learning algorithms, and domain-specific knowledge-based imputation. These methods can handle diverse data types and missing data patterns.

Categories of Data Imputation Methods: Data Imputation methods can be categorized into several main categories:

1. **Mean/Median Imputation:** This simple approach replaces missing values with the mean or median of the observed values for a given attribute.
2. **Regression Imputation:** Regression-based imputation methods use regression models to predict missing values based on other attributes.
3. **K-Nearest Neighbors (K-NN) Imputation:** K-NN imputation involves finding the K most similar data points to the one with missing values and averaging or interpolating their values.
4. **Matrix Factorization:** These methods factorize the dataset into latent factors and use them to predict missing values.
5. **Machine Learning Imputation:** Advanced imputation techniques involve machine learning algorithms, such as decision trees, random forests, or deep learning models, to predict missing values based on the relationships between attributes.

Applications: Data Imputation finds applications in various domains, including:

- Healthcare: Imputing missing medical records or patient data for clinical research.
- Finance: Estimating missing financial data or stock prices for portfolio analysis.
- Social Sciences: Imputing survey responses or census data to perform statistical analysis.
- Machine Learning: Preprocessing datasets for machine learning models that require complete data.

Recent Advances: Recent advancements in Data Imputation include the use of deep learning models, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), to generate realistic imputations. Additionally, imputation models that can handle sequential data and time-series data have gained attention.

Challenges and Future Directions: Challenges in Data Imputation involve handling high-dimensional data, imputing missing values in time-series data, and addressing the impact of imputation on downstream analyses. Future directions may include developing imputation methods that can handle complex data types, such as image or geospatial data, and further automation of the imputation process.

Conclusion: Data Imputation is a fundamental data preprocessing technique that plays a pivotal role in ensuring the completeness and utility of datasets. It bridges gaps in data caused by missing values, making data analysis and modeling more effective. As data collection and analysis continue to grow in complexity, Data Imputation remains a dynamic field with opportunities for innovation and improved techniques to handle diverse data types and challenges.

2.2 Background on Foundation Models

We now give an overview of language FMs, starting from very early, smaller FMs (i.e. PLMs) and moving to large-scale FMs, the latter of which is the focus of this paper.

Pretrained Language Models (PLMs):

- **What are PLMs?** Pretrained Language Models (PLMs) are neural network models that are pretrained on vast corpora of publicly available text, such as web pages, books, articles, and other textual sources. They have become a cornerstone in natural language processing (NLP) and artificial intelligence (AI) research.
- **Semantic Learning:** The first generation of PLMs, including ELMo, BERT, and RoBERTa, have transformed the field of NLP by learning the semantics of natural language. They do this by predicting masked words within sentences during the pretraining phase. This process enables them to grasp the contextual meaning of words and phrases.
- **Model Scale:** These PLMs are characterized by their substantial scale. They typically consist of neural networks with hundreds of millions to billions of parameters. The large number of parameters allows them to capture intricate patterns and relationships in language.
- **Notable PLMs:**
- **BERT (Bidirectional Encoder Representations from Transformers):** BERT, developed by Google, is one of the most influential PLMs. Its bidirectional training revolutionized NLP by capturing contextual information effectively.
- **GPT-3 (Generative Pretrained Transformer 3):** GPT-3, developed by OpenAI, is renowned for its impressive text generation capabilities and versatility. It's one of the largest and most powerful PLMs.
- **RoBERTa:** RoBERTa, created by Facebook AI, is a variant of BERT that fine-tunes the model's architecture and training methodology, achieving state-of-the-art performance on various NLP benchmarks.
- **T5 (Text-to-Text Transfer Transformer**): T5, developed by Google AI, frames NLP tasks as text-to-text tasks, simplifying the approach and achieving remarkable results across a wide range of applications.
- **XLNet:** XLNet, developed by Google AI and Carnegie Mellon University, combines bidirectional and autoregressive training, resulting in improved performance on various NLP tasks.

Adaptation to Downstream Tasks: PLMs are not limited to pretraining; they are versatile in adapting to specific NLP tasks. This adaptation typically involves two key steps:

- **Task-Specific Prediction Layer:** A task-specific prediction layer, such as a classification layer, is added on top of the pretrained PLM. This layer is designed to make predictions for the target task.
- **Fine-Tuning:** During fine-tuning, all model weights, including the pretrained weights, are updated using task-specific data. This fine-tuning step allows PLMs to specialize in various downstream tasks, such as text classification, question answering, or language generation.

In summary, Pretrained Language Models (PLMs) represent a groundbreaking approach in NLP. They are neural networks pretrained on extensive textual data and have the capacity to understand the semantics of natural language. These models are then fine-tuned for specific tasks, making them highly adaptable and powerful tools for a wide range of NLP applications. Their development has marked a significant advancement in AI and NLP research.

Large Autoregressive Language Models:

- **Introduction of GPT-3:** In 2020, the field of machine learning experienced a groundbreaking shift with the introduction of GPT-3 (Generative Pretrained Transformer 3). GPT-3 represented a novel class of large-scale language models, which were autoregressive in nature. These models marked a significant leap in terms of scale, performance, and capabilities.
- **Autoregressive Language Models:** Large autoregressive language models are pretrained neural networks designed to predict the next word in a sequence of words or tokens. They are characterized by their ability to generate coherent and contextually relevant text, one word at a time. This autoregressive property allows them to maintain context and produce human-like language.
- **Scale and Parameter Count:** One of the defining features of these models is their sheer scale. GPT-3, for instance, boasts billions of parameters, which is an order of magnitude larger than its predecessors. The extensive parameter count enables these models to capture intricate language patterns and nuances.
- **Language Generation Tasks:** Large autoregressive language models have found wide application in various language generation tasks. Some of the tasks they excel at include:
    - **Question Answering:** They can generate detailed and contextually accurate answers to questions posed in natural language.
    - **Summarization:** They are proficient at condensing lengthy text into concise and coherent summaries.
    - **Text Completion:** They can predict and complete text prompts, making them valuable in natural language understanding tasks.**Conversational Agents:** They

serve as the foundation for chatbots and conversational AI systems, enabling human-like interactions.

- **Ongoing Development:** Since the release of GPT-3, the field has witnessed a rapid progression in the development of larger and more powerful autoregressive language models. These newer models aim to outperform their predecessors in terms of both quality and efficiency. Notable examples include GPT-4 and beyond.
- **Applications Beyond NLP:** The impact of large autoregressive language models extends beyond natural language processing. They have also been applied in various other domains, including code generation, image generation, and even scientific research.

In summary, large autoregressive language models, exemplified by GPT-3 and subsequent iterations, have reshaped the machine learning landscape. Their ability to generate human-like text at an unprecedented scale has opened up new possibilities in language understanding, generation, and a wide array of applications across multiple fields.

Emergent Behaviors and Few-Shot Learning:

- **Introduction:** Emergent behaviors in the context of large language models refer to the remarkable capabilities that these models exhibit, particularly in terms of few-shot learning. Few-shot learning is the ability of a model to perform a task or solve a problem with very few examples or minimal task-specific information.
- **GPT-3's Capacity:** The largest variant of GPT-3, with a staggering 175 billion parameters, possesses a remarkable capacity for few-shot prompting. It can solve a wide range of natural language tasks with just a few examples or, in some cases, solely based on a task description. For example, it can translate from French to English with minimal guidance.
- **No Fine-Tuning Required:** What sets this few-shot learning apart is that it doesn't rely on traditional fine-tuning approaches. In typical fine-tuning, model parameters are updated to adapt to a specific task. However, with few-shot prompting, GPT-3 demonstrates the ability to perform tasks without any parameter updates. It showcases a form of in-context learning.
- **Diverse Range of Tasks:** Few-shot prompting has proven to be highly effective across a diverse range of tasks, many of which are quite different from the model's original pretraining objective. Some examples include code generation, trivia question answering, and even arithmetic calculations. This adaptability showcases the model's versatility.
- **Locating Learned Behaviors:** The behavior of large language models in few-shot prompting scenarios can be conceptualized as the model "locating" or accessing an already-learned behavior from its extensive pretraining. It demonstrates the ability to apply learned knowledge in context, even for tasks it wasn't explicitly trained on.
- **Smaller Models:** While the largest models exhibit impressive few-shot capabilities, smaller variants with fewer parameters (typically less than 10 billion) may require some

form of task-specific fine-tuning to perform effectively on data tasks. Research into optimizing the performance of smaller models on specific tasks is an ongoing area of study.

In summary, emergent behaviors in large language models like GPT-3, particularly their few-shot learning abilities, have pushed the boundaries of what AI models can achieve. These models can perform a wide range of tasks with minimal examples or task descriptions, demonstrating their adaptability and capacity to apply previously learned behaviors in context. This capability opens up new avenues for AI applications and continues to be a subject of active research.

3 GENERATIVE AI / AI IN CREATIVITY

Generative AI refers to AI systems primarily designed to generate content (text, images, audio and videos). It sets apart from AI systems with different functions, like classifying data (e.g., labeling images), grouping data (e.g., identifying customer segments with similar purchasing behavior), or making decisions (e.g., guiding an autonomous vehicle). Some common examples of generative AI systems are image generators (Midjourney or stable diffusion), Chatbots (ChatGPT, Bard, Palm), code generators (CodeX, Co-Pilot [32]) and audio generators (VALL-E). Generative AI works by leveraging complex algorithms and statistical models to generate new content that mimics the patterns and characteristics of the training data. Generative AI systems can employ different techniques like Variational Autoencoders (VAEs) [33], Generative Adversarial Networks (GANs) [34], or autoregressive models [35] to achieve the desired generation capabilities. These techniques allow the model to capture the underlying data distribution and generate new content that is similar to the training data, as shown in Fig. 3. LLMs, such as Chat GPT, are a type of generative AI that is specifically designed to generate human-like language in response to a given prompt. These models are trained on massive amounts of textual data (see Table II), using techniques such as unsupervised learning to learn the statistical patterns of language. However, many people accord the capabilities provided by GPT models to "more data and computing power" instead of "better ML research".

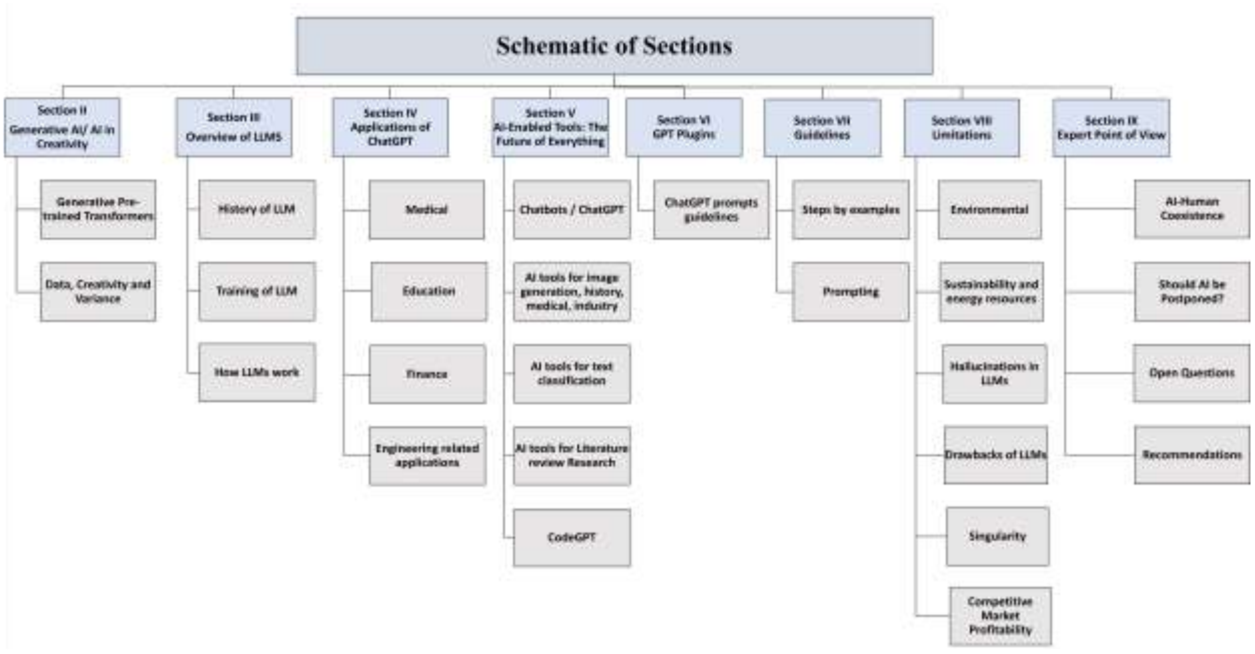
3.1 Generative Pre-trained Transformers - GPT 3.5

The transformer-based architecture is a type of neural network that is well-suited for NLP tasks. It uses a stackof self-attention layers to learn long-range dependencies in text. Self-attention [36] is a mechanism that allows the model to learn the importance of each word in the input sequence, regardless of its position. This is important for NLP tasks, such as translation and question answering, where the meaning ofa sentence can depend on the words that are far apart.
The architecture of GPT 3.5 is shown in Fig. 4, whichconsists of six major steps.

