# KNOWTRANS: Boosting Transferability of Data Preparation LLMs via Knowledge Augmentation

Yuhang Ge†, Fengyu Li†, Yuren Mao†*, Yanbo Yang†, Congcong Ge†, Zhaoqiang Chen♯, Jiang Long†, Yunjun Gao†
†*Zhejiang University, Hangzhou, China*
♯*Huawei Cloud Computing Technologies Co., Ltd, China*
†{*yuhangge,fengyuli,yuren.mao,yanboyang,gcc,longjiang1,gaoyj*}*@zju.edu.cn* ♯*chenzhaoqiang1@huawei.com*

***Abstract*—Data Preparation (DP), which involves tasks such as data cleaning, imputation and integration, is a fundamental process in data-driven applications. Recently, Large Language Models (LLMs) fine-tuned for DP tasks, i.e., DP-LLMs, have achieved state-of-the-art performance. However, transferring DP-LLMs to novel datasets and tasks typically requires a substantial amount of labeled data, which is impractical in many real-world scenarios. To address this, we propose a knowledge augmentation framework for data preparation, dubbed KNOWTRANS. This framework allows DP-LLMs to be transferred to novel datasets and tasks with a few data points, significantly decreasing the dependence on extensive labeled data. KNOWTRANS comprises two components: *Selective Knowledge Concentration* and *Automatic Knowledge Bridging*. The first component re-uses knowledge from previously learned tasks, while the second automatically integrates additional knowledge from external sources. Extensive experiments on 13 datasets demonstrate the effectiveness of KNOWTRANS. KNOWTRANS boosts the performance of the state-of-the-art DP-LLM, Jellyfish-7B, by an average of 4.93%, enabling it to outperform both GPT-4 and GPT-4o.**

***Index Terms*—data preparation, large language model**

## I. INTRODUCTION

Data Preparation (DP) aims to transform raw data into a form that is ready for analysis, which is one of the most important problems in data management [2]. It often requires extensive manual intervention and is time-consuming. To avoid the labor-intensive operations, deep learning models have been widely used to automatically solve the data preparation tasks, such as Ditto [3] for Entity Matching (EM), SMAT [3] for Schema Matching (SM), Doduo [4] for Column Type Annotation (CTA) and IPM [5] for Data Imputation (DI). However, these models are typically dataset-specific or task-specific, necessitating specialized model training on each dataset. Consequently, the transferability of these methods is constrained.

To improve the transferability, recent studies focus on unifying diverse input formats into a text-to-text form, and fine-tuning open-source Large Language Models (LLMs) to handle multiple DP tasks concurrently [6]. These methods typically implement multi-task supervised fine-tuning (SFT) on various data preparation datasets, achieving state-of-the-art performance [7, 8, 9, 10]. We refer to this multi-task SFT stage as *upstream learning*, and the fine-tuned LLMs as *upstream DP-LLMs*. Although upstream DP-LLMs can perform well on trained datasets, they still require a significant amount of labeled data to adapt to new datasets and tasks. It is often expensive and impractical in many real-world situations. To relieve the dependence on labeled data, MELD [10] proposes a heuristic data augmentation technique in few-shot settings. However, it still requires a significant number of labeled examples, which is typically up to 10% of the dataset. In some cases, thousands of labeled samples are needed, such as for SM, CTA, and DI. Overall, the existing methods are data-greedy and have inferior few-shot transferability.

The code and full version of this paper can be found in [1].

To boost the few-shot transferability of DP-LLMs, this paper investigates the reasons that necessitate a large number of labeled data and proposes a novel framework for adapting DP-LLMs to novel datasets and tasks with a small amount of data (e.g., only twenty labeled data). We present that inferior few-shot transferability of DP-LLMs is primarily caused by **Cross-Datasets/Tasks Knowledge Distraction** and **Dataset-Informed Knowledge Gap**.

- **Cross-Datasets/Tasks Knowledge Distraction.** During upstream learning, DP-LLM simultaneously learns from multiple data preparation datasets or tasks, each containing unique knowledge. However, the fine-tuning process forces the model to update all datasets/tasks within a shared parameter space, leading to gradient conflicts. As shown in Fig. 1 (left), when learning two datasets/tasks, $\mathcal{L}_1$ and $\mathcal{L}_2$, conflicting gradient update directions can create a "tug-of-war" effect [11, 12], preventing the model parameters from converging effectively to an optimal state. Ultimately, this conflict results in overlapping parameter representations for different datasets/tasks. When we perform cross-datasets/tasks transfer, DP-LLM is susceptible to interference from the knowledge of others stored in the model weights. We refer to this phenomenon as "knowledge distraction".
- **Dataset-Informed Knowledge Gap.** Different data preparation datasets may have unique data-informed rules for processing, which is hard to capture from limited labeled samples. In Fig. 1 (right), we illustrate an example using the EM dataset Walmart-amazon, where the discrimination of whether two entities match typically involves the following knowledge: 1) primary identifiers are the product's model numbers; 2) in case of missing or "NaN" values, focus on comparing other attributes; 3) product prices can be disregarded. However, if the DP-LLM model has not encountered this dataset during upstream learning, it may lack this specific knowledge. Previous methods have tried to manually construct this knowledge [7], but this approach is both labor-intensive and costly. Thus, this gap hinders the DP-LLM's performance in a few-shot setting.

2375-026X/25/$31.00 ©2025 IEEE
DOI 10.1109/ICDE65448.2025.00214

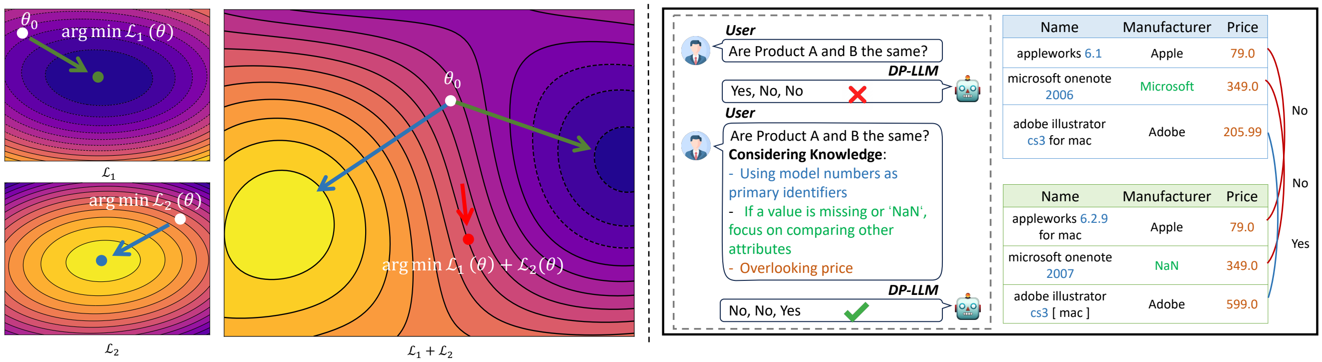



Fig. 1: (Left) Illustration of the loss landscape for $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_1 + \mathcal{L}_2$. The "tug-of-war" effect arises when conducting multi-task upstream learning, such as the two tasks in $\mathcal{L}_1 + \mathcal{L}_2$ have nearly opposite gradient update directions, i.e. the angle is obtuse. (Right) Comparison without v.s. with knowledge for DP-LLM on EM task. DP-LLM cannot handle all cases without knowledge, but it succeeds when knowledge is applied. Text colors represent corresponding matched knowledge.

To address the above issues, we propose a novel framework, **Know**ledge Augmentation for boosting the few-shot **Trans**ferability of DP-LLM, dubbed KNOWTRANS. KNOWTRANS only requires a small amount of labeled data—specifically, just twenty labeled samples. KNOWTRANS integrates two essential components: the *Selective Knowledge Concentration* (SKC) and the *Automatic Knowledge Bridging* (AKB). Specifically, SKC extracts knowledge from the upstream dataset into modular "knowledge patches" that are isolated from each other in parameter space. These knowledge patches are integrated into the upstream model and selectively re-used for few-shot fine-tuning, ensuring that the DP-LLM concentrates on valuable knowledge for new datasets and tasks. On the other hand, AKB employs a more capable closed-source LLM (e.g., GPT-4o) to incorporate knowledge derived from the dataset. Through an iterative process driven by an error feedback mechanism, it generates accurate, dataset-informed knowledge. This optimized knowledge is subsequently utilized as supplementary prompts during inference, effectively addressing the dataset-informed knowledge gap.

We conducted extensive experiments on 13 datasets, including both novel datasets and tasks in a few-shot setting (e.g., 20 samples per dataset). Our results show that the KNOWTRANS significantly improves the transferability of 4 different backbone DP-LLMs across datasets. For example, KNOWTRANS improves the state-of-the-art DP-LLM Jellyfish-7B's performance by **4.93%** on average, surpassing other DP-LLMs of the same size and outperforming GPT-4 and GPT-4o. Furthermore, KNOWTRANS boosts Jellyfish-13B by **6.1%**, significantly outperforming GPT-4 and GPT-4o by **7.03%** and **6.07%**, respectively. Furthermore, the framework shows impressive scalability, consistently outperforming the performance of the backbone DP-LLM as the labeled sample size increases.

## II. PRELIMINARIES

### A. Large Language Model

Large Language Models (LLMs), such as GPT-3 [13], LLaMA [14] and Mistral [15], are pre-trained on large-scale amounts of data, acquiring extensive world knowledge [16]. Thanks to the scale-up of parameters and data, LLMs exhibit emergent abilities [17], which refer to unexpected capabilities that arise only when the model reaches a certain scale, allowing it to perform tasks it was not explicitly trained for. As a result, these models demonstrate remarkable generalization capabilities across various natural language processing tasks [18]. Recently, some works [19, 20] begin to apply LLMs to data preparation tasks. However, LLMs are trained on general text data, which remains a gap between tabular data used in DP tasks. To bridge the gap, instruction tuning is commonly employed to adapt LLMs to DP tasks to achieve better performance.

**Definition 1.** *(Instruction Tuning) A learning paradigm where an LLM is trained to follow natural language instructions for a variety of tasks by fine-tuning instruction datasets. The LLM learns to generalize across tasks by understanding and performing based on the provided instructions. For instance, the prompt below demonstrates a task where the model is asked to generate the antonym of a given word according to the instruction [21].*

$$\textit{LLM}(\underbrace{\textit{Instruction: ``Give the antonym of fast.''}}_{\textbf{\textit{Instruction}}}) \rightarrow \underbrace{\textit{slow}}_{\textbf{\textit{Response}}}$$

### B. Multi-Task Learning

Multi-task learning is a training paradigm where a model learns multiple tasks simultaneously, using different datasets for each task. The aim is to improve the model's performance on all tasks by sharing useful information across them. Given a set of **tasks** $\mathcal{T}$ with associated datasets $\mathcal{D}$, a **multi-task model** $f$ is trained to be capable of handling all tasks [11, 22, 23].

### C. Model Fusion

Model Fusion (a.k.a. model merging) aims to integrate the parameters of multiple pre-trained models into a single model. This process combines the knowledge and capabilities of the individual models, resulting in a unified model that exhibits enhanced performance and generalization relative to the constituent models [24, 25, 26].

**Vanilla Model Fusion.** The vanilla model fusion method is a straightforward method where parameters from multiple models are combined by averaging. Formally, given a set of models $\{M_1, M_2, \ldots, M_n\}$, where each model $M_i$ is represented by a parameter set $\boldsymbol{\theta}_i$, the averaged merged model $M^\star$ is defined by the parameter vector $\boldsymbol{\theta}^\star$, formulated as follows:

$$\boldsymbol{\theta}^\star = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\theta}_i, \tag{1}$$

where $n$ is the number of models being merged. This equation signifies that each parameter of $\boldsymbol{\theta}^\star$ is the arithmetic mean of the corresponding parameters across all models.

