# ReFRED: Reliable Function REtrieval for Data Transformation

Yueting Chen
Seattle University
Seattle, United States
ychen24@seattleu.edu

Nick Koudas
University of Toronto
Toronto, Canada
koudas@cs.toronto.edu

Xiaohui Yu
York University
Toronto, Canada
xhyu@yorku.ca

## Abstract

Increasingly, Large Language Models (LLMs) are utilized for diverse data processing tasks. In this paper, we focus on leveraging LLMs for reliable data transformation in heterogeneous datasets, a core step in data analysis pipelines. We address the challenges of retrieving appropriate data transformation functions for tasks such as converting Unix timestamps to human-readable dates or extracting specific elements from complex data formats. Current solutions rely on code analysis and manually designed workflows, which are limited in flexibility and semantic understanding.

We propose a novel framework that prioritizes retrieving trusted transformation functions from a pre-defined set using LLMs. If no appropriate function is found, the system falls back on code generation by LLMs, though this is a last resort due to concerns about accuracy, hallucinations, and production reliability. Our framework balances function retrieval accuracy with validation costs using a two-stage approach: an indexing phase for generating function descriptions and embeddings, and a query phase for matching transformations. We consider three variants of the reliable function retrieval problem, introducing conformal prediction techniques, enhanced with abstention and size constraints, to ensure retrieval coverage while controlling validation costs.

Empirical evaluations demonstrate the effectiveness of our solutions, offering a trade-off between retrieval size and validation efficiency, attesting that our solutions to the problems introduced handle abstention scenarios and size constraints effectively.

## Keywords

Function Retrieval, Conformal Prediction, Data Transformation

## 1 Introduction

Data transformations play a pivotal role in database research, enabling the seamless integration of heterogeneous datasets from diverse sources and formats [14, 15, 45]. The heterogeneity of data types and formats often complicates the transformation process, requiring substantial time and expertise from end-users like data scientists and analysts. To mitigate these complexities, the self-service data transformation paradigm has been proposed [14, 15], allowing users to autonomously discover and leverage reusable data transformation functions with greater efficiency.

Figure 1 shows a data transformation case with two datasets from different sources to be analyzed. For the same attribute, such as date, source A encodes it as Unix timestamps, while source B stores it as a time string following a specific format. To extract the day of the week, distinct transformation functions are needed for each source. For example, `dayofweek_from_timestamp()` would be applied to source A, whereas `dayofweek_from_str()` would be used for source B. Once these transformations are executed, further analysis, such as performing statistical operations on

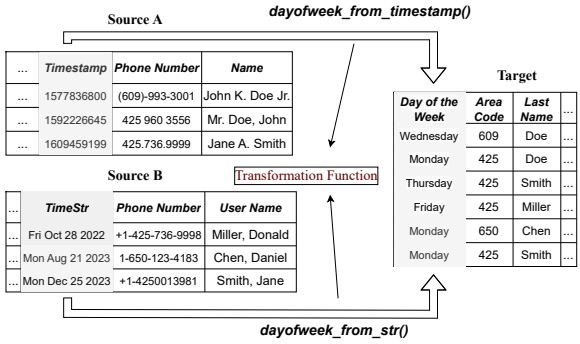



**Figure 1: Data Transformation Examples**

the data by weekday, can proceed. Similarly, we may want to analyze data based on other attributes, such as extracting the area code from phone numbers or isolating last names from full names. Such transformations are common in real-world scenarios and can be time-consuming and labor-intensive if performed manually.

The data transformations described above are frequently necessary when different formats are required for tasks like visualization or joinability search [45]. Crafting the appropriate function for such transformations often demands coding expertise (e.g., proficiency in programming languages) and can be time-consuming, especially when dealing with numerous datasets from varied sources. To minimize manual effort, we aim to allow users to provide a few paired input-output examples as a demonstration, enabling the system to automatically create/identify a transformation function.

**Background:** With the recent success of Large Language Models (LLMs) in a wide range of domains such as natural language question answering (QA) [8], text-to-SQL translation [26], and code generation [29], it is conceivable to prompt an LLM to generate a function that performs the required transformation when presented with input-output example pairs, converting the input into the desired output [7], which is referred to as the *code generation* approach. However, despite the demonstrated strengths of LLMs, the code generation approach faces several challenges, including hallucinations [7], which may result in erroneous code generation [6, 24, 29, 30, 37, 39, 43].

Existing studies demonstrate that code generation for tasks described in text using an LLM achieves an accuracy of up to 89% for classic algorithms [6], which may drop to 75% on a large set of questions, even with additional feedback [37]. Moreover, as the DS-1000 benchmark [24] demonstrates, for in-line code completion in the data science domain, the state-of-the-art models can only achieve 60% accuracy. To corroborate these findings, following the same approach, we generate code for the data transformation tasks of the TDE benchmark [15] utilizing `gpt-4o`, and then validate the code by actual execution, resulting in only 52% accuracy. A detailed manual analysis of the errors shows that the minority of them are due to minor issues such

*EDBT '26, Tampere (Finland)*



10.48786/edbt.2026.22

as representation precision (e.g., using a different precision level for unit transformation) and formatting issues (e.g., using numeric instead of string for output), which are specifically caused by the characteristics of data transformation tasks and can be easily fixed by human inspection. Nonetheless, around 40% of the errors originate from flawed logic, necessitating thorough and detailed code revision. Although in theory errors can be identified by running the generated code on provided examples and prompting the LLM to regenerate the code and test cases [30, 37] until successful, integrating such code into a production environment raises significant reliability and security concerns due to the inherent nondeterministic behavior of LLMs [6]. In practice, any LLM-generated code must undergo human verification to meet production standards. Due to these stringent quality control requirements, leveraging LLMs for direct code generation is considered unreliable, hinders automation, and requires manual inspection [6, 29, 40].

We explore leveraging LLMs to perform function retrieval from a pre-defined set of trusted functions, referred to as *Data Transformation Function Retrieval*, or simply *Function Retrieval*, focusing on retrieving the appropriate function for the transformation task. As invoking such functions may incur costs (e.g., when accessed via APIs) or performance overhead, the brute-force approach that exhaustively tests each function can be expensive [15], especially considering the large number of test cases to be invoked to validate each function. Prior work [14, 15] explored function indexing with code analysis techniques. However, these approaches were developed before the advent of LLMs and exhibit limited automated semantic understanding. As a result, they require complex workflows and extensive preparation, and rely on case execution and example generation to approximate semantic modeling.

**Our Proposal:** We propose a framework that utilizes LLMs to perform function retrieval, resorting to code generation only when necessary, thus combining the advantages of both approaches. We assume that human inspection will always be required to ensure the correctness and reliability of any code generated for production use. To minimize the reliance on code generation and reduce the need for human validation, we assemble a comprehensive set of trusted transformation functions beforehand. Such a pre-assembled set can be extensive, comprising generic functions from widely used libraries and packages, which are assumed to be tested, trusted, and reliable. For any given data transformation request, we first attempt to retrieve the appropriate function from the trusted set using LLMs. The retrieved functions will then undergo a validation step to identify the correct function, which involves applying each retrieved function to the input provided in the query and comparing the output with the desired result. The validation process incurs a quantifiable *validation cost*, which may include both monetary and performance overhead. Code generation, including additional steps such as test case generation and human inspection, is only invoked if the correct function cannot be found. The trusted transformation functions can also be periodically updated to accommodate evolving needs. For simplicity, we assume a comprehensive function set; however, our framework naturally extends to non-comprehensive sets by mapping unmatched queries to a placeholder function that triggers code generation.

For clarity, we use the term *query* to denote the example pair specifying the desired transformation from input to output, and *target function* to refer to the function capable of performing the required data transformation on the example. Our proposed framework: (1) retrieves a set of functions for the given query, (2) validates the retrieved set to determine the target function, and (3) invokes code generation and human inspection if the target function is not identified in the previous steps. To measure the quality of the retrieved set, two common metrics from the literature can be utilized [2, 33]: *coverage*, which indicates whether the target function is included in the retrieved set, and the *average size* of the retrieved set, which reflects the number of functions that need to be verified.

We propose using LLM and embeddings to perform function retrieval, as illustrated in the first part of Figure 2(B). The framework begins with generating textual descriptions for functions and queries, which are subsequently transformed into embeddings. For functions, two approaches are available: (a) if the source code is accessible, we employ an LLM with prompting techniques, where the function code is input, and the LLM generates a text description detailing the function signature and core functionality; (b) if the code is unavailable, we extract text description from human-authored documentation (e.g., textual descriptions of function signatures and behaviors from libraries). These descriptions typically include key information such as input/output data types and the fundamental transformation logic of each function. Similarly, for a given query, we utilize LLMs directly to obtain textual descriptions of functions that perform the transformation from the input to the output. Once the text descriptions for functions and queries are generated, they are transformed into embeddings using an embedding model.

A straightforward approach to perform function retrieval is to utilize embeddings and retrieve the $k$-nearest functions for each given query in the embedding space. In such cases, the average size of the retrieved set is fixed to $k$, while there is no coverage guarantee. Furthermore, the $k$ value for each query required to cover the target function could differ. In practice, one may wish to balance $k$ and coverage based on individual needs.

To solve the above issue, we propose our ReFRED framework and specify an additional calibration step (shown in Figure 2(A)) utilizing Conformal Prediction [41]. During calibration, we assume that a set of (query, target function) pairs has been collected from past examples. Based on the distances between the query and its target function, we perform a calibration step to aid reliable retrieval (detailed in Section 3), introducing coverage guarantees; such distances are stored and utilized for handling new queries.

Before handling new queries, a preprocessing step is performed to obtain, store and index function embeddings for all predefined functions (Section 3.1.1). When a user submits a query by providing an example of the desired transformation, we prompt an LLM to generate a description of a function capable of performing the transformation and perform retrieval utilizing our calibration results. The typical pipeline operates as follows: after the candidate functions are retrieved (Ⓐ in Figure 2, labeled *Retrieval*), we validate the set by executing each function on the provided input-output example. The function that produces the correct output is deemed the *target function*. If none of the retrieved functions can perform the transformation, we instruct the LLM to generate code to accomplish the task (Ⓑ in Figure 2, labeled *Fallback*). In the fallback case, the generated code undergoes a quality control review by a human before being integrated.

