A ADDITIONAL MATERIAL

A.1 Definition of Experimental Related Issues

A.1.1 Definition of Experimental Queries. To apply in-database K-Shape on a time series root.test.d0.s0, we apply the following SQL statement.

```
select lsmKShape(s0)
from root.test.d0
where time >= 2020-02-01T00:00:00
and time < 2020-03-01T00:00:00
```

Similarly, to apply in-database Medoid-Shape on this time series, we apply the following SQL statement.

```
select lsmMShape(s0)
from root.test.d0
where time >= 2020-02-01T00:00:00
and time < 2020-03-01T00:00:00
```

Both statements calculate k centroids with respect to all the subsequences with length l in the time range of [2020-02-01T00:00:00, 2020-03-01T00:00:00) in time series root.test.d0.s0. Note that subsequence length l and cluster number k are set in database configuration, since pre-computed metadata depend on these parameters.

A.1.2 Definition of Clustering Quality Measures. As shown in Figure 8 in D.3 of Appendix, we utilize Compactness [16] for two unlabeled private datasets as effectiveness metrics, defined by

$$\text{Compactness} = \frac{1}{n}\sum_{i=1}^{n} \min_{C_j} \text{SBD}(X_i, C_j).$$

For two labeled public datasets, we choose Rand Index [27] to measure clustering quality, defined by

$$R = \frac{TP + TN}{TP + TN + FP + FN},$$

where TP denotes the number of time series belonging to the same class and assigned to the same cluster, TN denotes the number of time series belonging to different classes but assigned to different clusters, FP denotes the number of time series belonging to different classes but assigned to the same cluster, and FN denotes the number of time series belonging to the same class but assigned to different clusters. Note that lower Compactness or higher Rand Index implies a more accurate clustering result.

A.2 Extended Effectiveness and Efficiency Evaluation

In addition to the existing 4 baselines in the submission, we further compare 2 SOTA effective baselines, called Partition Around Medoids with Move Split Merge (PAM+MSM) and Partition Around Medoids with Dynamic Time Warping (PAM+DTW). Referring to previous works [m1], PAM+MSM outperforms other elastic time series measures and clustering prototypes. As a classic and effective time series measure, DTW measure is widely applied in a lot of existing works on time series clustering [m2, m3]. Therefore, we also take DTW into account and evaluate its effectiveness on time series clustering.

We conduct evaluation on effectiveness and efficiency with varying data sizes in Figures 13 and 14, respectively. We find that PAM+MSM and PAM+DTW can indeed achieve slightly better clustering quality in Figure 13. However, they both take significantly more time to execute, as shown in Figure 14. Given more than 2000 subsequences, they need at least 1000 seconds to perform. Therefore, we only evaluate 3 data points for PAM+MSM and PAM+DTW in Figures 13 and 14, given limited time. This is because their time complexities are both $O(kn^2l^2)$, and thus, they cannot handle huge volume of data. Nevertheless, our in-database K-Shape and in-database Medoid-Shape can achieve at least 3 orders of magnitude improvements compared to PAM-MSM and PAM-DTW, while still obtain comparable clustering quality.

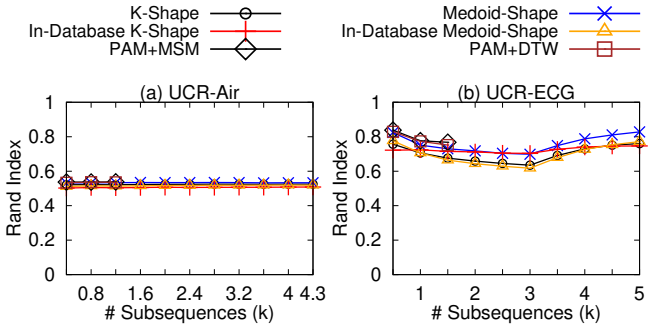

Figure 13: Effectiveness under different data sizes

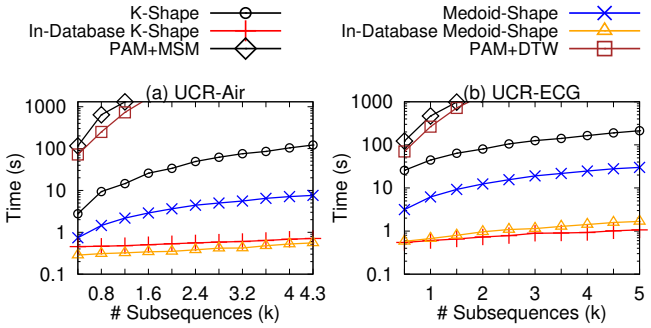

Figure 14: Time costs under different subsequence numbers

[m1] Christopher Holder, David Guijo-Rubio, Anthony J. Bagnall: Clustering Time Series with k-Medoids Based Algorithms. AALTD@ECML/PKDD 2023: 39-55

