Supplementary Appendix: Anomaly Resilient Temporal QoS Prediction using Hypergraph Convoluted Transformer Network

Suraj Kumar, Soumi Chattopadhyay, Chandranath Adak

Appendix A
GreySheep Discrepancy Index (GDI)

As outlined in Section III-C1 of the main paper, the primary goal of introducing GDI is to determine how a user or service may exhibit unique characteristics that deviate from the general norms of other users and services. The GDI for a user u_i (denoted as $\mathbb{G}^t(u_i)$) or a service s_j (denoted as $\mathbb{G}^t(s_j)$) at time-step t is a scalar value calculated by comparing the QoS invocation profile of u_i (i.e., $\mathcal{Q}^t(i,.)$) or s_j (i.e., $\mathcal{Q}^t(.,j)$) with the overall QoS mean for u_i (denoted as $\mu^t(u_i) = mean(\mathcal{Q}^t(i,.))$) or s_j (denoted as $\mu^t(s_j) = mean(\mathcal{Q}^t(.,j))$). This comparison incorporates individual means for each service when u_i interacts with a service, or for each user when s_j is invoked by a user. The mathematical definitions of $\mathbb{G}^t(u_i)$ and $\mathbb{G}^t(s_j)$ are provided in Eq.s 1 and 2, respectively.

$$\mathbb{G}^t(u_i) = \frac{\sum_{s_j \in \mathcal{S}_i^t} \Big(|\mathcal{Q}^t(i,j) - \mu^t(u_i) - \bar{\mu}^t(s_j)| \times \hat{\mathcal{N}}^t(s_j) \Big)}{|\mathcal{S}_i^t|} \tag{1}$$

$$\mathbb{G}^t(s_j) = \frac{\sum_{s_j \in \mathcal{U}_j^t} \Big(|\mathcal{Q}^t(i,j) - \mu^t(s_j) - \bar{\mu}^t(u_i)| \times \hat{\mathcal{N}}^t(u_i) \Big)}{|\mathcal{U}_j^t|} \tag{2}$$

where, $\mathcal{S}_i^t$ represents the set of services invoked by u_i at time-step t, and $\mathcal{U}_j^t$ represents the set of users that invoked service s_j at t. The mean-centered QoS values for u_i and s_j, denoted by $\bar{\mu}^t(s_j)$ and $\bar{\mu}^t(u_i)$ respectively, are defined as follows:

$$\bar{\mu}^t(s_j) = \frac{\sum_{u_i \in \mathcal{U}_j^t} \left(\mathcal{Q}^t(i,j) - \max_{u_i \in \mathcal{U}_j^t} \mathcal{Q}^t(i,j) - \min_{u_i \in \mathcal{U}_j^t} \mathcal{Q}^t(i,j) \right)}{|\mathcal{U}_j^t| - 2} \tag{3}$$

$$\bar{\mu}^t(u_i) = \frac{\sum_{s_j \in \mathcal{S}_i^t} \left(\mathcal{Q}^t(i,j) - \max_{s_j \in \mathcal{S}_i^t} \mathcal{Q}^t(i,j) - \min_{s_j \in \mathcal{S}_i^t} \mathcal{Q}^t(i,j) \right)}{|\mathcal{S}_i^t| - 2} \tag{4}$$

Additionally, $\hat{\mathcal{N}}^t(u_i)$ and $\hat{\mathcal{N}}^t(s_j)$ represent the normalized standard deviation for u_i and s_j at time-step t, as shown in Eq.s 5 and 6, respectively.

$$\hat{\mathcal{N}}^t(u_i) = 1 - \left(\frac{\sigma^t(u_i) - \min_{u_k \in \mathcal{U}} \sigma^t(u_k)}{\max_{u_k \in \mathcal{U}} \sigma^t(u_k) - \min_{u_k \in \mathcal{U}} \sigma^t(u_k)} \right) \tag{5}$$

$$\hat{\mathcal{N}}^t(s_j) = 1 - \left(\frac{\sigma^t(s_j) - \min_{s_k \in \mathcal{S}} \sigma^t(s_k)}{\max_{s_k \in \mathcal{S}} \sigma^t(s_k) - \min_{s_k \in \mathcal{S}} \sigma^t(s_k)} \right) \tag{6}$$

where, $\sigma^t(u_i) = sd(\mathcal{Q}^t(i,.))$ and $\sigma^t(s_j) = sd(\mathcal{Q}^t(.,j))$ represent the standard deviation of the QoS vectors corresponding to u_i and s_j at time-step t, respectively.