**Contributions:** Our main contribution is establishing a quantifiable trade-off between average retrieval size and coverage, balancing retrieval success with validation cost. Moreover, by identifying low-quality queries without sacrificing coverage, we

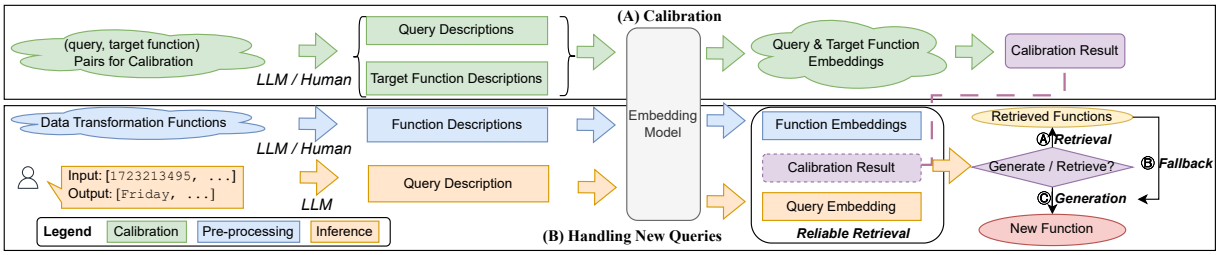



**Figure 2: ReFRED Framework Overview**

can skip validation and directly invoke LLM-based code generation (Ⓒ in Figure 2), reducing overhead. Our contributions are summarized as follows:

- We present the novel problem of utilizing LLM for Reliable Function Retrieval during data transformations.
- We formally define three variants of the reliable function retrieval problem. The first variant establishes quality guarantees for function retrieval in terms of the probability of retrieving the correct function. The second variant extends the problem by allowing the framework to abstain from retrieval when the expected retrieval result size is large, thereby reducing validation costs. The third variant further refines the problem by explicitly bounding the size of the retrieved set, which implicitly controls the validation costs.
- We propose formal solutions for these problems. In particular, we demonstrate how conformal techniques can be utilized to provide probabilistic guarantees in our problem context, and propose novel algorithms Function Retrieval with Abstention (FRA) and Size-Constrained Function Retrieval with Abstention (CFRA), showcasing their theoretical properties.
- We empirically assess the effectiveness of the proposed method on diverse datasets and demonstrate the trade-offs that are feasible using our approach.

## 2 Problem Definition

*Definition 2.1 (Data Transformation Function).* A data transformation function $f \in F$, where $F$ denotes the set of all trusted functions available, maps an input data space $\mathcal{X}_{in}$ to an output data space $\mathcal{X}_{out}$, with no restrictions imposed on the elements of these data spaces. Given an input $x_{in} \sim \mathcal{X}_{in}$, the function $f$ produces an output $x_{out}$. For any input-output pair, denoted as $x = (x_{in}, x_{out}) \sim \mathcal{X}_{in} \times \mathcal{X}_{out}$, the relationship $x_{out} = f(x_{in})$ holds. For simplicity, we refer to the pair $x$ as originating from $f$, denoted as $x \sim f$, where $f$ is the *target function* for $x$.

Given an input-output pair $x$, our focus is on the task of identifying the target function $f$ from a pre-defined set of functions, $F$. We refer to this task as *data transformation function retrieval*.

*Definition 2.2 (Data Transformation Function Retrieval).* For a given query, consisting of an input-output pair $x = (x_{in}, x_{out})$, and a set of transformation functions $F$, Data Transformation Function Retrieval returns a set of functions $F_x \subseteq F$ relevant to query $x$. The quality of $F_x$ can be measured by the metrics discussed below.

We abbreviate this problem as *function retrieval* (or simply *retrieval*). Each query $x$ is assumed to originate from at least one function, termed the target function[1], within the retrieved set $F_x$, i.e. $\exists f \in F_x$ such that $x \sim f$. However, this condition may not always hold, meaning that the target function may not be present in the retrieved set. To capture this, we use a coverage metric to indicate whether the target function is included in the retrieved set.

*Definition 2.3 (Function Retrieval Coverage).* The coverage of a retrieved set $F_x$ for a given query $x$ is defined as:

$$C(x, F_x) = \mathbb{1}_{\exists f \in F_x, x \sim f}$$

For any single query $x$, $C(x, F_x) \in \{0, 1\}$. Thus, for queries drawn from a distribution $P$, the expected retrieval coverage is $\mathbb{E}[C(x, F_x)] \in [0, 1]$. Assuming that the function that answers query $x$ is already in $F$, a straightforward way to achieve full coverage (i.e., $C(x, F_x) = 1$) is to include all available functions in $F_x$ for any query $x$, i.e., $F_x = F$. However, this approach incurs significant validation costs; in the worst case, every function must be verified against the query. Therefore, we aim to reduce the size of the retrieved set while maintaining high coverage.

*Definition 2.4 (Function Retrieval Size).* The size of the retrieval set is defined as the cardinality of the retrieved function set, i.e., $S(F_x) = |F_x|$.

Increasing the retrieval coverage often requires enlarging the retrieval set, which in turn increases the retrieval size. Our objective is to balance the trade-off between coverage and retrieval size.

We study retrieval with coverage guarantees, aiming to minimize retrieval size while ensuring marginal coverage, assuming queries come from a distribution $\mathcal{X}$. We first specify the problem of retrieval with a coverage guarantee.

Problem 1 (Function Retrieval with Marginal Coverage Guarantee). *Given a specified mis-coverage rate $\alpha \in (0, 1)$, retrieval aims to provide a marginal guarantee on coverage, i.e., $\mathbb{E}[C(x, F_x)] \geq 1 - \alpha$ for $x \in \mathcal{X}$.*

However, ensuring coverage alone does not guarantee reduced validation costs, as higher coverage typically entails a larger retrieval set. Ideally, we would like to maintain marginal coverage guarantees for all queries while minimizing retrieval size, allowing for efficient validation without compromising the retrieval of the target function. For queries where this balance cannot be achieved, we can abstain from providing a solution (thus avoiding validation costs) and instead resort to generating the code using a large language model (LLM). To formalize this, we introduce the concept of retrieval with abstention and constraints on retrieval size.

[1]Note that there may be multiple target functions for the same query. Since identifying at least one target function suffices to address the query, we focus on this case without loss of generality.

Problem 2 (Function Retrieval with Abstention). *Given an abstention ratio $\beta \in (0,1)$, only a subset of queries $\mathcal{X}' \subseteq \mathcal{X}$ will be answered, where $\mathbb{P}(\mathcal{X}' \mid \mathcal{X}) = 1-\beta$. For the answered queries $x \sim \mathcal{X}'$, the marginal coverage still satisfies the mis-coverage rate $\alpha$, i.e., $\mathbb{E}[C(x, F_x)] \geq 1-\alpha$, while the expected retrieval size for queries to be abstained is greater than the rest, i.e., $\mathbb{E}[S(F_x) \mid x \in \mathcal{X} \setminus \mathcal{X}'] > \mathbb{E}[S(F_x) \mid x \in \mathcal{X}']$.*

Although our solution to Problem 2 will introduce a tunable abstention ratio while still guaranteeing marginal coverage, we still lack a way to control the retrieval size and thus implicitly control the validation cost. Next, we introduce constraints on retrieval size while maintaining marginal coverage guarantees.

Problem 3 (Size-Constrained Function Retrieval with Abstention). *For a specified retrieval size $\kappa$, only a subset of queries from the original distribution, $P' \subseteq P$, will be answered, such that for $x \sim P'$, the expected coverage satisfies $\mathbb{E}[C(x, F_x)] \geq 1-\alpha$ and the expected retrieval size satisfies $\mathbb{E}[S(x)] \leq \kappa$.*

# 3 Reliable Data Transformation Function Retrieval

## 3.1 Framework Overview

As shown in Figure 3, our framework for handling a new query has two components: Ⓐ Embedding Generation, which maps queries and functions into the same embedding space, and Ⓑ Reliable Function Retrieval, which uses conformal prediction to select candidate functions. Large retrieval sets are abstained to reduce validation cost, triggering direct code generation (B.2), while smaller sets allow direct validation (B.1). If validation fails, the framework falls back to code generation, optionally human-validated before adding to the function set. We discuss each component below.

*3.1.1 Embedding Generation.* For a given query $x = (x_{in}, x_{out})$, both $x_{in}$ and $x_{out}$ can be arbitrary data types. Our approach is to derive natural language descriptions of all functions in $F$ as well as of the function $f$ with property $x_{out} = f(x_{in})$ utilizing LLMs [7].

Specifically, we utilize prompts to instruct LLM to generate two types of descriptions: (1) in a pre-processing step we generate a description[2] for each function in the function set $F$, denoted as $D_f$ (depicted as the Preprocessing Step in Figure 3) and (2) at query time we inquire from an LLM to generate a natural language description of the transformation required to transform $x_{in}$ to $x_{out}$, denoted as $D_x$, (depicted as the Query Step in Figure 3). We impose the same narrative for both types of descriptions and include information such as the signature of the function and a description of the procedure the function implements to conduct the transformation.

To assess the closeness of natural language function descriptions, we utilize embedding models to produce embeddings of descriptions in the same latent space and measure the distance between the query embedding and function embeddings. We use $e_x$ to represent the embedding for query $x$, while $e_f$ represents the embedding for a function $f$. In our framework, we utilize GPT and text embedding models from OpenAI to obtain descriptions and embeddings[3].

[2]This can be accomplished with the assistance of an LLM or utilizing standard function textual descriptions from libraries.

[3]Specifically, we choose `gpt-3.5-turbo-0125` and `text-embedding-3-small`. The specific LLM and associated embeddings utilized are orthogonal to our approach.

Note that during the generation process, both the descriptions and embeddings may be imperfect, potentially introducing noise and erroneous results. We account for this issue and mitigate its impact through reliable function retrieval, which we discuss in the following section.

*3.1.2 Reliable Function Retrieval.* For a given query, we compute its embedding and rank all indexed functions by distance in the embedding space. A common top-$k$ selection fixes $k$, but this limits flexibility, as the retrieval size cannot adapt to query-specific characteristics. Moreover, embedding noise may cause the target function to appear close for some queries yet distant for others.

