Flexim: A flexible similarity function for time series patterns

Zhuo Wang
Brown University
Providence, Rhode Island
zhuo_wang@brown.edu

ABSTRACT

Pattern based analysis tools are becoming an important part of time-series database systems. Algorithms used for this type of analysis usually require an underlying similarity function. The function quantifies the perceived visual closeness of any given pair of patterns. The widely used similarity functions are either mathematically defined or are based on hard assumptions. These functions attempt to approximate how all or most humans perceive similarity. However, these functions fail to generalize well to all tasks. In other words, the perception of similarity is often dependent on the task; hence, in reality, there is no one-size-fits-all approach to perceived similarity quantification. In this paper, we demonstrate Flexim which is a learned and therefore flexible similarity function that is designed to not only emulate any standard similarity measure, but also designed to learn task-specific notions of similarity. Flexim first presents a small number of simple questions through its user interface to the user. It then learns general augmentation rules based on user feedback. Using these rules Flexim generates a large dataset of pattern pairs labeled with their approximate similarity. Flexim then trains a deep model on the generated data to estimate the task-specific similarity function.

ACM Reference Format:
Zhuo Wang. 2023. **Flexim**: A flexible similarity function for time series patterns. In *Proceedings of ACM Conference (Conference'23)*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Time series data is collected from sources such as financial markets, weather monitoring, and biodata capturing devices. To convert the raw time series data into insights, predictions, or actionable steps, time series analysis is applied. Time series analysis includes pattern matching, clustering, classification, embedding, frequent pattern search, concept drift detection, anomaly detection, etc. Most algorithms in analyses, specially pattern matching, require a function for quantifying the similarity between a pair of patterns.

Euclidean Distance (*ED*) and Dynamic Time Warping (*DTW*) based similarity functions are the most popular methods [2]. A distance value $\mathfrak{d}$ can be converted into similarity $\mathfrak{s}$ as $\mathfrak{s} = 1 - \frac{\mathfrak{d}}{\mathfrak{D}}$ where $\mathfrak{D}$ is the maximum distance between any pair of patterns in the dataset. Accordingly, in this paper, we may use terms distance and similarity interchangeably.

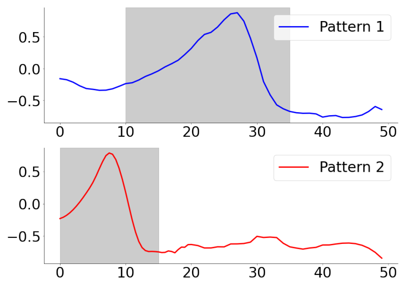


Figure 1: Task-specific perceived similarity

A similarity function can be thought of as a **quantification** method for **perceived similarity** between two patterns. Most of the related work focuses on improving the quantification assuming that there is a single way to perceive similarity that all humans agree upon with negligible variations. In this paper, we argue that a universal formulation for perceived similarity is not practical and the perception is primarily dependant on the task (e.g., stock analysis, vital signals, weather forecast, anomaly detection, etc). Below, we discuss further:

Similarity quantification: In short, the main challenge is to define an algorithm or a mathematical formula $\mathcal{F}$ that generates a single number $\mathfrak{s}$ that measures the perceived similarity between patterns a and b (e.g., $0 \leq \mathfrak{s} \leq 1$). If p denotes the statement "similarity function $\mathcal{F}$ generates a high score for a given pair a and b" and q denotes the statement "perceived similarity between a and b is high", for $\mathcal{F}$ to be a useful similarity function the following is the hardest to satisfy necessary condition: $\forall a \forall b : p \Leftrightarrow q$. Note that while p is a quantitative statement, q is qualitative. Using Euclidean Distance is a straightforward way to satisfy $p \Rightarrow q$. In other words, a pair with a very high pointwise similarity are expected to be perceptually similar. The issue lies in $q \Rightarrow p$: two pairs with a low pointwise similarity are not necessarily perceptually dissimilar to one another. For example, *ED* is very sensitive to shifting, stretching, and compressing which may result in a low similarity score. To alleviate this problem, *DTW* which is less sensitive to shifting and signal size is often used as an alternative.