1) Input Encoding: The input to the GPT model is a sequence of tokens representing the data. Each token is converted into a high-dimensional vector representation through an embedding layer.
2) Transformer Encoder: The GPT model consists of multiple layers of transformer encoders. Each encoder layer has a self-attention mechanism and feed-forward neural networks. Self-attention allows the model to capture dependencies between different words in the input sequence, while the feed-forward networks process and transform the representations.
3) Contextual Embedding: As the input sequence passes through the transformer encoders, each token's representation is updated in a contextualized manner. This meansthat the

representation of each token is influenced by the surrounding tokens and their contextual information.

4) Decoding and Language Generation: Once the input sequence has been encoded through the transformer encoders, the GPT model can generate new text by predicting the probability distribution of the next token given the context. This process is typically done using a softmax activation function, which produces a probability distribution over all possible tokens in the vocabulary.



Schematic of Sections
Section II Generative AI/ AI in Creativity
Generative Pre-trained Transformers
Data, Creativity and Variance
Section III Overview of LLMS
History of LLM
Training of LLM
How LLMs work
Section IV Applications of ChatGPT
Medical
Education
Finance
Engineering related applications
Section V AI-Enabled Tools: The Future of Everything
Chatbots / ChatGPT
AI tools for image generation, history, medical, industry
AI tools for text classification
AI tools for Literature review Research
CodeGPT
Section VI GPT Plugins
ChatGPT prompts guidelines
Section VII Guidelines
Steps by examples
Prompting
Section VIII Limitations
Environmental
Sustainability and energy resources
Hallucinations in LLMs
Drawbacks of LLMs
Singularity
Competitive Market Profitability
Section IX Expert Point of View
AI-Human Coexistence
Should AI be Postponed?
Open Questions
Recommendations

Fig. V : Schematic of the overview of the survey at a glance.

5) Training with Masked Language Modeling: During the pre-training phase, GPT models often employ a technique called Masked Language Modeling (MLM) [37]. MLM involves randomly masking certain tokens in the input sequence and training the model to predict those masked tokens based on the context. This helps the model learn contextual relationships and improves its ability to generate coherent text.

6) The 6th layer represents the output of the model.

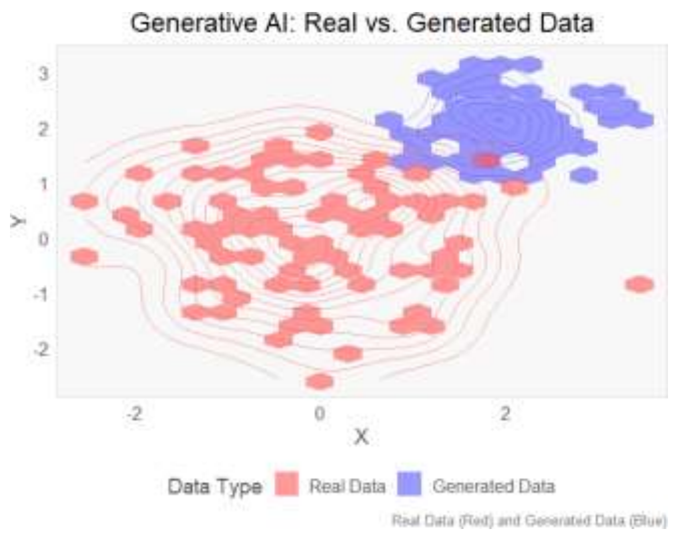


Fig. VI : A multivariate normal distribution applied to estimate the mean vector and covariance matrix of the underlying distribution using maximum likelihood estimation. Finally,we generate new synthetic data from the learned probability distribution. This demonstrates the concept of generative AI using probability distributions, where the AI learns to generatenew data that is similar to the original data by modeling the underlying probability distribution

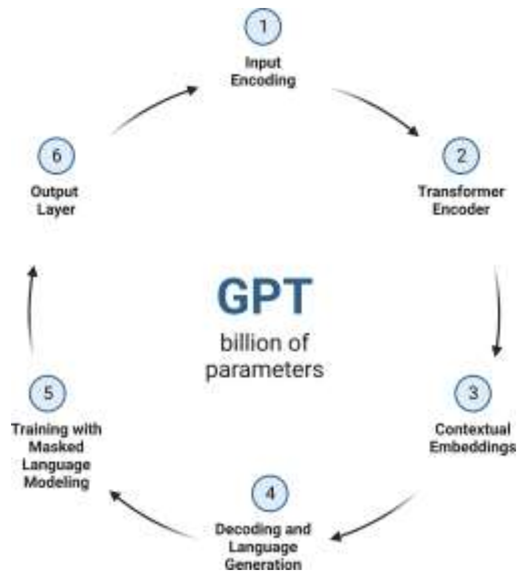


Fig. VII : The architecture of Generative Pre-trained Transformers

Table III: Comparison of LLMs' Reasoning Performance. Notations: MMLU [24]: high school and college knowledge, GSM8K: elementary school math, MATH: very hard math and natural science. All current models struggle, BBH [25]: a collection of 27 hard reasoning problems, HumanEval [26]: a classical dataset for evaluating coding capability, C-Eval [27]: a collection of 52 disciplines of knowledge test in Chinese, TheoremQA [28]: a question-answering dataset driven by STEM theorems. [29], [24], [23], [30], [31], [27]

Model	Param.	Type	GSM8K	MATH	MMLU	BBH	HumanEval	C-Eval	TheoremQA
GPT-4	-	RLHF	92.0	42.5	86.4	-	67.0	68.7*	43.4
claude-v1.3	-	RLHF	81.8*	-	74.8*	67.3*	-	54.2*	24.9
PaLM-2	-	Base	80.7	34.3	78.3	78.1	-	-	31.8
GPT-3.5-turbo	-	RLHF	74.9*	-	67.3*	70.1*	48.1	54.4*	30.2
claude-instant	-	RLHF	70.8*	-	-	66.9*	-	45.9*	23.6
text-davinci-003	-	RLHF	-	-	64.6	70.7	-	-	22.8
code-davinci-002	-	Base	66.6	19.1	64.5	73.7	47.0	-	-
text-davinci-002	-	SIFT	55.4	-	60.0	67.2	-	-	16.6
Minerva	540B	SIFT	58.8	33.6	-	-	-	-	-
Flan-PaLM	540B	SIFT	-	-	70.9	66.3	-	-	-
Flan-U-PaLM	540B	SIFT	-	-	69.8	64.9	-	-	-
PaLM	540B	Base	56.9	8.8	62.9	62.0	26.2	-	-
LLaMA	65B	Base	50.9	10.6	63.4	-	23.7	38.8*	-
PaLM	64B	Base	52.4	4.4	49.0	42.3	-	-	-
LLaMA	33B	Base	35.6	7.1	57.8	-	21.7	-	-
InstructCodeT5+	16B	SIFT	-	-	-	-	35.0	-	11.6
StarCoder	15B	Base	8.4	15.1	33.9	-	33.6	-	12.2
Vicuna	13B	SIFT	-	-	-	-	-	-	12.9
LLaMA	13B	Base	17.8	3.9	46.9	-	15.8	-	-
Flan-T5	11B	SIFT	16.1*	-	48.6	41.4	-	-	-
Alpaca	7B	SIFT	-	-	-	-	-	-	13.5
LLaMA	7B	Base	11.0	2.9	35.1	-	10.5	-	-
Flan-T5	3B	SIFT	13.5*	-	45.5	35.2	-	-	-

LLMs have captured significant interest in recent years due to their remarkable performance across an extensive array of NLP tasks, including text generation, translation, summarization, question-answering, and sentiment analysis [38]. Constructed upon the foundation of the transformer architecture [5], these models exhibit an extraordinary capacity to process and generate human-like text by leveraging massive volumes of training data.

3.2 Data, Creativity, and Variance

As discussed previously, LLMs are developed by training large deep neural networks using text data made up of multiple different sources including but not limited to books, social media and also text from the internet such as poetry, songs, news articles etc. This diversity in the training mix allows the model to provide output text that is coherent just as a human written text may read. However, it should be noted that the “creativity” exhibited in LLMs goes beyond the regurgitation of data that it may have seen during the training process. To produce text creatively, the deep learning model of the LLM needs to form an understanding of the text used in its training in aspects related to language, tone and writing patterns etc. This way the LLM is able to generate responses to user queries that are creative and genuine in terms of language writing by combining the different types of input information it ingested during training together to generate meaningful results for the provided query.

A fundamental criterion for gaining the capability of this creativity is to have sufficient variance, which indicates to the model’s ability to produce an unexpected output. In short, variance ensures that there is randomness in a model’s output, and it is introduced to enable it to generate sufficiently good results over a range of output results. By introducing variance in the model, one can increase the diversity of output content

generated which goes beyond the scope of what the training data consisted of.

It is acknowledged that since the release and mainstreaming of LLMs by users of all walks of life, some have complained of LLM getting stuck in a cycle of similar answers, especially if a complex query has been asked of it multiple times. For e.g., Microsoft found that its Bing AI powered by ChatGPT tends to get repetitive in its responses after 15 consecutive chats [1]. While this problem has mitigated since then by taking mea- sures such as refreshing context and/or introducing limits to the questions asked per session, this does question the variance capability of LLMs. There is a philosophical consideration here, LLMs have been associated with being “God-like AI” due to their exhibited creativity in different fields. From a human scenario, an example could be considered fingerprints, each one of the more than eight billion people in the world has unique fingerprints which can be used to identify them. However, from a philosophical perspective, it could be of interest how the size of the human population affects the variance of this space of potential of different fingerprints for humans? Therefore, it is important to appreciate the total “space” of creativity that LLMs have by design and consider creativity in that respect

4 OVERVIEW OF LLMS

LLMs have revolutionized the field of artifical intelligence and have found applications in various domains, including communication, content generation, and knowledge dissemination. In this Section, we briefly discuss the history, training, and working of LLMs.

4.1 A History of LLM

LLMs are a type of AI model that can process and generate natural language text. These models are typically trained on massive amounts of text data and use deep learning techniques to learn the patterns and structures of language [39]. The history of LLMs can be traced back to the early days of NLP research [40]. The first language models were developed in the 1950s and 1960s. These models were rule-based and relied on handcrafted linguistic rules and features to process language. They were limited in their capabilities and were not able to handle the complexity of NLP [41]. In the 1980s and 1990s, statistical language models were developed. These models used probabilistic methods to estimate the likelihood of a sequence of words in a given context. They were able to handle larger amounts of data and were more accurate than rule-based models [42]. However, they still had limitations in their ability to understand the semantics and context of language [43]. The next major breakthrough in language modeling came in the mid-2010s with the development of neural language models [44]. These models used deep learning techniques to learn the patterns and structures of language from large amounts of text data. The first neural language model was the recurrent neural network language model (RNNLM), which was developed in 2010. RNNLM was able to model the context of words and produce more natural-sounding text than previous models [45].