To overcome this limitation, we propose an alternative strategy that determines a query-specific distance threshold. Based on this threshold, we can retrieve all functions whose distances to the query fall within the defined range, resulting in a variable number of retrieved functions per query. This approach enables more precise control over coverage quality for different queries (depicted as Reliable Function Retrieval in Figure 3). A key challenge is determining the appropriate distance threshold for reliable function retrieval (Figure 3.B.1). Utilizing the concept of conformal prediction [41], we gather a set of query examples with the corresponding target functions as the calibration set and perform a dedicated calibration step to derive the necessary thresholds. Moreover, based on the problem formulation (Problems 2 and 3), we also wish to abstain from answering certain queries and perform code generation directly based on the calibration result (Figure 3.B.2). In the following sections, we explore this topic in depth.

## 3.2 Function Retrieval with Marginal Coverage Guarantee

In this section, we provide a solution to Problem 1. We first introduce conformal predictions in Section 3.2.1, which forms the basis of our subsequent solution in Section 3.2.2.

*3.2.1 Conformal Prediction.* Let $z_i = (x_i, y_i)$ be an *example* observed from some distribution $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the *object space* and $\mathcal{Y}$ is the *label space*. Let $\alpha \in (0,1)$ be the given *error level*. Then conformal prediction [4, 25, 41] provides a marginal guarantee[4] of the prediction set $I(X) \subseteq \mathcal{Y}$ for a new example, $Z \sim \mathcal{Z}$, such that the following holds.

$$\mathbb{P}(Y \in I(X)) \geq 1-\alpha \tag{1}$$

In its standard form, conformal prediction assumes data *exchangeability*, which is weaker than the i.i.d assumption [41], to provide a valid guarantee. The framework is generalized even if the exchangeability assumption does not hold [5], thus providing a general solution framework under any data distribution.

Let $\{z_i = (x_i, y_i) \mid i \in \{1 \dots n\}\}$ be the observations. Conformal prediction works as follows [3]: (a) Define a non-conformity measure $s \in \mathbb{R}$, which is used to quantify the error made by the model. (b) Compute $\hat{q}$ as the $\frac{\lceil (n+1)(1-\alpha) \rceil}{n}$ quantile of the calibration scores $\{s(x_i, y_i) \mid i \in \{1 \dots n\}\}$. (c) For a new example $X$, the prediction set is computed as $C(X) = \{y : s(X, y) \leq \hat{q}\}$.

[4]The guarantee is marginal because it holds on average over the entire data distribution, not necessarily for every single data point. For example, even though the overall coverage is 95%, for some individual points, the actual coverage could be slightly higher or lower.

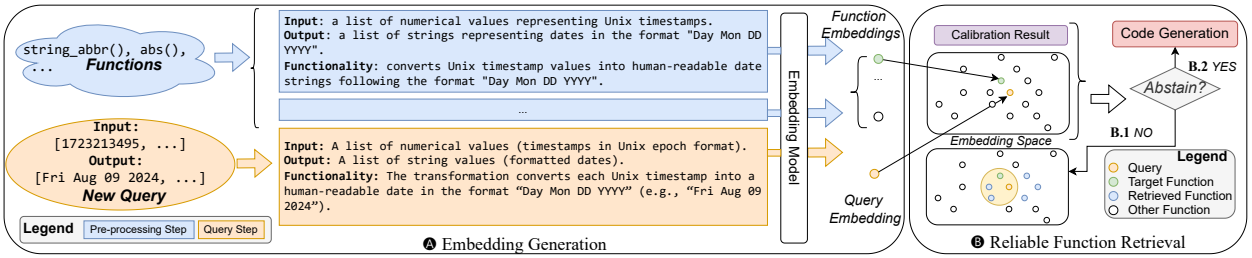



**Figure 3: The Overall Framework with Embedding Generation and Reliable Function Retrieval.**

Both the above calibration step and model training could be costly for each new example; thus, a commonly applied algorithm is the split conformal prediction and its numerous variants [31], where the model is trained and calibrated only once, while the calibration result will be applied for all data points during inference.

*3.2.2 Function Retrieval with Conformal Prediction.* In our setting, we can first build a calibration set by collecting query and target function pairs. This can be accomplished by imposing data transformation requests, providing example input and output (forming a query), and utilizing the set of functions at our disposal to retrieve the true target function; for this, we rank the set of functions with respect to the query, utilizing embeddings and conducting validation. This is a task that is performed once as part of the setup. We then proceed by calibrating a predictor using split conformal prediction [41] using the calibration set. Thus, in this setting, the object space is the queries in the embedding space, $\mathcal{X} = \mathbb{R}^N$, where $N$ is the number of dimensions, while the label space $\mathcal{Y} = \mathbb{R}$ is the space of distances of the queries to their target functions. For each query embedding $e_x \in \mathcal{X}$, the ground-truth distance of the query $x$ to its target function is denoted $d_x \in \mathcal{Y}$. We use the ground-truth distance as our non-conformity score, formally,

$$s(e_x, d_x) = d_x \tag{2}$$

We use $s_x$ to denote the non-conformity score for simplicity in the remaining sections. Then, we compute $\hat{q}$ as the $\frac{\lceil (n+1)(1-\alpha) \rceil}{n}$ quantile of the non-conformity scores in the calibration set (as described in Section 3.2.1). For a new query $q'$, we compute our prediction interval as $I(q') = [0, \hat{q}]$. Conformal prediction guarantees that $\mathbb{P}(d_{q'} \in I(q')) \geq 1 - \alpha$. The functions to be retrieved are collected based on the prediction interval, $F_{q'} = \{f \in F \mid d_f \in I(q')\}$. It is evident that $\mathbb{E}[C(q', F_{q'})] \geq 1 - \alpha$. We refer to this method as Function Retrieval with Conformal Prediction, or simply FRCP.

## 3.3 Function Retrieval with Abstention

In this section, we propose our solution to the problem outlined in Problem 2. Given an abstention ratio $\beta$, our objective is to consistently abstain from queries associated with larger retrieval sizes. However, this presents a challenge, as the retrieval size for a specific query is unknown until the retrieval process has been executed.

One approach is to estimate the retrieval size with conformal prediction and subsequently abstain from queries with larger estimated retrieval sizes. However, this may compromise conformal prediction's coverage guarantee of including the target function. This is because the calibration process focuses on optimizing retrieval size rather than preserving conformal prediction coverage.

To ensure the desired coverage guarantee, the key intuition is that if we can partition the query space $\mathbb{R}^n$ into groups based on their retrieval sizes, and then achieve conditional coverage guarantees within each group, we can safely abstain from queries belonging to groups with larger retrieval sizes. Building on this intuition, we first present a learned approach that provides conditional coverage guarantees for each group in Section 3.3.1. We then introduce our proposed solution and describe how to estimate the retrieval sizes for each group in Section 3.3.2.

*3.3.1 Conformal Prediction with Learned Features.* Kiyani et al. [23] propose a general approach, where the ultimate goal is to provide *full conditional coverage* that for every $x \in \mathcal{X}$:

$$\mathbb{P}(Y \in I(X) \mid X = x) = 1 - \alpha \tag{3}$$

This goal aims to provide a coverage guarantee of $1 - \alpha$ for each query $x$. However, such a full conditional coverage is impossible with a finite-size calibration set [23]. Thus it was proposed to relax it and learn a partitioning of the covariate space such that queries in the same partition are similar in terms of their prediction sets.

Specifically, this partitioning can be represented by a function class $\mathcal{H}$, which outputs a vector of the probabilities of an object $x$ belonging to $m$ groups[5]. Consider a single group and let $S$ be the distribution of the non-conformity scores in that group. It is known that for the random variable $s \in S$ minimizing the expected pinball loss over $\hat{q}$ yields an $(1 - \alpha)$-quantile of the distribution [23].

$$\ell_\alpha(\hat{q}, s) = \begin{cases} \alpha(\hat{q} - s) & \text{if } \hat{q} \geq s, \\ (1-\alpha)(s - \hat{q}) & \text{if } \hat{q} < s. \end{cases}$$

By design, pinball loss aims to optimize by penalizing less when we overestimate and penalizing more when we underestimate the true value. We refer to $q$ as the learned quantile value.

With that in mind, we formulate the following optimization problem. Let $\mathbf{q} = (\hat{q}_1, \ldots \hat{q}_m) \in R^m$, where $m$ is the number of desired groups. Consider a calibration set $(x_i, s_i), 1 \leq i \leq n$ consisting of queries along with their non-conformity scores, and a function class $\mathcal{H}$. The joint optimization problem is formulated as follows:

$$h^*, \hat{q}^* = \underset{\hat{q} \in \mathbb{R}^m, h \in \mathcal{H}}{\arg\min} \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{m} h^i(x_j) \ell_\alpha(\hat{q}_i, s_j) \tag{4}$$

[5] In practice, this can be a neural network with a multi-class output layer or a set of functions such as linear functions.

where $h^i$ is the probability that a query belongs to group $i$, and $s_j$ denotes the non-conformity score for $j$-th example.

To quantify the error of such an approach, *Mean Squared Conditional Error (MSCE)* is introduced, which measures the deviation of prediction sets $I(X)$ from Equation 3, defined as

$$MSCE(\mathcal{Z}, \alpha, I) = \mathbb{E}[(\mathbb{P}(Y \in I(X) \mid X = x) - (1 - \alpha))^2] \quad (5)$$

where $\mathcal{Z}$ is the distribution of the examples (Section 3.2.1). Next, we explore approaches that simultaneously control the proportion of examples in certain groups and guide each group to have distinct retrieval sizes.

In practice, the optimization problem in Equation 4 can be solved using neural networks (i.e., utilizing $h_\theta$ to derive $\mathcal{H}$, where $\theta$ represents the network parameters). The training process can be conducted through alternating gradient descent, where a few gradient descent steps are taken on both $q$ and $\theta$ in each iteration [23].

However, as we will show in our experiments (Section 4), this general approach is not suitable in our context. The primary limitation is that the resulting joint optimization, leading to the learned network $h_\theta$, fails to: (a) control the proportion of examples assigned to each group, and (b) consistently generate partitions where each group exhibits a distinct retrieval size. This occurs because neither the group size nor the retrieval size within each group are explicitly included in the optimization objective.