## III. Problem Definition

Data preparation (DP) typically involves cleaning, transforming and organizing raw data for analysis. Formally, let $T$ be a table (e.g., a relational or web table) with rows $\{r_1, r_2, \dots\}$ and columns $\{c_1, c_2, \dots\}$. Each cell $v_{i,j}$ represents the value in row $i$ and column $j$. We define a set of DP tasks, each with inputs $\mathcal{X}$ and labels $\mathcal{Y}$. A multi-task model $f$ is applied to each task, taking specific inputs and producing corresponding labels. This work focuses on the following DP tasks:

**Entity Matching (EM).** Given two rows $r_1$ and $r_2$ from table $T$, determine whether they refer to the same real-world entity.

**Data Imputation (DI).** Given a missing cell value $v_{i,j}$ of row $r$, infer the missing value based on other values.

**Schema Matching (SM).** Given a pair of column names of $(c_j, c_k)$ with its corresponding description $(d_j, d_k)$, determine whether they refer to the same attribute.

**Error Detection (ED).** Given a row $r$ and a specific cell $v_{i,j}$, identify whether the value in $v_{i,j}$ is erroneous based on the other cell values in the same row.

**Data Cleaning (DC).** Given a row $r$ and a specific erroneous cell $v_{i,j}$, correct the erroneous based on other cell values.

**Column Type Annotation (CTA).** Given a column $c_j$ in table $T$, assign a semantic type $\mathcal{C}$ to the entire column.

**Attribute Value Extraction (AVE).** Given a text $s$ and a target attribute $c_j$, extract the corresponding value $v_j$ from $s$.

To develop a versatile LLM capable of addressing diverse data preparation tasks, it is essential to train a multi-task **upstream model**. Let $N$ upstream tasks $\{\mathcal{T}_1, \dots, \mathcal{T}_N\}$ be associated with a collection of labeled datasets $\mathcal{D}$, referred to as upstream data. A multi-task **upstream DP-LLM** can be trained using these upstream datasets. However, despite its capability to handle multiple tasks, the upstream DP-LLM often struggles to generalize to novel datasets and tasks with limited labeled data. This limitation primarily arises due to cross-dataset/task knowledge distraction and dataset-informed knowledge gaps. To address these challenges, our goal is to enhance the transferability of DP-LLMs in few-shot settings.

**Few-Shot DP-LLM Transfer.** Building on the upstream DP-LLM, we introduce a set of novel **downstream data**, comprising novel datasets $\{\mathcal{D}'_1, \mathcal{D}'_2, \dots\}$ and tasks $\{\mathcal{T}'_1, \mathcal{T}'_2, \dots\}$ with few-shot labeled samples. Our goal is to boost the transferability of the upstream DP-LLM, allowing it to adapt effectively to novel downstream data after a few-shot fine-tuning. Importantly, these downstream data are not included in the backbone upstream DP-LLM, ensuring a fair evaluation of transferability.

## IV. Overview

Fig. 2 illustrates the knowledge augmentation KnowTrans. KnowTrans consists of two key components: the *Selective Knowledge Concentration* (SKC) component and the *Automatic Knowledge Bridging* (AKB) component. They work at training time and inference time, respectively. In the SKC component, the upstream data are reused, enabling the upstream DP-LLM to concentrate on relevant knowledge when fine-tuning the novel downstream data and mitigating the knowledge distraction issue. The AKB component generates proper knowledge as supplementary prompts for each novel dataset during inference to alleviate the dataset-informed knowledge gap.

**SKC Component.** In this component, we propose a novel method to fine-tune the upstream DP-LLM to concentrate on relevant knowledge for novel downstream data, with the following three stages: ① *Upstream knowledge patches extraction*. It first dynamically extracts knowledge of the upstream datasets into independent knowledge patches, using a cross-model low-rank parameterize method. ② *Dynamic knowledge patches Fusion*. It incorporates extracted knowledge patches and an additional new knowledge patch, dynamically weighting them to integrate transferable knowledge from upstream datasets and shared knowledge for the new data. ③ *Few-shot fine-tuning*. It keeps the upstream model fixed and fine-tunes only the knowledge patches and weights on few-shot downstream data. After these stages, the upstream model effectively concentrates relevant knowledge for novel datasets and tasks.

**AKB Component.** This component automatically generates dataset-informed knowledge in an iterative process, which involves the following four steps: ④ *Generation*. A subset of the dataset is sampled and transformed into prompt-response pairs. These pairs are concatenated with a seed prompt containing initial knowledge and fed into a more capable closed-source LLM to generate the initial knowledge candidates pool. ⑤ *Evaluation*. It evaluates these knowledge candidates using the fine-tuned DP-LLM on a validation set, identifies the best knowledge, and collects error cases. ⑥ *Feedback*. It generates error feedback information based on these error cases. ⑦ *Refinement*. It refines the knowledge based on error feedback and then incorporates the refined knowledge into the pool for the next iteration. After completing the iteration, the most appropriate data-informed knowledge should be generated to closely align with the dataset characteristics.

Together, these components form an integrated framework that effectively enhances the transferability of DP-LLMs by optimizing both training and inference, resulting in better performance on novel downstream data.

## V. Selective Knowledge Concentration

This section introduces the Selective Knowledge Concentration (SKC) component, which enables the DP-LLM to concentrate knowledge relevant to novel datasets and tasks. Fig.2 (left) illustrates an overview of the component.

### A. Upstream Knowledge Patches Extraction

To prevent conflicts in the parameter space, we extract knowledge from upstream datasets and store it in separate

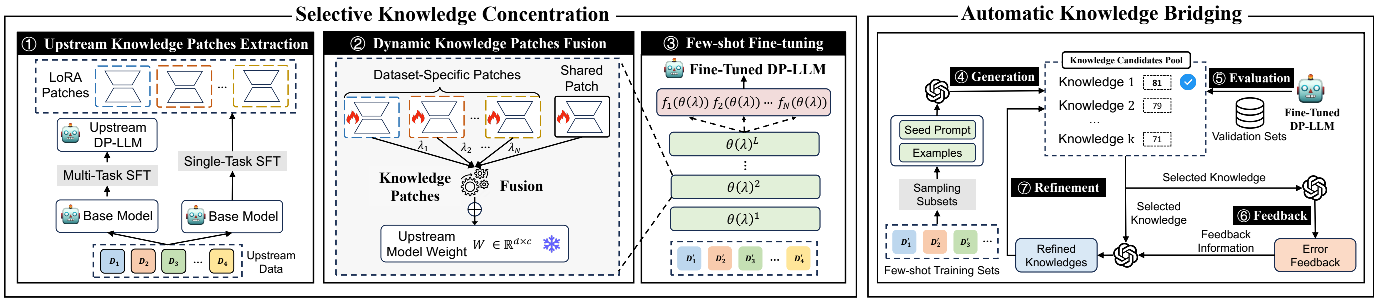



Fig. 2: The illustration of our proposed knowledge augmentation framework KnowTrans, which consists of two components: Selective Knowledge Concentration (SKC) component and Automatic Knowledge Bridging (AKB) component.

knowledge patches. These patches remain isolated, ensuring no overlap. They can then be reused to transfer dataset-specific knowledge to downstream data. However, extracting knowledge patches from an upstream DP-LLM is challenging since the model has already been fine-tuned on upstream data. Further fine-tuning does not effectively isolate valuable information into knowledge patches. To address this challenge, we propose reusing the upstream data and fine-tuning it on the base model of the upstream DP-LLM using LoRA [27]. For instance, if we employ the upstream DP-LLM Jellyfish-7B [7], the corresponding base model can be Mistral-7B [15]. We refer to the LoRA module obtained after fine-tuning as a "knowledge patch". LoRA's plug-and-play nature makes it an ideal choice for serving as a modular knowledge patch, and its parameter efficiency allows us to maintain only the lightweight module for each dataset. This approach benefits from the shared architecture and pre-trained data between the base and upstream DP-LLMs, enabling an effective transfer of knowledge patches from the base model to the upstream DP-LLM.

**Cross-Model Low-Rank Parameterize.** Assume we have a base model of the upstream DP-LLM. Given a pre-trained weight matrix $\boldsymbol{\theta}_0 \in \mathbb{R}^{d\times c}$ of the base model and a various of upstream tasks $\{\mathcal{T}_1, \cdots, \mathcal{T}_N\}$. For each upstream task $\mathcal{T}_i$, we initialize a module $\boldsymbol{\Delta W}_i$ on the base model and fine-tune it with data $\mathcal{D}_i$ using a low-rank parameter update, i.e.,

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_0 + \alpha \boldsymbol{\Delta W}_i = \boldsymbol{\theta}_0 + \alpha \boldsymbol{B}_i \boldsymbol{A}_i. \qquad (2)$$

This allows $\boldsymbol{\Delta W}_i$ to extract knowledge patches from each dataset by introducing only a low-rank transformation $\boldsymbol{\Delta W} = \boldsymbol{B} \times \boldsymbol{A}$ for effective knowledge encoding, where $\boldsymbol{B} \in \mathbb{R}^{d\times r}$ and $\boldsymbol{A} \in \mathbb{R}^{r\times c}$ with rank $r \ll min(d, c)$. $\boldsymbol{B}$ and $\boldsymbol{A}$ are initialized with a random Gaussian distribution and zero respectively. $\alpha$ is the scaling factor that controls the strength of the updates.

**Knowledge Patch Training.** To train knowledge patch for each upstream task $\mathcal{T}_i$, we keep the main transformer layers of $\mathcal{M}_\text{b}$ fixed, while updating only the parameters in $\boldsymbol{\Delta W}_i$. The training objective can be formulated as the standard maximum-likelihood training for conditional language modeling, i.e.,

$$\min_{\boldsymbol{\Delta W}_i} \sum_{(x,y)\in\mathcal{D}_i} \sum_{t=1}^{|y|} -\log\left(P_{\boldsymbol{\theta}_0+\boldsymbol{\Delta W}_i}(y_t \mid x, y_{<t})\right), \qquad (3)$$

where $|y|$ is the length of the output sequence $y$, and $y_t$ is the target token at position $t$ in the output sequence. Fig. 2 ① depicts the training process of our cross-model knowledge patch training. We use the dataset from the Jellyfish-Instruct [7] for training these knowledge patches, detailed in Section VII-A. Our experiments in Section VII-B show that this cross-model training method substantially enhances the performance of novel datasets and tasks. This confirms that the analogous base model can effectively serve as a proxy for creating transferable knowledge patches. After training, these knowledge patches are then integrated into the upstream DP-LLM, providing an initialization point for subsequent dynamic fusion.

### *B. Dynamic Knowledge Patch Fusion*

To ensure that the DP-LLM focuses on relevant knowledge for new datasets and tasks, we propose reusing the extracted knowledge patches during fine-tuning. To achieve this, we introduce a dynamic knowledge patch fusion module that seamlessly integrates upstream knowledge patches while incorporating newly added knowledge patches. By fusing these patches through a weighted ensemble approach, the module dynamically combines dataset-specific transferable knowledge with shared knowledge, enabling upstream DP-LLM effective transfer to novel datasets and tasks.