In 2015, Google introduced the first large-scale neural language model called the Google Neural Machine Translation (GNMT) system [46]. This model was trained on massive amounts of bilingual text data and was able to achieve stateof-the-art performance on machine translation tasks. Figure 5 shows the evolution of LLMs. The development of LLMs continued with the introduction of the Transformer model in 2017 [5]. The Transformer was able to learn the longer-term dependencies in language and allowed for parallel training on multiple Graphical Processing Units (GPUs), making it possible to train much larger models [47]. The release of OpenAI's GPT-1 [48] in 2018, marked a significant advance in NLP with its transformer-based architecture. With 117 million parameters, GPT-1 could generate contextually relevant sentences, demonstrating the potential of transformers in revolutionizing NLP tasks [49]. It was trained using a two-step process of unsupervised pre-training and supervised fine-tuning, a methodology that generated considerable interest in the academic and research community. Although GPT-1 had its limitations, it set the stage for subsequent, more powerful models, propelling a new era of AI research and highly-competitive research in LLMs (see Fig. 5). In 2020, OpenAI released the largest language model to date, GPT-3, which was trained on a massive amount of text data and was able to generate highly coherent and naturalsounding text [50]. GPT-3 demonstrated the potential of LLMs for a wide range of NLP tasks [51]. Inspired by the success of GPT-3, OpenAI recently announced and began working on the development of the next iteration of their language model, GPT-4 [52]. GPT-4 is expected to be even larger and more powerful than GPT-3, with the ability to generate even more coherent and natural sounding text

While LLMs have found use in a number of different applications, they can also be biased and produce inaccurate or harmful outputs. There are a number of factors that can contribute to the weakness of an LLM. One factor is the size of the training dataset. If the dataset is too small, the model may not be able to learn to generalize to new situations. Another factor is the quality of the training dataset. If the dataset contains biased or inaccurate information, the model will learn to reflect that bias or inaccuracy. There are also a number of factors that can contribute to the strength of an LLM. One factor is the size of the model. Larger models have more parameters, which allows them to learn more complex relationships between words and concepts. Another factor is the architecture of the model. Some architectures are better suited for certain tasks than others. In order to evaluate the performance of an LLM, it is important to consider the following factors: • The size of the training dataset. • The quality of the training dataset. • Number of parameters • Complexity of the architecture of the model. • The task that the model is being evaluated on. It is also important to note that LLMs are still under development, and their performance can vary depending on the specific task and the environment in which they are used. The following subsection discusses the most common LLMs, their training and working principles.

4.2 Training of LLMs

Training large language models involves several key steps that are fundamental to their successful development. The process typically begins with the collection and preprocessing of a massive amount of text data from diverse sources, such as books, articles, websites, and other textual corpora (see Table. IV). The curated dataset [54] serves as the foundation for training the LLMs. The training process itself revolves around a technique known as unsupervised learning, where the model learns to predict the next word in a sequence given the preceding context. This task is commonly referred to as language modeling. LLMs utilize sophisticated neural network architectures, such as Transformers, which enable them to capture complex patterns and dependencies in language. The training objective is to optimize the model's parameters to maximize the likelihood of generating the correct next word in a given context [9]. This optimization is typically achieved through an algorithm called stochastic gradient descent (SGD) or its variants, combined with backpropagation, which com- putes gradients to update the model's parameters iteratively.

- **Generative Pre-trained Transformer 3 (GPT-3):** GPT-3 is one of the most advanced and largest language models developed by OpenAI [50]. It represents a significant breakthrough in NLP and has garnered considerable attention due to its impressive capabilities [9]. GPT-3 follows a transformer-based architecture, which allows it to capture complex linguistic patterns and dependencies in text [55]. The model consists of a stack of transformer layers, enabling it to process and generate text at various levels of abstraction.

With a staggering number of approximately 175 billion parameters, GPT-3 is currently one of the largest language models ever created. The training process of GPT-3 involves unsupervised learning on a massive corpus of publicly available text data from the internet. By leveraging its enormous size and extensive training data, GPT-3 has acquired a broad understanding of language and can generate human-like text across various topics [56].

- **Bidirectional Encoder Representations from Transformer (BERT):** BERT is a prominent language model with significantly advanced NLP tasks. Its training process comprises pretraining and fine-tuning stages [57]. During pretraining, BERT learns a general language representation from large-scale unlabeled text data. It employs masked language modeling (MLM) and next-sentence prediction (NSP) tasks. MLM involves masking a portion of input tokens and training the model to predict the original masked tokens, fostering bidirectional context understanding [58]. NSP trains BERT to predict whether a second sentence follows the first, enhancing coherence comprehension. After pretraining, BERT undergoes finetuning on specific tasks with labeled data. Fine-tuning tailors BERT's learned representations to target tasks, such as sentiment analysis or named entity recognition. It employs backpropagation and gradient descent optimization to update model parameters. Training BERT demands significant computational resources [49], utilizing highperformance hardware like GPUs or Tensor Processing Units (TPUs) or field programmable gate arrays (FPGAs) [59], [60], [61]. Techniques such as layer normalization, residual connections, and attention mechanisms inherent in the transformer architecture further enhance BERT's capacity to capture intricate dependencies and long-range contextual relationships.

- **eXtreme Language understanding Network (XLNet):** XLNet is a generalized autoregressive pre-training method that surpasses the limitations of traditional left-to-right or right-to-left language modeling. XLNet is trained using a permutation-based approach that differs from traditional autoregressive models [62]. In the training process, rather than predicting the next word given the previous words in a fixed order, XLNet considers all possible permutations of the input sequence and models the probability of each permutation. This allows XLNet to capture dependencies in both directions, thus addressing the limitations of sequential left-to-right or right-to-left modeling [63]. The training of XLNet involves two key steps: unsupervised pretraining and supervised finetuning. During unsupervised pretraining, XLNet learns to predict words conditioned on the entire input context by maximizing the expected log-likelihood over all possible permutations. This is achieved using a variant of the transformer architecture, similar to models like BERT. The permutation-based objective function used in XLNet training presents unique challenges. Unlike traditional autoregressive models that can rely on the causal order of words for prediction, XLNet needs to consider all possible permutations, resulting in an exponentially large number of training instances. This makes the training process computationally intensive and requires efficient strategies, such as ”factorized sampling,” to sample a subset of permutations during each training iteration. Another difficulty in training XLNet is the need for large-scale computing resources [9], [64], [65]. The vast number of possible permutations and the large model size contribute to increased memory and computation requirements. Training XLNet often necessitates distributed training on multiple GPUs or TPUs and can take significant time [9].

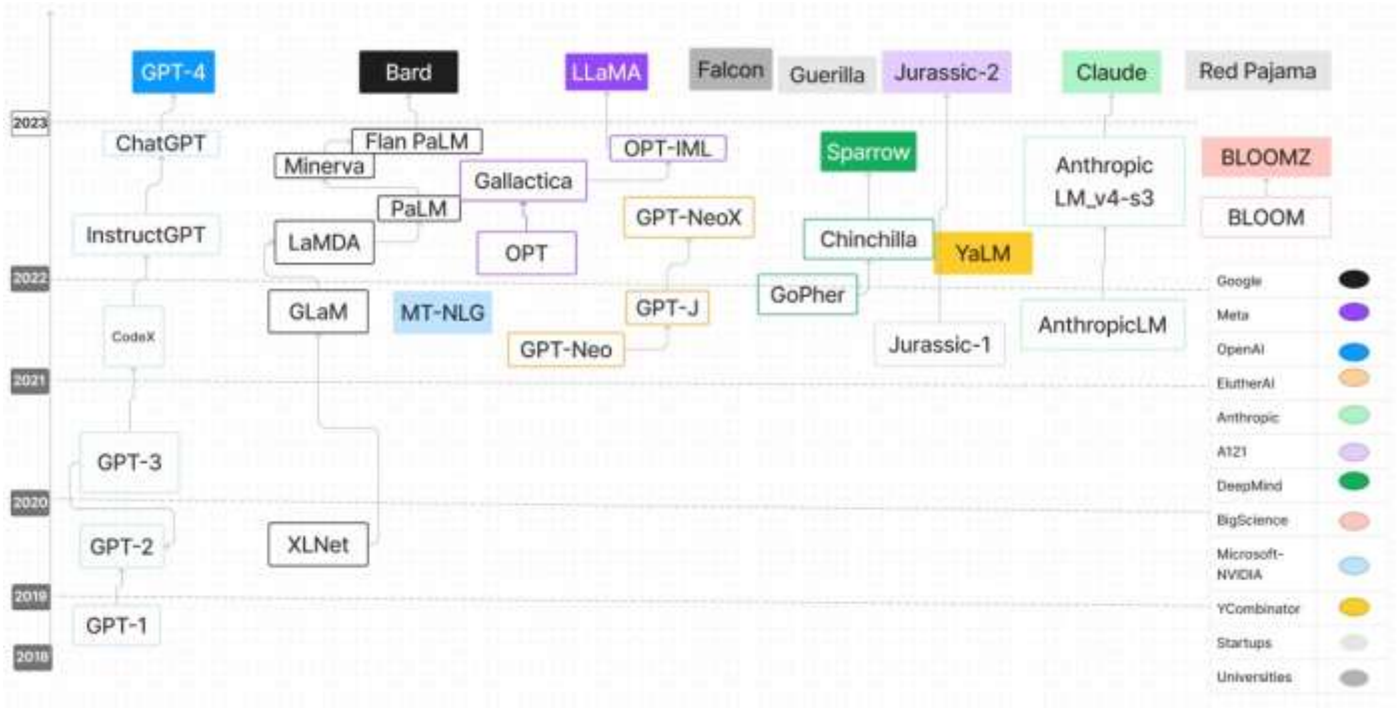


Fig. VIII : Evolution of LLMs across different research and commercial organizations.

TABLE IV: State-of-the-art for LLM training pipeline [53]. Notations: RM: Reward Modeling, RL: Reinforcement Learning, SFT: Supervised Fine-tuned.

Stage	Pretraining	SupervisedFinetuning	Reward Modeling	Reinforcement Learning
Dataset	Raw Internet II	Demonstration	Comparisons	Prompts
Algorithm	Language Modeling	Language Modeling	Binary Classification	Reinforcement Learning
Model	Base Model	SFT Model	RM Model	RL Model
Resources	100s of GPUs months of training deployable	1-100 of GPUs days of training deployable	1-100 of GPUs days of training not deployable	1-100 of GPUs days of training deployable

- **Text-to-Text Transfer Transformer (T5):** T5, developed by Google, is a versatile language model that is trained in a ”text-to-text” framework. The training process of T5 involves two main steps: pretraining and fine-tuning. During pretraining, T5 is trained on a massive corpus of publicly available text from the internet. The objective is to learn a generalized representation of language that can be applied to a wide range of tasks.The key innovation of T5 is the formulation of all tasks as text generation problems. This means that every task, including text classification, summarization, translation, and question answering, is cast into a text-to-text format. For example, instead of training T5 to answer questions directly, it is trained to generate the complete answer given the question and relevant context. In the pretraining phase, T5 is trained using a variant of the transformer architecture. The transformer model allows T5 to capture long-range dependencies and effectively model the contextual relationships in the input text [66]. The pretraining objective is typically based on maximum likelihood estimation, where T5 is trained to predict the target text given the source text. Once pretraining is complete, T5 undergoes fine-tuning on specific downstream tasks [66]. One of the challenges in training T5 is the availability of large-scale labeled datasets for various tasks. Finetuning requires task-specific labeled data, and the quality and quantity of the data play a crucial role in the model's performance [9]. Additionally, the computational resources required to train T5 can be substantial, as the model is computationally intensive due to its transformer architecture and the size of the pre-trained parameters.