In Problem 2, our specific goal is to ensure that at least one group has an expected retrieval size larger than that of the other groups. If we can identify such a group, comprising a $\beta$ proportion of samples from the distribution $\mathcal{Z}$, the problem can be addressed by abstaining from this group while maintaining coverage guarantees for the remaining groups. In the following section, we explore methods that allow for controlling the proportion of examples in specific groups while ensuring that each group has a distinct retrieval size.

*3.3.2 Function Retrieval with Abstention.* To address the challenges outlined in Section 3.3.1, we introduce a guided approach for selecting the function class $\mathcal{H}$ during calibration. The key intuition is to partition the space such that: (a) approximately $\beta$ proportion of the samples from the original distribution is assigned to a specific group, denoted as $G_a$, and (b) the queries within group $G_a$ yield retrieval sets that are larger in expectation compared to those in the remaining groups.

The proposed method involves decoupling the joint optimization problem in Equation 4 into two distinct steps: (a) first, partition the calibration set and define the function class $\mathcal{H}$ by training a neural network $h_\theta$ on an *auxiliary classification task*, and (b) once the partitioning is established using the network $h_\theta$, proceed to optimize only for $\hat{q}^*$ in Equation 4. This separation simplifies the optimization process while ensuring that the partitioning aligns with the objective of managing retrieval sizes across groups. We next discuss these two steps separately.

**Step 1. Auxiliary Classification Task.** We now outline the process for preparing the training data for the auxiliary classification task. A key challenge is that the retrieval size for a query cannot be directly obtained, as it is only available when the retrieval is performed with the specified mis-coverage rate $\alpha$. To address this, we propose using a *minimal retrieval size* as a proxy to guide the partitioning (and we refer to the steps below collectively as *minimal retrieval size computation*), defined as follows:

(1) First, compute the non-conformity score $s_x$ for each query $x$ as specified in Equation 2.
(2) Next, construct an interval $\hat{I}(x) = [0, s_x]$ for each query $x$. This interval is the minimum interval that guarantees coverage of the ground-truth distance between the query and its target function.
(3) Using the interval $\hat{I}(x)$, define a set of functions $\hat{F}_x = \{f \in F \mid d_f \in \hat{I}(x)\}$, referred to as the *minimal retrieval function set*. This set contains the smallest number of functions that ensures inclusion of the target function for query $x$.
(4) The size of the set $|\hat{F}_x|$ is used as the *minimal retrieval size*.

This minimal retrieval size serves as a proxy to guide the partitioning of queries during the auxiliary classification task. With the minimal retrieval size, we can construct a dataset to train $\mathcal{H}$ on the auxiliary task. To achieve this, we introduce the Function Retrieval with Abstention (FRA) Algorithm, shown in Algorithm 1.

**Algorithm 1:** FRA

1 $L_x \leftarrow$ the list of query embeddings in the calibration set;
2 $L_s \leftarrow$ a list of *minimal retrieval sizes*;
3 $s_\beta \leftarrow$ the $1-\beta$ quantile of $L_s$; $L_l \leftarrow \{\}$;
4 **foreach** $s \in L_s$ **do**
5 $\quad label \leftarrow s \leq s_\beta \ ? \ 0 : 1$; $L_l$.append($label$);
6 $h_\theta \leftarrow$ the neural network for partitioning;
7 Train $h_\theta$ on $(L_x, L_l)$ as a classification task;

We assume, without loss of generality, that the query space is partitioned into two groups, with one group representing $\beta$ proportion of the examples. The algorithm operates as follows: First, compute the minimal retrieval size for each query (Steps 1-2), and then determine the $(1-\beta)$ quantile from the computed retrieval sizes (Step 3), which will serve as the split point for partitioning. Next, queries in the calibration set are labeled based on this split value (Step 5). Specifically, a neural network $h_\theta$ is trained using the labeled data (Steps 6-7). For instance, with $\beta = 0.2$, the algorithm identifies the 0.8 quantile of the minimal retrieval sizes, which serves as the threshold ($s_\beta$) for group assignment. Queries with a minimal retrieval size less than $s_\beta$ are assigned a label of 0, while those with a minimal retrieval size greater than or equal to $s_\beta$ are assigned a label of 1. In this case, queries labeled as 1 constitute approximately $\beta$ proportion of the set and will be treated as the group for abstention.

The algorithm can be readily extended to support more groups, though we omit the details here for brevity. The network $h_\theta$ is trained using labeled data, employing Binary Cross Entropy loss for two groups or Cross Entropy loss when handling multiple groups. Dropout can be applied during training to prevent overfitting.

It is important to note that the labels generated in the algorithm (Steps 4-7) are used solely to guide the training of the partitioning network $h_\theta$. These manually assigned labels may not fully capture the underlying structure of the original distribution space. Therefore, after training $h_\theta$, we use it to re-assign labels to the queries in the calibration set. The intuition here is that $h_\theta$ can learn structural patterns in the covariates, enabling consistent partitioning of future queries based on these learned patterns.

**Step 2. Obtaining quantile values.** After obtaining the partitioning network $h_\theta$, it remains necessary to determine the appropriate quantile values within the groups we choose not to abstain from in order to solve Problem 2. We explore two potential approaches:

***Learned Approach (LA)***: We fix the network $h_\theta$ and use gradient descent to optimize $\hat{q}^*$, as specified in Equation 4. This approach modifies only the training process of $\mathcal{H}$, and as a result, the theoretical analysis presented in Section 3.3.1 remains valid.

***Conformal Prediction Approach (CPA)***: After training $h_\theta$, we can directly apply conformal prediction within the groups we opt not to abstain from. In this scenario, $h_\theta$ is treated as a partitioning function based on covariates, aligning with the framework of group-conditional conformal prediction [32]. We can thus apply our proposed FRCP method (Section 3.2.2) for the group that chooses not to abstain; we refer to this approach as CPA. This approach provides marginal coverage guarantees within each group, ensuring coverage for the remaining non-abstaining groups. Formally, let $G_a$ denote the group that $h_\theta$ chooses to abstain. Under exchangeability, for $X \sim \mathcal{X}$, it's trivial that the following holds:

$$\mathbb{P}(Y \in I(X) \mid X \notin G_a) \geq 1 - \alpha$$

**Analysis.** We utilize the *minimal retrieval size* to guide the partitioning network in learning how to partition based on covariates. We manually construct the training set for the classification task, which serves as a guide for partitioning. Within this training set, we assign a proportion $\beta$ of the queries with the largest *minimal retrieval size* to a group, denoted as $G_a$. The partitioning network then learns, based on query features, to partition future queries such that one group will consistently contain approximately $\beta$ of the queries, characterized by the largest expected retrieval sizes.

Formally, let $G_a$ (labeled as 1) and $G_b$ (labeled as 0) be two groups of queries from the calibration set utilizing Algorithm 1, drawn from distribution $\mathcal{X}$, and let $S'(x)$ denote the minimal retrieval size for query $x$. Since we partition groups based on the minimal retrieval size, it follows that $\mathbb{E}[S'(F_x) \mid x \in G_a] > \mathbb{E}[S'(F_x) \mid x \in G_b]$.

From the steps in the *minimal retrieval size computation*, for a query $x$, a larger $S'(x)$ corresponds to a higher non-conformality score $s_x$, and conversely, a smaller $S'(x)$ implies a lower $s_x$. Without loss of generality, we can state that $\mathbb{E}[s_x \mid x \in \mathcal{X}]$ increases as $\mathbb{E}[S'(F_x) \mid x \in \mathcal{X}]$ increases. Let $\hat{q}_a$ be the $1-\alpha$ quantile of non-conformity scores for group $G_a$, and $\hat{q}_b$ be the corresponding quantile for group $G_b$. We have $\hat{q}_a > \hat{q}_b$. Since the retrieval size is computed based on the quantile, a higher quantile $\hat{q}$ results in a larger retrieval size. Therefore, as $\hat{q}$ increases, the expected retrieval size $\mathbb{E}[S(F_x) \mid x \in \mathcal{X}]$ also increases. Consequently, we have $\mathbb{E}[S(F_x) \mid x \in G_a] > \mathbb{E}[S(F_x) \mid x \in G_b]$.

Let $\mathcal{L}$ be the loss function of the partitioning network $h_\theta$. When $\mathcal{L} = 0$ and the calibration set is sufficiently large, under the i.i.d. assumption, for any query drawn from the origin distribution, where $x \in \mathcal{X}$, the following holds:

$$\mathbb{E}[S(F_x) \mid h_\theta(x) = 1] > \mathbb{E}[S(F_x) \mid h_\theta(x) = 0] \tag{6}$$

As the model converges in practice, $\mathcal{L}$ approaches zero, ensuring that Equation 6 remains valid. We will empirically validate this approach in the experiments section.

*3.3.3 Function Retrieval with Abstention for New Queries.* After calibration, we should obtain: (a) the partitioning network $h_\theta$, which assigns a given query to a group, and (b) the quantile value $\hat{q}_i$ for each group $i$. To provide reliable function retrieval, for a new query $q'$, we first assign it to a group using $h_\theta$. Two cases arise: (1) if $q'$ is assigned to the group which we wish to abstain from, $G_a$, we perform abstention directly; (b) if $q'$ is assigned to any other group $i$, we calculate our prediction interval $I(q') = [0, \hat{q}_i]$, where $\hat{q}_i$ is the quantile value associated with the group $i$, and produce retrieval set $F_{q'} = \{f \in F \mid d_f \in I(q')\}$.

## 3.4 Size-Constrained Function Retrieval with Abstention

We now address Problem 3, where the goal is to specify a target retrieval size $\kappa$ such that queries with a retrieval size larger than $\kappa$ are subject to abstention. For simplicity, we continue to denote the abstaining group of queries as $G_a$ and the non-abstaining group as $G_b$. This problem is challenging because the partitions $G_a$ and $G_b$ must be determined, and the exact expected retrieval size for queries in $G_b$ can only be computed—given a mis-coverage rate $\alpha$—once the partitioning is finalized. To address this, we propose a method aimed at ensuring that the expected retrieval size for $G_b$ is as close as possible to the specified target $\kappa$.

In Section 3.3.2, we introduced the concept of the *minimal retrieval size* for partitioning the calibration set. A straightforward strategy involves using the target retrieval size $\kappa$ as a threshold to partition the calibration set: samples with a *minimal retrieval size* exceeding $\kappa$ are assigned to $G_a$, while the remaining samples are assigned to $G_b$. We refer to this strategy as the *initial partition*.