A high value of $\hat{\mathcal{N}}^t(u_i)$ or $\hat{\mathcal{N}}^t(s_j)$ generally indicates a consistent QoS invocation pattern for u_i or s_j at time-step t. Therefore, when calculating the GDI of a user u_i in relation to its individual service invocation s_j, we use $\hat{\mathcal{N}}^t(s_j)$ as a weight. This weight reflects the consistency of s_j and helps to determine whether an abnormal value is attributable to u_i or s_j. Similarly, the GDI for s_j is calculated. A high GDI value for u_i or s_j indicates a significant abnormality. Based on these GDI values, we labeled the greysheep users and services in the main paper.

Appendix B
Datasets

In this section, we provide more information about the datasets utilized in our experiments. We leverage two large-scale QoS datasets from the publicly available WSDREAM-2 [1] repository, namely, Response Time (RT) and Throughput (TP). These datasets capture QoS interactions for 142 users across 4500 web services over 64 time-steps, with a 15-minute interval between each time-step. These datasets are commonly used for time-aware QoS prediction [2]–[7]. We now discuss the characteristics of the datasets.

(i) Dataset Challenges: In real-life scenarios, a user invokes very few services, eventually leading to situations where a large percentage of QoS data is unknown at a particular time slice, which causes high sparsity in the QoS log matrices. The sparsity is further increased when new users or services, often called cold-start, are added to the QoS ecosystem. The unknown QoS values and cold-start turn out to be data deficiency issues. The datasets may further have the issue of data credibility, where the data itself is unreliable due to the presence of outliers and greysheep (GS). The actual cause of generating outliers, however, is unknown, but it is led by network loads and service status. Moreover, the greysheep is known for its pattern of QoS invocation, which differentiates it from other users or services. Here, we exemplify the concept of GS using a subset of the RT dataset [1]. Fig. B.1 depicts an instance of a GS user compared to three other non-GS (NGS) users across twelve services based on their response time values. Notably, the QoS pattern of GS1 significantly diverges from NGS1, NGS2, and NGS3. The presence of such users in the QoS logs complicates the learning process and leads to low QoS prediction performance. Therefore, handling such issues is essential for high-prediction accuracy.

(ii) Dataset Distribution: As reported in Section IV of the main paper, the value ranges in the RT and TP datasets are $[0.001, 19.999]$ and $[3.6534\times10^{-5}, 6726.8335]$, respectively. Table B.1 provides further details on the distribution of both datasets. Notably, both datasets are positively skewed, with

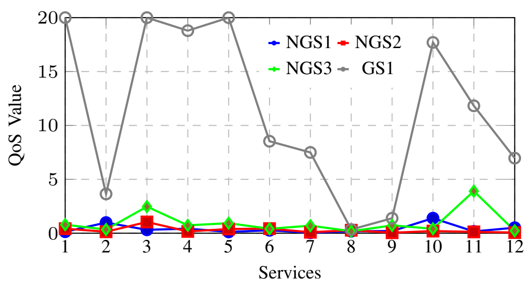


Fig. B.1: Example of a greysheep user

TABLE B.1: Range-wise analysis of QoS datasets

RT (seconds)	Range	(0.001, 0.5)	(0.5, 1)	(1, 2)	(2, 3)	(3, 5)	(5, 10)	(10, 15)	(15, 19.999)
	Percentage	56.44%	15.55%	8.89%	3.04%	3.51%	4.02%	1.80%	6.75%
TP (kbps)	Range	(3.6534e-5, 25)	(25, 50)	(50, 100)	(100, 250)	(250, 500)	(500, 1000)	(1000, 2500)	(2500, 6726.8335)
	Percentage	90.66%	3.81%	2.57%	1.97%	0.69%	0.26%	0.0378%	0.0008%

most QoS values concentrated on the left side of the distribution. This asymmetry complicates the interpretation of complex features, necessitating non-linear models to capture the underlying patterns, which is the focus of our paper.