[m2] Hussein El Amouri, Thomas Andrew Lampert, Pierre Gançarski, Clément Mallet: Constrained DTW preserving shapelets for explainable time-series clustering. Pattern Recognit. 143: 109804 (2023)

[m3] John Paparrizos, Sai Prasanna Teja Reddy: Odyssey: An Engine Enabling The Time-Series Clustering Journey. Proc. VLDB Endow. 16(12): 4066-4069 (2023)

A.3 Evaluation on Aim Function Error

To prove that Medoid-Shape can obtain relatively low aim function errors as in Proposition 7, we evaluate the relative errors between optimal aim function scores and Medoid-Shape aim function scores. That is, relative errors are defined as $e = (f^* - \tilde{f})/f^*$, where f^* denotes the optimal aim function score and $\tilde{f}$ denotes the aim

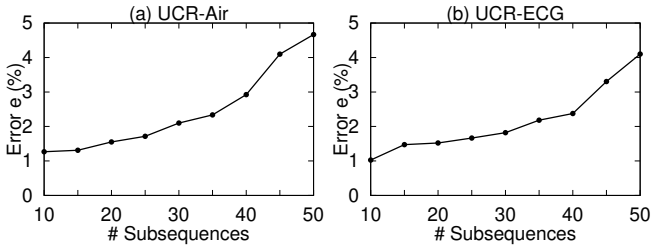

Figure 15: Aim function errors under different data sizes

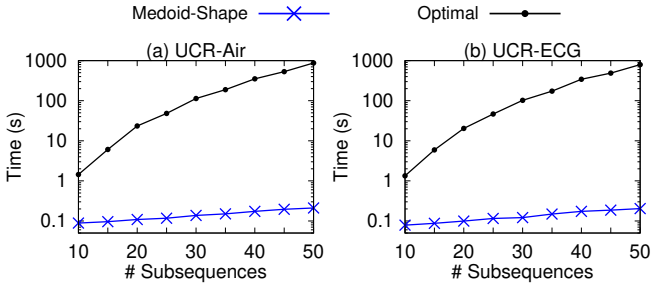

Figure 16: Time costs of Medoid-Shape and optimal solver (corresponding to Figure 15)

function score returned by Medoid-Shape. To be specific,

$$f^* = \max_{\mathbb{C}\subseteq\mathbb{T}_l} \sum_{X_i\in\mathbb{T}_l} \max_{C\in\mathbb{C}} g(X_i, C),$$

$$\tilde{f} = \sum_i w_i \max_{C_h} g(U_i, C_h),$$

where w_i and U_i denote the weights and centroids of approximate clustering, C_h denotes the centroids returned by Medoid-Shape.

Figure 15 reports the aim function errors between Medoid-Shape aim function scores and optimal aim function scores. With more subsequences, the aim function error increases, since more subsequences are approximated. Nevertheless, the errors are relatively small.

Figure 16 reports the corresponding execution time of Medoid-Shape and optimal solver. Note that optimal aim function score can only be solved by brute-forcedly searching all possible combinations of medoids, extremely time-consuming as shown in Figure 16. Therefore, these experiments are only conducted on a small amount of data. The brute-forced optimal solver obviously takes significantly consuming time.

A.4 Evaluation on Disk Space Overheads

Figure 17 reports the disk space overheads of different numbers of subsequences. The disk space overheads include the space overheads of compressed time series data points and pre-computed data. With more subsequences on disk, the according space overheads of all methods consistently increase with no doubt. For out-of-database methods, i.e., K-Shape and Medoid-Shape, they do not need any pre-computation and their space overheads are both equal to the compressed sizes of time series data points. In-database K-Shape