Non-universality of perceived similarity: The perceived similarity between time series patterns is not only dependent on variations in human judgment but also it depends on the task. In fact, the latter introduces more variation than the former. As an example, consider the two patterns in Figure 1. Humans with no particular bias, medical doctors, and stock analysts may have their own unique ways of perceiving similarity. The members of the same group may also have variations within themselves but it is less significant specially if the members are experts on the same task. At a first glance, judging based on the point-wise similarity between those patterns, they look significantly different; an unbiased

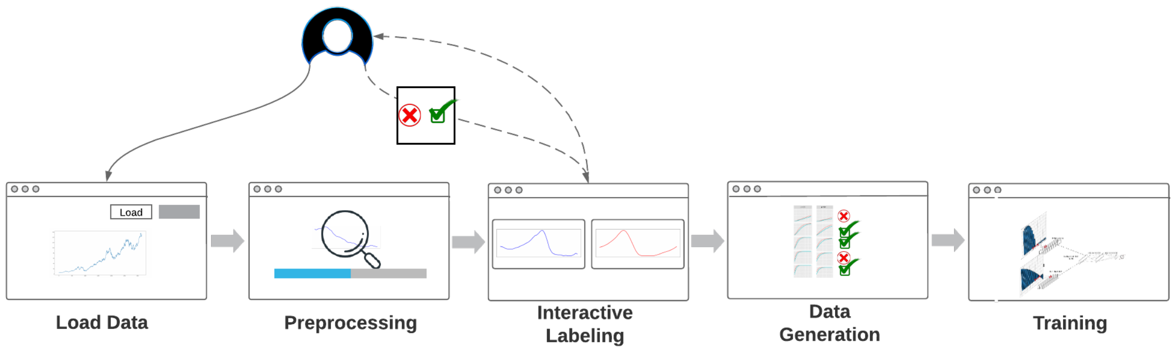


Figure 2: Flexim session workflow and user interactions

human may assume that by similarity we mean a strict matching. A medical doctor, on the other hand, may think that shifting does not matter too much, hence a method such as *DTW* might be a proper candidate. Besides being less sensitive to shifting and compression, when judging similarity, a stock analyst may put more weight on regions with higher amplitude variations (e.g., the areas highlighted in Figure 1). In the last scenario, *DTW* will not be able to accurately reflect the requirements of the task. This example shows that assuming the existence of a universal similarity perception does not seem realistic. Besides, neither *DTW* nor *ED* are sufficient when our aim is to quantify more nuanced and task-specific similarities.

A key observation in this project is that humans judge the similarity or difference between two patterns based on "how much" and "what types" of transformations are required to convert one to another. Examples of transformations are compression, rotation, addition of noise, and stretching. Each transformation has a relative importance in a given task. Based on that observation, Flexim using its internal transformation functions, provides the users with sample pairs of patterns some of which are transformed (i.e., distorted) versions of each other to first learn what transformations and to what degree are important in deciding the similarity. Flexim uses the relative importance of the transformations as augmentation rules to generate a synthetic training dataset that is used to train a final model that is able to quantify the perceived similarity between any input pair.

Steps that are taken by Flexim and the user to train and output a final similarity function are illustrated in Figure 2. The user first chooses a dataset. Flexim loads and preprocesses the input. This step includes dividing the input patterns into clusters. After that step, a new session starts which sequentially presents the user with pair of patterns and asks the user if the the two patterns are similar. After enough data is collected from the user, Flexim generates a large dataset of pattern pairs that are automatically labeled. Finally, a regression model is trained on the generated data. The predictions of the model are used as similarity (values are clipped if necessary). After the final similarity function is learned, Flexim allows the user to perform search queries using its interface to test the quality of the similarity function. The last step is not present in Figure 2.

The rest of the paper is organized as following: Section 2 discusses the related work, Section 3 provides a more detailed description of Flexim's architecture. We provide a demonstration scenario and some sample output generated by Flexim in Section 4. Finally, Section 5 discusses the limitations of the current project and possible future directions.

2 RELATED WORK

Dynamic Time Warping (DTW) [1] is one of the similarity measures that is algorithmically defined. The method is based on dynamic programming. *DTW* is extremely slow and that is one of the reasons that brute-force use of it is impractical on large datasets. Shape-based Disrance (SBD) [5] is another similarity measure based on normalized cross-correlation. If x and y are different time-series that have different lengths, the longer sequence is shifted to match the shorter. After matching, the series may have to be truncated or extended and padded with zeros if needed. SBD is sensitive to amplitude of its input signals. ShapeSearch [6] is a tool for visual exploration of trend lines which has its own similarity measure. Qetch [4] is another visual exploration tool that is resilient to distortions that may occur when humans draw the patterns manually.

3 SYSTEM ARCHITECTURE

Flexim consists of four main components that are demonstrated in Figure 3. The source code for the current prototype is available on Github [3]. Below, we explain the functions of each component, communications between them, and their sub-components.