**Knowledge Patches Fusion.** Formally, given a pre-trained weight matrix $\hat{\boldsymbol{\theta}}_0$ of upstream DP-LLM, we learn $\boldsymbol{\lambda} = [\lambda_1, \cdots, \lambda_N]$ that control the contribution of each upstream knowledge patch to the model, i.e.,

$$\hat{\boldsymbol{\theta}}(\boldsymbol{\lambda}) = \hat{\boldsymbol{\theta}}_0 + \alpha\left(\sum_{i=1}^{N} \lambda_i \boldsymbol{\Delta W}_i + \boldsymbol{\Delta W}_{N+1}\right). \qquad (4)$$

Intuitively, $\hat{\boldsymbol{\theta}}_0$ is the backbone model, and each $\Delta W_N$ is the upstream knowledge patch with dataset-specific knowledge while $\Delta W_{N+1}$ is the newly added knowledge patch that is expected to capture the shared information across multiple downstream datasets. The knowledge patch fusion module attempts to elicit the most relevant knowledge from upstream tasks and transfer it to downstream datasets, effectively preventing the issue of knowledge distraction. The illustration of the combination process can be found in Fig. 2 ②.

**Remark.** Compared to traditional multi-task SFT, the weighted ensemble approach for knowledge patches fusion offers several advantages: (ii) Avoiding Gradient Conflicts: multi-task SFT updates all tasks in a shared parameter space, leading to gradient conflicts. Our approach isolates task-specific knowledge in modular knowledge patches, preventing interference.

TABLE I: Structured explanation of the input and output for the SKC component.

| Stage | Input | Output |
|---|---|---|
| **1. Upstream Knowledge Patches Extraction** | Pre-trained base model $\boldsymbol{\theta_0}$ and multiple upstream task datasets $\{\mathcal{D}_1, \mathcal{D}_{\in}, ..., \mathcal{D}_N\}$. | LoRA-based knowledge patches $\{\boldsymbol{\Delta W_1}, ..., \boldsymbol{\Delta W_N}\}$ extract for each datasets. |
| **2. Dynamic Knowledge Patch Fusion** | Knowledge patches $\{\boldsymbol{\Delta W_1}, ..., \boldsymbol{\Delta W_{N+1}}\}$, initial weights $\{\lambda_1, ..., \lambda_N\}$. | Fused model parameters: $\hat{\boldsymbol{\theta}}(\boldsymbol{\lambda}) = \hat{\boldsymbol{\theta}}_0 + \alpha(\sum_{i=1}^{N} \lambda_i \boldsymbol{\Delta W}_i + \boldsymbol{\Delta W}_{N+1})$. |
| **3. Few-shot Fine-tuning** | Fused model parameters $\hat{\boldsymbol{\theta}}(\boldsymbol{\lambda})$, few-shot dataset $\mathcal{D}'$. | Fine-tuned model with updated knowledge and optimized fusion weights. |

(ii) Better Knowledge Reuse: Instead of forcing a single shared representation, our method dynamically fuses relevant knowledge patches, improving transferability to novel tasks. (iii) Reducing Overfitting: multi-task SFT risks overfitting, especially with limited data. Our adaptive fusion selectively integrates knowledge, enhancing generalization. (iv) Providing Better Initialization: our fusion strategy constructs a well-informed starting point, enabling faster convergence and better fine-tuning performance. Furthermore, our experimental results in Table VI further confirm the effectiveness of knowledge patch fusion, as discussed in Section VII-E.

**Algorithm 1** Selective Knowledge Concentration

**Input:** Pre-trained base model $\mathcal{M}_{\text{b}}$ with parameters $\boldsymbol{\theta}_0$, upstream tasks $\{\mathcal{T}_1, \cdots, \mathcal{T}_N\}$ with respective datasets $\{\mathcal{D}_1, \cdots, \mathcal{D}_N\}$, few-shot data $\mathcal{D}'$ of novel datasets and tasks

*Stage 1: Upstream Knowledge Patches Extraction*
**for** each upstream task $\mathcal{T}_i$ **do**
  Initialize random $\boldsymbol{\Delta W}_i$ on $\mathcal{M}_{\text{b}}$
  Fine-tune $\boldsymbol{\Delta W}_i$ on $\mathcal{D}_i$ using Eq. 2
  Extracting low-rank knowledge patch $\boldsymbol{\Delta W}_i$
**end for**
*Stage 2: Dynamic Knowledge Patches Fusion*
Initialize interpolation weights $\{\boldsymbol{\lambda}_1, \cdots, \boldsymbol{\lambda}_N\}$ on the upstream dataset-specific knowledge patches $\{\boldsymbol{\Delta W}_1, \cdots, \boldsymbol{\Delta W}_N\}$
Initialize shared knowledge patch $\boldsymbol{\Delta W}_{N+1}$
Compute fused weight matrix $\hat{\boldsymbol{\theta}}$ using Eq. 4
*Stage 3: Few-shot Fine-Tuning*
**for** each mini-batch $\boldsymbol{\xi} \in \mathcal{D}'$ **do**
  Update knowledge patches and $\{\boldsymbol{\lambda}_1, \cdots, \boldsymbol{\lambda}_N\}$ using gradient descent by optimizing Eq. 5
**end for**
**return** Fine-tuned model $\hat{\boldsymbol{\theta}}$

### C. Few-shot Fine-Tuning

After integrating the knowledge patches and fusion module into the upstream DP-LLM, we fine-tune it for new datasets and tasks. Similar to the earlier training of individual knowledge patches, we keep the parameters of the backbone model fixed and use cross-entropy to maximize the likelihood in conditional language modeling. However, in this stage, we fine-tune the upstream DP-LLM across all datasets. Ultimately, the objective function for multi-task learning can be described as:

$$\min_{\boldsymbol{\lambda}, \boldsymbol{\Delta W_1}, \cdots, \boldsymbol{\Delta W_{N+1}}} \mathbb{E}_{\boldsymbol{\xi}} \left[ \sum_{i=1}^{N} f_i(\boldsymbol{\theta}(\boldsymbol{\lambda}); \boldsymbol{\xi}) \right], \tag{5}$$

where $\boldsymbol{\xi}$ is a mini-batch of multi-task training data $\left\{ \left( x_i^1, y_i^1 \right), \ldots, \left( x_i^N, y_i^N \right) \right\}_{i=1}^{q}$ and $q$ is the batch size. The illustration of few-shot fine-tuning can be found in Fig. 2 ③.

Algorithm 1 summarizes the SKC process, while Table I provides a structured explanation of the input and output at each stage. SKC consists of three key stages: Upstream Knowledge Patches Extraction, Dynamic Knowledge Patch Fusion, and Few-shot Fine-tuning. This process enables DP-LLMs to effectively concentrate relevant knowledge, mitigate knowledge distraction, and enhance adaptability to novel datasets and tasks.

## VI. Automatic Knowledge Bridging

This section presents the Automatic Knowledge Bridging (AKB) component, which is designed to integrate dataset-informed knowledge and address the knowledge gap in DP-LLM. Fig.2 (right) illustrates an overview of the component.

### A. Automatic Knowledge Bridging

Manually Deriving knowledge for a target dataset is time-consuming and impractical, especially in scenarios where novel datasets evolve rapidly. Thus, we attempt to frame the knowledge search process as an optimization problem to achieve automated searching. However, due to the vast search space, finding the right knowledge can be extremely difficult, making the knowledge optimization process intractable. Recent advances in prompt engineering have shown that LLMs can write better prompts than humans [28, 29, 30]. Inspired by this, we adopt closed-source LLM (e.g., GPT-4o) with comprehensive world knowledge to generate dataset-informed knowledge, which can be seen as a segment of the prompt.

Considering we have the fine-tuned upstream DP-LLM $\mathcal{M}$, a novel data preparation dataset $\mathcal{D}'_i = \{(x, y)\}$ of input-output pairs, which associated with a seed prompt $P_i$ for the dataset. Our goal is to find the most suitable knowledge $\rho$ from a closed-source LLM $\mathcal{M}_{\text{gpt}}$. When $\mathcal{M}$ is prompted with the concatenation $[P_i(\rho)||x]$, where $P_i(\rho)$ denote knowledge insert into the prompt. Providing this as input, we expect $\mathcal{M}$ to generate the corresponding response $y$. To achieve this goal, we formulate it as a prompt optimization problem. Formally, we search for the optimal knowledge that maximizes the expected per-sample score $\mathcal{S}(\rho, x, y)$ across all possible $(x, y)$:

$$\rho^\star = \arg\max_{\rho} \mathcal{S}(\rho) = \arg\max_{\rho} \mathbb{E}_{(x,y)}[\mathcal{S}(\rho, x, y)], \tag{6}$$

where $p^\star$ is the founded optimal knowledge. Next, we will instantiate the optimization problem using a four-step workflow.

### B. Workflow

In practice, AKB consists of four steps: *Generation*, *Evaluation*, *Feedback* and *Refinement*, as illustrated in Fig 2 (right).

Specifically, given a seed prompt and a subset of examples from the few-shot training set, AKB generates an initial pool of knowledge candidates. Then, it evaluates each candidate in the pool on a validation set with the fine-tuned upstream DP-LLM and selects the top performer. Subsequently, the best knowledge is used to identify error cases in the training set. Based on various errors, the closed-source LLM is utilized to generate diverse error feedback. Finally, based on feedback information, the selected knowledge is refined several times and re-added into the knowledge pool for the next iteration. Once the convergence criteria are met, AKB will provide high-quality knowledge as a supplement to the prompt for each inference. Next, we will discuss the details of each step.

**Generation.** To create an initial pool of potential knowledge candidates $\mathcal{K}$, we use a seed prompt with a set of examples as input. The seed prompt corresponds to the task prompt of the dataset and contains initial handcrafted knowledge. The examples $X_{\text{examples}}$ are a set of input-output pairs sampled from the dataset $\mathcal{D}'_i$, similar to the role of demonstrations in ICL. In this work, examples implicitly convey data-informed dataset knowledge, which is provided to the closed-source LLM to generate data-informed knowledge as text descriptions, it can be described as follows:

$$\rho = \mathcal{M}_{\text{gpt}}(P_{\text{gen}} || X_{\text{examples}}), \tag{7}$$

where $P_{\text{gen}}$ is the knowledge generation prompt to guide the $\mathcal{M}_{\text{gpt}}$ in generating knowledge candidates. To illustrate, we take the EM task as an example, the sketch of its generation prompt is shown in Fig. 3 (a), and the complete prompt can be found in the full version [1].

**Evaluation.** To obtain the rank of knowledge in the candidates, we need a score function that accurately measures the alignment between the task performance and the generated knowledge. Following previous works [28, 29], we directly choose the task metric as the score function, since our goal is to improve the performance of the target task, making the metric a suitable measure for our purposes.

Specifically, we first integrate each piece of knowledge $\rho$ from the set $\mathcal{K}$ into the task prompt $P_i$, resulting in $P_i(\rho)$. Next, we evaluate this knowledge-enhanced prompt using a validation dataset $\mathcal{D}_{\text{valid}}$. We use the validation dataset the same as dataset $\mathcal{D}'_i$. For each pair $(x, y)$ in $\mathcal{D}_{\text{valid}}$, we concatenate $x$ with the prompt $P_i(\rho)$, i.e. $[P_i(\rho)||x]$, and feed it into the model $\mathcal{M}$ to generate an output. Finally, we compare the output to the ground truth $y$ using the task's specific metric. This process can be summarized as follows:

$$\mathcal{S}(\rho, x, y) = \mathcal{E}(\mathcal{M}(P_i(\rho)||x), y), \tag{8}$$

where $\mathcal{E}(\cdot)$ denotes the metric of the target task (e.g. binary-F1 for EM). Ultimately, we rank the knowledge in the pool and select the $\rho$ with the highest score for further refinement.