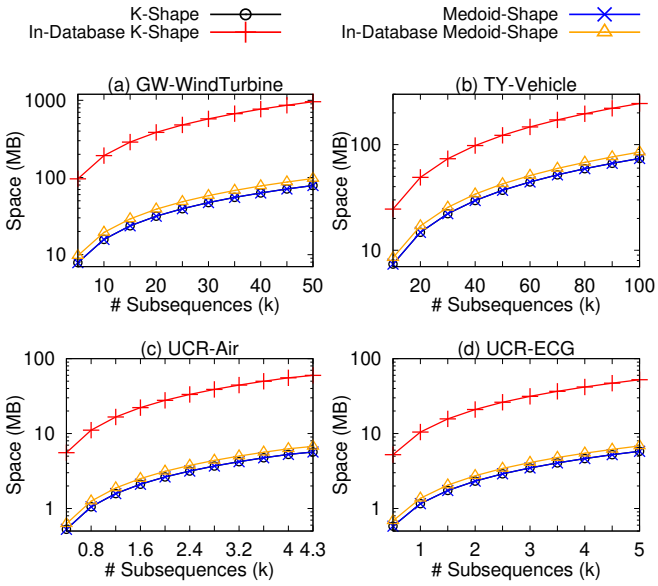

Figure 17: Space costs under different subsequence numbers

takes much more extra space overheads than others, since it need to store the sum matrices as pre-computed metadata. In-database Medoid-Shape takes slightly more extra space overheads than out-of-database methods, since it only needs to store some approximate centroids.

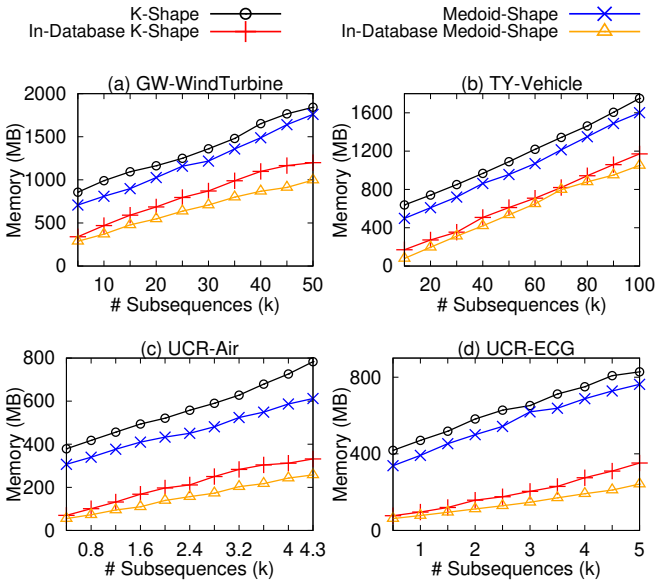

Figure 18: Peak Memory under different subsequence numbers

Methods	Flush Time Complexity	Query Time Complexity	Space Complexity
K-Shape	–	$O(\max\{nkl\log l, nl^2, kl^3\} \times iter)$	$O(nl + kl^2)$
Medoid-Shape	–	$O(W + k^2srl\log l)$	$O(sl + kl)$
in-database K-Shape	$O(\max\{kl\log l, l^2, kl^3\}N \times iter)$	$O(k(N-M)l^2 + k(M+1)l^3)$	$O(kNl^2)$
in-database Medoid-Shape	$O(NW)$	$O(r(N-M)l + rM\ell + k^2srl\log l)$	$O(kNl)$

Table 3: Time and space complexity of baselines in end-to-end process

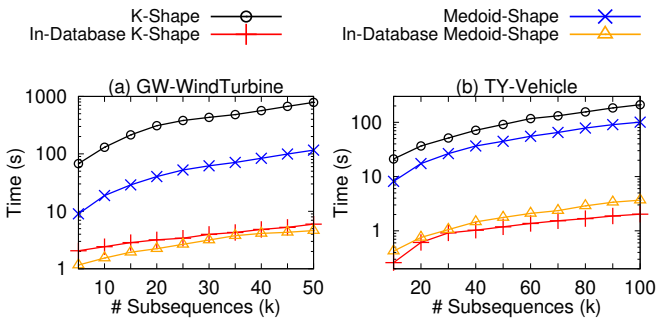

Figure 19: Time costs on large datasets with sufficient memory

A.5 Evaluation on Query Peak Memory

Figure 18 reports the query peak memory of different numbers of subsequences. We assign sufficient memory budget for the database to execute queries. As the number of subsequences increases, all methods cost higher memory. However, in-database methods cost less memory than out-of-database methods under all subsequence numbers. Because in-database methods do not need to load all relevant data points, and thus cost less memory.

A.6 Extended Evaluation with Sufficient Memory Budget

We remove the limited memory budget and evaluate the out-of-memory data points in the submission. Figure 19 shows the complete evaluated time costs. As shown, given large number of subsequences, K-Shape is extremely time-consuming. However, our in-database adaptions are much more efficient.