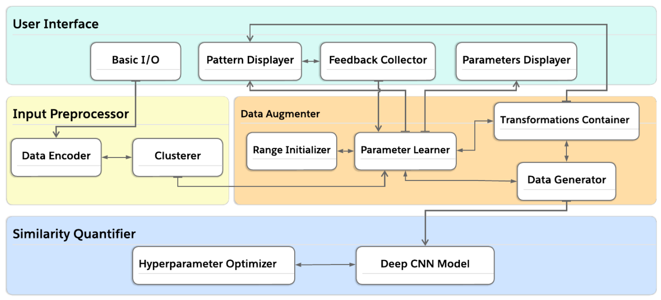


Figure 3: Flexim system architecture

3.1 User Interface

This component allows the user to interact with Flexim until a final similarity model is trained and returned (i.e., the steps shown in Figure 2). *Basic I/O* manages interface parts that are used to load, save, and take backups. Other parts include the following:

```python
# choose the transformation name to be displayed
# and the type of the transformation (global or local)
@register('scale', 'global')
def scale(x, alpha):
  y = 3 * alpha * x
  return y
```

Figure 4: A custom transformation function

3.1.1 Pattern Displayer. It displays the original data and highlights the parts of the data that are sampled to be transformed. In each round, it overlays the original pattern with its transformed version and presents them to the user.

3.1.2 Feedback Collector. The user decides if the two samples look similar or not. This sub-component collects all of these feedback each time a new question is answered.

3.1.3 Parameters Displayer. The transformed pattern is a distorted version of the original pattern using some augmentation parameters. The degree of distortion is displayed by this sub-component in the form of slider widgets. Each slider corresponds to a particular transformation function and the name of that function appears beside the slider.

3.2 Input Preprocessor

The input is first divided into small chunks of patterns of particular length (default is 50). Z-normalization is applied to convert the patterns into a standard form. After that, the following sub-components complete additional steps.

3.2.1 Data Encoder. has an auto-encoder at its core which is used to convert the patterns into a lower dimensional representations. These representations are used to capture the high level properties of the patterns.

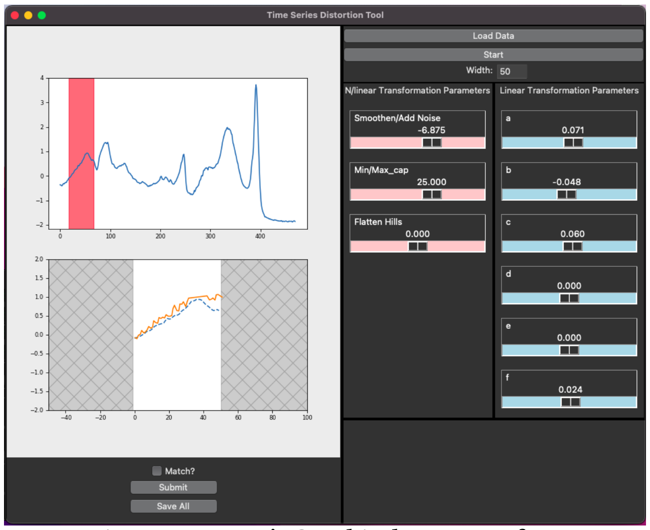
3.2.2 Clusterer. clusters the lower dimensional representations in order to assign a group code to each pattern. It uses the K-means clustering algorithm. The reason we divide the input into clusters is that there might be variations in augmentation rules for different types of shapes. As an example, straight lines might not too sensitive to stretching but patterns with hills might be. It also ensures that FLEXIM chooses a wide variety of pattern types when asking for feedback.

3.3 Data Augmenter

This component is responsible for learning augmentation rules and generating a dataset using these rules. The rules are derived from parameters of different transformation functions that are also defined within this component. After learning the rules, it generates a dataset of pattern pairs as training data and their corresponding approximate similarity as labels.

3.3.1 Transformations Container. stores a set of transformation functions. Theses functions accept a time series pattern as input and apply the transformation defined within the body of the function. The intensity of each transformation is decided based on another input argument $\alpha \in [-1, 1]$. This container is extendable meaning that if a particular task requires additional functions, they can be added to the container. The code fragment in Figure 4 is an example that shows how to add a new function that scales the input pattern by 3α. The decorator *@register* is used to indicate that this is a transformation function.