**Feedback.** To further enhance the quality of the knowledge, we refine the selected knowledge by incorporating an error feedback mechanism. Before doing so, it is essential to analyze the identified errors and generate meaningful feedback for knowledge refinement. To achieve this, we leverage the closed-source LLM $\mathcal{M}_{\text{gpt}}$ again to systematically analyze the errors and produce informative feedback.

Specifically, we first test the selected knowledge $\rho_t$ on the training dataset $\mathcal{D}_{\text{train}}$ to obtain the error set $E$ in the $t$ iteration. Then, we sample a subset $X_{\text{errors}}$ of $E$ for producing the feedback information. We choose to identify error cases from the training set rather than directly using errors from the validation set for two key reasons. First, refining knowledge based on training errors ensures that the validation set remains an independent benchmark for evaluating improvements, preventing potential overfitting. Second, training data typically contains more diverse error patterns, providing a richer feedback signal that enhances the robustness of the refined knowledge. This design allows AKB to iteratively improve knowledge while maintaining a reliable validation process. Given the selected knowledge $\rho_t$ and error subset $X_{\text{errors}}$, the feedback prompt $P_{\text{fb}}$ (as shown in Fig. 3 (b)) is employed to guide $\mathcal{M}_{\text{gpt}}$ in producing feedback. The process is formalized as follows:

$$f_{b_t} = \mathcal{M}_{\text{gpt}}(P_{\text{fb}} \parallel P_i(\rho_t) \parallel X_{\text{errors}}). \tag{9}$$

Here, the feedback $f_{b_t}$ is a text response from $\mathcal{M}_{\text{gpt}}$ that includes error feedback information summarized of error examples. Intuitively, error examples highlight deficiencies in the current knowledge $\rho_t$, helping the model produce useful feedback information. This feedback information will help $\mathcal{M}_{\text{gpt}}$ generate more accurate knowledge through refinement.

**Refinement.** With the error feedback in hand, we refine the selected knowledge. Given the knowledge $\rho_t$, error set $X_{\text{errors}}$ and error feedback information $f_{b_t}$, the refinement prompt $P_{\text{refine}}$ (refer to Fig. 3 (c)) is used to guide $\mathcal{M}_{\text{gpt}}$ in generating refined knowledge $\hat{\rho}_t$. This process is formalized as follows:

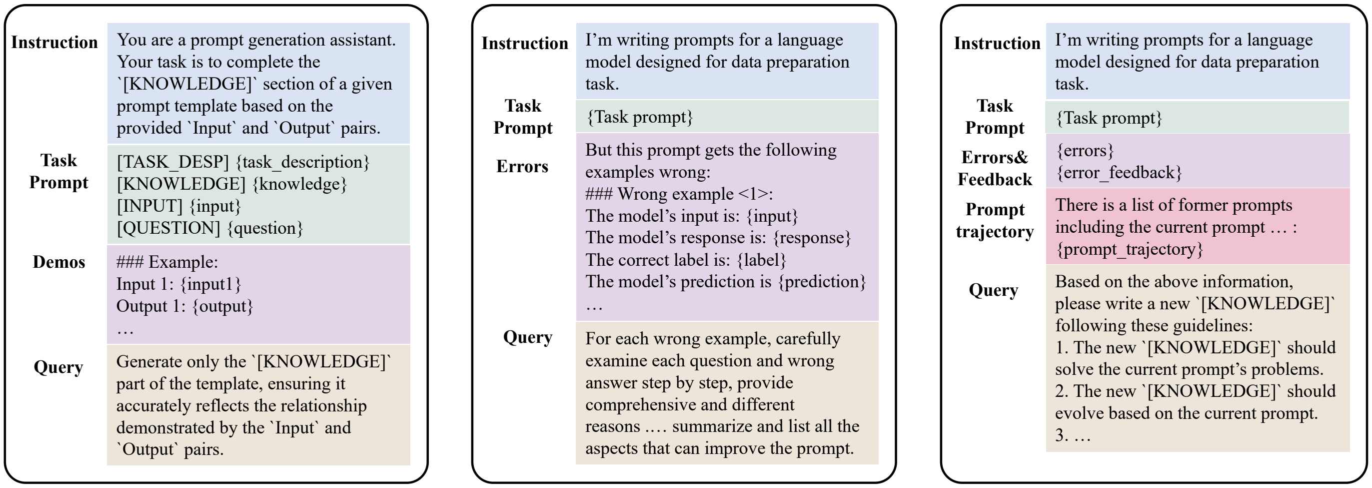
$$\hat{\rho}_t = \mathcal{M}_{\text{gpt}}(P_{\text{refine}} \parallel P_i(\rho_t) \parallel X_{\text{errors}} \parallel f_{b_t}). \tag{10}$$

The knowledge refinement process alternates between feedback and refinement multiple times to cover diverse error types. Specifically, we sample different subsets from error example set $E$, and use them to get different knowledge refinement. Additionally, we retain the knowledge optimization trajectory $\rho_{0:t-1}$ involves all generated knowledge from the previous iterations. By considering the full knowledge optimization trajectory, AKB implicitly summarizes the common mistakes from past solutions and avoids repeating them. Thus, Eq. 10 is in fact implemented as:

$$\hat{\rho}_t = \mathcal{M}_{\text{gpt}}(P_{\text{refine}} \parallel X_{\text{errors}} \parallel f_{b_t} \parallel \rho_{0:t-1}). \tag{11}$$

After processing all error subsets, the refined knowledge is aggregated and incorporated into the candidate pool $\mathcal{K}$ for re-evaluation in the next iteration. Once the specified number of iterations is completed, the highest-scoring knowledge candidate is selected as the final output. This iterative AKB process ensures progressive enhancement of $\rho$, optimizing task performance and effectively addressing the dataset-informed knowledge gap in the upstream DP-LLM for novel datasets.

Algorithm 2 describes the optimization process of AKB in detail. It begins by generating an initial pool of knowledge




$(a)$ $P_{gen}$: Generation prompt $(b)$ $P_{fb}$: Feedback prompt $(c)$ $P_{refine}$: Refinement prompt

Fig. 3: The illustration of three prompts used in the AKB component.

**Algorithm 2** Automatic Knowledge Bridging

**Input:** $\mathcal{M}_{\text{gpt}}$: closed-source LLM, $\mathcal{M}'$: knowledge fused DP-LLM , $\mathcal{D}_{\text{valid}}$: validation dataset

1: Randomly sample subsets $X_{\text{demos}} \subset \mathcal{D}_{\text{valid}}$
2: Generate initial candidate knowledge pool $\mathcal{K}$ using Eq. 7
3: $t \leftarrow 0$
4: **while** not converged **do**
5: $\quad \rho_t = \arg\max_{\rho \in \mathcal{K}} f(\rho, \mathcal{D}_{\text{valid}})$ // Score (Eq. 8)
6: $\quad E = \{(x, y) \in \mathcal{D}_{\text{valid}} \mid \mathcal{M}'(\rho_t, x) \neq y\}$ // Error set
7: $\quad$ **for** iteration $j \in \{0, 1, \ldots\}$ **do**
8: $\quad\quad$ Choose a random errors subset $X^{j}_{\text{errors}} \subset E$
9: $\quad\quad$ Generate feedback $f_{b_j}$ using Eq. 9
10: $\quad\quad$ Get refined knowledge $\rho'_t$ using Eq. 10
11: $\quad\quad$ $\mathcal{K} \leftarrow \mathcal{K} \cup \rho'_t$
12: $\quad$ **end for**
13: $\quad t \leftarrow t + 1$
14: **end while**
15: **Return:** $\rho_t$

candidates (lines 1–3). At each iteration, the algorithm selects the most promising knowledge candidate using a scoring function (line 5) and identifies the error set, which consists of instances where the model's predictions diverge from the ground truth (line 6). The knowledge pool is then updated by integrating refined knowledge derived from feedback (lines 7–11). This iterative process continues until convergence, ultimately yielding the optimized data-informed knowledge.

## VII. Experiments

In this section, we conduct extensive experiments to evaluate our proposed framework KnowTrans.

### A. Experiments Setup

**Datasets.** For upstream training, we utilize the publicly available training data from the Jellyfish Benchmark[7], which includes four data preparation tasks: ED, DI, SM, and EM. To comprehensively evaluate the effectiveness and transferability of KnowTrans, we conduct experiments on 13 novel datasets in a few-shot setting, ensuring that none of these datasets were included in the upstream training phase. Specifically, we consider: four previously studied tasks with eight novel datasets: **ED:** Flights, Rayyan, Beer [31]. **DI:** Flipkart [32], Phone [33]. **SM:** CMS [3]. **EM:** Abt-Buy, Walmart-Amazon [34]; and three entirely novel tasks with five datasets: **CTA:** SOTAB [35]. **AVE:** AE-110k, OA-mine. [36] **DC:** Rayyan, Beer [37]. These datasets span diverse domains and pose various challenges, enabling a rigorous assessment of KnowTrans's generalization capability. Table IX provides a summary of the dataset statistics for the novel downstream datasets. The complete upstream data information can be found in the full version [1].

**Baselines.** We compare KnowTrans with three types of methods: *(1) Open-source LLM-based Methods (DP-LLMs)*: We include several state-of-the-art DP-LLMs, including Mistral [15], TableLLaMA [9], MELD [10], and Jellyfish [7] (7B/8B/13B). Additionally, we evaluate Jellyfish-ICL, an enhanced version of Jellyfish-7B with in-context learning. *(2) Closed-source LLM-based Methods*: We compare against proprietary LLMs from OpenAI, including GPT-3.5, GPT-4, and GPT-4o, utilizing in-context learning via the OpenAI API. *(3) Non-LLM Methods*: We also include traditional non-LLM baselines across different tasks: Raha [31] (ED), IPM [5] (DI), SMAT [3] (SM), Ditto [38] (EM), Doduo [4] (CTA), MAVE [39] (AVE), and Baran [37] (EC). *(4) Our Methods*: We present two variants of the proposed framework: KnowTrans-U and KnowTrans. Both are designed to enhance the transferability of the Jellyfish backbone model. The key difference is that KnowTrans-U integrates the AKB component into the upstream training stage, while KnowTrans utilizes it only in the downstream phase.

**Implementation.** We implement KnowTrans in Pytorch. We fine-tuned the upstream model for the SKC component using the LlamaFactory framework [40]. Specifically, For the SKC component, we apply LoRA with a default rank of 32. The learning rate is set to 6e-5, with a batch size of 4 and a gradient accumulation step of 4. The model is trained for 3 epochs,

TABLE II: Statistic of Downstream Datasets.

| Task | Dataset | Training Set | Few-shot | Test Set |
|---|---|---|---|---|
| ED | Flights | 12,256 | 20 | 2,000 |
| | Rayyan | 9,000 | 20 | 2,000 |
| | Beer | 10,050 | 20 | 2,000 |
| DI | Flipkart | 11,460 | 20 | 2,675 |
| | Phone | 2,547 | 20 | 1,194 |
| SM | CMS | 23,068 | 20 | 2,564 |
| EM | Abt-buy | 5,743 | 20 | 1,916 |
| | Walmart-Amazon | 6,144 | 20 | 2,049 |
| CTA | SOTAB | 356 | 20 | 250 |
| AVE | AE-110k | 4,405 | 20 | 1,495 |
| | OA-mine | 7,360 | 20 | 2,451 |
| DC | Rayyan | 9,000 | 20 | 2,000 |
| | Beer | 10,050 | 20 | 2,000 |

with a 9:1 split for the train/validation dataset. For the AKB component, we use the closed-source LLM GPT-4o (gpt-4o-2024-08-06 API) for generation, feedback and refinement steps, configured with a temperature of 0.9. For evaluation, we use DP-LLM for inference with a temperature setting of 0.35, a top-k value of 10, and a top-p value of 0.9. We provide 10 examples for knowledge generation, with 4 error examples for refinement. The default setting runs 3 iterations, with 5 error random samples in each iteration. All experiments are conducted 3 times and the averaged performances are reported. All experiments are conducted on a server equipped with 256GB RAM, Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz and NVIDIA A40 GPU. The software environment consists of Ubuntu 20.04 LTS operating system, Python 3.9, PyTorch 2.3.1, Hugging Face Transformers Library 4.46.1 and LLM serving engine vLLM 0.5.3. To evaluate these data preparation tasks, we use accuracy for DI. For EM, ED, SM, DC and AVE, we use the F1-score, while for CTA, we use the micro-F1 score. All metrics are reported on a 100-point scale.