Appendix C
Configuration of Hyperparameters

Table C.1 details the hyperparameters used for experimental analysis in the main manuscript. We aim to optimize model performance across various configurations by systematically adjusting these hyperparameters through an exhaustive grid search over the specified parameter ranges, conducted on the validation set using a five-fold cross-validation strategy. The configuration achieving the lowest average error in terms of MAE and RMSE was selected as the final model setting. During hyperparameter tuning, each configuration was trained for up to 20K epochs using an AdamW optimizer [8] with a stepwise learning rate schedule. The learning rate was initialized at 10^{-3} and progressively reduced to facilitate smoother convergence and prevent premature stagnation. Early stopping (patience = 250) was employed to avoid overfitting and ensure stable training. A detailed sensitivity analysis of these hyperparameters is provided in Appendix G.

TABLE C.1: Hyperparameter settings of HCTN

Hyperparameters	Description	Values
τ	Time Window	{2, 4, 6, 8, 10, 12, 14}
l	HCN layers	{1, 2, 3, 4}
h_n	No. of heads	{1, 2, 3, 4, 5}
d_k	Head size	{32, 64, 128, 256, 512}

Appendix D
Comparison Methods

The experiments in the main manuscript are compared with the following representative methods:
(i) WSPred [9]: This paper adopts a Tensor Factorization (TF) approach under the average QoS value regularization constraint for time-aware personalized QoS prediction.
(ii) CTF [10]: This method proposed an outlier-resilient TF method incorporating the Cauchy loss function.
(iii) NNCP [11]: This paper proposes non-negative TF via canonical polyadic decomposition, multiplicative updating (MU) algorithm are leveraged to optimized the latent factors.
(iv) BNLFT [12]: Extending NNCP, this method introduced linear biases to non-negative TF, and incorporates an alternating direction method along with MU for faster convergence.
(v) TUIPCC [5]: This paper proposed a similarity-based approach that prioritizes recent QoS values using temporal decay.
(vi) TRCF [13]: Combining ratio-based similarity and Pearson correlation, this method employs QoS averaging and deviation migration for temporal QoS prediction.
(vii) TaTruSR [14]: This method integrates a similarity-based approach with an entropy-based reliability estimation to effectively identify and exclude unreliable users.
(viii) DeepTSQP [3]: Utilizing binary invocation and similarity-based features, this paper used multi-layer GRU-based QoS sequence learning.
(ix) GFEN [4]: This method uses a GAN model, an enhanced GRU generator, and a fully connected discriminator to mitigate the features in the probabilistic matrix factorization (PMF).
(x) RNCF [6]: This paper employs a multi-layer GRU framework that uses one-hot encoded features as input.
(xi) SCATSF [15]: This approach utilizes similarity-based neighbor selection that capture high-order features using a multi-layer deep network, and GRU for temporal QoS forecasting.
(xii) PLMF [16]: This technique uses personalized LSTM and matrix factorization (MF) to learn user and service latent factors for online QoS prediction.
(xiii) GMCL [17]: Combining tripartite graph modeling and contrastive learning, this method effectively captures both local and global behaviors improves robust features representation.
(xiv) STGCN [18]: Originally proposed for traffic forecasting. STGCN introduces a spatio-temporal graph convolutional framework that simultaneously models both spatial and temporal dependencies from time-series sensor data.
(xv) STGFT [19]: The method was initially introduced as a spatio-temporal graph fusion transformer for modeling water-quality time-series in river networks.
(xvi) TPMCF [2]: This method proposed a real-time outlier-resilient approach leveraging a GCN and an enhanced transformer encoder to captures temporal fluctuations.

Positioning of HCTN: In this paper, we propose HCTN, an end-to-end anomaly-resilient framework for real-time QoS prediction that utilizes high-order multi-granularity features through a hypergraph convoluted transformer encoder. This approach addresses various challenges, such as data deficiency, data credibility, and data representation in temporal QoS prediction. HCTN effectively mitigates various underlying challenges for highly accurate, time-sensitive QoS prediction, surpassing the limitations of conventional methods.

- WSPred [9], CTF [10], NNCP [11], and BNLFT [12] employ TF-based approach to implicitly address sparsity and cold-start issues, while exploiting the triadic relationships among users, services, and time. [9], [11], [12] are sensitive to outliers due to their reliance on outlier-sensitive objective functions. Further, some of these methods incorporate regularization terms with the loss function to partially mitigate outlier effects. On the other hand, [10] uses the Cauchy loss function to address outliers. However, all of these methods struggle to capture higher-order features and handle greysheep users. In contrast, our method, HCTN, employs a hypergraph convolution transformer network which not only enhances collaborative filtering by capturing more nuanced higher-order neighborhoods, it also captures fine-to-coarse-grained complex temporal dependencies. Further, to mitigate the greysheep problem, it leverages local features derived from QoS distributions to enhance the performance of greysheep instances. Thereby, demonstrating superior prediction performance compared to these methods.