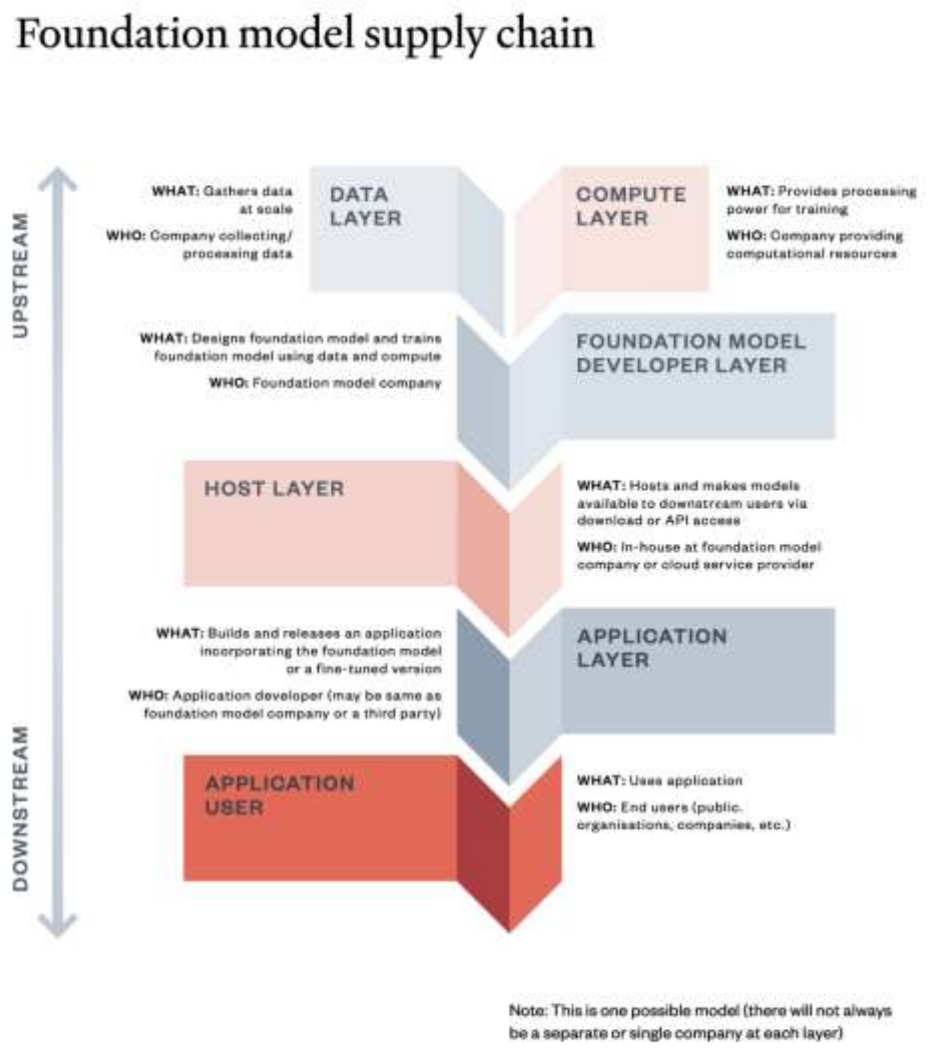
- **Conditional Transformer Language Model (CTRL):** CTRL is a language model designed to generate text based on specific control codes or prompts. It is trained using a two-step process: pretraining and fine-tuning. During pretraining, CTRL is trained on a large corpus of publicly available text data [67]. The objective of pretraining is to teach the model to understand and generate coherent text based on different control codes or prompts [68]. The training data includes diverse sources such as books, articles, websites, and other text documents. The training process involves utilizing the transformer architecture, similar to models like BERT and GPT. The model is trained to predict the next word or phrase in a given context, learning the statistical patterns and linguistic structures of the language. One of the unique aspects of CTRL is its conditioning of control codes or prompts. These control codes guide the model's text generation process, allowing users to specify the desired style, topic, or other characteristics of the generated text. The control codes act as explicit instructions to guide the model's behavior during both training and inference. The fine-tuning phase of CTRL is crucial for adapting the model to specific tasks or domains. Fine-tuning involves training the pre-trained CTRL model on task-specific datasets with control codes. The model is exposed to taskspecific prompts and is trained to generate text that aligns with the desired output or behavior for the given task.

4.3 How LLMs work

At their core, LLMs, are a type of AI that can mimic human intelligence. They function by employing advanced statistical models and deep learning techniques to process and understand extensive amounts of text data [69]. These models learn the intricate patterns and relationships present in the data, enabling them to generate new content that closely resembles the style and characteristics of a specific author or genre [70]. The process begins with pre-training, during which the LLM is exposed to a massive corpus of text from various sources such as books, articles, and websites. Through unsupervised learning, the model learns to predict the next word in a sentence based on the context of the preceding words. This allows the model to develop an understanding of grammar, syntax, and semantic relationships [67]. As shown in the LLMs pre-training pipeline in Figure 6, the first step is pretraining corpus sources which can be roughly divided into two categories: general data and specialized data. Following the collection of a huge amount of text data, it is critical to preprocess the data in order to generate the pre-training corpus, particularly by removing noisy, redundant, unnecessary, and potentially poisonous material [71] [72].The second stage involves quality filtering to remove the low quality and unwanted data from the training corpus using some techniques such as the language filtering, statistic filtering and keyword filtering [72]. Third stage is deduplication, where previous research [73] discovered that duplicate data in a corpus reduces the diversity of LMs, causing the training process to become unstable and thereby affecting model performance [73]. The fourth stage privacy reduction, it is crucial to address privacy concerns related to the use of web-based data for pre-training language models. This data often includes user-generated content containing sensitive or personal information, thereby posing a potential risk of privacy breaches [74]. Therefore, it is essential to undertake privacy redaction measures to remove personally identifiable information (PII) from the pre-training corpus. The last step is tokenization. It is an important step in data preprocessing as well. It seeks to segment raw text into sequences of individual tokens, which are then fed into LLMs. Following pre-training, the LLM undergoes fine-tuning, which involves training the model on a specific task or domain. During this phase, the model is provided with labeled examples and guided to generate more accurate and contextually appropriate responses for the target task [67]. Fine-tuning allows the LLM to specialize in various applications, such as language translation, question-answering, or text generation. The success of LLMs lies in their ability to capture the statistical patterns and linguistic nuances present in the training data [68]. By processing and analyzing vast amounts of text, LLMs gain a comprehensive understanding of language and are able to generate coherent and contextually relevant responses. During the inference stage, when interacting with an LLM, a user inputs a prompt or query. The model processes the input and generates a response based on its learned knowledge and context. This response is generated using probabilistic methods that consider the likelihood of various words or phrases given the input context.

5 FOUNDATION MODELS FOR DATA TASKS

Our goal is to understand whether FMs can benefit data cleaning and integration tasks. The procedure for applying FMs to data tasks requires adapting structured data inputs to textual inputs (Section 5.1), casting data tasks as text generation tasks (Section 5.2) and, for few-shot prompting, constructing demonstrative task examples to help the FM learn new data tasks (Section 5.3).



Foundation model supply chain
UPSTREAM
DOWNSTREAM
WHAT: Gathers data at scale
WHO: Company collecting/processing data
DATA LAYER
COMPUTE LAYER
WHAT: Provides processing power for training
WHO: Company providing computational resources
WHAT: Designs foundation model and trains foundation model using data and compute
WHO: Foundation model company
FOUNDATION MODEL DEVELOPER LAYER
HOST LAYER
WHAT: Hosts and makes models available to downstream users via download or API access
WHO: In-house at foundation model company or cloud service provider
WHAT: Builds and releases an application incorporating the foundation model or a fine-tuned version
WHO: Application developer (may be same as foundation model company or a third party)
APPLICATION LAYER
APPLICATION USER
WHAT: Uses application
WHO: End users (public, organisations, companies, etc.)
Note: This is one possible model (there will not always be a separate or single company at each layer)

Fig. IX : Foundation model supply chain

5.1 Tabular Data Serialization

Tabular Data Serialization is a process of converting structured tabular data, typically found in databases or spreadsheets, into text representations that can be used as input for Foundation Models (FMs). FMs are large-scale language models like GPT, or RoBERTa that are primarily designed to process and generate text. To leverage the power of FMs for data-related tasks, structured data needs to be serialized into a format that these models can understand and work with.

Here are the key aspects of Tabular Data Serialization:

1. **Structured Data Input**: Tabular data consists of structured information organized into rows and columns. Each row represents a record or an entry, while each column represents an attribute or a field. Examples of structured data include databases, CSV files, Excel spreadsheets, and dataframes.

2. **Conversion to Text**: The goal of Tabular Data Serialization is to convert the structured data into a text format that FMs can process. This conversion involves turning each entry (row) and its associated attribute-value pairs into a textual representation.

3. **Serialization Format**: The common serialization format used involves representing each attribute-value pair in the following way:

   serialize(e) ≔ attr1 : val1 . . . attrm : valm

   - **attr1**, **attr2**, ..., **attrm** are the attributes or column names.
   - **val1**, **val2**, ..., **valm** are the corresponding attribute values.

4. **Handling NULL Values**: If an attribute value is NULL (indicating missing or undefined data), it is serialized as an empty string. This ensures that even missing data is represented in the text format.

5. **Task-Relevant Attributes**: Depending on the specific data task and dataset, serialization may happen only for a subset of attributes that are relevant to the task. Not all attributes need to be included in the serialized representation.

6. **Prompt Format Sensitivity**: FMs can be sensitive to the format of the prompt or input text. The way data is serialized, the choice of attribute-value pairs, and how they are structured can impact the model's performance. Experimentation with different serialization methods is often required to optimize FM performance for a particular task.

7. **Sensitivity to Sub-Selection Choices**: In some cases, not all attributes are relevant to the task at hand. The choice of which attributes to include in the serialization can affect the model's ability to understand and generate meaningful responses. This sensitivity is studied to determine the most effective sub-selection choices for specific tasks.

In summary, Tabular Data Serialization is a critical preprocessing step that allows structured data to be fed into Foundation Models for various data-related tasks. It involves converting structured data into a text format that FMs can comprehend and work with effectively. This process is highly task-dependent and requires careful consideration of attribute selection and serialization format to achieve optimal results with Foundation Models.

The **serialize_row** function can be used to serialize a structured row of data into a string:

```
def serialize_row(
    row: pd.core.series.Series,
    column_map: Dict[str, str],
    sep_tok: str,
    nan_tok: str,
) -> str:
    """Turn structured row into string."""
    res = []
    for c_og, c_map in column_map.items():
        if str(row[c_og]) == "nan":
            row[c_og] = nan_tok
        else:
            row[c_og] = f"{row[c_og].strip()}"
        res.append(f"{c_map}: {row[c_og]}".lstrip())
    if len(sep_tok) > 0 and sep_tok != ".":
        sep_tok = f" {sep_tok}"
    return f"{sep_tok} ".join(res)
```

the **serialize_row** function can be used to convert structured data into a human-readable text format, making it suitable for input to natural language models or for generating user-friendly reports.

5.2 Data Tasks as Natural Language Tasks

The concept of "Data Tasks as Natural Language Tasks" involves transforming data-related tasks into natural language tasks that can be processed by Foundation Models (FMs) such as large-scale language models like GPT, or RoBERTa. This approach enables FMs to generate text-based responses as answers to data-related questions or tasks.

Here are the key steps and components of this approach:

1. **Data Task Transformation**: The first step is to convert data-related tasks into a format that can be expressed as natural language questions or prompts. These tasks can include tasks like entity matching, error detection, and imputation.

2. **Serialized Representations**: As mentioned in Section 3.1 (likely a previous section of the document), the structured data is serialized into a text format. This serialized representation serves as the input to the natural language task.
3. **Prompts**: Prompts are natural language descriptions or questions that encapsulate the data task. These prompts use the serialized representations from the serialized data to frame the task in a way that FMs can understand. The prompts are designed to elicit specific responses from the FM.
4. **FM Processing**: The prompts are passed as input to the Foundation Model. The FM generates text-based responses as answers to the tasks based on its understanding of the prompts and the context provided by the serialized data.
5. **Task-Specific Outputs**: The type of response generated by the FM depends on the specific data task:
   - For **entity matching**, the model typically generates a "Yes" or "No" response indicating whether two entities match or not.
   - For **error detection**, the model generates responses indicating whether an error is detected ("Yes") or not ("No").
   - For **imputation**, the model generates the missing value that needs to be filled in the data.
6. **Enumeration of Prompts**: The document enumerates and describes the prompts for each data task. These prompts are carefully crafted to ensure that the FM understands the task's requirements and can generate appropriate responses.
7. **Natural Language Output**: The FM's generated output is in natural language, making it human-readable and interpretable. This output can be used for decision-making, data validation, or further analysis.

In summary, "Data Tasks as Natural Language Tasks" is a methodology that leverages the capabilities of Foundation Models to process and generate natural language responses for data-related tasks. It bridges the gap between structured data and natural language understanding, enabling FMs to provide answers and insights for a wide range of data tasks.

We now enumerate the prompts for each task and we explain each one in detail .

Entity matching: given two entries (e,e') the template is

Product A is serialize(e). Product B is serialize(e').

Are Product A and Product B the same?

Data imputation: given an entry e and attribute j to infer,

____

attr1 : val1 . . . attr$_j$?

Error detection: given an entry e and attribute j to classify as erroneous,

____

Is there an error in attr$_j$: val$_j$?