### *B. Effectiveness Evaluation*

**Performance Against Open-Source LLMs-Based and non-LLM Methods.** To evaluate the effectiveness of KnowTrans, we compared it with 7B open-source DP-LLMs and non-LLM methods. As shown in Table III, KnowTrans achieved state-of-the-art performance on 13 novel datasets in few-shot settings, with an average performance of **79.26%**, surpassing the best DP-LLM, Jellyfish, by **4.93%**. Additionally, KnowTrans significantly outperforms non-LLM methods by an average of **33.7%**. Non-LLM methods often overfit in few-shot scenarios because they typically depend on feature learning or smaller pre-trained models. Although DP-LLMs use larger models to mitigate overfitting, they still face issues like cross-datasets/tasks knowledge distraction and dataset-informed knowledge gap. In contrast, our proposed KnowTrans concentrates on transferable knowledge selectively and can effectively bridge the knowledge gap by integrating SKC and AKB components.

For **novel datasets** in ED, DI, SM, and EM, KnowTrans excels by leveraging its AKB component to incorporate dataset-specific knowledge. For example, it achieves **85.37%** and **85.88%** on ED and EM tasks, improving by **9.98%** and **3.96%**, respectively. While MELD can acquire transferable knowledge by assigning weights to the top-k experts within a MoE module, it employs an instance-level expert combination approach that fails to utilize dataset-level knowledge. For **novel tasks** such as CTA, AVE, and DC, KnowTrans addresses knowledge distraction with its SKC component, which reuses relevant knowledge from upstream tasks. On the CTA task, KnowTrans achieves **83.61%**, outperforming Mistral (**80.08%**). For AVE, it achieves **63.90%**, surpassing Mistral (**62.65%**). In DC, it delivers competitive results, such as **96.27%** on Rayyan and **98.54%** on Beer datasets. Additionally, the results indicate that integrating AKB into the upstream training process (KnowTrans-U) improves performance over Jellyfish, achieving **75.75%** compared to **74.33%**, demonstrating its ability to enhance dataset-specific knowledge incorporation. However, its performance remains below KnowTrans(**79.26%**), suggesting that applying AKB during upstream training introduces premature generalization rather than allowing dynamic adaptation at the fine-tuning stage. These findings further validate that dynamically applying AKB in the downstream phase is more effective than embedding it into the upstream model.

**Performance Against Closed-Source LLMs-Based Methods.** We also compare KnowTrans's performance against open-source LLM-based methods, particularly the GPT-series models such as GPT-3.5, GPT-4, GPT-4o and the newly added GPT-4o-mini. The results are summarized in Table V. We present three model sizes—7B, 8B, and 13B—boosted with the corresponding version of Jellyfish. All these enhanced models demonstrate superior performance compared to GPT-3.5, GPT-4, GPT-4o and GPT-4o-mini across multiple datasets and tasks. For example, on average, KnowTrans -13B outperforms GPT-4 and GPT-4o by **7.03%** and **6.07%**, respectively, while achieving a **9.48%** improvement over GPT-4o-mini. Additionally, it shows a substantial improvement of **13.54%** over GPT-3.5. This improvement underscores the advantage of our knowledge augmentation framework in a few-shot scenario.

While powerful in general applications, GPT-series models face challenges when directly applied to domain-specific tasks without further adaptation. Additionally, API calls to GPT can be quite costly, and we will provide a detailed comparison in the next section. Our proposed approach mitigates these challenges by leveraging SKC and AKB, enabling the model to specialize effectively in data preparation tasks. The results confirm that the KnowTrans provides robust solutions for adapting data preparation LLMs to novel datasets and tasks, bridging the gap between general-purpose LLMs and specialized tasks. This makes KnowTrans an efficient choice for scenarios where task-specific tuning is essential but access to closed-source LLMs may be limited or costly.

### *C. Efficiency and Cost Analysis*

We further explore the cost-effectiveness of KnowTrans -7B compared to closed-source LLMs like GPT-3.5, GPT-4o, and GPT-4, as demonstrated in Table IV. Our analysis focuses

TABLE III: Comparison of state-of-the-art 7B open-source DP-LLMs and non-LLM methods on thirteen datasets. Each dataset is evaluated in a few-shot setting. The best result is highlighted in **bold**, while the second-best is underlined.

| Task | Dataset | Model | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Non-LLM-Methods | Mistral | TableLLaMA | MELD | Jellyfish | Jellyfish-ICL | KNOWTRANS-U | KNOWTRANS |
| ED | Flights | 44.00 | 45.67 | 53.02 | 66.48 | 68.65 | 64.67 | 71.25 | **74.38** |
| | Rayyan | 62.00 | 45.00 | 36.99 | 79.79 | 78.89 | 74.17 | 84.68 | **89.40** |
| | Beer | 70.00 | 12.99 | 38.06 | 77.84 | 78.62 | 45.27 | 82.41 | **92.33** |
| | Average | 58.67 | 34.55 | 42.69 | 74.70 | 75.39 | 61.37 | 79.44 | **85.37** |
| DI | Flipkart | 2.54 | 81.27 | 42.59 | 79.74 | 78.09 | 82.47 | 78.84 | **82.88** |
| | Phone | 8.20 | 84.09 | 70.35 | 85.09 | 83.17 | 83.92 | 84.76 | **85.68** |
| | Average | 5.37 | 82.68 | 56.62 | 82.42 | 80.63 | 83.20 | 81.80 | **84.28** |
| SM | CMS | 2.10 | 18.75 | 1.86 | 26.67 | 27.59 | **30.30** | 29.63 | 27.69 |
| EM | Abt-buy | 57.14 | 20.09 | 42.58 | 85.52 | 77.62 | 74.56 | 83.33 | **87.86** |
| | Walmart-Amazon | 80.00 | 39.83 | 34.70 | 78.31 | 82.74 | 79.08 | 81.87 | **83.89** |
| | Average | 68.57 | 29.96 | 38.64 | 81.92 | 80.18 | 76.82 | 82.60 | **85.88** |
| CTA | SOTAB | 25.13 | 80.08 | 20.31 | 58.78 | 79.22 | 42.75 | **85.25** | 83.61 |
| AVE | AE-110k | 3.91 | 65.08 | 18.93 | 60.54 | 59.27 | 59.51 | 60.38 | **67.86** |
| | OA-mine | 1.63 | **60.22** | 17.01 | 57.16 | 57.57 | 42.76 | 47.29 | 59.93 |
| | Average | 2.77 | 62.65 | 17.97 | 58.85 | 58.42 | 51.14 | 53.84 | **63.90** |
| DC | Rayyan | 63.00 | **96.82** | 84.23 | 91.57 | 96.37 | 92.69 | 96.66 | 96.27 |
| | Beer | 87.00 | 95.83 | 99.68 | **99.72** | 98.54 | 95.10 | 98.42 | 98.54 |
| | Average | 75.00 | 96.33 | 91.96 | 91.57 | 97.46 | 93.90 | **97.54** | 97.41 |
| Average | | 45.56 | 57.36 | 43.11 | 70.62 | 74.33 | 66.71 | 75.75 | **79.26** |

TABLE IV: Comparison of input tokens, output tokens, and cost per instance for KNOWTRANS and other GPT models.

| | Input Tokens | Output Tokens | Price |
|---|---|---|---|
| GPT-3.5 | 751.08 | 2.86 | \$0.0004 |
| GPT-4o | 751.08 | 2.86 | \$0.0038 |
| GPT-4 | 751.08 | 2.86 | \$0.0227 |
| KNOWTRANS | 20.41 | 8.21 | \$0.0002 |

on the number of input tokens required, the number of output tokens generated, and the cost implications for each method.

- **Input and Output Tokens:** KNOWTRANS requires significantly fewer input tokens, averaging **20.41** tokens, compared to the 751.08 tokens required by GPT-3.5, GPT-4o, and GPT-4. The reduction in token usage is primarily attributed to our approach of incorporating few-shot examples into model parameters through training, whereas GPT requires few-shot samples to be placed in the context of demonstrations. This reduction in token usage not only minimizes computational load but also enhances processing speed. Additionally, due to the additional knowledge and error feedback in the AKB component, the output token count for KNOWTRANS is **8.21**, slightly higher than the other models (**2.86** tokens), suggesting more detailed and comprehensive output generation. However, since the AKB process only requires searching once for each dataset, the increase can be essentially ignored.
- **Cost Efficiency:** The cost per instance for KNOWTRANS is \$0.0002, markedly lower than that of GPT-3.5 (\$**0.0004**), and significantly less than the costs associated with GPT-4o (\$**0.0038**) and GPT-4 (\$**0.0227**). This pricing advantage makes KNOWTRANS an economical choice for large-scale implementations where multiple queries are processed.

### D. Scalability Analysis

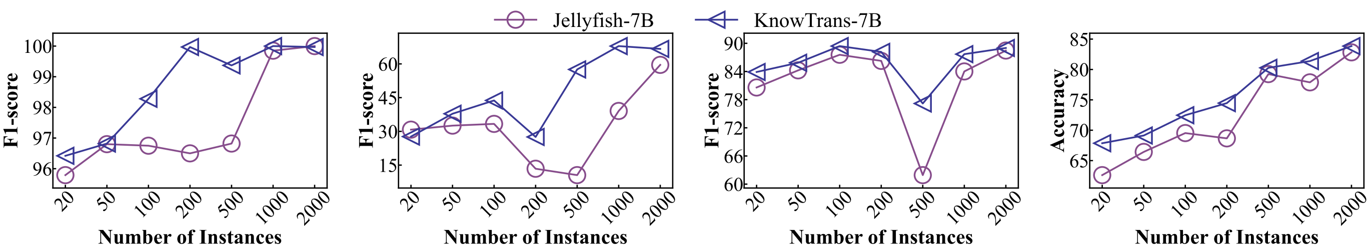
To further assess the effectiveness of the proposed framework KNOWTRANS across varying levels of data availability, we conducted a detailed scalability analysis using four datasets: DC (Rayyan), SM (CMS), EM (Walmart-Amazon), and AVE (AE-110k). This analysis examines the performance as the number of instances increases, simulating different resource conditions ranging from low to high data availability.

The results, presented in Fig. 4, demonstrate that the KNOWTRANS consistently outperforms the baseline model, Jellyfish-7B, across all datasets. As the number of instances grows, the performance of both models converges, reducing the gap in F1 scores. However, the KNOWTRANS maintains a consistent advantage in the low-to-medium range of instances, emphasizing its robustness in scenarios where data resources are constrained. This scalability is attributed to the framework's SKC and AKB components, which enable efficient utilization of limited labeled data and mitigate challenges such as knowledge distraction and gap. These results underline KNOWTRANS's scalability and effectiveness, making it a practical solution for diverse data preparation tasks, particularly in real-world applications where data availability often varies widely.