- TUIPCC [5], TRCF [13], and TaTruSR [14] primarily rely on similarity computation and weighted averaging. These approaches, despite incorporating domain knowledge, often struggle with data deficiency, data credibility, and limited utilization of higher-order features. Although [14] introduces an entropy-based mechanism to identify unreliable users, it partially addresses the greysheep issue by eliminating them during evaluation rather than mitigating the root causes. In contrast, HCTN not only identifies such anomalous users (and services) but also enhances their representation through dedicated feature learning, effectively improving robustness and expressiveness.
- DeepTSQP [3], GFEN [4], RNCF [6], SCATSF [15], and PLMF [16] utilize GRU/ LSTM to capture complex features effectively. While [3], [4], [6], [16] rely on outlier-sensitive loss functions, [15] adopts the Huber loss to partially mitigate the impact of outliers. Further, [3] leverages similarity and binary invocation features, [6] uses ID embeddings, [4] applies PMF, [15] incorporates contextual user-service features, and [16] employs latent MF features. Although some of these methods address a few more challenges, such as domain knowledge integration, data sparsity, and cold start, they still struggle with greysheep behavior and modeling long-range dependencies. Conversely, our method addresses both limitations effectively. Specifically, a dedicated greysheep mitigation module is proposed to identify and address the issue faced by such instances. Furthermore, to model the long-range dependencies in HCTN, we adopt a fine-to-coarse transformer-based module, leading to superior performance over these methods.
- GMCL [17] introduced a tripartite graph-based learning framework trained with contrastive and L1 loss, which effectively addresses the outlier issue and captures the higher-order features. However, this approach does not explicitly address the cold start and greysheep problem and further relies on ID embeddings, resulting in declined prediction performance compared to HCTN, which is specifically designed to handle these issues effectively.
- In contrast to STGCN [18] and STGFT [19], which were originally developed for traffic forecasting and water-quality prediction, respectively, by modeling spatio–temporal dependencies over homogeneous graphs constructed from physical sensors or river monitoring stations, we adapt these frameworks to the time-aware QoS prediction setting by applying them separately on user and service graphs. The node embeddings obtained from message passing on both graphs are concatenated row-wise and fed into an MLP, followed by an inner product to estimate the QoS matrix. STGCN [18] employs first-order graph convolutions and temporal gated CNNs, which restrict spatial modeling to pairwise edges and limit temporal modeling to fixed-size receptive fields; as a result, it cannot capture multi-way collaborative structures or long-range QoS dynamics. STGFT [19] improves temporal attention and introduces adaptive adjacency matrices but still relies on conventional graph structures that assume stationary spatial relationships, an assumption mismatched to the latent, irregular, and highly heterogeneous interaction patterns in QoS tensors. In contrast, HCTN introduces a hypergraph-based spatial module capable of modeling higher-order user–service co-invocation relationships beyond pairwise connectivity, allowing it to uncover richer collaborative dependencies even when direct observations are sparse. Furthermore, by integrating greysheep detection and local pattern adaptation, HCTN explicitly accounts for atypical or noisy interaction behaviors that STGCN and STGFT treat as unstructured noise. Its transformer-based temporal granularity module further captures fine-to-coarse temporal variations that conventional temporal convolution or single-level attention cannot model effectively. Theoretically, the combination of (i) higher-order hypergraph message passing, (ii) anomaly-aware feature refinement, and (iii) multi-resolution temporal attention yields a more expressive function class capable of approximating non-linear triadic relationships among users, services, and time. This increased representational capacity allows HCTN to exploit the same input features more effectively than STGCN and STGFT, ultimately leading to its superior performance on WSDREAM.
- TPMCF [2] combines a GCN-based framework and transformer encoder architectures to capture spatial and temporal patterns. However, separate training of these two distinct architectures increases the number of parameters and prolongs training time. Although it uses a robust loss function to handle outliers, it fails to address the greysheep issue. In contrast, HCTN enjoys an end-to-end deep framework that incorporates a hypergraph collaborative filtering, eliminating the need for separately trained components and providing a sparsity-resilient solution to handle anomalous instances effectively.