To ensure marginal coverage, we compute the quantile value for group $G_b$ after forming the initial partition. This value will be used to calculate the actual retrieval size for queries in $G_b$. However, the actual retrieval size may differ significantly from $\kappa$. To better align with $\kappa$, we propose an *iterative adjustment* to the initial partition. We refer to such a method Size-Constrained Function Retrieval with Abstention (CFRA), detailed in Algorithm 2.

We first partition the calibration set into two groups based on $\kappa$ using *minimal retrieval size* (i.e., we derive the *initial partition*) (Lines 1-3). Then, we collect the unique distance values between each query and its target function in each group (Lines 4-5) and compute $\hat{q}$ for group $G_b$ with the given mis-coverage rate $\alpha$ (Line 6), based on which we can obtain the actual retrieval size for group $G_b$ (Line 7). The computation works as follows: (a) we first obtain the actual retrieval set for each query $x \in G_b$ with the quantile value $\hat{q}$, (b) we then compute the retrieval size for each query $x$, and aggregate the average value among all queries in $G_b$, denote as *current average retrieval size* $\hat{n}$. Depending on the value of $\hat{n}$, we either remove data points from $G_a$ (Lines 9-13) or add more data points to $G_b$ (Lines 15-21). For example, if $\hat{n}$ exceeds the target $\kappa$, we iteratively remove points with the highest distances (Lines 9-11) and stop once $\hat{n}$ falls just below $\kappa$ (Lines 12-13, lines 19-21). We then construct the training labels accordingly and train $h_\theta$ as outlined in Section 3.3.2 (Lines 22-23). Similarly, the quantile value for the non-abstain group can be obtained using the same FRCP method discussed in Section 3.2.2.

With Algorithm 2, we construct the labels for groups $G_a$ and $G_b$, such that $\mathbb{E}[S(F_x) \mid x \in G_b] \leq \kappa$, and use these labels to train the partitioning network $h_\theta$. Under the i.i.d. assumption, for queries drawn from the original distribution $x \in \mathcal{X}$, with a sufficiently large calibration set, as the loss of the partitioning network approaches zero (i.e., $\mathcal{L} \to 0$), the objective $\mathbb{E}[S(F_x) \mid h_\theta(x) = 0] \leq \kappa$ can be achieved. We will empirically demonstrate

**Algorithm 2:** CFRA

```
L_x ← the list of query embeddings in the calibration set;
G_a ← x ∈ L_x with minimal retrieval size > κ;
G_b ← x ∈ L_x with minimal retrieval size ≤ κ;
D_a ← unique distances between
  queries and the groud-truth function in G_a;
D_b ← unique distances between
  queries and the groud-truth in G_b;
q̂ ← compute_q_hat(G_b, α) ;
n̂ ←compute_average_size(G_b, q̂);
if n̂ > κ then
    slist ←sort D_b in descending order;
    for s ∈ slist do
        remove examples with distance s from G_b;
        recompute q̂ and n̂;
        if n̂ < κ then break ;
else
    slist ←sort D_a in ascending order;
    for s ∈ slist do
        add examples with distance s to G_b;
        recompute q̂ and n̂;
        if n̂ > κ then
            remove the example added in the previous step ;
            break;
L_l ← labels based on G_a and G_b;
train h_θ on (L_x, L_l) as a classification task;
```

in the experiments that this method yields an average retrieval size that is less than $\kappa$.

After calibration, we obtain the partitioning network $h_\theta$ and quantile $\hat{q}_i$ for each group. For a new query $q'$, inference follows Section 3.3.3 and is omitted for brevity.

## 3.5 Extensions

In this section, we further discuss two extensions to the problems.

*3.5.1 Reliable Retrieval Beyond Exchangeability.* In the general case, conformal methods accommodating non-exchangeability [5] can also be applied, with our framework still applicable. The main change lies in computing the quantile in FRCP (Section 3.2.2).

Specifically, in the non-exchangeability setting, we assign a weight to each example in the calibration set during query time to compute the threshold value. Let $e_q$ represent the embedding of the new query $q$, and the weight of an example $x$ from the calibration set is computed as $w_x = exp(-d(e_q, e_x)/\tau)$ for a hyper-parameter $\tau$ (default value of 1). This weight controls the influence of each calibration, with the intuition that points closer to the query point should exert more influence. The weights are normalized as $\hat{w}_x = w_x/(1 + \sum_{x \in \mathcal{X}_{cal}} w_x)$, where $\mathcal{X}_{cal}$ is the calibration set. The cutoff threshold $\hat{q}$ is computed as $\hat{q} = \inf\{s \mid \mathbb{1}_{(s_x \le s)} \hat{w}_x \ge 1 - \alpha\}$, where $s_x$ is the non-conformity score as defined in Section 3.2.2, and $\mathbb{1}_{cond}$ is the indicator function that equals 1 if the condition *cond* is met and 0 otherwise.

With the non-exchangeability assumption, the bound of Equation 1 becomes slightly looser. Further details of the analysis can be found in [5]. By modifying the quantile computation in FRCP, our framework extends to non-exchangeability settings while still providing a marginal coverage guarantee for all three problems we discussed so far. We will compare these methods in the experiments.

*3.5.2 Reliable Retrieval with an Additional Regression Model.* Our framework operates in the embedding space, leveraging natural language descriptions. However, due to the imprecision and vastness of natural language, performing retrieval based solely on the nearest neighbors may not always yield relevant functions. It is natural to expect that the quality of descriptions of the query transformation and function descriptions, directly impacts the quality of nearest neighbor retrieval. Assuming we can achieve quality descriptions, a regression model can be introduced to predict the expected distance to the target function in the embedding space before retrieval, thereby refining the retrieval set. Let $M_{reg}$ be a regression model predicting the distance to the target function, denoted $d'_x = M_{reg}(x)$, for a given query $x$. Rather than directly using nearest neighbor retrieval, we define a prediction band around $d'_x$ to construct the retrieval set.

Our framework supports this extension with two modifications: (a) redefining the non-conformity score to be the residual between the predicted and actual distance to the target function, i.e., $s'(e_x, d_x) = |M_{reg}(x) - d_x|$; and (b) defining the prediction interval $I'(x) = [d'_x - \hat{q}, d'_x + \hat{q}]$, from which the retrieval set is constructed as $F'_x = \{f \mid d_f \in I'(x)\}$.

In this setting, we expect that $|F'_x| < |\hat{F}_x|$, where $\hat{F}_x$ is the retrieval set constructed using the FRCP method in Section 3.2.2. The intuition is that, if the regression model performs well, the prediction interval will be narrower than without $M_{reg}$, meaning that $|d'_x - d_x| < d_x$. Assuming a uniform distribution of functions in the embedding space, this should result in $|F'_x| < |\hat{F}_x|$. We will empirically demonstrate the impact of including and excluding the regression model in our framework in the experiments.

# 4 Experimental Evaluation

## 4.1 Experiments Setting

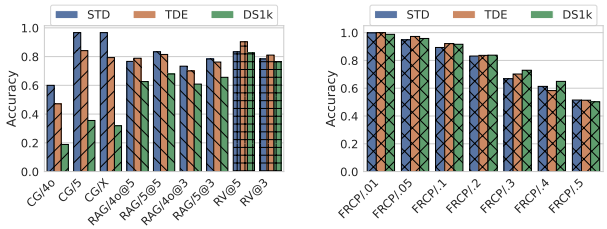
We employ `GPT-3.5-turbo-0125` as the large language model (LLM) in our experiments, to generate descriptions of functions and queries, and `text-embedding-3-small` to obtain embeddings. All algorithms are implemented in Python. We use three datasets with different complexities: (a) **STD**: We created the Simple Transformation Dataset (STD) with 60 functions over numerical, string, and boolean data. Each function has 20 queries, totaling 1,200 queries. (b) **TDE**: We use the dataset from [15], containing complex transformations (e.g., unit conversion, pattern extraction, URL encoding), and augment it with additional generated queries, yielding 2,250 queries over 227 functions. (c) **DS1K**: To enable large-scale validation, we adapt DS-1000 [24], yielding 13,129 queries over 771 functions. Unlike simple transformations, these functions target specific code-completion scenarios, where code snippets serve as functions and task descriptions as queries.

The transformation functions for each dataset are implemented in Python, and LLM-generated descriptions are provided for each function. Note that our method is independent of the LLM choice and function implementations, as we focus on reliable retrieval in the embedding space. All the datasets are made publicly available [6]. Unless otherwise specified, each dataset is split into calibration and test sets evenly. To demonstrate statistical trends, we perform 10 independent runs with different random seeds for each method and report the results from all runs.

The experiments aim to achieve three *primary objectives*:

- *Coverage Validation.* To ensure that the empirical coverage across all methods (Sections 3.2 to 3.4) aligns with the

[6] https://github.com/dbllm/refred




(a) Baseline Methods (b) FRCP with Different $\alpha$ Levels

**Figure 4: End-to-End Evaluation Results on All Datasets**

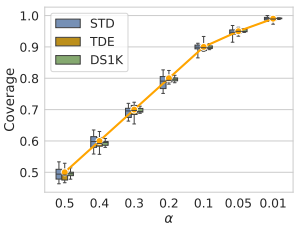



**Figure 5: Coverage for Varying $\alpha$ with FRCP**

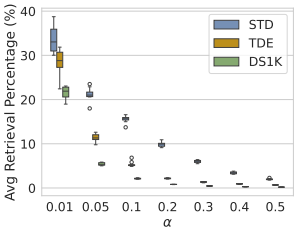



**Figure 6: Retrieval Percentage for Varying $\alpha$ with FRCP**

target level, maintaining the specified mis-coverage rate, $\alpha$.

- *Average Retrieval Percentage Comparison.* For methods incorporating an abstention option (Sections 3.3 and 3.4), we aim to demonstrate that queries with abstention consistently result in a larger empirical average retrieval size if the retrieval is performed, than those without. To better visualize the results, we show the average retrieval size as a percentage of the total number of pre-defined functions for all queries, denoted as average retrieval percentage.
- *Method Extensions Evaluation.* To demonstrate that our proposed extension methods (Section 3.5) enhance either the empirical coverage or the average retrieval percentage.

Ultimately, we seek to show that by tuning the mis-coverage rate $\alpha$, one can balance the trade-off between function execution cost during validation and function generation cost.