Furthermore, the datasets used in our study cover a wide range of practical data preparation tasks, as shown in Table IX. These tasks are essential components in data integration, cleaning, and transformation workflows, which are critical for real-world applications such as enterprise data management, automated ETL pipelines, and AI-driven data analytics. Additionally, our dataset selection includes heterogeneous sources (e.g., structured relational databases, web tables, and noisy real-world datasets), ensuring that the framework is evaluated in scenarios commonly encountered in industrial and academic data processing pipelines. Moreover, our method has already been successfully applied in Huawei's industrial scenarios,

TABLE V: Comparison of state-of-the-art closed-source LLMs on thirteen datasets. Each dataset is evaluated in a few-shot setting. The best result is highlighted in **bold**, while the second-best is underlined.

| Task | Dataset | Model | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | GPT-3.5 | GPT-4 | GPT-4o | GPT-4o-mini | KnowTrans-7B | KnowTrans-8B | KnowTrans-13B |
| ED | Flights | 61.39 | 68.65 | 66.22 | 73.21 | *74.38* | 66.91 | **84.53** |
| | Rayyan | 51.87 | 63.80 | 52.54 | 52.90 | **89.40** | 78.13 | 84.40 |
| | Beer | 75.91 | 83.89 | 76.90 | 78.52 | 92.33 | **98.93** | 83.93 |
| | Average | 63.06 | 72.11 | 65.22 | 68.21 | **85.37** | 81.32 | 84.29 |
| DI | Flipkart | 82.47 | **85.64** | 85.57 | 82.99 | 82.88 | 81.87 | 81.20 |
| | Phone | 85.43 | 85.51 | **86.18** | 85.43 | 85.68 | 83.33 | 84.34 |
| | Average | 83.95 | 85.58 | **85.88** | 84.21 | 84.28 | 82.60 | 82.77 |
| SM | CMS | 30.77 | 30.30 | 28.57 | 22.22 | 27.69 | 35.9 | **40.91** |
| EM | Abt-buy | 37.21 | 92.49 | **94.09** | 76.66 | 87.86 | 87.98 | 92.42 |
| | Walmart-Amazon | 80.80 | 84.07 | 87.37 | 87.47 | 83.89 | 83.55 | **91.15** |
| | Average | 59.01 | 88.28 | 90.73 | 82.07 | 85.88 | 85.77 | **91.79** |
| CTA | SOTAB | 90.76 | 96.40 | **98.00** | 83.23 | 83.61 | 81.60 | 81.48 |
| AVE | AE-110k | 50.99 | 49.72 | 61.10 | 52.90 | 67.86 | 66.67 | **72.47** |
| | OA-mine | 55.12 | 46.47 | 60.47 | 54.92 | 59.93 | 55.08 | **63.57** |
| | Average | 53.06 | 48.10 | 60.79 | 53.91 | 63.90 | 60.88 | **68.02** |
| DC | Rayyan | 91.87 | 90.82 | 86.50 | 87.08 | 96.27 | 95.65 | **96.82** |
| | Beer | **99.75** | 99.17 | 94.35 | 97.35 | 98.54 | 97.69 | 99.52 |
| | Average | 95.81 | 95.00 | 90.43 | 92.22 | 97.41 | 96.67 | **98.17** |
| | Average | 67.85 | 74.76 | 75.32 | 71.91 | 79.40 | 77.87 | **81.39** |




(a) DC (Rayyan) (b) SM (CMS) (c) EM (Walmart-Amazon) (d) AVE (AE-110k)

Fig. 4: Comparison of the performance of Jellyfish-7B and KnowTrans-7B across four tasks (datasets). The results illustrate F1-score and accuracy for varying numbers of labeled instances (20, 50, 100, 200, 1000 and 2000).

TABLE VI: Performance for ablation study.

| Task | Dataset | Model | | | | |
|---|---|---|---|---|---|---|
| | | Base | SFT | w/o SKC | w/o AKB | KnowTrans |
| ED | Rayyan | 74.33 | 78.89 | 73.94 | 87.30 | **89.40** |
| DI | Flipkart | 78.36 | 78.09 | 81.79 | 81.42 | **82.88** |
| | Phone | 83.50 | 83.17 | 84.76 | 85.34 | **85.68** |
| SM | CMS | 27.59 | 27.59 | **30.77** | 24.76 | 27.69 |
| EM | Abt-buy | 77.36 | 77.62 | 82.80 | 81.17 | **87.86** |
| CTA | SOTAB | 79.47 | 79.22 | 82.08 | 82.60 | **83.61** |
| AVE | AE-110k | 56.48 | 59.27 | 61.58 | 65.79 | **67.86** |
| | OA-mine | 53.20 | 57.57 | 57.89 | 58.63 | **59.93** |
| DC | Rayyan | 90.39 | 96.37 | **96.82** | 95.99 | 96.27 |
| | Beer | 93.56 | 98.54 | 97.98 | 98.15 | **98.54** |
| **Average** | | 71.42 | 73.63 | 75.04 | 76.12 | **77.97** |

demonstrating its effectiveness in real-world applications.

### E. Ablation Study

To evaluate the impact of SKC and AKB components in KnowTrans, as detailed in Table VI, we performed an ablation study on KnowTrans-7B across multiple datasets. Each task included at least one dataset and was evaluated under four different configurations: (i) Base – Removing both the SKC and AKB components (i.e., using the original upstream Jellyfish-7B model). (ii) SFT – Removing both SKC and AKB components while applying multi-task SFT. (iii) w/o SKC – Removing only the SKC component. (iv) w/o AKB – Removing only the AKB component. The results demonstrate the critical role of both the SKC and AKB components in enhancing the overall performance of KnowTrans. Removing the SKC component leads to a noticeable performance drop from **77.97%** to **75.04%**, highlighting its importance in knowledge concentration. Furthermore, a comparison between configurations (ii) and (iv) shows an improvement from **73.63%** to **76.12%** when training with SKC alone, underscoring its effectiveness in mitigating knowledge distraction, enhancing transferability, reducing overfitting, and outperforming multi-task SFT. Similarly, eliminating the AKB component results in a decline from **77.97%** to **76.12%**, reinforcing its role in generating dataset-informed knowledge and bridging knowledge gaps in novel tasks. These findings confirm that the integration of SKC and AKB significantly enhances the transferability of the upstream DP-LLM, enabling superior few-shot generalization.

### F. Hyper-parameter Analysis

**Effect of different backbone models.** The results in Fig. 5 and Fig. 6 demonstrate the effectiveness of KnowTrans in boost-

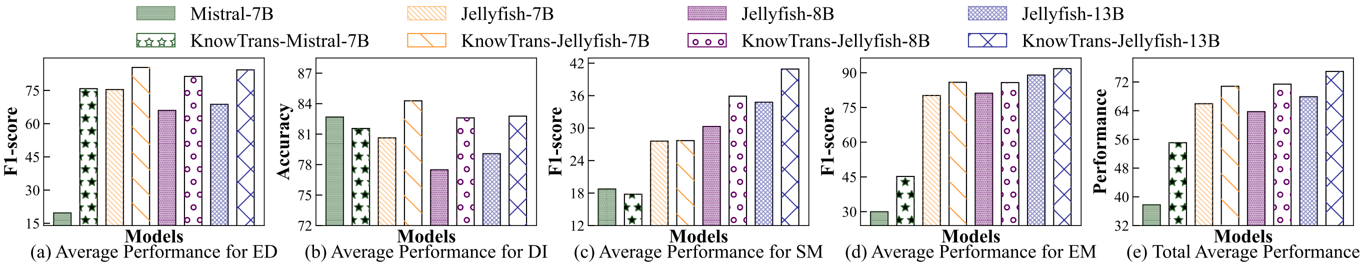



Fig. 5: Comparison of different backbone models using our proposed KNOWTRANS on novel datasets.

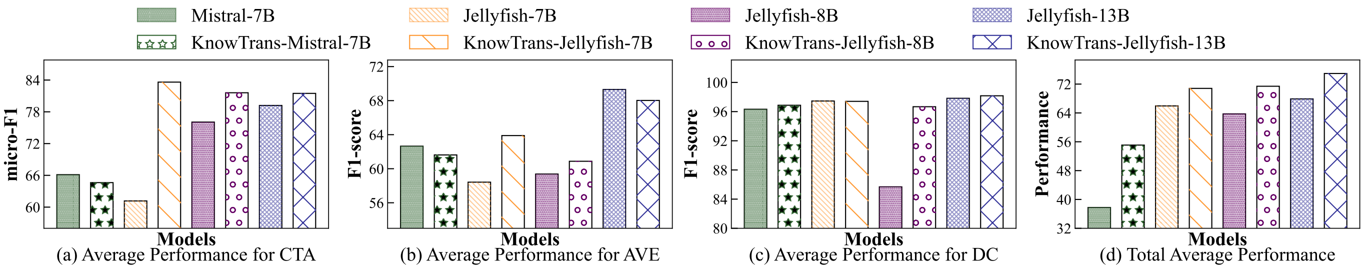



Fig. 6: Comparison of different backbone models using our proposed KNOWTRANS on novel tasks.

TABLE VII: Performance of using different weight strategies.

| Task | Dataset | Model | | | |
|---|---|---|---|---|---|
| | | **Single** | **Uniform** | **Adaptive** | **KNOWTRANS** |
| ED | Flights | 65.99 | 69.08 | 71.69 | **74.38** |
| | Rayyan | 74.33 | 81.79 | 87.30 | **89.40** |
| EM | Abt-buy | 77.36 | 80.43 | 81.17 | **87.86** |
| AVE | AE-110k | 56.48 | 63.09 | 65.79 | **67.86** |
| Average | | 69.00 | 73.60 | 76.49 | **79.90** |

ing various backbone models, including Mistral-7B, Jellyfish-7B, Jellyfish-8B, and Jellyfish-13B, across novel datasets and tasks. KNOWTRANS consistently delivers superior performance compared to the baselines. Specifically, KNOWTRANS exhibits consistent and significant performance improvements for all model sizes, though the extent of improvement varies by model capacity. Larger models, such as Jellyfish-13B, benefit the most from the KNOWTRANS framework within the same architecture, achieving the highest scores across multiple datasets and tasks. This is attributed to their larger parameter capacity, which allows them to better utilize the augmented knowledge introduced by KNOWTRANS 's SKC and AKB components.

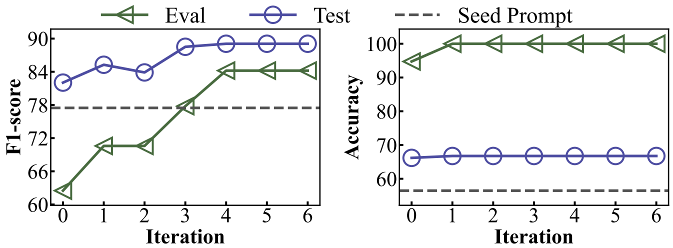
Smaller models, also experience substantial performance gains, indicating the versatility of KNOWTRANS. Particularly, Mistral-7B exhibits the most significant performance gain after applying the KNOWTRANS . This is particularly notable because Mistral-7B had not undergone upstream multi-task training on data preparation tasks, unlike the Jellyfish models. These findings highlight the general applicability and efficiency of KNOWTRANS in boosting the transferability of DP-LLMs, regardless of model size. The remarkable improvement can be attributed to KNOWTRANS 's ability to dynamically incorporate knowledge patches and refine dataset-specific knowledge through the SKC and AKB components, effectively compensating for the absence of prior upstream training.