These templates highlight the generality of this framework which could be extended beyond the tasks considered in this paper.

Entity Matching (EM)

Entity Matching is a fundamental process in data management and integration, where the primary goal is to determine if two entries, referred to as *e* and *e*', correspond to the same real-world entity. It plays a crucial role in ensuring data consistency and accuracy when combining data from various sources. This task is made more efficient by leveraging Pretrained Language Models (PLMs) like GPT-3.5. Let's explore the steps and components of entity matching:

Entity Matching Prompt:

The prompt for entity matching is a structured query that the PLM will process to determine whether two entries match. It is constructed as follows:

Product A is serialize(*e*). Product B is serialize(*e*'). Are Product A and Product B the same?

This prompt comprises several key components:

1. **"Product A" and "Product B":** These placeholders represent the two entries *e* and *e*' that are under consideration for matching.
2. **"serialize(*e*)" and "serialize(*e*')":** These phrases indicate that the entries are represented in a serialized format. Serialization is a method to convert structured data into a textual format suitable for processing by the PLM. It involves encoding the attributes and values of an entity in a structured manner.
3. **"Are Product A and Product B the same?":** This question serves as the query to the model, and it seeks a "Yes" or "No" response indicating whether the two entries represent the same entity.

Response:

The PLM generates a response, which is typically "Yes" or "No," to indicate whether Product A and Product B represent the same real-world entity. This response is determined based on the model's analysis of the serialized information provided in the prompt.

Additional Function - serialize_match_pair:

```
def serialize_match_pair(
    row: pd.core.series.Series,
    column_mapA: Dict[str, str],
    column_mapB: Dict[str, str],
    add_instruction: bool,
    instruction: str,
    suffix: str,
    prod_name: str,
    sep_tok: str,
    nan_tok: str,
) -> str:
    """Turn structured pair of entities into string for
matching."""
    res = (
        f"{prod_name} A is {serialize_row(row, column_mapA,
sep_tok, nan_tok)}."
        f" {prod_name} B is {serialize_row(row, column_mapB,
sep_tok, nan_tok)}."
        f"{suffix} "
    )
    if add_instruction:
        res = f"{instruction} {res}"
    return res
```

In this context, a specialized function named **serialize_match_pair** is utilized to format and structure the prompt. This function takes care of organizing the data, adding instructions if needed, and creating the prompt that is presented to the PLM. It allows for the seamless interaction between structured data and the language model.

By following this structured approach, you can effectively harness the capabilities of PLMs to perform entity matching tasks. The natural language interface provides flexibility and efficiency in data integration processes, ensuring data consistency and quality.

Error Detection (ED)

Error detection is a critical step in data cleaning pipelines. It involves the use of Pretrained Language Models (PLMs), such as GPT-3.5, to identify attributes in a given entry that contain errors. The task is framed as a classification problem, where the goal is to predict whether a particular attribute, denoted as *j*, has an error. Here's a detailed explanation of how error detection is performed using PLMs:

Error Detection Prompt:

The prompt for error detection is designed to identify errors in a specific attribute of an entry. It is structured as follows:

Is there an error in attr*j*: val*j*?

This prompt consists of the following components:

1. **"attr*j*: val*j*":** This part of the prompt specifies the attribute *j* and its corresponding value val*j* that are under scrutiny for errors.
2. **"Is there an error in...":** This question serves as the query to the model, asking whether there is an error in the specified attribute-value pair.

Response:

The PLM generates a response, typically "Yes" or "No," to indicate whether an error exists in the specified attribute-value pair. This determination is based on the model's analysis of the information provided in the prompt.

Additional Function - serialize_error_detection:

```
def serialize_error_detection(
    row: pd.core.series.Series,
    add_prefix: bool,
    instruction: str,
    suffix: str,
    sep_tok: str,
    nan_tok: str,
) -> str:
    """Turn single cell into string for error detection."""
    column_map = {
        c: c
        for c in row.index
        if str(c) not in ["Unnamed: 0", "text", "col_name",
"label_str", "is_clean"]
```

```
    }
    entire_row = serialize_row(row, column_map, sep_tok,
nan_tok)
    column_map = {row["col_name"]: row["col_name"]}
    res = f"{entire_row}\n\nIs there an error in
{serialize_row(row, column_map, sep_tok, nan_tok)}{suffix} "
    if add_prefix:
        res = f"{instruction} {res}"
    return res
```

In this context, a specialized function called **serialize_error_detection** is employed to format and structure the prompt for error detection. This function prepares the data, adds any necessary prefixes or instructions, and constructs the prompt that is presented to the PLM.

By following this structured approach, you can effectively utilize PLMs to perform error detection tasks. The natural language interface simplifies the process of identifying errors in data attributes, making it a valuable component of data cleaning pipelines.

Data Imputation(DI)

Data imputation is a critical component in data handling, particularly when dealing with incomplete datasets. Pretrained Language Models (PLMs) like GPT-3.5 can be leveraged for data imputation tasks. Data imputation involves the process of inferring or predicting the missing values of a specific attribute in an entry. Here's an in-depth explanation of how data imputation is performed using PLMs:

Data Imputation Prompt:

The prompt for data imputation is designed to infer missing values for a specific attribute. It is structured as follows:

attr1: val1 . . . attr*j* ?

This prompt encompasses the following key elements:

1. **"attr1: val1 . . . attr*j*":** This portion of the prompt specifies the attributes and values provided in the entry. These attributes are used to help the model understand the context and predict the missing value for attr*j*.
2. **"attr*j* ?":** This part of the prompt specifies the attribute attr*j* for which data imputation is required.

Response:

The PLM generates a response that includes the predicted value for the missing attribute attr*j*. The response is based on the model's analysis of the provided attributes and values.

Additional Function - serialize_imputation:

```
def serialize_imputation(
    row: pd.core.series.Series,
    column_map: Dict[str, str],
    impute_col: str,
    add_instruction: bool,
    instruction: str,
    suffix: str,
    sep_tok: str,
    nan_tok: str,
) -> str:
    """Turn single entity into string for imputation."""
    assert impute_col not in column_map, f"{impute_col} cannot
be in column map"
    # Rename to avoid passing white spaced sep token to
serialize_row
    sep_tok_ws = sep_tok
    if len(sep_tok) > 0 and sep_tok != ".":
        sep_tok_ws = f" {sep_tok}"
    res = f"{serialize_row(row, column_map, sep_tok,
nan_tok)}{sep_tok_ws}{suffix} "
    if add_instruction:
        res = f"{instruction} {res}"
    return res
```

In this context, a specialized function called **serialize_imputation** is employed to format and structure the prompt for data imputation. This function is responsible for preparing the data, adding any necessary prefixes or instructions, and constructing the prompt that is presented to the PLM.

Utilizing this structured approach simplifies the process of inferring missing data values and enables the effective use of PLMs for data imputation tasks.

5.3 Task Demonstrations

Task Demonstrations in Prompt Tuning: Task demonstrations play a crucial role in prompt-based training of language models, such as GPT-3.5. These demonstrations are used to teach the model how to perform specific tasks and understand the nuanced semantics of the data. The key objective is to guide the model to generate desired responses, whether it involves producing Yes/No answers or imputing missing values. Here's an overview of how task demonstrations are employed and two different approaches for selecting them:

Purpose of Task Demonstrations:

Task demonstrations are essential for two primary reasons:

- **Teaching the Model:** Demonstrations are used to train the model to execute a particular task as required. For instance, they help the model learn when to generate Yes/No answers, when to predict missing values, and how to format responses.
- **Semantic Understanding:** Demonstrations also enable the model to grasp the finer-grained semantics of the input data (D). This is crucial for the model to accurately interpret and process the data provided in the prompts.

2. Two Approaches for Selecting Task Demonstrations:

There are two main approaches for selecting task demonstration examples:

1. Random Sampling for Task Demonstrations:

Random sampling of task demonstrations involves selecting examples from a labeled dataset without specific ordering or criteria. While this method is straightforward, it comes with certain challenges and characteristics:

1. Random Selection:

- In this approach, a set number of examples is randomly chosen from the labeled dataset. These examples will be used as task demonstrations for the language model.

2. High Variance in Model Performance:

- One of the challenges of random sampling is that it often leads to high variance in the model's performance. The quality and relevance of the randomly selected examples can vary widely.

3. Impact of Prompt Order:

- The order in which examples are presented in the prompt can have a notable impact on the performance of the language model. This is because the model's responses are influenced by the order of examples in the input.

4. Ongoing Research:

- In recent times, there have been efforts to systematize the prompt tuning process in the context of random sampling. However, this remains an active area of research.

Using the get_random_prompt Function:

```
def get_random_prompt(train_data: pd.DataFrame, num_examples:
int = 10) -> str:
    """Get random examples for prompt from trian data."""
    prefix_exs_rows = sample_train_data(train_data,
num_examples)
    serialized_prefixes = [
        (txt + label).strip()
        for txt, label in zip(prefix_exs_rows["text"],
prefix_exs_rows["label_str"])
    ]
    prefix_exs = "\n\n".join(serialized_prefixes) + "\n"
    return prefix_exs
```

In practice, a function like **get_random_prompt** is used to generate prompts for the model. This function randomly selects a specified number of examples from the training data. These examples are typically selected without any particular pattern or criteria. The selected examples are then formatted and combined into a prompt that can be presented to the language model.

It's important to note that while random sampling is a simple way to obtain task demonstrations, it may result in varied model performance and often requires careful tuning to achieve consistent and reliable results. Researchers and practitioners continue to explore strategies for making random sampling more effective in the context of prompt-based language models.

TABLE V: Entity Matching Results Measured by F1 Score

Dataset	**Magellan**	**Ditto**	**GPT3-175B (k=0)**	**GPT3-175B (k=10)**
Fodors-Zagats	100	100	87.2	100
Beer	78.8	94.37	78.6	100
iTunes-Amazon	91.2	97.06	65.9	98.2
Walmart-Amazon	71.9	86.76	60.6	87.0
DBLP-ACM	98.4	98.99	93.5	96.6
DBLP-Google	92.3	95.60	64.6	83.8
Amazon-Google	49.1	75.58	54.3	63.5

In this table, we compare the F1 scores for entity matching across different datasets for different models. The performance is measured with and without task demonstrations (k=0 and k=10).

The table provides a comparative view of entity matching performance with and without task demonstrations across various datasets and models.

Manual Prompt Construction for Task Demonstrations:

In the manual construction of prompts, task demonstrations are carefully created and tailored to provide specific guidance to the language model. Unlike random sampling, manual prompt construction is a more deliberate and controlled process. Here's how this approach works and how the **get_manual_prompt** function is utilized:

1. Contextual Prompt Generation:

- Manual prompt generation is typically performed when there is a need for more specific context or fine-tuned guidance for the language model. The goal is to create prompts that are highly relevant to the task at hand.

2. Data-Driven Prompt Design:

- The content and structure of the prompt are often driven by the characteristics of the data and the specific task. It may involve providing detailed information, context, or task-specific instructions.

3. Handling Data Sources:

- The **get_manual_prompt** function takes two inputs: **data_dir** and **example**. **data_dir** specifies the data source or domain, and **example** represents an individual data point or entry as a Pandas Series.

4. Data-Driven Prompt Content:

- The function uses the **data_dir** to determine the appropriate format for the prompt based on predefined constants. The prompt content is constructed based on the specific data source and the example provided.

5. Handling Different Data Sources:

- If the data source is recognized (exists in the predefined constants), the function retrieves the corresponding prompt content. If there are subkeys or variations within the data source, it selects the appropriate prompt based on the **example** attributes.

6. Error Handling:

- The function includes error handling to ensure that the provided data source is recognized and the content is available in the constants. It also checks if the content format is as expected.

7. Customization and Control:

- Manual prompt construction provides users with finer-grained control over the prompt's content and structure. This approach allows for tailoring prompts to the specific needs of the task and the characteristics of the data.

8. Task-Specific Prompts:

- The goal is to create prompts that are well-suited to guide the language model in performing the task accurately and effectively.

In summary, manual construction of prompts is a valuable approach when precise and task-specific guidance is required for the language model. It offers flexibility and customization, allowing researchers and practitioners to leverage the full potential of language models for various applications.