## 4.2 Reliable Function Retrieval

In this section, we follow the objectives outlined in Section 4.1, and evaluate the *Coverage Validation* and *Average Retrieval Percentage Comparison* objectives first across all methods (Sections 4.2.2 to 4.2.4). We then address the *Method Extensions Evaluation* in Sections 4.2.5 and 4.2.6. Additionally, in Section 4.2.7, we vary the embedding models to demonstrate that our method is not sensitive to the specific choice of embedding model. Finally, in Section 4.2.8, we present the trade-off between function execution and code generation.

*4.2.1 Comparison with other Baselines.* To demonstrate the benefit of our proposed methods, we compare against popular approaches for the function transformation task, including: (1) **CG (Code Generation):** LLM directly generates Python code from input/output pairs. (2) **RAG (Retrieval-Augmented Generation):** Queries and functions are first embedded (as in Section 4.1), the top-$k$ functions are then retrieved, and an LLM selects one as the transformation function. (3) **RV (Retrieval-and-Validation):** Similar to RAG, but instead of LLM selection, functions are validated against the query, guaranteeing correctness if the target function is retrieved. We also evaluate our proposed **FRCP** method (Section 3.2.2), which applies conformal prediction with mis-coverage rate $\alpha \in [0.01, 0.5]$ and reports validation accuracy.

We show our experiment results on the above baselines in Figure 4a. For both approaches CG and RAG, the result quality depends on various factors, including model, prompts, etc. We report our results using the best prompt we tried. Results are shown for three LLMs, including `gpt-4o-mini` (denoted as `CG/4o` and `RAG/4o`), `gpt-5-mini` (denoted as `CG/5` and `RAG/5`), and `codex-mini-latest` model finetuned on code generation tasks (denoted as `CG/X` and `RAG/X`). The CG method achieves varying levels of accuracy across different datasets: among them, it performs best on the STD dataset and worst on the DS1K dataset, highlighting the intrinsic challenge of the task when employing the code generation approach.

For RAG and RV, we also vary $k \in \{3, 5\}$ to examine its impact, denoted as `RAG@k` and `RV@k`. The accuracy of CG and RAG depends heavily on the underlying LLM, with `gpt-5-mini` performing best. For retrieval-based methods (i.e., RAG and RV), accuracy is limited by LLM generation errors, so RAG cannot exceed RV. Increasing $k$ improves both RAG and RV, but all baseline methods operate on a best-effort basis, leaving accuracy beyond user control.

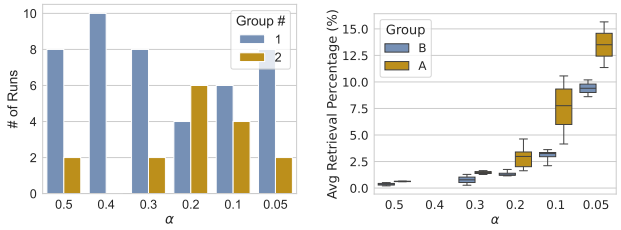
As can be observed from Figure 4b, our proposed method FRCP allows users to specify a mis-coverage rate $\alpha$ (denoted as `FRCP/`$\alpha$, where $\alpha$ varies from 0.01 to 0.5), enabling the achieved accuracy to closely match the user's target. Such results can be consistently observed on all these datasets. The parameter $\alpha$ indirectly controls the size of the retrieved function set, thereby influencing either the cost of LLM invocations for code generation or the cost of function invocations. We will further discuss such topics in detail in the following sections.

*4.2.2 Function Retrieval with Marginal Coverage Guarantee.* We present the results of our proposed method, FRCP, introduced in Section 3.2.2, where we vary the mis-coverage rate $\alpha$ from 0.5 to 0.01. Figure 5 illustrates the expected coverage rate as a diagonal line. Each box plot corresponds to a different $\alpha$ value and represents the coverage over 10 independent runs. The boxes depict the interquartile range, which encompasses the middle 50% of the coverage values, with whiskers extending to the minimum and maximum observed values. Note that while the x-axis labels appear evenly spaced for better visualization, the corresponding $\alpha$ values are not uniformly distributed. In both data sets, we observe that the expected coverage rate is statistically maintained for each $\alpha$.

From Figure 6, as $\alpha$ decreases to 0.01, the coverage improves, but the average retrieval percentage increases to more than 40% for STD, 30% for TDE, and 20% for DS1k, imposing significant validation costs. This suggests that while function retrieval using large language models (LLMs) yields practical results, there is a tradeoff: higher coverage comes at the cost of larger validation sets. Our method allows users to manage this tradeoff by adjusting the mis-coverage rate ($\alpha$), offering control over the balance between coverage and validation cost.

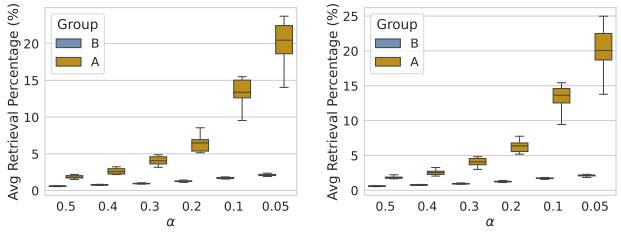
*4.2.3 Function Retrieval with Abstention.* We begin by presenting the performance of the general approach, CPLF, as outlined in Section 3.3.1. Following this, we discuss the results of our proposed methods in detail.

**CPLF**: We implement the CPLF method proposed by Kiyani, et. al. [23] and utilize it for our task, using two groups (i.e., setting the number of output classes to 2 for the partitioning network). For each value of $\alpha$, we run CPLF 10 times independently and




(a) # of Groups Obtained (b) Avg. Retrieval Percentage

**Figure 7: CPLF on TDE Dataset.**




(a) FRA+LA (b) FRA+CPA

**Figure 8: Avg. Retrieval Percentage per Group on TDE**

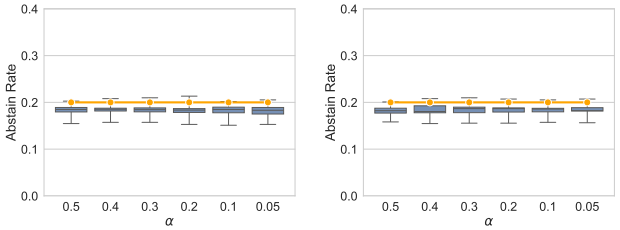
count how many groups are obtained at the end of each run. Due to space constraints, we present the results for the TDE dataset in Figure 7; the results on other datasets exhibit a similar trend. Figure 7a presents the actual number of groups obtained for different values of $\alpha$. For example, when $\alpha = 0.05$, 2 out of 10 runs produce 2 groups, while 8 runs only produce one group. CPLF does not consistently generate the desired two groups across both datasets. The best performance is observed with the TDE dataset when $\alpha = 0.2$, where two groups are obtained in the result for 6 out of 10 runs. Further increasing the expected number of groups produces similar results.

This behavior can be attributed to two key factors: (a) optimizing the joint problem involving both the partitioning network and the quantile value is complex and may require careful hyperparameter tuning, and (b) the large embedding space for natural language descriptions introduces multiple possible solutions to the joint optimization problem. Additionally, the inherent randomness in the gradient descent process can lead to suboptimal outcomes.

In Figure 7b, we also plot the average retrieval percentage for each group when two groups are successfully formed. Since the average retrieval size is not explicitly optimized by CPLF, the percentages of the two groups may not vary significantly. For example, when $\alpha = 0.1$, there is a noticeable gap between the average retrieval percentages of groups A and B. However, as $\alpha$ increases to 0.3, the retrieval percentages for the two groups become more similar. These findings suggest that CPLF is not suited for our specific task, emphasizing the necessity of our proposed method.

**FRA:** We present our FRA-based methods, described in Section 3.3.2, in Figures 8 to 11. Specifically, FRA is used to obtain the partitioning network while applying either LA (referred to as FRA+LA) or CPA (referred to as FRA+CPA) to compute the quantile value. The abstention rate is set by default to $\alpha = 0.2$, meaning that approximately 20% of queries are expected to be abstained.

Figure 8 shows the average retrieval percentage on the TDE dataset; the results on other datasets show a similar trend and are omitted due to space constraints. Since the partitioning network is guided by the grouping results utilizing minimal retrieval




(a) FRA+LA (b) FRA+CPA

**Figure 9: Abstain Rate with FRA (TDE)**

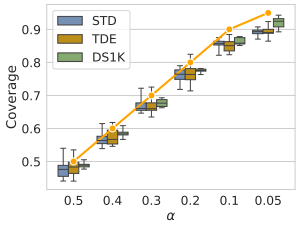



**Figure 10: Coverage with FRA+LA**

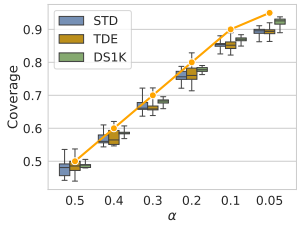
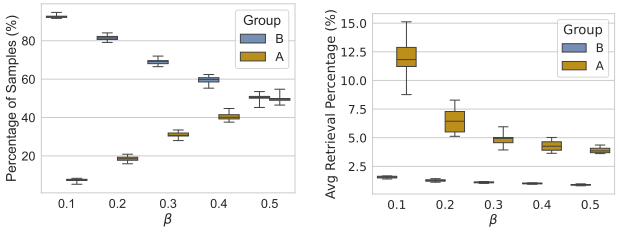



**Figure 11: Coverage with FRA+CPA**




(a) Percentage of Samples (b) Avg. Retrieval Percentage

**Figure 12: Varying $\beta$ on TDE (FRA+CPA)**

percentage, our method consistently identifies two groups with significantly different average retrieval percentages, as reflected in both figures. We denote group A as the set of queries that will be abstained from answering, while group B represents the remaining queries. Additionally, the specified abstention rate is met, as shown in Figure 9. The results for the STD dataset exhibit the same trend and are omitted for brevity. In the remaining experiments, we focus on FRA+CPA, as FRA+LA yields similar results.

Furthermore, we plot the coverage for all datasets using both approaches in Figures 10 and 11. As shown, the empirical coverage is maintained near the target value. However, compared to the base conformal prediction method applied to the entire dataset (Figure 5), the empirical coverage for non-abstained queries tends to be lower than expected. This discrepancy stems from the bias introduced by the partitioning network, which violates the exchangeability assumption. Later in this section, we will demonstrate how methods that do not assume exchangeability can address this issue.