**Effect of weight strategy.** To evaluate the impact of different adaptive strategies on model performance, we present the performance improvement comparison across different adaptive strategies: single (i.e. no upstream knowledge patches), uniform and adaptive (i.e. used in SKC), over Jellyfish-7B. As shown in Table VII, the overall average performance of uniform and adaptive strategies surpasses that of the single model, demonstrating that dynamically integrating knowledge from upstream tasks can effectively enhance the transferability of DP-LLMs. Furthermore, the adaptive strategy outperforms uniform, indicating that our proposed adaptive knowledge integration module, SKC, plays a crucial role in optimizing the use of dataset-specific and shared knowledge. Notably, KNOWTRANS, which incorporates SKC, achieves the best performance across all datasets, further validating the effectiveness of this approach.

**Effect of round times for refinement.** We then explore the effect of the iterative round times for error feedback on the performance of KNOWTRANS. We demonstrate the experiment results on two tasks: Attribute Value Extraction (AVE) and Error Detection (ED), as shown in Fig. 7.

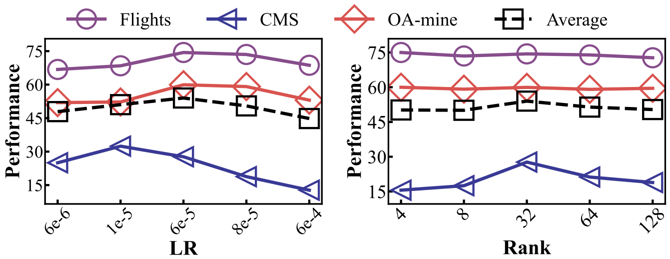
- **Error Detection (ED):** In contrast, Fig. 7(a) shows a notable improvement in both eval and test performance for the ED task. Initially, eval performance increases sharply, indicating rapid assimilation of feedback in early iterations. After the third round, the eval performance begins to plateau, while the test performance continues to improve slightly, suggesting enhanced model transferability from iterative refinements.
- **Attribute Value Extraction (AVE):** As depicted in Fig. 7 (b), the eval and test performance for AVE remain constant over 6 rounds of feedback, which suggests that the model quickly stabilizes and additional feedback does not contribute further to significant changes in model behavior for this metric. This indicates that for certain data, additional knowledge may not be helpful. Nonetheless, AKB does not impair its performance, and we can also benefit from the SKC.

**Effect of learning rate and LoRA rank.** To evaluate the



(a) Peformance for ED (Beer) (b) Peformance for AVE (AE-100k)

Fig. 7: Performance of different round times for refinement.



(a) Effect of Learning Rate (b) Effect of Rank

Fig. 8: Performance of different learning rate and LoRA rank.

impact of hyperparameter choices on model performance, we conduct a sensitivity analysis focusing on learning rate (LR) and LoRA rank. Fig. 8 (a) shows the effect of varying the learning rate, where we observe that excessively small or large LR values degrade performance, with an optimal range around $1e^{-5}$ to $6e^{-5}$. Similarly, Fig. 8 (b) presents the impact of different LoRA ranks, indicating that performance improves with increasing rank up to 32, beyond which the gains plateau or slightly decline. These findings highlight the need to carefully adjust these parameters to maintain optimal model performance.

## VIII. Related Work

In this section, we introduce related works including Non-LLM methods, LLM-based methods and model fusion.

### *A. Non-LLM Methods*

Early works in data preparation focused on rule-based methods [41, 42, 43] and statistical models [44, 45, 46]. Rule-based approaches included manual rules [47, 48, 49], consistency checks [50], and similarity constraints [51], while probabilistic models [52] were used in ED, DC, DI and AVE. For EM and SM, early works employed similarity functions [53, 54] or hashing [55, 56]. CTA leveraged feature engineering techniques, including hand-crafted [57, 58], statistical [59], and semantic features [60]. Recent works have increasingly adopted ML techniques for data preparation. In ED, DC, and DI, methods like few-shot learning [61], transfer learning [37], meta-learning [62], GANs [63] and PLMs [62] are introduced. For EM and SM, word embeddings [64], DNNs [65], GMMs [66], GNNs [67] and PLMs [38, 68, 69, 70, 71] are leveraged for better tabular representation. PLM-based methods also enhance CTA [4, 72, 73, 74, 75, 76, 77, 78, 79] and AVE [39, 80, 81, 82, 83] performance. However, most of them focus on a single task, whereas we propose an all-in-one multitask model applicable to various datasets and tasks.

### *B. LLM-based Methods*

More recently, some works employ LLMs for data preparation, including closed- and open-source models. Closed-source methods primarily use prompt engineering [34, 36, 84, 85, 86, 87, 88, 89] on ChatGPT and GPT-4o, but their reliance on APIs limits adaptability to tabular data. Thus, TableGPT [8] improves ChatGPT's performance on table-related tasks via instruction tuning, though closed models still pose challenges like high costs and privacy concerns. Open-source approaches fine-tune LLMs to address these issues. For example, AnyMatch [90] fine-tunes GPT-2 [91] for zero-shot EM, while Archetype [92] enhances Alpaca-7B [14] for CTA. DP-LLMs like Jellyfish [7], TableLLaMA [9] and MELD [10] propose to manage multiple tasks within a single model. Unlike these methods of training multiple tasks from scratch, we focus on enhancing DP-LLMs' transferability to novel datasets and tasks in a few-shot setting.

### *C. Model Fusion*

Model fusion enhances multi-task capabilities by combining trained models for better generalization [24, 93, 94, 95]. Early methods attempted to average model weights from multiple independent models, leveraging their diversity for improved generalization [24, 25, 93, 96, 97, 98]. However, these approaches are computationally intensive, particularly for large models. To address this, parameter-efficient fusion techniques like LoRA [27] and Adapter [99] enable functional transfer without merging entire models. LoRAHub [95] optimizes the LoRA fusion process, AdapterSoup [99] averages adapters for domain adaptation and LoRARetriever [100] dynamically retrieves relevant LoRAs. MoLE [101] uses a Mixture-of-Experts framework to learn optimal LoRA composition weights. Unlike these, our method fuses LoRAs from upstream models to boost DP-LLM transferability in a cross-model manner.

## IX. Conclusion

This paper proposes KnowTrans, a knowledge augmentation framework that enhances the transferability of DP-LLMs to novel datasets and tasks in few-shot settings. By integrating SKC and AKB components, KnowTrans mitigates knowledge distraction and dataset-informed gaps, ensuring robust performance and scalability with increasing data availability. Experimental results demonstrate its effectiveness in improving DP-LLM transferability across various models, highlighting its practicality for diverse data preparation tasks. Furthermore, this work establishes a strong foundation for future advancements in knowledge-augmented LLMs for data preparation.


## Acknowledgment

This work was supported in part by the NSFC under Grants No. (62302436, 62025206, and U23A20296), Zhejiang Province's "Lingyan" R&D Project under Grant No. 2024C01259, Ningbo Science and Technology Special Projects under Grant No. 2023Z212, Yongjiang Talent Programme "Research on key technologies in Intelligent Database", and Zhejiang University Education Foundation Qizhen Scholar Foundation. Yuren Mao is the corresponding author of the work.

## REFERENCES


[1] "Code, datasets and full version," https://github.com/a11en0/knowtrans, 2025.

[2] C. Chai, J. Wang, Y. Luo, Z. Niu, and G. Li, "Data management for machine learning: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 5, pp. 4646–4667, 2022.

[3] J. Zhang, B. Shin, J. D. Choi, and J. C. Ho, "Smat: An attention-based deep learning solution to the automation of schema matching," in *ADBIS*, 2021.

[4] Y. Suhara, J. Li, Y. Li, D. Zhang, Ç. Demiralp, C. Chen, and W.-C. Tan, "Annotating columns with pre-trained language models," in *SIGMOD*, 2022.

[5] Y. Mei, S. Song, C. Fang, H. Yang, J. Fang, and J. Long, "Capturing semantics for imputation with pre-trained language models," in *ICDE*, 2021.

[6] J. Tu, J. Fan, N. Tang, P. Wang, G. Li, X. Du, X. Jia, and S. Gao, "Unicorn: A unified multi-tasking model for supporting matching tasks in data integration," *SIGMOD*, 2023.

[7] H. Zhang, Y. Dong, C. Xiao, and M. Oyamada, "Jellyfish: Instruction-tuning local large language models for data preprocessing," in *EMNLP*, 2024.

[8] P. Li, Y. He, D. Yashar, W. Cui, S. Ge, H. Zhang, D. Rifinski Fainman, D. Zhang, and S. Chaudhuri, "Table-gpt: Table fine-tuned gpt for diverse table tasks," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–28, 2024.

[9] T. Zhang, X. Yue, Y. Li, and H. Sun, "Tablellama: Towards open large generalist models for tables," in *NAACL*, 2024.

[10] M. Yan, Y. Wang, K. Pang, M. Xie, and J. Li, "Efficient mixture of experts based on large language models for low-resource data preprocessing," in *KDD*, 2024.

[11] Y. Mao, Z. Wang, W. Liu, X. Lin, and W. Hu, "Banditmtl: Bandit-based multi-task learning for text classification," in *ACL*, 2021.

[12] B. Liu, X. Liu, X. Jin, P. Stone, and Q. Liu, "Conflict-averse gradient descent for multi-task learning," *NeurIPS*, 2021.

[13] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *NeurIPS*, 2020.

[14] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv:2307.09288*, 2023.

[15] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv:2310.06825*, 2023.

[16] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv:2303.18223*, 2023.

[17] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent abilities of large language models," *Transactions on Machine Learning Research*, 2022.

[18] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–40, 2023.

[19] S. Arora, B. Yang, S. Eyuboglu, A. Narayan, A. Hojel, I. Trummer, and C. Ré, "Language models enable simple systems for generating structured views of heterogeneous data lakes," *arXiv:2304.09433*, 2023.

[20] R. C. Fernandez, A. J. Elmore, M. J. Franklin, S. Krishnan, and C. Tan, "How large language models will disrupt data management," *VLDB*, 2023.

[21] S. Zhang, L. Dong, X. Li, S. Zhang, X. Sun, S. Wang, J. Li, R. Hu, T. Zhang, F. Wu *et al.*, "Instruction tuning for large language models: A survey," *arXiv:2308.10792*, 2023.

[22] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *NeurIPS*, 2018.

[23] Y. Mao, W. Liu, and X. Lin, "Adaptive adversarial multi-task representation learning," in *ICML*, 2020.

[24] A. Rame, M. Kirchmeyer, T. Rahier, A. Rakotomamonjy, P. Gallinari, and M. Cord, "Diverse weight averaging for out-of-distribution generalization," *NeurIPS*, 2022.

[25] A. Ramé, K. Ahuja, J. Zhang, M. Cord, L. Bottou, and D. Lopez-Paz, "Model ratatouille: Recycling diverse models for out-of-distribution generalization," in *ICML*, 2023.

[26] E. Yang, L. Shen, G. Guo, X. Wang, X. Cao, J. Zhang, and D. Tao, "Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities," *arXiv:2408.07666*, 2024.

[27] E. J. Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models," in *ICLR*, 2022.

[28] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large language models are human-level prompt engineers," *arXiv:2211.01910*, 2022.

[29] L. Chen, J. Chen, T. Goldstein, H. Huang, and T. Zhou, "Instructzero: Efficient instruction optimization for black-box large language models," in *ICML*, 2024.

[30] J. Cheng, X. Liu, K. Zheng, P. Ke, H. Wang, Y. Dong, J. Tang, and M. Huang, "Black-box prompt optimization: Aligning large language models without model training," *arXiv:2311.04155*, 2023.