Appendix E
Model Complexity Analysis

This section presents a detailed complexity analysis of HCTN regarding model size and computational efficiency. We first quantify the implementation-level complexity by reporting the total number of trainable parameters and estimating the floating-point operations (FLOPs) required per forward pass. We then complement these practical measurements with an asymptotic analysis of each major module to characterize HCTN's theoretical computational behavior. Together, these results provide a comprehensive assessment of HCTN's resource requirements and demonstrate its suitability for deployment under diverse hardware and memory constraints.

A. *Memory Efficiency and Computational Complexity*

Table E.1 summarizes the computational complexity of HCTN in comparison with two strong QoS prediction baselines,

TABLE E.1: Model parameter counts and FLOPs

Method	Params (K)	FLOPs (G)
CTF [10]	0.7059	1.8403
TPMCF [2]	4765.3880	126.6281
HCTN	551.5640	27.9539

CTF [10] and TPMCF [2], by reporting both the number of trainable parameters and the floating-point operations (FLOPs) required per forward pass. The results indicate that HCTN achieved an effective balance between representational capacity and computational efficiency. Specifically, HCTN contains 551.56K parameters, an 8.6× reduction relative to TPMCF, while requiring only 27.95 GFLOPs, which is 4.5× lower than TPMCF's computational cost. While CTF attained the smallest model size and the lowest FLOPs, its prediction accuracy on the WSDREAM QoS benchmark (Tables III-IV of the main paper) remained noticeably lower than that of HCTN. This suggests that CTF's highly compact design, although computationally attractive, may limit its ability to capture the richer spatio–temporal and collaborative patterns required for accurate QoS estimation. In contrast, HCTN offers a more favorable accuracy–efficiency trade-off: it delivers substantially improved prediction performance with only a modest increase in computation. Overall, the results demonstrate that HCTN remains computationally lightweight while providing strong predictive capability, making it suitable for large-scale, time-aware QoS prediction scenarios where both accuracy and deployability are essential.

B. *Asymptotic Complexity Analysis*

Let, n and m denote the number of users and services, respectively, and $N = n+m$ the total number of nodes. Let τ represents the number of temporal snapshots, and $E = \sum_{t=1}^{\tau} e_t$ denote the total number of observed QoS interactions across all τ time steps. The asymptotic computational cost of each component of HCTN is summarized below:

- *Non-negative Matrix Decomposition (NMD):* Decomposing each $Q^t \in \mathbb{R}^{n\times m}$ into latent matrices $X_u^t \in \mathbb{R}^{n\times f_1}$ and $X_s^t \in \mathbb{R}^{m\times f_1}$ costs $\mathcal{O}(nmf_1)$ per iteration. Over I_{NMD} iterations and τ time steps: $\mathcal{O}(I_{\text{NMD}} \cdot \tau \cdot nmf_1)$.
- *Hypergraph Collaborative Filtering Module (HCFM):* Each snapshot employs an HCN with one dense projection $\mathcal{O}(Nf_1f_2)$ and l hypergraph-convolution layers, each costing $\mathcal{O}(Nf_2^2 + e_tf_2)$. Summed across all τ snapshots: $\mathcal{O}(\tau Nf_1f_2 + l\tau Nf_2^2 + lf_2E)$.
- *Greysheep Mitigation Module (GMM):* Computing local statistical features scales as $\mathcal{O}(E)$, while dense fusion across τ time steps costs $\mathcal{O}(\tau Nf_2^2)$: $\mathcal{O}(E + \tau Nf_2^2)$.

TABLE E.2: Symbols used in complexity analysis

Symbols	Representations
n	Number of users
m	Number of services
τ	Number of past time steps
l	Number of hypergraph-convolution layers
e_t	Number of QoS interactions at time t
h	Number of attention heads in MHA
d	Transformer input dimension

- *Temporal Granularity Extraction Module (TGEM):* The multi-head attention (E-block) requires $\mathcal{O}(N\tau^2 f_2)$, and the subsequent dense transformations (T- and F-blocks) add $\mathcal{O}(N\tau f_2^2)$: $\mathcal{O}(N\tau^2 f_2 + N\tau f_2^2)$.
- *Collaborative QoS Prediction Module (CQPM):* Normalization and pooling incur $\mathcal{O}(N\tau f_2)$, dense projections contribute $\mathcal{O}(Nf_2f_3 + Nf_3f_4)$, and final QoS reconstruction $US^\top$ adds $\mathcal{O}(nmf_4)$: $\mathcal{O}(N\tau f_2 + Nf_2f_3 + Nf_3f_4 + nmf_4)$.