3.3.2 Range Initializer. performs a presearch to restricts the range of each parameter belonging to individual transformation functions. Initially, the parameters are in the range $[-1, 1]$. Using a binary search based algorithm, *Range Initializer* each time applies only one transformation type with varying degrees of intensity. This process gauges the maximum allowable distortion of each type and limits the range into $[l, r]$ where $-1 \leq l, r \leq 1$. Range initialization helps reduce the minimum number of feedback questions required by FLEXIM.



Width: 50

N/linear Transformation Parameters
Smoothen/Add Noise -6.875
Min/Max_cap 25.000
Flatten Hills 0.000

Linear Transformation Parameters
a 0.071
b -0.048
c 0.060
d 0.000
e 0.000
f 0.024

Figure 5: FLEXIM's Graphical User Interface

3.3.3 Parameter Learner. has an active learning model at its core. The model learns to predict the probability of a set of transformations applied on a pattern belonging to a particular cluster to result in a similar pattern. The *Range Initializer* helps *Parameter Learner* to reduce its exploration space. Each time that a new set of parameters need to be labeled, the *Parameter Learner* sends the set to *Pattern Displayer* to apply the transformations on the original pattern and display it to the user.

3.3.4 Data Generator. creates a large training dataset. Each row consists of two patterns and their corresponding similarity is used as label. The pattern pairs are of three categories: (1) *Naïvely Similar Pairs* which are pairs with almost identical pairwise similarity, (2) *Naïvely Not Similar Pairs* which are pair of patterns that are sampled from random parts of the data, and (3) *Complex Pairs* which consists of original patterns and their transformed form using parameter values sampled from the parameter space. Sampling of the parameter space is handled by *Parameter Learner.*

3.4 Active Learning Model

This component contains the Active Learning Model that is used to learn linear/non-linear transformation parameters and make predictions about whether specific linear/non-linear transformation parameters combination actually match user's perception based on limited label data. In order to reduce the number of limited label data and speed up the learning process, we use pre-train models as the basic structure and ensemble learning as the learning strategy and find that the experiment results shown in the following section prove that the modified active learning model's structure takes the effect.

3.4.1 Pre-train Model. The inspiration for loading pre-train model comes out that user may find the similarity function they wish to emulate really close to objective similarity function like euclidean distance, cosine similarity or dynamic time wrapping. Compared with training the similarity function from scratch, fine-tuning a machine learning model that loads a pre-train model is not only efficient but also effective.

3.4.2 Ensemble Learning. Traditional machine learning training approaches are limited to learning only a subset of the structure of train data, thereby limiting the predictive performance of machine learning models. Ensemble methods is a technique for machine learning that integrates multiple base models into one optimal predictive model. One main advantage of the ensemble learning approach is that we can combine multiple weaker models together instead of a single master model to perform machine learning tasks so that each model accurately captures some aspect of data structure.

Ensemble learning is also be classified as **sequential ensemble learning** and **parallel ensemble learning** based on training parallelism. The basic motivation of **sequential ensemble learning** is to exploit the dependence between base learners, which indicates that overall performance may be improved by assigning a greater weight to previously mislabeled examples. For **parallel ensemble learning method**, base learners are generated simultaneously without knowledge of each other, and the motivation of parallel methods is to exploit independence between the base learners since averaging can reduce error dramatically. In our case, we use a parallel ensemble learning approach to train the active learning model because we use different similarity function's pre-train model as base learners and generate output based on different base learners without letting each base learner influence others.

3.5 Similarity Quantifier

This component contains the final regression model that is trained on the data generated by *Data Generator*.

3.5.1 Deep CNN Model. Is a deep convolutional model that accepts two one dimensional patterns of fixed size as its input and predicts the similarity of the pair as its output. Intuitively, the model learns to detect what transformations and to what degree are applied on one of the patterns to convert it to the second pattern, and based on that, predicts the similarity score. Figure 6 shows how deep CNN model outputs similarity score

3.5.2 Hyperparameter Optimizer. uses Bayesian Optimization method finds the set of right hyperparameters such as drop-out rate, learning rate, and regularization parameter. Bayesian optimization is an effective search method for finding the best combination of hyperparameters from a range of possible values. Before each trial, an acquisition function predicts the next most promising candidate combination of hyperparameters for the algorithm to try on the next step based on past exploration. Together with the set of candidate values, the algorithm receives a budget that specifies the number of calls to some expensive function that we wish to optimize.