Using the get_manual_prompt Function:

```
def get_manual_prompt(data_dir: str, example: pd.Series) -> str:
    """Get manual prompt for data name."""
    if data_dir not in constants.DATA2TASK.keys():
        raise ValueError(f"{data_dir} not recognized for
prompts")
    subkey_attr = constants.DATA2EXAMPLE_SUBKEY_ATTR[data_dir]
    if subkey_attr is None:
        if not isinstance(constants.PREFIXES[data_dir], str):
            raise ValueError(f"Prefix was not a string for
{data_dir}")
        return constants.PREFIXES[data_dir]
    else:
        if not isinstance(constants.PREFIXES[data_dir], dict):
            raise ValueError(
                f"Prefix was not a dict with {subkey_attr}
subkeys for {data_dir}"
     )
     return
constants.PREFIXES[data_dir][str(example[subkey_attr])]
```

In practical application, the **get_manual_prompt** function is employed to generate prompts for the model, and it operates differently from random sampling methods. Here's how this function works:

Purpose of the Function: The **get_manual_prompt** function is designed for creating prompts that provide specific and controlled guidance to the language model. Unlike random sampling, it's focused on producing tailored prompts that are well-aligned with the nature of the data source and the task at hand.

In summary, the **get_manual_prompt** function is a valuable tool when precision and task-specific guidance are essential. It empowers users to craft prompts that align with the intricacies of the data and the unique demands of the task, leading to more effective interactions with the language model.

6 EXPERIMENTS

In the "Experiments" section of a research paper or study, the authors typically describe the specific experiments they conducted to evaluate their approach or method

6.1 Experimental Setup

the "Experimental Setup" section in a research paper or experimental study provides critical details about how experiments were conducted, including the models, datasets, metrics, and baselines used. Here's a breakdown of what you might typically find in this section:

1. Models:

- This section outlines the machine learning or statistical models used in the experiments. It may include:
    - Model Architectures: A description of the specific model architectures, such as neural networks, decision trees, or regression models.
    - Hyperparameters: Details about the hyperparameters used to configure the models, including learning rates, batch sizes, and optimization algorithms.
    - Pretrained Models: If pretrained models were used, mention the source and version.
    - Any modifications or customizations made to the models for the experiments.

2. Datasets:

- Information about the datasets used for training, validation, and testing. This typically includes:
    - Dataset Names: The names and sources of the datasets.
    - Data Splitting: How the data was split into training, validation, and test sets.
    - Data Preprocessing: Any data preprocessing steps applied, such as cleaning, tokenization, or feature engineering.
    - Dataset Size: The number of samples or instances in each dataset split.
    - Data Statistics: Summary statistics of the datasets, such as mean, standard deviation, or class distribution.

3. Metrics:

- Metrics used to evaluate the performance of the models. Common metrics include:
    - Accuracy: The proportion of correctly classified instances.

    - Precision, Recall, F1-Score: Metrics for binary classification that provide insights into true positives, false positives, and false negatives.
    - Mean Absolute Error (MAE) or Mean Squared Error (MSE): Metrics for regression tasks.
    - Additional domain-specific metrics if applicable.

In the context of error detection, schema/entity matching, imputation, and data transformation, evaluation metrics are used to assess the performance of algorithms or models. The choice of evaluation metrics depends on the specific task and the nature of the problem. Here's a detailed explanation of the evaluation :

1. F1 Score:

- **Definition**: The F1 score is a metric that combines Precision and Recall into a single value. It is the harmonic mean of Precision and Recall and is used primarily in binary classification tasks.
- **Formula**:
    - F1 Score = 2 * (Precision * Recall) / (Precision + Recall)
- **Use Case**: F1 score is commonly used when you want to balance Precision and Recall, especially in scenarios with imbalanced datasets. It's particularly useful when both false positives and false negatives need to be considered.
- **Precision**:
    - Precision measures the proportion of true positive predictions out of all positive predictions made by the model.
    - Formula: Precision = True Positives / (True Positives + False Positives)
    - High precision indicates that the model has fewer false positives.
- **Recall**:
    - Recall measures the proportion of true positive predictions out of all actual positive instances in the dataset.
    - Formula: Recall = True Positives / (True Positives + False Negatives)
    - High recall indicates that the model can identify most of the actual positive instances.

2. Precision:

- **Definition**: Precision is a metric that measures the accuracy of positive predictions made by the model. It focuses on the number of true positives relative to all positive predictions.
- **Formula**: Precision = True Positives / (True Positives + False Positives)
- **Use Case**: Precision is important when you want to minimize false positives. It is commonly used in tasks where the cost of making a false positive prediction is high.

3. Recall:

- **Definition**: Recall, also known as Sensitivity or True Positive Rate, measures the model's ability to identify all relevant instances of a particular class. It focuses on the number of true positives relative to all actual positive instances.
- **Formula**: Recall = True Positives / (True Positives + False Negatives)
- **Use Case**: Recall is crucial when you want to ensure that you capture as many of the actual positive instances as possible. It is used in tasks where missing a positive instance has severe consequences.

4. Accuracy:

- **Definition**: Accuracy is a metric that measures the overall correctness of predictions made by the model. It calculates the proportion of correctly predicted instances (both true positives and true negatives) out of all instances.
- **Formula**: Accuracy = (True Positives + True Negatives) / Total Instances
- **Use Case**: Accuracy is commonly used when you want to assess the overall performance of a classification model. However, it may not be suitable for imbalanced datasets because it can be skewed by the majority class.

Interpretation:

- **F1 Score**: A higher F1 score indicates better overall performance in binary classification tasks. It is particularly useful when you need a balanced trade-off between Precision and Recall.
- **Precision**: High precision means that when the model predicts a positive instance, it's more likely to be correct. It focuses on minimizing false positives.
- **Recall**: High recall means that the model can identify most of the actual positive instances. It focuses on minimizing false negatives.
- **Accuracy**: Accuracy provides a measure of overall correctness. A high accuracy score suggests that the model is making correct predictions for a significant portion of instances. However, it may not be suitable for imbalanced datasets.

In summary, these metrics are essential tools for evaluating the performance of classification models and understanding the trade-offs between different aspects of model behavior, such as correctness, false positives, and false negatives. The choice of which metric to prioritize depends on the specific goals and requirements of your machine learning task.

4. Baselines:

- A description of the baseline methods used for comparison. Baselines can include:
    - Existing state-of-the-art models in the field.
    - Simpler or traditional algorithms used as benchmarks.
    - A discussion of why specific baselines were chosen.

6.2 Zero/Few-shot Performance of Large

FMs In this section we explore the zero and few-shot performance of GPT-3-175B. Our goal is to understand whether large FMs transfer to data tasks in zero- and few-shot settings.

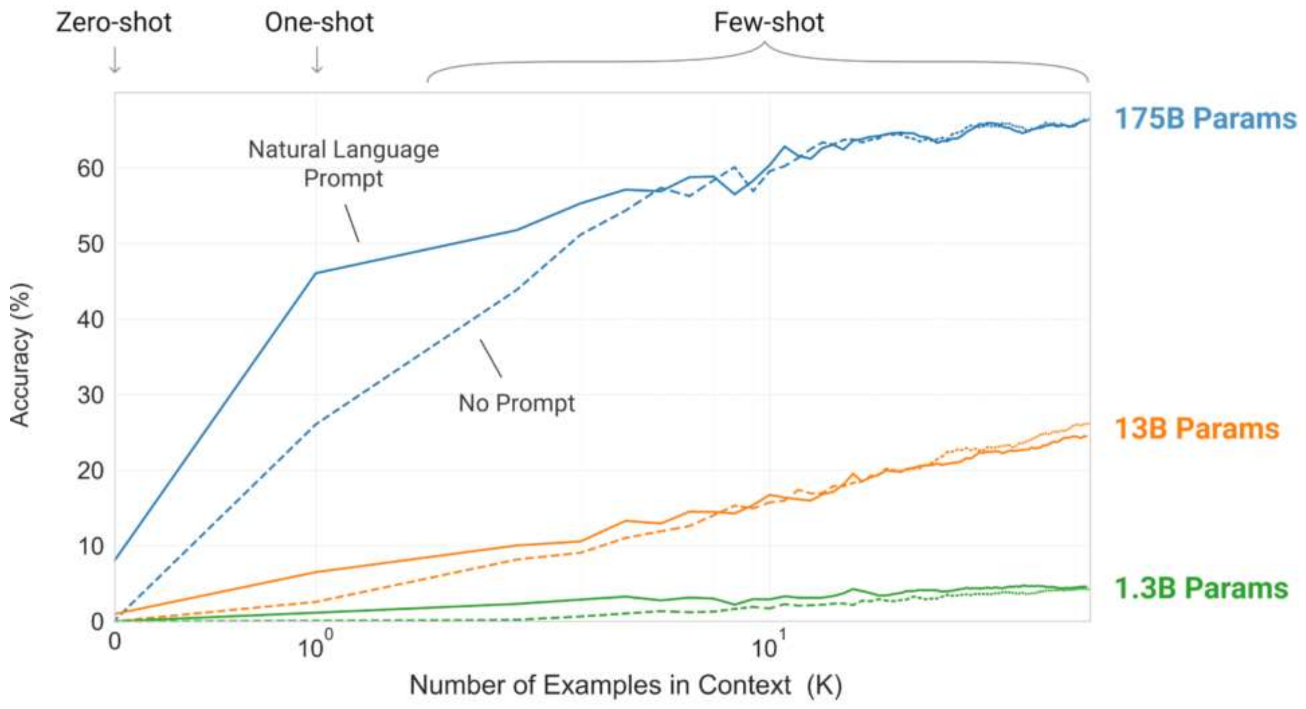


FIG. X

Few-shot Performance Our results show that in the few-shot setting, with manually curated task demonstrations, GPT-3-175B achieves SoTA performance on 4 entity matching, 2 imputation, 2 data transformation, 2 error detection and 1 schema matching benchmark dataset(s) (see Table 1, Table 2, Table 3). The FM outperforms fullyfinetuned, SoTA PLM-based approaches for entity matching [57], and data imputation [72]. For error detection, the few-shot approach outperforms the ML-based SoTA method, HoloDetect, which uses data augmentation and weak supervision to perform well.

Zero-shot Performance In the zero-shot setting, we observe that the FM outperforms statistical-based approaches and standard data repair engines [84] for imputation. On entity matching, the zero-shot performance is significantly lower than the few shot performance. This performance gap suggests that demonstrations are very important for the task.

Discussion. These results show that large FMs can transfer to data tasks. These results are particularly exciting given that FMs are trained to model English language and have no prior exposure to the semantics of data tasks nor the syntax of tabular data. Furthermore, the zero-shot performance on imputation suggests that large FMs not only have an understanding of how to complete the tasks, but also have encoded knowledge that is needed to correct and complete records (e.g. functional dependencies between address and zip code). We analyze encoded large FM knowledge in the full report [75]. On the entity matching datasets that the FM does not achieve SoTA on, we find that the FM struggles on data domains that contain jargon not commonly found in text. In such cases, the model lacks a strong semantic understanding of the input and has difficulty reasoning over the data. For example, in the Amazon-Google dataset, the model has difficulty matching samples due to the high volume of product-specific identifiers in the descriptions. Here, for instance, the model fails to accurately match the two entries: “name: pcanywhere 11.0 host only cd-rom xp 98 nt w2k me. manufacturer: symantec. price: NULL” and “name: symantec pcanywhere 11.0 windows. manufacturer: NULL. price: 19.99.”.

6.3 Prompt Tuning Ablations

In this section, we analyze the performance impact of the three different choices made during prompt tuning: attribute selection, prompt formatting, and task demonstration curation. 4.3.1 Results. We run our ablations on three entity matching datasets (see Table 4). For all datasets, we evaluate on up to 200 samples for cost purposes. We discuss our findings next.

Attribute Selection First, we find through experimentation that sub-selecting attributes during row serialization can have a nontrivial impact on entity matching performance. In particular, we observe that sub-selecting attributes that are essential in determining whether two entities match (e.g. name) and removing noisy attributes improves model accuracy. To better illustrate this point, we evaluate FM performance on three datasets when not sub-selecting attributes. We find that, on average, attribute selection results in a 13.7 F1 point performance improvement (Table 4).