Overall complexity: Combining all components, the total asymptotic cost of HCTN is:

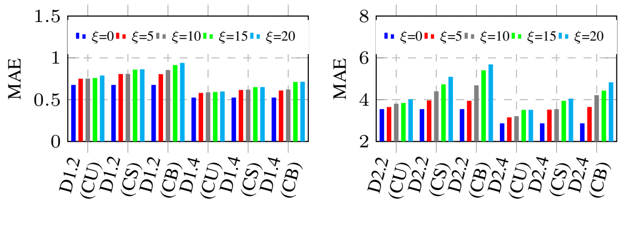
$$\mathcal{O}\Big(I_{\text{NMD}} \cdot \tau \cdot nmf_1 + l\tau Nf_2^2 + lf_2E$$
$$+ N\tau^2 f_2 + N\tau f_2^2 + nmf_4\Big).$$

In sparse QoS settings ($E \ll N^2$) and small τ, the runtime is dominated by the hypergraph convolution term $\mathcal{O}(lf_2E)$ and the dense transformation term $\mathcal{O}(N\tau f_2^2)$, ensuring linear scalability with the number of observed interactions and maintaining efficient training and inference.

APPENDIX F
IMPACT OF COLD-START

To assess the impact of cold-start conditions on HCTN, we simulate controlled cold-start scenarios by randomly selecting a fixed percentage $\xi \in \{5, 10, 15, 20\}$ of users or services and removing all their historical QoS records. Three evaluation settings are constructed: (a) *Cold-start Users (CU)*, where QoS histories for ξ% of users are removed; (b) *Cold-start Services (CS)*, where QoS histories for ξ% of services are removed; and (c) *Cold-start Both (CB)*, where QoS histories for ξ% of both users and services are removed. After applying the respective cold-start mask, the model is trained using the remaining valid entries following the same train–validation–test protocol described in Section IV of the main paper. For each ξ and each scenario, the complete training and evaluation pipeline is repeated five times with different random selections, and the mean performance across the five runs is reported. Fig. F.1 shows the results of these scenarios, and our key observations are summarized below:

(i) As ξ increased from 0 to 20 for all cases, HCTN performance declined. Further, for the same value of ξ, the performance degradation was highest in the CB scenario, followed by CS; with CU showing the least decline possibly due to the impact of service on the QoS value of a user-service pair is more prominent than that of a user.



(a) D1 Dataset (b) D2 Dataset

Fig. F.1: Performance of HCTN on cold-start with varying ξ

TABLE F.1: Cold-start users and service comparison of HCTN with CTF and TPMCF

ξ	D1.2			D1.4			D2.2			D2.4		
	CTF	TPMCF	HCTN	CTF	TPMCF	HCTN	CTF	TPMCF	HCTN	CTF	TPMCF	HCTN
0	1.8908	0.8132	0.6765	1.5337	0.6159	0.5256	10.9274	5.7447	3.5486	8.6924	4.5136	2.8671
5	1.9283	0.8490	0.8050	1.6072	0.6945	0.6099	11.7714	7.0479	3.9443	9.4462	5.4211	3.6499
10	1.9462	0.8993	0.8541	1.6173	0.7069	0.6221	12.7103	7.2463	4.6851	9.9722	5.5479	4.2163
15	1.9538	0.9419	0.9140	1.6235	0.7259	0.7119	12.6588	7.2771	5.4062	10.0847	5.6973	4.4329
20	1.9624	0.9590	0.9397	1.6303	0.7708	0.7142	12.8640	7.3544	5.6886	10.4121	5.9460	4.8293

(ii) As the training density increased, performance increased for all scenarios – CU, CS, and CB, for the same value of ξ. This suggests that even with cold-starts in QoS logs, the effects can be reduced by increasing training density, supplementing the matrix factorization to generate better latent features to handle cold-start. This justifies the requirements of matrix factorization features.

To show the effectiveness of HCTN in addressing the cold-start problem, we compare it with two prominent past methods, CTF [10] and TPMCF [2], both handle temporal QoS prediction. CTF leverages tensor factorization to handle cold-start situations, while TPMCF uses non-negative matrix factorization to extract user/ service features, helping to alleviate the cold-start issue.