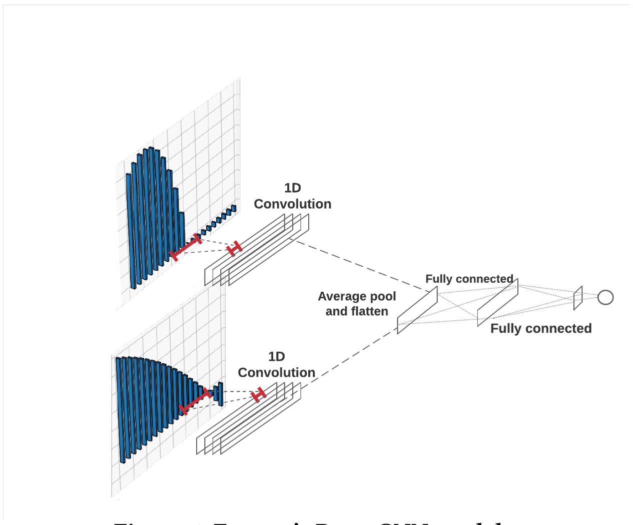


Figure 6: FLEXIM's Deep CNN model

4 DEMONSTRATION

Demonstration part presents the first prototype of FLEXIM [3] which allows the users to select from the following scenarios to explore the data. The dataset we will use for demonstration is historical daily closing prices of SP500 from January 1993 to January 2023.

4.1 User Interface Interaction

Users may choose to interact with FLEXIM to complete the initial three steps illustrated in Figure 2. It allows them to choose the source data and provide feedback to FLEXIM until it stops asking more questions. This scenario is designed to familiarize the user with the feedback mechanism and show the simplicity of the questions. Also, it is designed to demonstrate how quickly FLEXIM becomes confident enough to stop asking further questions.

Figure 5 illustrates a snapshot of FLEXIM's GUI after a sample dataset is loaded. After the user presses the *Start* button, FLEXIM randomly samples different parts of the data and applies various transformations on the patterns. The sampled part of the data is highlighted by red. The original pattern and its transformations are overlaid on top each other for an easier comparison. Blue dotted pattern is the original and the orange one is the transformed version.

On the right hand side, each slider corresponds to a separate transformation function. The red sliders are linked to non-linear and blue ones are linked to linear transformations. The sliders display the current level of transformation of each kind that is applied to acquire the orange pattern. Since the parameters are automatically calculated by the active learning component, the sliders are locked to be prevented from being altered by the users. If the user decides to define a custom transformation function, the GUI updates itself by adding a new slider for that new function.

4.2 Emulated *DTW* vs. *DTW*

FLEXIM is designed to adapt to the specific requirements of any task. Even though the assumptions made by algorithms such as *DTW* are not generalizable to all tasks, we can assume that some tasks would work well with *DTW*. In this scenario, we want to test if Flexim is able to emulate a standard similarity (distance) algorithm.

We have trained a model on a virtual user that uses *DTW* to determine the similarity between patterns. Just like a real user, the emulated user was asked the exact same number and type of questions (i.e., deciding if the distorted version of a pattern is similar to itself). If the similarity generated by *DTW* exceeds a threshold, then the emulated user labels the pair as *similar*and as *not similar*, otherwise. We run the whole pipeline on these emulated feedback data in advance to generate and train a final model.

To test how accurate our emulation is, we use earth mover's distance to evaluate Flexim 's emulation effect. Earth mover's distance is a measure of the distance between two probability distributions. The closer the values, the better indication that two probability distributions are similar. In this scenario, earth mover's distance is a better indicator that shows how effective Flexim is able to emulate other objective similarity functions such as Dynamic Time Wraping.

Figure 7 illustrates the emulation result of training Flexim 's active learning model with/without loading pre-training model based on earth mover's distance metrics. Based on experiment result, we can draw conclusion that within same number of query time limit, Flexim which uses other objective similarity measure's pre-train model as part of its active learning model structure, can emulate other similar objective similarity measure more effectively than train whole model from scratch.