Table VI : Entity Matching Ablation Results (F1 Score) for Different Prompt Formats

Prompt Format	**Beer**	**iTunes-Amazon**	**Walmart-Amazon**	**Amazon**
Prompt 1 (w. Attr. & Example Select.)	100 ± 0.00	98.2 ± 0.00	88.9 ± 0.00	
Prompt 1 (w/o Example Select.)	91.1 ± 0.05	86.6 ± 0.02	65.2 ± 0.04	
Prompt 1 (w/o Attr. Select.)	76.9 ± 0.00	94.1 ± 0.00	75.0 ± 0.00	
Prompt 1 (w. Attr. & w/o Attr. names)	80.0 ± 0.00	94.5 ± 0.00	84.2 ± 0.00	
Prompt 2 (w. Attr. & Example Select.)	96.3 ± 0.00	84.7 ± 0.00	100 ± 0.00	

Prompt 1: "Are Product A and Product B the same?"

Prompt 2: "Are Product A and Product B equivalent?"

In this table, we present the results of ablation experiments for entity matching. The F1 scores are calculated for different prompt formats, and the evaluations are conducted on up to 200 samples for cost considerations.

This table provides a comparison of F1 scores for different prompt formats in the context of entity matching. The evaluation is carried out for various datasets, including Beer, iTunes-Amazon, Walmart-Amazon, and Amazon.

Prompt Formatting Second, we observe that FMs can be brittle to subtle variations in prompt templates. We investigate this brittleness by replacing the span “Are Product A and Product B the same?” (Prompt 1) with “Are Product A and Product B equivalent?” (Prompt 2). This minor modification results in an average 9.4 F1 point performance gap in the datasets in Table 4**.**

Task Demonstrations Finally, we find that the choice of task demonstrations has a significant impact on downstream performance. We conduct an ablation where we replace manually curated task demonstrations with randomly selected demonstrations (see Prompt 1 (w/o Example Select.) in Table 4). We run this experiment over three different random seeds and report the results in Table 4. Across all cases, manually curated examples outperform randomly selected examples by an average of 14.7 F1 points. 4.3.2 Discussion. The aforementioned results demonstrate that successful prompt tuning requires (1) selecting an informative set of attributes required for the task, (2) crafting a well-formatted prompt that the FM understands, and (3) constructing a set of instructive task demonstrations that condition the model to the data at hand. For (1), we find that attribute sub-selection boosts performance by removing noisy attributes that hurt performance. For (2), our ablations show that prompt formatting (e.g. word choice, punctuation) can have significant impact on model performance. For (3), our results indicate that examples need to be carefully crafted for FMs to learn new tasks. We conjecture that on more reasoning-intensive tasks (e.g. matching), prompts are important as they help teach the model how to reason about entities and how to complete the task. Moreover, we emphasize that all three steps require some form of iteration to develop the most effective prompt (e.g. passing various inputs to the model and inspecting its outputs). These findings are aligned with existing literature on prompt tuning that observe non-trivial variance in prompt-based learning settings [105]. This performance variance suggests that iterative prompt programming is an essential human-in-the-loop process for FM usage [63]. However, some works suggest that smarter, automatic example selection methods can help close the gap between random example selection and human-in-the-loop prompt selection [62]. These results highlight the paradigm shift induced by building systems centered around FMs: instead of spending time tuning models, we now need to invest time finding the right examples and engineering useful prompts for each task.

7 RESEARCH AGENDA

Because of their natural language interface and vast internal knowledge, FMs provide an interface for unifying a wide-range of siloed, hand-engineered data integration pipelines. Consequently, we envision that the data orchestration workbenches of the future will be centered around FMs. We discuss the opportunities, practical considerations, and technical challenges associated of this vision next.

7.1 Opportunities for FMs

Natural Language Interactions FMs usher in a new era of human machine collaborative workflows wherein users spend less time labeling data and finetuning models and more time writing natural 742 language prompts that are representative of the task at hand. As the necessity of writing code decreases, we envision systems that are more accessible to non-machine learning experts (e.g., business users). In future work, we seek to better understand the human-inthe-loop prompt engineering pipeline, especially in the context of data management practices.

Model Prototyping The data integration and management pipeline can be categorized into three distinct stages: discovery and design, development, and deployment [43]. We propose that FMs will be most useful in the discovery and design phase when training data is less available. In this setting, FMs enable rapid prototyping of data models via prompting. In some cases, the FM's out-of-thebox performance will be sufficiently high, obviating the need to train a task-specific model. In others, we can use the FM to label and generate data with human-in-the-loop feedback. When a sufficient amount of data has been collected, transitioning to the fully-supervised model development regime is the optimal choice.

Passive Learning From User Exhaust In the enterprise setting, organizations accumulate a staggering amount of data exhaust—the informational byproduct that streams from devices, products, and workforce management practices [1]. Because FMs are pre-trained in an unsupervised fashion with a simple token prediction objective , they can learn over any raw and unlabeled sources of data [34, 83, 101]. As such, FMs can effectively ingest the exhaust from the entire data stack (from system logs to structured data). Learning from data analyst exhaust (e.g., clicks over GUI) is also an opportunity to improve FM performance for these data tasks.

7.2 Practical Considerations of FMs

Integration in Data Management Workflows FMs take text as input and generate text as output. As a result, they are limited in their ability to directly take actions over the graphical user interfaces (GUIs) of data management software (DMS) (e.g., Snowflake). Given that a majority of data analyst time is spent working in these environments, we need ways of translating natural language specifications of data tasks (e.g., “remove all nan cells”), to actionable operations on these GUIs. Excitingly, new work demonstrates how to augment FM capabilities such that they can effectively take actions on web and mobile interfaces [10, 95]. These works suggest the possibility of directly integrating FMs with DMS.

Interaction with Existing Systems FMs can interact with existing DMS by either replacing them or utilizing their outputs to conduct downstream tasks. In terms of system replacement, for tasks where the required rules and logic are not encoded in the FMs knowledge, it is an open question [31] as to how to systematically translate the rules and logic (e.g., domain-specific dependencies) to natural language inputs to the FMs. For system integration, we need ways of systematically incorporating the outputs of existing systems (e.g., dataset pattern discovery) in natural language prompts. Prior work has demonstrated how to “ground” FM-based text-to SQL prompts with database schemas and metadata [29]. Similar ideas could be used when incorporating system outputs with FMs.

Debuggability FMs are non-deterministic and can make unexpected errors. In order to use these models in data management pipelines, we need mechanisms for increasing transparency and debuggability of pipelines. One possible approach is to collect and monitor model confidence scores. Prior work demonstrates that a FM can “learn to express uncertainty about its own answers in natural language” [60]. Another approach is to decompose tasks into chains of “primitive operations”, enabling for more transparency and visibility of specific failure points [99].

7.3 Technical Challenges of FMs

Domain Specificity A key challenge in applying FMs to data management tasks is operating over highly specialized domains (e.g., medical, financial, and insurance data). The existing literature suggests that domain-specialization is best achieved by continuously pretraining on relevant data (e.g., earning reports, medical reports) [37, 102]. However, training models with billions of parameters can be costly. This has motivated several works which attempt to more quickly and cheaply adapt models to new domains by finetuning only a few layers of the model or simply training a small neural network (e.g., adaptor) on top of a frozen FM [50].

Privacy Organizations are often unable to pass sensitive data to third party APIs for privacy reasons [13, 15]. Unfortunately, this constraint is in conflict with the contemporary state of FMs where the best performing models (e.g., GPT-3), can only be accessed via API requests. This motivates the need for better open-source models which are competitive with closed-source models. Excitingly, recent work [14] has demonstrated that techniques such as prompt ensembling and prompt reframing can enable open-source models like GPT-J-6B [94] to out-perform GPT3-175B on popular natural language understanding benchmarks. Extending this work to data management tasks is an interesting research direction

Prompt Engineering and Automation Prompting is the primary mechanism for adapting FMs to new tasks. However, prompting requires some manual effort to design performant prompts. In the DI setting, prompts can be sensitive to things such as data schemas and minor formatting variations. A few automated approaches have been developed which reduce the manual effort needed to construct prompts: soft prompt tuning (e.g., optimizing a sequence of taskspecific vectors which are appended to the text prompt) [52, 56] and learning to retrieve better in-context examples [86]. Adapting these works to the challenges of prompting for data tasks is an open area of research.

8 CONCLUSION

In this comprehensive investigation, we delved into the potential of harnessing Factorization Machines in the realm of classical data tasks. Our exploration led to compelling discoveries regarding the capabilities of large FMs, which exhibit state-of-the-art (SoTA) performance across a multitude of data tasks. Even more intriguing is their aptitude for excelling in scenarios where they receive minimal to no natural language task demonstrations. This remarkable adaptability is a testament to the power of these models, which are primarily pretrained for predicting the next words in a sequence.

The significance of our findings is amplified by their implications. By demonstrating that FMs can excel without task-specific fine-tuning, we lay the groundwork for more efficient and versatile approaches in data management. These outcomes not only highlight the potential of advanced models but also underscore the convergence of traditional data tasks with cutting-edge natural language processing (NLP) techniques.

Our research stands on the shoulders of years of dedicated work within the data management community, particularly in tasks related to integration and data cleaning. We extend the boundaries of this established domain, showcasing how language-guided models can enrich human-in-the-loop data integration practices. Our work acts as a bridge, connecting two previously distinct domains, and suggests that the advantages of NLP models may be harnessed across a broader range of data management tasks.

As we look ahead, our findings resonate with the vision of data management practices intertwined with advanced AI capabilities. The symbiotic relationship between humans and AI, particularly language models, has the potential to revolutionize how we approach data-related challenges. Beyond the specific tasks we explored, we envision a future where data integration becomes more streamlined, data cleaning more efficient, and data management more accessible across a spectrum of domains and applications.

This work not only underscores the transformative potential of advanced language models but also invites further inquiry and experimentation. It is a step toward realizing the vision of intelligent, language-guided data management practices that leverage the power of AI to tackle the diverse and evolving data needs of our digital age.

REFERENCES

[1] 2018. How to turn data exhaust into a competitive edge. https: //knowledge.wharton.upenn.edu/article/turn-iot-data-exhaust-nextcompetitive-advantage/
[2] 2020. Federal Judicial Caseload Statistics 2020. https://www.uscourts.gov/ statistics-reports/federal-judicial-caseload-statistics-2020
[3] 2022. California Consumer Privacy Act (CCPA). https://oag.ca.gov/privacy/ccpa
[4] 2022. Decreasing cost of storage. https://www.iotone.com/term/decreasingcost-of-storage/t172
[5] 2022. Tamr | Enterprise Data Mastering at Scale - Tamr Inc. https://www.tamr. com/
[6] 2022. Trifacta: Data Wrangling Software and Tools. https://www.trifacta.com/
[7] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? Proceedings of the VLDB Endowment 9, 12 (2016), 993–1004.
[8] Abubakar Abid, Maheen Farooqi, and James Zou. 2021. Large language models associate Muslims with violence. Nature Machine Intelligence 3, 6 (2021), 461– 463.
[9] Amina Adadi. 2021. A survey on data-efficient algorithms in big data era. Journal of Big Data 8, 1 (2021), 1–54.
[10] Adept. [n. d.]. ACT-1: Transformer for Actions. https://www.adept.ai/act
[11] Julia Adler-Milstein, Jason S Adelman, Ming Tai-Seale, Vimla L Patel, and Chris Dymek. 2020. EHR audit logs: a new goldmine for health services research? Journal of biomedical informatics 101 (2020), 103343.
[12] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. 2022. Flamingo: a Visual Language Model for Few-Shot Learning. arXiv preprint arXiv:2204.14198 (2022).
[13] Simran Arora, Patrick Lewis, Angela Fan, Jacob Kahn, and Christopher Ré. 2022. Reasoning over Public and Private Data in Retrieval-Based Systems. arXiv:2203.11027 [cs.IR]
[14] Simran Arora, Avanika Narayan, Mayee F Chen, Laurel J Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. 2022. Ask Me Anything: A simple strategy for prompting language models. arXiv preprint arXiv:2210.02441 (2022).
[15] Simran Arora and Christopher Ré. 2022. Can Foundation Models Help Us Achieve Perfect Secrecy? arXiv preprint arXiv:2205.13722 (2022).
[16] Felix Biessmann, Tammo Rukat, Phillipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. 2019. DataWig: Missing Value Imputation for Tables. Journal of Machine Learning Research 20, 175 (2019), 1–6. http://jmlr.org/papers/v20/18-753.html [17] Felix Biessmann, Tammo Rukat, Philipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. 2019. DataWig: Missing Value Imputation for Tables. J. Mach. Learn. Res. 20, 175 (2019), 1–6.
[18] Sid Black, Gao Leo, Phil Wang, Connor Leahy, and Stella Biderman. 2021. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow. https: //doi.org/10.5281/zenodo.5297715
[19] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258 (2021).
[20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. Advances in neural information processing systems 33 (2020), 1877–1901.