[31] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Raha: A configuration-free error detection system," in *SIGMOD*, 2019.

[32] PromptCloud, "Flipkart products dataset," https://www.kaggle.com/datasets/PromptCloudHQ/flipkart-products, 2017.

[33] ——, "Amazon reviews: Unlocked mobile phones,"

https://www.kaggle.com/datasets/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones, 2016.

[34] A. Narayan, I. Chami, L. Orr, S. Arora, and C. Ré, "Can foundation models wrangle your data?" *arXiv:2205.09911*, 2022.

[35] K. Korini and C. Bizer, "Column type annotation using chatgpt," 2023.

[36] A. Brinkmann, R. Shraga, and C. Bizer, "Product attribute value extraction using large language models," *arXiv:2310.12537*, 2023.

[37] M. Mahdavi and Z. Abedjan, "Baran: Effective error correction via a unified context representation and transfer learning," *VLDB*, 2020.

[38] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan, "Deep entity matching with pre-trained language models," *arXiv:2004.00584*, 2020.

[39] L. Yang, Q. Wang, Z. Yu, A. Kulkarni, S. Sanghai, B. Shu, J. Elsas, and B. Kanagal, "Mave: A product dataset for multi-source attribute value extraction," in *WSDM*, 2022.

[40] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, Z. Luo, Z. Feng, and Y. Ma, "Llamafactory: Unified efficient fine-tuning of 100+ language models," 2024.

[41] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "The llunatic data-cleaning framework," *VLDB*, 2013.

[42] X. Ding, H. Wang, J. Su, M. Wang, J. Li, and H. Gao, "Leveraging currency for repairing inconsistent and incomplete data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 3, pp. 1288–1302, 2020.

[43] W. Fan, P. Lu, and C. Tian, "Unifying logic rules and machine learning for entity enhancing," *Science China Information Sciences*, vol. 63, pp. 1–19, 2020.

[44] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "Holoclean: Holistic data repairs with probabilistic inference," *VLDB*, 2017.

[45] S. De, Y. Hu, M. V. Vamsikrishna, Y. Chen, and S. Kambhampati, "Bayeswipe: A scalable probabilistic framework for cleaning bigdata," *arXiv:1506.08908*, 2015.

[46] A. Lew, M. Agrawal, D. Sontag, and V. Mansinghka, "Pclean: Bayesian data cleaning at scale with domain-specific probabilistic programming," in *AISTATS*, 2021.

[47] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Towards certain fixes with editing rules and master data," *The VLDB journal*, vol. 21, pp. 213–238, 2012.

[48] D. Vandic, J.-W. Van Dam, and F. Frasincar, "Faceted product search powered by the semantic web," *Decision Support Systems*, vol. 53, no. 3, pp. 425–437, 2012.

[49] L. Zhang, M. Zhu, and W. Huang, "A framework for an ontology-based e-commerce product information retrieval system." *J. Comput.*, vol. 4, no. 6, pp. 436–443, 2009.

[50] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma, "Improving data quality: Consistency and accuracy," in *VLDB*, 2007.

[51] S. Song, Y. Sun, A. Zhang, L. Chen, and J. Wang, "Enriching data imputation under similarity rule constraints," *IEEE transactions on knowledge and data engineering*, vol. 32, no. 2, pp. 275–287, 2018.

[52] D. Putthividhya and J. Hu, "Bootstrapped named entity recognition for product attribute extraction," in *EMNLP*, 2011.

[53] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava, "Fast indexes and algorithms for set similarity selection queries," in *ICDE*, 2008.

[54] A. Doan, P. Konda, P. Suganthan GC, Y. Govind, D. Paulsen, K. Chandrasekhar, P. Martinkus, and M. Christie, "Magellan: toward building ecosystems of entity matching solutions," *Communications of the ACM*, vol. 63, no. 8, pp. 83–91, 2020.

[55] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl, "Meta-blocking: Taking entity resolution to the next level," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1946–1960, 2013.

[56] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra, "Magellan: Toward building entity matching management systems," *VLDB*, 2016.

[57] S. K. Ramnandan, A. Mittal, C. A. Knoblock, and P. Szekely, "Assigning semantic labels to data sources," in *ESWC*, 2015.

[58] M. Pham, S. Alse, C. A. Knoblock, and P. Szekely, "Semantic labeling: a domain-independent approach," in *ISWC*, 2016.

[59] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, Ç. Demiralp, and W.-C. Tan, "Sato: Contextual semantic type detection in tables," *arXiv:1911.06311*, 2019.

[60] M. Hulsebos, K. Hu, M. Bakker, E. Zgraggen, A. Satyanarayan, T. Kraska, Ç. Demiralp, and C. Hidalgo, "Sherlock: A deep learning approach to semantic data type detection," in *KDD*, 2019.

[61] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas, "Holodetect: Few-shot learning for error detection," in *SIGMOD*, 2019.

[62] Z. Miao, Y. Li, and X. Wang, "Rotom: A meta-learned data augmentation framework for entity matching, data cleaning, text classification, and beyond," in *SIGMOD*, 2021.

[63] J. Peng, D. Shen, N. Tang, T. Liu, Y. Kou, T. Nie, H. Cui, and G. Yu, "Self-supervised and interpretable data cleaning with sequence generative adversarial networks," *VLDB*, 2022.

[64] J. MESTS and M. Tang, "Distributed representations of tuples for entity resolution," *VLDB*, 2018.

[65] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "Deep learning for entity matching: A design space exploration," in *SIGMOD*, 2018.

[66] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan, "Zeroer: Entity resolution using zero labeled examples," in *SIGMOD*, 2020.

[67] B. Li, W. Wang, Y. Sun, L. Zhang, M. A. Ali, and Y. Wang, “Grapher: Token-centric entity resolution with graph convolutional neural networks,” in *AAAI*, 2020.
[68] C. Zhao and Y. He, “Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning,” in *WWW*, 2019.
[69] R. Peeters and C. Bizer, “Dual-objective fine-tuning of bert for entity matching,” *VLDB*, 2021.
[70] J. Tu, J. Fan, N. Tang, P. Wang, C. Chai, G. Li, R. Fan, and X. Du, “Domain adaptation for deep entity resolution,” in *SIGMOD*, 2022.
[71] P. Wang, X. Zeng, L. Chen, F. Ye, Y. Mao, J. Zhu, and Y. Gao, “Promptem: prompt-tuning for low-resource generalized entity matching,” *arXiv:2207.04802*, 2022.
[72] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, “Turl: Table understanding through representation learning,” *ACM SIGMOD Record*, vol. 51, no. 1, pp. 33–40, 2022.
[73] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos, “Tapas: Weakly supervised table parsing via pre-training,” *arXiv:2004.02349*, 2020.
[74] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, “Tabert: Pretraining for joint understanding of textual and tabular data,” *arXiv:2005.08314*, 2020.
[75] H. Iida, D. Thai, V. Manjunatha, and M. Iyyer, “Tabbie: Pretrained representations of tabular data,” *arXiv:2105.02584*, 2021.
[76] Y. Zhou, S. Singh, and C. Christodoulopoulos, “Tabular data concept type detection using star-transformers,” in *CIKM*, 2021.
[77] Y. Sun, H. Xin, and L. Chen, “Reca: Related tables enhanced column semantic type annotation framework,” *VLDB*, 2023.
[78] Z. Miao and J. Wang, “Watchog: A light-weight contrastive learning based framework for column annotation,” *SIGMOD*, 2023.
[79] Y. Wang, H. Xin, and L. Chen, “Kglink: A column type annotation method that combines knowledge graph and pre-trained language model,” *arXiv:2406.00318*, 2024.
[80] H. Xu, W. Wang, X. Mao, X. Jiang, and M. Lan, “Scaling up open tagging from tens to thousands: Comprehension empowered attribute value extraction from product title,” in *ACL*, 2019.
[81] Q. Wang, L. Yang, B. Kanagal, S. Sanghai, D. Sivakumar, B. Shu, Z. Yu, and J. Elsas, “Learning to extract attribute value from product via question answering: A multi-task approach,” in *KDD*, 2020.
[82] X. Zhang, C. Zhang, X. Li, X. L. Dong, J. Shang, C. Faloutsos, and J. Han, “Oa-mine: Open-world attribute mining for e-commerce products with weak supervision,” in *WWW*, 2022.
[83] J. Yan, N. Zalmout, Y. Liang, C. Grant, X. Ren, and X. L. Dong, “Adatag: Multi-attribute value extraction from product profiles with adaptive decoding,” *arXiv:2106.02318*, 2021.
[84] R. Peeters and C. Bizer, “Using chatgpt for entity matching,” in *ADBIS*, 2023.
[85] Z. Cheng, J. Kasai, and T. Yu, “Batch prompting: Efficient inference with large language model apis,” *arXiv:2301.08721*, 2023.
[86] N. Guan, K. Chen, and N. Koudas, “Can large language models design accurate label functions?” *arXiv:2311.00739*, 2023.
[87] M. Fan, X. Han, J. Fan, C. Chai, N. Tang, G. Li, and X. Du, “Cost-effective in-context learning for entity resolution: A design space exploration,” in *ICDE*, 2024.
[88] C. Fang, X. Li, Z. Fan, J. Xu, K. Nag, E. Korpeoglu, S. Kumar, and K. Achan, “Llm-ensemble: Optimal large language model ensemble method for e-commerce product attribute value extraction,” in *SIGIR*, 2024.
[89] N. Seedat and M. van der Schaar, “Matchmaker: Self-improving large language model programs for schema matching,” *arXiv:2410.24105*, 2024.
[90] Z. Zhang, P. Groth, I. Calixto, and S. Schelter, “Any-match - efficient zero-shot entity matching with a small language model,” *arXiv:2409.04073*, 2024.
[91] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, 2019.
[92] B. Feuer, Y. Liu, C. Hegde, and J. Freire, “Archetype: A novel framework for open-source column type annotation using large language models,” *VLDB*, 2024.
[93] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, “Averaging weights leads to wider optima and better generalization,” *arXiv:1803.05407*, 2018.
[94] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. G. Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt, “Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time,” in *ICML*, 2022.
[95] C. Huang, Q. Liu, B. Y. Lin, T. Pang, C. Du, and M. Lin, “Lorahub: Efficient cross-task generalization via dynamic lora composition,” *arXiv:2307.13269*, 2023.
[96] T. Li, Z. Huang, Q. Tao, Y. Wu, and X. Huang, “Trainable weight averaging: Efficient training by optimizing historical solutions,” in *ICLR*, 2022.
[97] S. Jain, S. Addepalli, P. K. Sahu, P. Dey, and R. V. Babu, “Dart: Diversify-aggregate-repeat training improves generalization of neural networks,” in *CVPR*, 2023.
[98] A. Tang, L. Shen, Y. Luo, N. Yin, L. Zhang, and D. Tao, “Merging multi-task models via weight-ensembling mixture of experts,” *arXiv:2402.00433*, 2024.
[99] A. Chronopoulou, M. E. Peters, A. Fraser, and J. Dodge, “Adaptersoup: Weight averaging to improve generalization of pretrained language models,” *arXiv:2302.07027*, 2023.
[100] Z. Zhao, L. Gan, G. Wang, W. Zhou, H. Yang, K. Kuang, and F. Wu, “Loraretriever: Input-aware lora retrieval and composition for mixed tasks in the wild,” *arXiv:2402.09997*, 2024.
[101] X. Wu, S. Huang, and F. Wei, “Mixture of lora experts,” in *ICLR*, 2024.