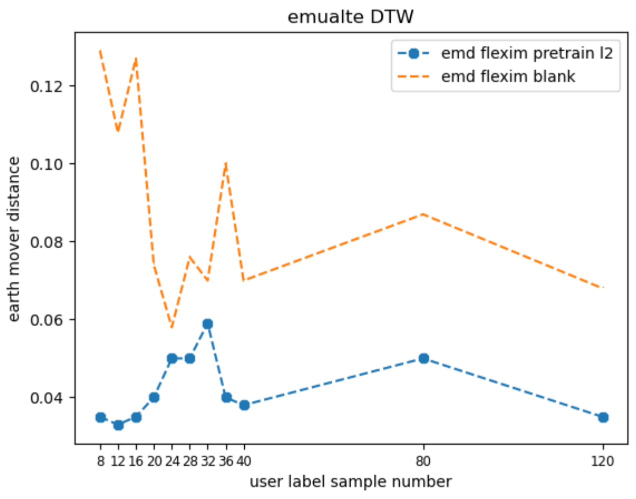


Figure 7: emulate Dynamic Time Wraping similarity function

Apart from using pre-training model, ensemble learning also plays important role in process of training learnable similarity function. Figure 8 and Figure 9 shows the experiment result of training Flexim 's active learning model with/without using ensemble learning based on accuracy and earth mover distance metrics. We can observe from the experiment result that within the same number of query time limits, Flexim 's ensemble learning-trained model achieves significantly higher accuracy level than using other training strategies.

Based on the experiment result, we draw the conclusion that Flexim 's active learning model, trained with ensemble learning strategy, can emulate objective similarity measures more effectively. In addition, we also implement additional relevant experiments, such as using the earth mover's distance to evaluate the emulation effect which is shown in Figure 10. These experiment results demonstrate that active learning model with stacked structure enables Flexim to emulate objective similarity functions more effectively than using other model structures.

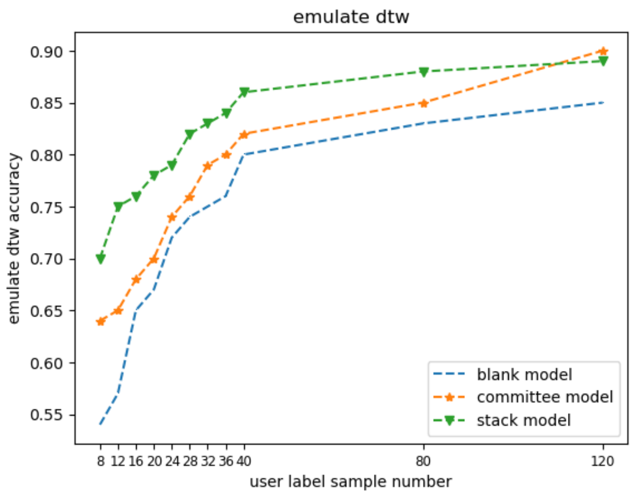


Figure 8: emulate Dynamic Time Wraping similarity function (2)

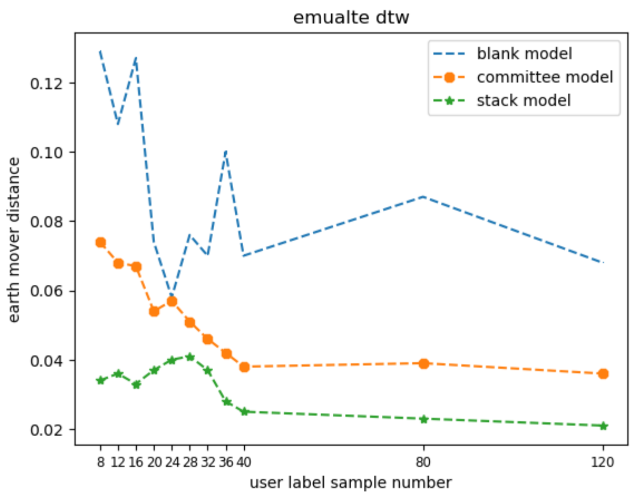


Figure 9: emulate Dynamic Time Warping similarity function (3)

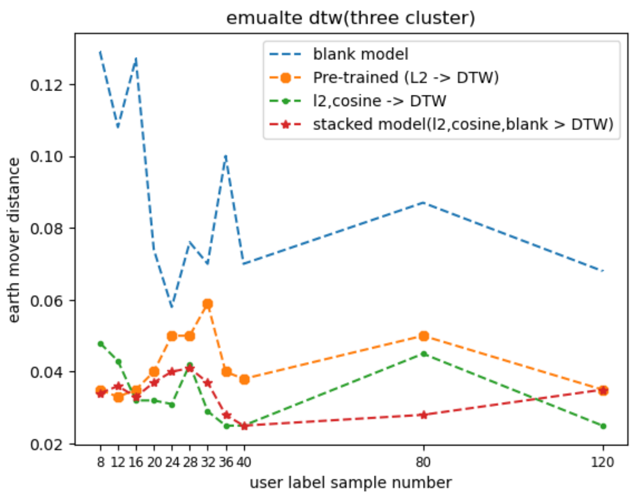


Figure 10: emulate Dynamic Time Wraping similarity function (3)

4.3 Search Results Comparison

4.3.1 query pattern with simple shape. This scenario enables users to digitally draw a pattern to be searched using different similarity functions. We perform a simple linear scan to avoid errors that might be introduced by faster search methods. We will define several tasks with various requirements and ask the users which method's results are most comparable with regard to these specifications. As an example, assuming that our task is not sensitive to smoothening, consider Figure 11 that illustrates top results returned by *Cosine*, *DTW*, and FLEXIM. The first two methods are too rigidly defined to adapt to the task and therefore fail to find the most similar pattern according to the defined rule, whereas the third method, FLEXIM , returns a pattern that seems like the closest match among the three.