**Varying $\beta$:** We also vary the value of $\beta$ from 0.1 to 0.5 to demonstrate the flexibility of the FRA method and present the results on TDE in Figure 12 (STD also shows similar results). Regardless of the specific value of $\beta$, the partitioning network consistently maintains the desired proportion of samples to abstain from answering (Group A in Figure 12a). Furthermore, we observe that the average retrieval percentage of each group remains significantly different across all $\beta$ values in Figure 12b, suggesting that queries likely to produce larger retrieval percentages are more prone to being abstained.

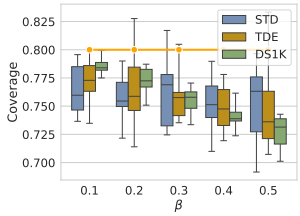



**Figure 13: Coverage Varying $\beta$ (FRA+CPA)**

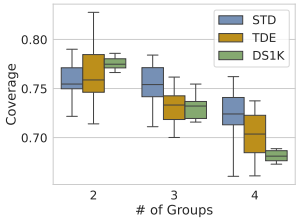
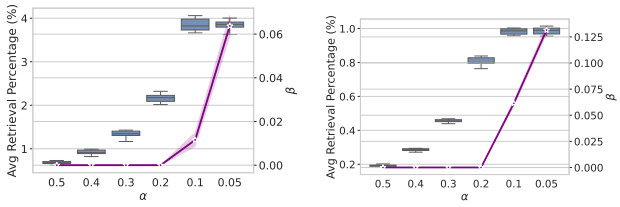



**Figure 14: Coverage Varying # of Groups (FRA+CPA)**




**(a) TDE** **(b) DS1K**

**Figure 15: Avg. Retrieval Percentage with CFRA+CPA Varying $\alpha$**

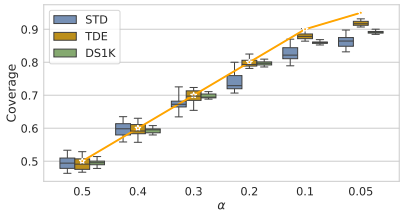



**Figure 16: Coverage with CFRA+CPA Varying $\alpha$**

As the abstention rate $\beta$ increases, we also notice a drop in empirical coverage, as shown in Figure 13. This decrease in coverage is attributed to two factors: (a) as more queries are abstained, fewer samples remain in each group for calibration, leading to poorer calibration results; and (b) the partitioning network may introduce bias, violating the exchangeability assumption.

**Varying Number of Groups:** The FRA method can be extended to support more than two groups. We increase the number of groups from 2 to 4 and compute the overall coverage for non-abstained queries, as shown in Figure 14. As the number of groups increases, the overall coverage decreases. This trend is consistent with our earlier observation in Figure 11, where empirical coverage tends to fall below the expected coverage. As more groups are introduced, the gap between empirical and expected coverage widens. These findings suggest that, for reliable function retrieval with abstention, using two groups yields the best results.

*4.2.4 Size-Constrained Function Retrieval with Abstention.* For the solution to Problem 3, we set $\kappa = 4\%$ of the number of predefined functions ($\kappa = 1\%$ for the DS1K dataset due to the high number of functions) and evaluate the CFRA method (Section 3.4) with CPA to compute quantile values across varying values of $\alpha$. Figure 15 presents the results for TDE and DS1K. The results for STD are omitted as they exhibit a trend similar to those observed with TDE. The box plot shows the average retrieval percentage for non-abstained queries (left y-axis), while the line plot presents the empirical abstention rate $\beta$ (right y-axis). When $\alpha = 0.5$, the abstention rate for both datasets is 0, which means that all queries are answered. As $\alpha$ decreases from 0.4 to 0.1, the empirical abstention rate increases. Furthermore, as shown in Figure 16, the gap between the empirical and expected coverage increases as $\alpha$ approaches 0.05. This pattern aligns with our earlier observations regarding the impact of varying $\beta$ in Section 4.2.3. Later, we will demonstrate how incorporating non-exchangeability can help reduce the coverage gap.

We also vary $\kappa$ from 2% to 8% of $|F|$, and Figure 17 presents that the empirical average retrieval percentage remains controlled within the $\kappa$ threshold on TDE. As $\kappa$ increases, the empirical abstention rate $\beta$ decreases (right y-axis), as more queries are answered without abstention. The same trend is also observed on STD and omitted for brevity.

*4.2.5 Reliable Function Retrieval with Non-Exchangeability.* We implemented all three of our methods utilizing non-exchangeability, as described in Section 3.5.1; we append the suffix "+W" to their notation to signify this (denoting a weighted approach). The results on TDE are presented in Figure 18, while the same trend can be observed on STD as well (omitted due to space constraints). Compared to the original methods, incorporating non-exchangeability reduces the gap between the empirical coverage and the target coverage. This suggests that conducting estimations under non-exchangeability is the preferred methodology in practice.

*4.2.6 Reliable Function Retrieval with Additional Regression Models.* We also conducted experiments with regression models, as discussed in Section 3.5.2. Specifically, we evaluated two models: Support Vector Regression (SVR) and Multi-Layer Perceptron (MLP); we compared them to a baseline that does not utilize a regression model (NoReg) on the TDE dataset using the FRCP method. The comparison is shown in Figure 19, with other methods and datasets exhibiting similar trends.

For this experiment, the dataset was split in a 4:3:3 ratio, where 40% of the data was used to train the regression models, while the remaining was split evenly for calibration and testing. Regardless of the regression model used, target coverage was consistently maintained, as shown in Figure 19a. For improved visualization, we used a line plot where the solid line represents the mean value, and the shaded area around it indicates the range between the minimum and maximum values at each $\alpha$ level. However, the advantage of using regression models is not evident when $\alpha$ is large; in fact, the average retrieval percentage may increase significantly if models like SVR are applied. When the target coverage is low, results based on nearest neighbors are sufficiently effective. On the other hand, when a higher coverage rate is needed (i.e., with lower $\alpha$ values), regression models, such as SVR and MLP, can reduce the average retrieval percentage, as observed when $\alpha = 0.1$.

*4.2.7 Varying Embedding Methods.* We also demonstrate that our proposed approach is orthogonal to the choice of embedding methods Specifically, we switch the embedding model to `all-mpnet-base-v2` for function and query descriptions. Figure 20 presents experimental results on the DS1K dataset using the FPA+CPA method, as discussed in Section 4.2.3. As shown, the observed trends are consistent with our previous experiments, confirming that our methods are compatible with varying embedding models. Similar trends were observed in all other experiments, which are omitted here for brevity.

*4.2.8 Reliable Data Transformations with Code Generation.* Finally, we demonstrate that varying $\alpha$ can effectively balance the number of function validation invocations, which occur during the validation phase, against the number of code generations. In

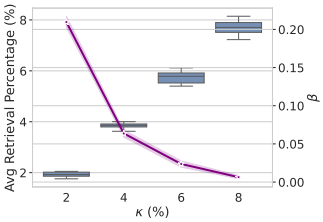
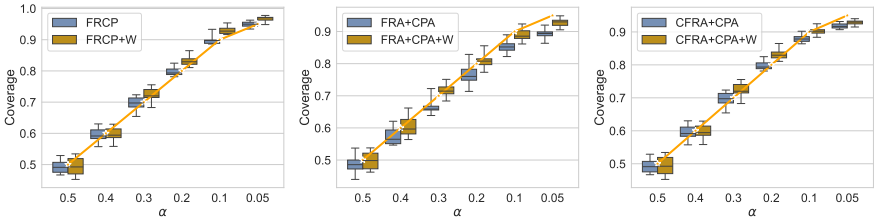
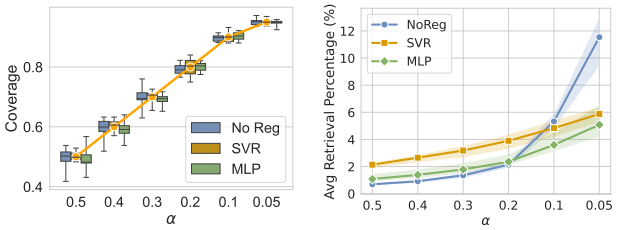



**Figure 17: Avg. Retrieval Percentage Varying $\kappa$ (CFPR+CPA)**



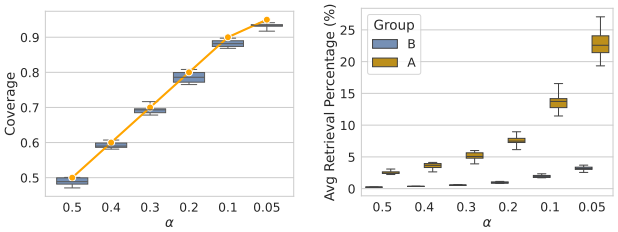

**(a) FRCP+W** **(b) FRA+CPA+W** **(c) CFRA+CPA+W**

**Figure 18: Coverage on TDE Dataset with Non-Exchangeability Assumption**



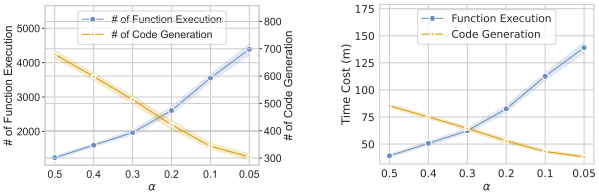

**(a) Coverage** **(b) Avg. Retrieval Percentage**

**Figure 19: Reliable Function Retrieval with Regression Models (with FRCP method, on TDE dataset)**




**(a) Coverage Vary $\alpha$** **(b) Retrieval Percentage Vary $\alpha$**

**Figure 20: Results with Sentence Transformers (with FPA+CPA method, on DS1K dataset)**




**(a) Numbers of Invocations** **(b) Execution Time**

**Figure 21: Function Execution and Generation Invocations on TDE with FRA+CPA**

Figure 21a, the left y-axis presents the number of function validation invocations, while the right y-axis presents the number of times we opt for code generation. Note that in practice, the cost of each code generation invocation can be significant, as it involves considerable time and effort to execute test cases and perform human verification. For example, considering the case of performing function validation in 10 test cases per function, on the TDE dataset (Figure 21b), as we increase the target coverage rate (i.e., decrease $\alpha$), the cost of function validation decreases, while the cost of code generation increases, and vice versa. Even when $\alpha = 0.05$, the function execution cost remains significantly lower than validating all functions, which takes approximately over 30 hours. Our proposed framework allows empirically selecting an appropriate $\alpha$ value, along with a specific abstention method, to best meet specific application needs.