[21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021).

[22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311 (2022).

[23] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In 2013 IEEE 29th International Conference on Data Engineering (ICDE). IEEE, 458–469.

[24] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. Katara: A data cleaning system powered by knowledge
bases and crowdsourcing. In Proceedings of the 2015 ACM SIGMOD international
conference on management of data. 1247–1261.

[25] EasyClosets Coupons. 2021. 21 best GPT-3 tools, examples and use cases nogood™: Growth Marketing Agency. https://nogood.io/2021/06/25/gpt-3tools/

[26] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In Proceedings of the 2013 ACM SIGMOD International Conference
on Management of Data. 541–552.

[27] Nilesh Dalvi, Vibhor Rastogi, Anirban Dasgupta, Anish Das Sarma, and Tamás Sarlós. 2013. Optimal hashing schemes for entity matching. In Proceedings of
the 22nd international conference on world wide web. 295–306.

[28] Tamraparni Dasu and Ji Meng Loh. 2012. Statistical Distortion: Consequences
of Data Cleaning. Proceedings of the VLDB Endowment 5, 11 (2012).

[29] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. [n. d.]. Structure-Grounded Pretraining for Text-to-SQL. ([n. d.]).

[30] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. ACM SIGMOD Record 51, 1 (2022), 33–40.

[31] Michael Desmond, Evelyn Duesterwald, Vatche Isahagian, and Vinod Muthusamy. 2022. A No-Code Low-Code Paradigm for Authoring Business Automations Using Natural Language. arXiv preprint arXiv:2207.10648 (2022).

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).

[33] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[34] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. 2020. Jukebox: A Generative Model for Music. CoRR abs/2005.00341 (2020). https://arxiv.org/abs/2005.00341

[35] Eric Ghysels, Arthur Sinko, and Rossen Valkanov. 2007. MIDAS regressions: Further results and new directions. Econometric reviews 26, 1 (2007), 53–90.

[36] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-off crowdsourc-
ing for entity matching. In Proceedings of the 2014 ACM SIGMOD international
conference on Management of data. 601–612.

[37] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't stop pretraining: adapt language
models to domains and tasks. arXiv preprint arXiv:2004.10964 (2020).

[38] Suchin Gururangan, Ana Marasovic, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL). 8342–8360.

[39] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. arXiv preprint arXiv:2002.08909 (2020).
[40] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. 2018. Transform-Data-by-Example (TDE): An Extensible Search
Engine for Data Transformations. Proceedings of the VLDB Endowment 11, 10
(2018).
[41] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In Proceedings of the 2019
International Conference on Management of Data. 829–846.
[42] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP.
https://doi.org/10.48550/ARXIV.
1902.00751
[43] IBM. [n. d.]. Data Integration Phases. https://www.ibm.com/docs/en/iis/11.7?
topic=SSZJPZ_11.7.0%2Fcom.ibm.swg.im.iis.productization.iisinfsv.overview.
doc%2Ftopics%2Fcisocapabilities.html
[44] Xisen Jin, Dejiao Zhang, Henghui Zhu, Wei Xiao, Shang-Wen Li, Xiaokai Wei, Andrew Arnold, and Xiang Ren. 2021. Lifelong pretraining: Continually adapting language models to emerging corpora. arXiv preprint arXiv:2110.08534
(2021).
[45] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In Proceed-
ings of the sigchi conference on human factors in computing systems. 3363–3372.
[46] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020).
[47] Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, et al. 2022. MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. arXiv preprint arXiv:2205.00445 (2022).
[48] Zixuan Ke, Bing Liu, Nianzu Ma, Hu Xu, and Lei Shu. 2021. Achieving Forgetting Prevention and Knowledge Transfer in Continual Learning. Advances in Neural Information Processing Systems 34 (2021).
[49] Pradap Konda, Sanjib Das, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, et al. 2016.
Magellan: toward building entity matching management systems over data science stacks. Proceedings of the VLDB Endowment 9, 13 (2016), 1581–1584.
[50] AI21 Labs. 2022. STANDING ON THE SHOULDERS OF GIANT FROZEN LANGUAGE MODELS. preprint (2022).
[51] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush
Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In International Conference on Learning Representations.
https://openreview.net/forum?id=H1eA7AEtvS
[52] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for
Parameter-Efficient Prompt Tuning. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 3045–3059. https://doi.org/10.18653/v1/2021.emnlp-main.243
[53] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for
Parameter-Efficient Prompt Tuning. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 3045–3059.
[54] Yoav Levine, Itay Dalmedigos, Ori Ram, Yoel Zeldes, Daniel Jannai, Dor Muhlgay, Yoni Osin, Opher Lieber, Barak Lenz, Shai Shalev-Shwartz, et al. 2022. Standing on the shoulders of giant frozen language models. arXiv preprint arXiv:2204.10019 (2022).

[55] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference
on Natural Language Processing (Volume 1: Long Papers). 4582–4597.
[56] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). Association for Computational Linguistics, Online, 4582–4597. https://doi.org/10.18653/v1/2021.acllong.353
[57] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. Proceedings of
the VLDB Endowment 14, 1 (2020), 50–60.
[58] Paul Pu Liang, Chiyu Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2021. Towards understanding and mitigating social biases in language models.
In International Conference on Machine Learning. PMLR, 6565–6576.
[59] Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. TruthfulQA: Measuring How Models Mimic Human Falsehoods. (2021).
[60] Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Teaching Models to Express Their Uncertainty in Words. arXiv preprint arXiv:2205.14334 (2022).
[61] Alisa Liu, Swabha Swayamdipta, Noah A Smith, and Yejin Choi. 2022. WANLI: Worker and AI Collaboration for Natural Language Inference Dataset Creation. arXiv preprint arXiv:2201.05955 (2022).
[62] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What Makes Good In-Context Examples for GPT-3? arXiv preprint arXiv:2101.06804 (2021).
[63] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. arXiv preprint arXiv:2107.13586 (2021).
[64] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019).
[65] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017).
[66] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming
few-shot prompt order sensitivity. arXiv preprint arXiv:2104.08786 (2021).
[67] Li Lucy and David Bamman. 2021. Gender and Representation Bias in GPT-3 Generated Stories. NAACL HLT 2021 (2021), 48.
[68] Rabeeh Karimi Mahabadi, Luke Zettlemoyer, James Henderson, Marzieh Saeidi, Lambert Mathias, Veselin Stoyanov, and Majid Yazdani. 2022. PERFECT: Promptfree and Efficient Language Model Fine-Tuning. In Proceedings of the 60th Annual
Meeting of the Association for Computational Linguistics (ACL).
[69] Bernard Marr. 2021. What is unstructured data and why is it so important to businesses? https://www.forbes.com/sites/bernardmarr/2019/10/16/whatis-unstructured-data-and-why-is-it-so-important-to-businesses-an-easyexplanation-for-anyone/?sh=999b04d15f64
[70] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. 2010. ERACER: a database approach for statistical inference and data cleaning. In Proceedings of the 2010
ACM SIGMOD International Conference on Management of data. 75–86.
[71] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On Faithfulness and Factuality in Abstractive Summarization. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL). 1906–1919.
[72] Yinan Mei, Shaoxu Song, Chenguang Fang, Haifeng Yang, Jingyun Fang, and Jiang Long. 2021. Capturing Semantics for Imputation with Pre-trained Language Models. In 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 61–72.
[73] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D

Manning. 2022. Fast Model Editing at Scale. In International Conference on
Learning Representations. https://openreview.net/forum?id=0DcZxeWfOPt [74] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In Proceedings
of the 2018 International Conference on Management of Data. 19–34.
[75] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? arXiv preprint arXiv:2205.09911 (2022).
[76] OpenAI. 2021. OpenAI API. https://openai.com/api/
[77] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas.
2020. Blocking and filtering techniques for entity resolution: A survey. ACM
Computing Surveys (CSUR) 53, 2 (2020), 1–42.
[78] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In Proceedings of the 2018 Conference of the North American
Chapter of the Association for Computational Linguistics: Human Language
Technologies, Volume 1 (Long Papers). Association for Computational Linguistics, New Orleans, Louisiana, 2227–2237. https://doi.org/10.18653/v1/N18-1202
[79] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin,
Yuxiang Wu, and Alexander Miller. 2019. Language Models as Knowledge
Bases?. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 2463–2473.
[80] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal
of Machine Learning Research 21 (2020), 1–67.
[81] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 3505–3506.
[82] Simon Razniewski, Andrew Yates, Nora Kassner, and Gerhard Weikum. 2021. Language Models As or For Knowledge Bases. arXiv preprint arXiv:2110.04888
(2021).
[83] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. 2022. A Generalist Agent. arXiv preprint arXiv:2205.06175 (2022).
[84] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. Proceedings of the VLDB Endowment 10, 11 (2017).
[85] Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In Extended Abstracts of the
2021. CHI Conference on Human Factors in Computing Systems. 1–7.
[86] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2021. Learning to retrieve
prompts for in-context learning. arXiv preprint arXiv:2112.08633 (2021).
[87] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. 2022. Multitask Prompted Training Enables Zero-Shot Task Generalization. In
International Conference on Learning Representations. https://openreview.net/ forum?id=9Vrb9D0WI4
[88] Timo Schick and Hinrich Schütze. 2021. It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2339–2352.
[89] Michael Stonebraker, Daniel Bruckner, Ihab F Ilyas, George Beskales, Mitch Cherniack, Stanley B Zdonik, Alexander Pagan, and Shan Xu. 2013. Data Curation at Scale: The Data Tamer System.. In Cidr, Vol. 2013. Citeseer.

[90] Fan-Keng Sun and Cheng-I Lai. 2020. Conditioned natural language generation using only unconditioned language model: An exploration. arXiv preprint arXiv:2011.07347 (2020).
[91] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie 2.0: A continual pre-training framework for language understanding. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 8968–8975.
[92] Edhy Sutanta, Retantyo Wardoyo, Khabib Mustofa, and Edi Winarko. 2016. Survey: Models and Prototypes of Schema Matching. International Journal of Electrical & Computer Engineering (2088-8708) 6, 3 (2016).
[93] P. Voigt and A. Von dem Bussche. 2017. The EU General Data Protection Regulation (GDPR). In Springer International Publishing. https://doi.org/10. 1007/978-3-319-57959-7
[94] Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformerjax.
[95] Bryan Wang, Gang Li, and Yang Li. 2022. Enabling Conversational Interaction with Mobile UI using Large Language Models. arXiv preprint arXiv:2209.08655 (2022).
[96] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing Entity Resolution. Proceedings of the VLDB Endowment 5, 11 (2012).
[97] Albert Webson and Ellie Pavlick. 2021. Do Prompt-Based Models Really Under-
stand the Meaning of their Prompts? arXiv preprint arXiv:2109.01247 (2021).
[98] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large
language models. arXiv preprint arXiv:2201.11903 (2022).
[99] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. Ai chains: Transparent and controllable human-ai interaction by chaining large language model
prompts. In CHI Conference on Human Factors in Computing Systems. 1–22.
[100] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A Systematic Evaluation of Large Language Models of Code. In Deep Learning for Code Workshop.
[101] Wilson Yan, Yunzhi Zhang, Pieter Abbeel, and Aravind Srinivas. 2021. Videogpt: Video generation using vq-vae and transformers. arXiv preprint arXiv:2104.10157 (2021).
[102] Yi Yang, Mark Christopher Siy Uy, and Allen Huang. 2020. Finbert: A pretrained language model for financial communications. arXiv preprint arXiv:2006.08097 (2020).
[103] Ehtisham Zaidi and Sharat Menon. 2022. Magic Quadrant for Data Integration Tools.
[104] Jing Zhang, Bonggun Shin, Jinho D Choi, and Joyce C Ho. 2021. SMAT: An attention-based deep learning solution to the automation of schema matching.
Advances in databases and information systems. ADBIS 12843 (2021), 260.
[105] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In International Conference on Machine Learning. PMLR, 12697–12706.