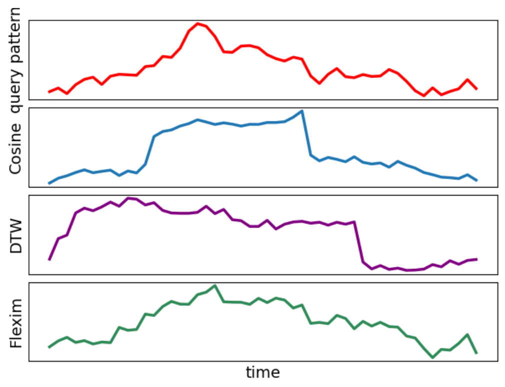


Figure 11: Top results using Cosine, DTW and Flexim

In order to demonstrate FLEXIM's searching performance on various time-series query patterns, we also conduct several experiments and present search results below.

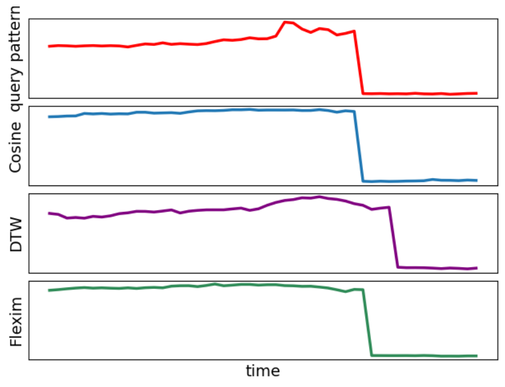


Figure 12: Top results using Cosine, DTW and Flexim (2)

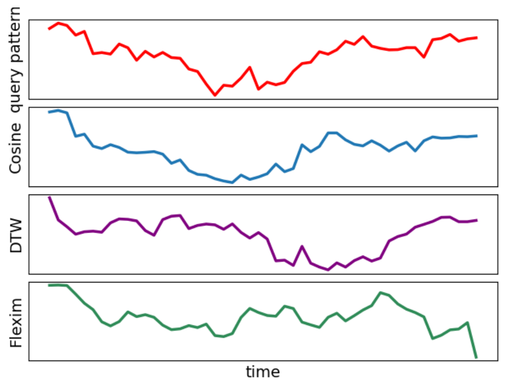


Figure 13: Top results using Cosine,DTW and Flexim (3)

Figure 12 and Figure 13 demonstrate that for other basic shapes of query patterns, Flexim not only achieves high-quality search results comparable to cosine and Dynamic Time Warping, but also captures query pattern's features that outperform other search results based on *cosine* similarity and *Dynamic Time Wraping* similarity.

4.3.2 query pattern with complex shape. However, for some complex query pattern's shape, FLEXIM 's search results are not ideal compared with using *Cosine* similarity function and *DTW* similarity function, as depicted in Figure 14 and Figure 15. The main reason of this problem is that linear and non-linear transformation functions learned by FLEXIM are not comprehensive enough to capture shape features and characteristics of the query pattern and return reasonable search results. To address this issue, FLEXIM provides users with APIs that enable them to add their self-defined linear or non-linear transformation function to improve search performance based on users' similarity perceptions.

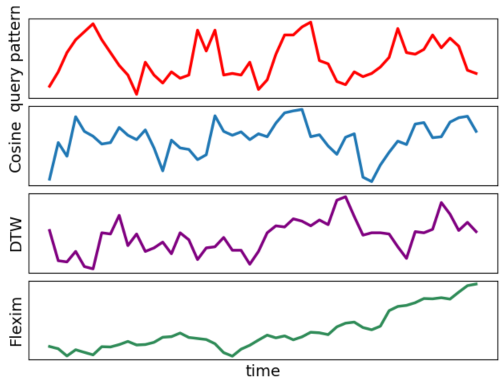


Figure 14: Top results using Cosine, DTW and Flexim (4)