A.7 Complexity Analysis

We list the time complexity and space complexity of K-Shape, Medoid-Shape and in-database K-Shape and in-database Medoid-Shape in Table 3, where n, l, k denote subsequence number, subsequence length, cluster number, respectively. And W, s, r denote approximate clustering time sample subsequence numbers and approximate cluster number, respectively. Last, N, M, ℓ denote the page number, overlapped page number and average overlap length, respectively.

A.8 Real-World Deployed Case

We plan to deploy our in-database clustering methods in a steel processing factory. There are around 10,000 forging machines in the factory. Each machine records a time series of 22 features during the working process, including operating current, forging temperature and so on. We have recently collected a dataset of 6 months from one forging machine with a sample rate of 1 data points per second. The dataset contains around 20 million data points each with 22 features, costing around 10 GB on disk. The forging machine processed 5 different types of steels in these 6 months, each with 2-4 working patterns.

Analysts need to cluster subsequences of different time periods into at least 4 clusters, to find the frequent working patterns of this machine on 5 different types of steels. One straightforward way is to transfer the data into clouds with high computing power, and then apply out-of-database clustering. However, such straightforward way is concerning due to the following 2 reasons.

(1) *Data movements with extraction, transformation and loading (ETL) indeed incur huge costs [m4].* Consider 10,000 machines in the factory, the overall size of 1-year data is around 0.2 PB. With a transmission speed 500 MB per second from edges to clouds, it takes around 5 days for ETL. It is infeasible to spend 5 days moving data, just for subsequence clustering.

(2) *Thanks to database facilities, such as metadata, in-database clustering is faster than out-of-database clustering on files of raw data.* Though increasing data transfer speeds is one feasible way, it will incur more costing on compression and calculation from scratch. For example, with proposed optimization, in-database K-Shape costs 0.71 s, while transferring outside and clustering from scratch costs 103.10 s.

(3) *Applications requiring clustering of many time series, such as anomalous subsequence detection, require high timeliness.* As stated in Section 1, the results of subsequence clustering can further serve anomalous subsequence detection [m5]. Such detection needs to timely alert the abnormal event. steel bars may be stuck when processing, leading to current subsequences deviating from clustering centroids. Obviously, moving data out of databases and then detecting cannot meet the demands on timeliness. However, in-database clustering can further help anomaly detection. A simple idea is to cluster subsequences of the past hour and utilize the clustering results for detecting anomalies in the next hour.

[m4] Panos Vassiliadis: A Survey of Extract-Transform-Load Technology. Int. J. Data Warehous. Min. 5(3): 1-27 (2009)

[m5] Hardy Kremer, Stephan Günnemann, Thomas Seidl: Detecting Climate Change in Multivariate Time Series Data by Novel Clustering and Cluster Tracing Techniques. ICDM Workshops 2010: 96-97

A.9 Clarification on Challenges and Contributions

Our challenges and according solutions are summarized as follows. We will further update these statements in Section 1 of revision.

Challenge (1). The out-of-order storage encumbers frequently loading data outside databases and applying existing out-of-database clustering methods.

Contribution (1). Therefore, we propose in-database K-Shape and in-database Medoid-Shape for LSM-Tree based time series databases to avoid loading data out of databases.

Empirical Evidence (1). Figures 11 and 12 of Appendix D.4 in current submission prove that in-database K-Shape and in-database Medoid-Shape perform more efficient than K-Shape given different degrees of out-of-order issues.

Challenge (2). The data size of time series may be so large that the memory cannot hold the whole time series.

Contribution (2). Therefore, we propose in-database K-Shape and in-database Medoid-Shape. In-database adaptions only need to load pre-computed metadata into memory, instead of loading all data points into memory.

Empirical Evidence (2). Figure 18 in Supplementary A.5 shows that in-database methods cost much less memory than out-of-database methods during querying, since they do not load all data points.

Challenge (3). Clustering may be repeatedly conducted with various time filters for different tasks. Existing methods need to cluster from scratch each time, and thus suffer from low efficiency. Therefore, we propose in-database K-Shape with pre-computation to support frequent queries, and propose Medoid-Shape and in-database Medoid-Shape for further acceleration.

Contribution (3). Therefore, we propose in-database K-Shape with pre-computation to efficiently support frequent queries. We further propose Medoid-Shape and in-database Medoid-Shape to accelerate queries under long subsequences.

Empirical Evidence (3). Figure 4 of Section 6.2 and Figure 5 of Section 6.3 in current submission demonstrate the high efficiency of Medoid-Shape and two in-database adaptions. Given long subsequences in Figure 5, Medoid-Shape performs much more efficient than K-Shape.