*4.2.9 Summary.* Through the above experiments, we show that for a given miscoverage rate $\alpha$, all our proposed methods addressing different problems achieve coverage at or near the desired level, thereby ensuring reliable function retrieval. By adjusting $\alpha$, one can balance the cost between function execution and code generation. We also evaluated the effectiveness of two extensions in this setting.

## 5 Related Work

Self-service data transformation systems have been extensively studied over the past few decades, with the primary goal of creating user-friendly interfaces for performing data transformations [1, 13–16, 20–22, 38, 45, 46]. These capabilities have been integrated into widely used commercial tools, such as Excel [13] and Power BI [15]. However, traditional approaches rely on techniques such as code analysis [15], predefined sets of transformation operators [20, 45], or heuristic ranking algorithms [14, 16, 22], which can limit their flexibility and suffer from noisy inputs [10]. Recent research has begun exploring the use of machine learning to enhance these systems [9]. The emergence of large language models (LLMs) [44] has expanded the potential for various applications, including data transformation tasks [36] and joinability search [10], with promising results. However, LLMs are known to suffer from issues like hallucinations [19], leading to the development such as prompting techniques [17, 27, 28, 34, 42] and Retrieval Augmented Generation [11, 12, 18] to mitigate these issues. Such approaches remain best-effort methods, where the cost and accuracy depend implicitly on the underlying model and parameters, and no theoretical guarantees can be provided. In contrast, conformal prediction [4, 31, 35, 41], a statistical framework that provides coverage guarantees, is well-suited for uncertainty quantification and has been successfully applied to various machine learning tasks [4]. Distinct from prior work, this paper proposes to abstain from answering a proportion of questions by leveraging conformal prediction, ensuring reliable retrieval results.

## 6 Conclusion

In this paper, we address the problem of reliable function retrieval using large language models (LLMs) for data transformation tasks. By leveraging conformal prediction techniques, we explore three methods that provide reliable retrieval results while offering trade-offs between code validation costs and code generation. Our experimental results demonstrate the effectiveness of these methods and highlight the benefits of the proposed framework.

## Acknowledgment

This work was supported in part by NSERC Discovery and NSERC-CSE Research Communities grants.

## Artifacts

All the artifacts for this paper are available at https://github.com/dbllm/refred

## References


[1] Ziawasch Abedjan, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2016. Dataxformer: A robust transformation discovery system. In 2016 IEEE 32nd International Conference on Data Engineering (ICDE). IEEE, 1134–1145.

[2] Anastasios Angelopoulos, Stephen Bates, Jitendra Malik, and Michael I Jordan. 2020. Uncertainty sets for image classifiers using conformal prediction. arXiv preprint arXiv:2009.14193 (2020).

[3] Anastasios N Angelopoulos and Stephen Bates. 2021. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. arXiv preprint arXiv:2107.07511 (2021).

[4] Anastasios N Angelopoulos, Stephen Bates, et al. 2023. Conformal prediction: A gentle introduction. Foundations and Trends® in Machine Learning 16, 4 (2023), 494–591.

[5] Rina Foygel Barber, Emmanuel J Candes, Aaditya Ramdas, and Ryan J Tibshirani. 2023. Conformal prediction beyond exchangeability. The Annals of Statistics 51, 2 (2023), 816–845.

[6] Robin Beer, Alexander Feix, Tim Guttzeit, Tamara Muras, Vincent Müller, Maurice Rauscher, Florian Schäffler, and Welf Löwe. 2024. Examination of Code generated by Large Language Models. arXiv preprint arXiv:2408.16601 (2024).

[7] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In Proceedings of the 34th International Conference on Neural Information Processing Systems. 1877–1901.

[8] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. ACM Transactions on Intelligent Systems and Technology 15, 3 (2024), 1–45.

[9] Sibei Chen, Yeye He, Weiwei Cui, Ju Fan, Song Ge, Haidong Zhang, Dongmei Zhang, and Surajit Chaudhuri. 2024. Auto-Formula: Recommend Formulas in Spreadsheets using Contrastive Learning for Table Representations. Proceedings of the ACM on Management of Data 2, 3 (2024), 1–27.

[10] Arash Dargahi Nobari and Davood Rafiei. 2024. DTT: An Example-Driven Tabular Transformer for Joinability by Leveraging Large Language Models. Proceedings of the ACM on Management of Data 2, 1 (2024), 1–24.

[11] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining. 6491–6501.

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997 (2023).

[13] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. ACM Sigplan Notices 46, 1 (2011), 317–330.

[14] William R Harris and Sumit Gulwani. 2011. Spreadsheet table transformations from examples. ACM SIGPLAN Notices 46, 6 (2011), 317–328.

[15] 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), 1165–1177.

[16] Jeffrey Heer, Joseph M Hellerstein, and Sean Kandel. 2015. Predictive Interaction for Data Transformation.. In CIDR. Citeseer.

[17] Dong Huang, Jie M Zhang, Qingwen Bu, Xiaofei Xie, Junjie Chen, and Heming Cui. [n. d.]. Bias testing and mitigation in llm-based code generation. ACM Transactions on Software Engineering and Methodology ([n. d.]).

[18] Yizheng Huang and Jimmy Huang. 2024. A Survey on Retrieval-Augmented Text Generation for Large Language Models. arXiv preprint arXiv:2404.10981 (2024).

[19] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. Comput. Surveys 55, 12 (2023), 1–38.

[20] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. 2017. Foofah: Transforming data by example. In Proceedings of the 2017 ACM International Conference on Management of Data. 683–698.

[21] Zhongjun Jin, Michael Cafarella, HV Jagadish, Sean Kandel, Michael Minar, and Joseph M Hellerstein. 2018. CLX: Towards verifiable PBE data transformation. arXiv preprint arXiv:1803.00701 (2018).

[22] Zhongjun Jin, Yeye He, and Surajit Chauduri. 2020. Auto-transform: learning-to-transform by patterns. Proceedings of the VLDB Endowment 13, 12 (2020), 2368–2381.

[23] Shayan Kiyani, George J Pappas, and Hamed Hassani. 2024. Conformal Prediction with Learned Features. In Forty-first International Conference on Machine Learning.

[24] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In International Conference on Machine Learning. PMLR, 18319–18345.

[25] Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J Tibshirani, and Larry Wasserman. 2018. Distribution-free predictive inference for regression. J. Amer. Statist. Assoc. 113, 523 (2018), 1094–1111.

[26] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. Advances in Neural Information Processing Systems 36 (2024).

[27] Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2025. Structured chain-of-thought prompting for code generation. ACM Transactions on Software Engineering and Methodology 34, 2 (2025), 1–23.

[28] Jia Li, Yunfei Zhao, Yongmin Li, Ge Li, and Zhi Jin. 2024. Acecoder: An effective prompting technique specialized in code generation. ACM Transactions on Software Engineering and Methodology 33, 8 (2024), 1–26.

[29] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. Advances in Neural Information Processing Systems 36 (2024).

[30] Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. In International Conference on Machine Learning. PMLR, 26106–26128.

[31] Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. 2002. Inductive confidence machines for regression. In Machine learning: ECML 2002: 13th European conference on machine learning Helsinki, Finland, August 19–23, 2002 proceedings 13. Springer, 345–356.

[32] Yaniv Romano, Rina Foygel Barber, Chiara Sabatti, and Emmanuel Candès. 2020. With malice toward none: Assessing uncertainty via equalized coverage. Harvard Data Science Review 2, 2 (2020), 4.

[33] Yaniv Romano, Matteo Sesia, and Emmanuel Candes. 2020. Classification with valid and adaptive coverage. Advances in Neural Information Processing Systems 33 (2020), 3581–3591.

[34] Gabriel Ryan, Siddhartha Jain, Mingyue Shang, Shiqi Wang, Xiaofei Ma, Murali Krishna Ramanathan, and Baishakhi Ray. 2024. Code-aware prompting: A study of coverage-guided test generation in regression setting using llm. Proceedings of the ACM on Software Engineering 1, FSE (2024), 951–971.

[35] Glenn Shafer and Vladimir Vovk. 2008. A tutorial on conformal prediction. Journal of Machine Learning Research 9, 3 (2008).

[36] Ankita Sharma, Xuanmao Li, Hong Guan, Guoxin Sun, Liang Zhang, Lanjun Wang, Kesheng Wu, Lei Cao, Erkang Zhu, Alexander Sim, et al. 2023. Automatic data transformation using large language model-an experimental study on building energy data. In 2023 IEEE International Conference on Big Data (BigData). IEEE, 1824–1834.

[37] Uddip Acharjee Shuvo, Sajib Acharjee Dip, Nirvar Roy Vaskar, and ABM Alim Al Islam. 2024. Assessing ChatGPT's Code Generation Capabilities with Short vs Long Context Programming Problems. In Proceedings of the 11th International Conference on Networking, Systems, and Security. 32–40.

[38] Rishabh Singh. 2016. Blinkfill: Semi-supervised programming by example for syntactic string transformations. Proceedings of the VLDB Endowment 9, 10 (2016), 816–827.

[39] Chuyue Sun, Ying Sheng, Oded Padon, and Clark Barrett. 2024. Clover: Closed-Loop Ver ifiable Code Generation. In International Symposium on AI Verification. Springer, 134–155.

[40] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In Chi conference on human factors in computing systems extended abstracts. 1–7.

[41] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. 2005. Algorithmic learning in a random world. Vol. 29. Springer.

[42] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems 35 (2022), 24824–24837.

[43] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2024. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering. 1–12.

[44] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. arXiv preprint arXiv:2303.18223 (2023).

[45] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. Proceedings of the VLDB Endowment 10, 10 (2017), 1034–1045.

[46] Chaoji Zuo, Sepehr Assadi, and Dong Deng. 2022. Spine: Scaling up programming-by-negative-example for string filtering and transformation. In Proceedings of the 2022 International Conference on Management of Data. 521–530.