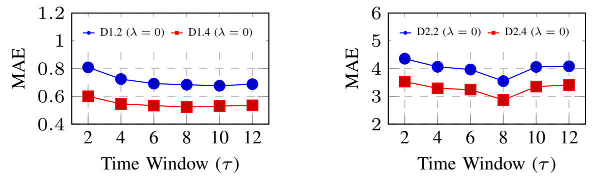
Table F.1 presents a comparison of the CB scenario across various cold-start percentages, $\xi = \{0, 5, 10, 15, 20\}\%$, on all datasets. On average, the performance gains of the HCTN over TPMCF on D1 and D2 datasets are 8.01% and 30.01%, respectively, which shows the effectiveness of HCTN over TPMCF. The performance gain of HCTN with respect to CTF is even higher, highlighting the benefits of utilizing higher-order features. This performance boost in HCTN is due to its effective feature utilization in cold-start situations and its ability to model non-linear features.

APPENDIX G
IMPACT OF HYPER-PARAMETERS

This section highlights the impact of different hyperparameters on the performance of HCTN.

A. Impact of Time Window

We explore the optimal number of time sequences needed for high QoS prediction performance. Fig.s G.1(a)-(b) show the experiments by varying the time window (τ) from 2 to 12 with a step size of 2 for all datasets. The performance improves as τ increases for both datasets. At relatively lower τ, performance deteriorates, which is due to not incorporating adequate temporal features essentially required for the high prediction performance. Further, incorporating more time slices enhances QoS prediction up to a certain τ. Specifically, for the D1.2 and D1.4 datasets, we achieved the highest prediction performance for $\tau = 10$ and $\tau = 8$, respectively. Similarly, for D2 datasets, a common $\tau = 10$ achieve the competitive performance. This experiment highlights that while a sufficient number of time slices is crucial for improving model performance by offering more time information, excessively large τ can negatively impact the model.

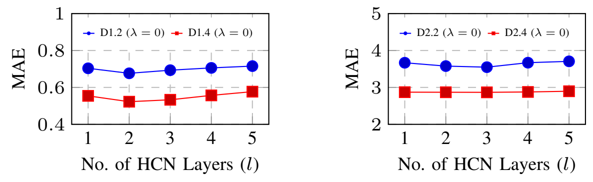


(a) D1 Dataset (b) D2 Dataset

Fig. G.1: Impact of number of time-windows (τ)

B. Impact of HCN Layers

This experiment shows the utilization of higher-order features obtained using hypergraph collaborative filtering. Fig.s G.2(a)-(b) illustrate the impact of the number of HCN layers (l) by varying from 1 to 5. We achieved competitive performance for $l = 2$ and $l = 3$ for D1 and D2, respectively. However, for higher l, performance deteriorates since, with increased hypergraph convolution layer, the node features propagation may aggregated to give indistinguishable node embedding.



(a) D1 Dataset (b) D2 Dataset

Fig. G.2: Impact of no. of HCN layers (l)

C. Impact of Number of Heads and Head Size in MHA

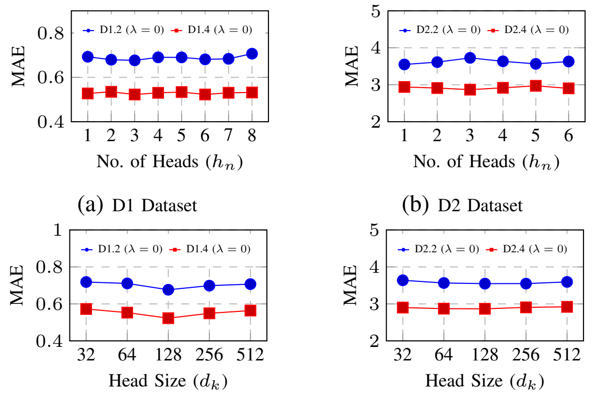
To optimize the parameters in multi-head attention (MHA), we investigate the sufficient requirements of the number of heads (h_n) and head size (d_k) affecting model performance. Our key observations are as follows.

(i) Utilizing multiple heads in MHA enables the capture of multi-dimensional information from the features. This experiment intends to find the optimal number of heads that balance prediction performance with the number of learnable parameters, since an increment in the number of heads also increases the learnable parameters. Fig.s G.3(a)-(b) display the impact of h_n for all datasets. We achieved the best performance for $h_n = 3$ and $h_n = 6$ for D1.2 and D1.4, respectively. However, $h_n = 1$ for D2.2, and $h_n = 3$ for D2.4 worked better.