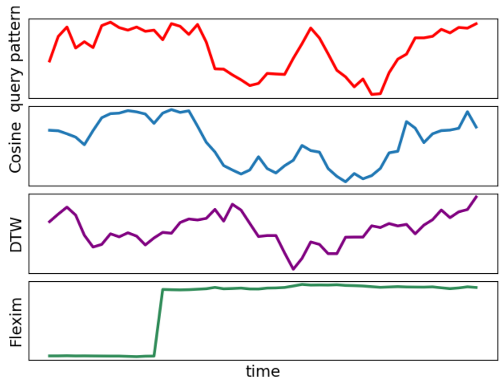


Figure 15: Top results using Cosine, DTW and Flexim (5)

4.3.3 relationship between the search result and training sample. Whether the search results' quality can be improved is also one of the problems we are concerned about. We design an experiment to evaluate search results for the same query pattern using different training samples. Based on search results shown in Figure 16 and Figure 17, we can find that Flexim 's search result's quality improves as the number of training samples increases. One of the project's extensions or concerned points remained in this project is how to improve search result quality with fewer training samples.

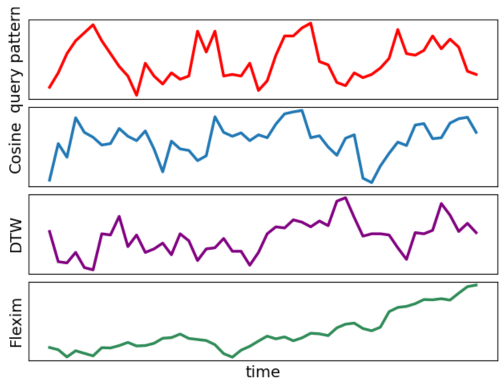


Figure 16: Top results using Cosine, DTW and Flexim (6) - 60 training sample

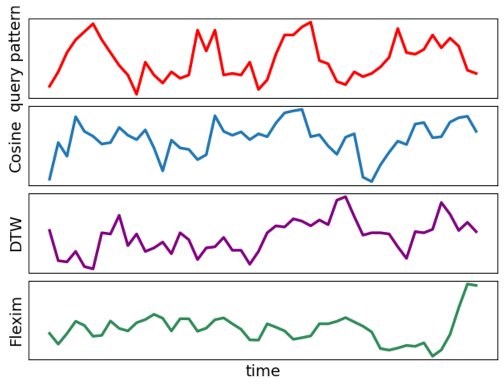


Figure 17: Top results using Cosine, DTW and Flexim (6) - 300 training sample

5 LIMITATIONS AND FUTURE WORK

In this project, the problem of quantifying time series similarity is reformulated as the problem of finding how much distortion (or transformation) is required to convert one pattern to another. This definition allows more flexibility to adapt to task-specific requirements. Our initial results show that Flexim is not only capable of emulating popular similarity measures but also quantifying non-standard notions of perceived similarity. However, we need to implement large-scale user studies to support these initial results more strongly. Another limitation is that Flexim accepts only simple forms of feedback (i.e., binary). We are working on developing more sophisticated and intuitive feedback mechanisms such as ranking different patterns. Moreover, Flexim current workflow's efficiency needs to be improved since the user still needs to provide a certain amount of label data to train Flexim 's machine learning model in order to achieve considerable performance. We are also working on improving the efficiency of the machine learning model's training pipeline so that model can achieve the satisfactory result without requiring lots of user-label data.

A future research direction is to develop a full search tool that uses Flexim at its core. This tool will require techniques to speed up the search such as indexing and embedding. Vector database indexes and stores vector embeddings for fast retrieval and similarity search, how to design suitable index structure based on vector embeddings and learnable similarity functions to speed up vector search and retrieval is also a potential research direction of database research area.

REFERENCES

[1] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series.. In *KDD workshop*, Vol. 10. Seattle, WA, USA:, 359–370.
[2] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1542–1552.
[3] Amir Ilkhechi and Zhuo Wang. 2022. Flexim Repository. https://github.com/chelsea97/Flexim_Final_Version
[4] Miro Mannino and Azza Abouzied. 2018. Expressive time series querying with hand-drawn scale-free sketches. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.
[5] John Paparrizos and Luis Gravano. 2015. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1855–1870.

[6] Tarique Siddiqui, Paul Luh, Zesheng Wang, Karrie Karahalios, and Aditya Parameswaran. 2020. Shapesearch: A flexible and efficient system for shape-based exploration of trendlines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 51–65.