(ii) The head size refers to the dimensionality of each attention head in MHA. Specifically, it controls the size of features each attention head incorporates. For a fixed input features

TABLE I.1: Comparison of HCTN with past methods

Methods	Loss Function	MAE					RMSE				
		D1.1	D1.2	D1.3	D1.4	D1.5	D1.1	D1.2	D1.3	D1.4	D1.5
TF	MSE	2.9184	2.7888	2.500	2.4800	2.2127	4.7508	4.5696	4.4100	4.3900	4.0169
AMF [20]	MRE	1.6328	1.5766	1.5527	1.5391	-	-	-	-	-	-
HCTN	**Cauchy**	**0.8453**	**0.6765**	**0.6133**	**0.5256**	**0.2237**	**2.4486**	**2.2336**	**2.1074**	**1.9754**	**1.2853**
		D2.1	D2.2	D2.3	D2.4	D2.5	D2.1	D2.2	D2.3	D2.4	D2.5
TF	MSE	8.7997	8.5080	8.400	8.350	7.8045	39.5133	39.2792	39.3000	39.2800	38.6964
AMF [20]	MRE	5.0942	4.6984	4.5398	4.4610	-	37.0818	36.0874	35.6750	35.5216	-
HCTN	**Cauchy**	**3.9693**	**3.5486**	**3.3071**	**2.8671**	**0.7404**	**23.8219**	**21.3090**	**20.8300**	**18.8000**	**4.8526**



(a) D1 Dataset (b) D2 Dataset (c) D1 Dataset (d) D2 Dataset

Fig. G.3: (a)-(b) Impact of no. of heads, and (c)-(d) head size dimension, a smaller head size captures finer details, while larger heads capture broader information. To assess the head size, we vary d_k in powers of 2, ranging from 2^5 to 2^9. Fig.s G.3(c)-(d) illustrate the performance with different d_k for all datasets. We achieved optimal performance for $d_k = 2^7 = 128$ across all datasets.

Appendix H
Model Deployability

We analyze the deployability of HCTN by measuring its inference latency across a diverse set of hardware configurations, ranging from mid-tier consumer CPUs to high-end workstation processors. As shown in Table H.1, HCTN consistently achieved low inference times, with mean latency ranging from 2.12×10^{-6} second to 2.58×10^{-5} second across all machines. These results demonstrate that HCTN introduces negligible computational overhead and delivers real-time predictions without requiring GPU acceleration. The low variance in latency further indicates stable execution behavior across different memory capacities and core counts. Overall, the study confirms that HCTN is an efficient, lightweight, and easily deployable in practical QoS-aware service platforms under resource-constrained environments.

TABLE H.1: HCTN Inference time on different machines

Processor Configuration	Core	RAM	Inference Time (second) (mean ± stdev)
10th Gen Intel® Core™ i7-10700@4.80 GHz	16	7.53 GB	2.58e-5 ± 1.19e-6
13th Gen Intel® Core™ i5-1335U@4.60 GHz	12	15.30 GB	1.67e-5 ± 1.66e-6
AMD Ryzen-9 7950X@5.70 GHz	16	32.00 GB	2.12e-6 ± 6.24e-8
13th Gen Intel® Core™ i9-13900H@5.40 GHz	20	30.98 GB	2.01e-5 ± 2.30e-6
12th Gen Intel® Core™ i7-12700@4.90 GHz	20	62.49 GB	1.06e-5 ± 1.64e-7
12th Gen Intel® Core™ i7-12700@4.90 GHz	20	125.49 GB	1.10e-5 ± 1.35e-7
2nd Intel® Xeon Gold 6226R CPU@3.90GHz	32	188.35 GB	2.43e-5 ± 1.05e-7

Appendix I
Supporting Results for the Main Manuscript

In this subsection, we have moved some less prominent baseline methods (refer to Table I.1) for the sake of completeness, while incorporating the most recent and highly relevant SOTA methods into the main manuscript.

References

[1] Z. Zheng *et al.*, "Investigating QoS of Real-World Web Services," *IEEE TSC*, vol. 7, no. 1, pp. 32–39, 2014.
[2] S. Kumar *et al.*, "TPMCF: Temporal QoS Prediction Using Multi-Source Collaborative Features," *IEEE TNSM*, pp. 1–1, 2024.
[3] G. Zou *et al.*, "DeepTSQP: Temporal-Aware Service QoS Prediction via Deep Neural Network and Feature Integration," *Know.-Based Syst.*, vol. 241, no. C, 2022.
[4] P. Zhang *et al.*, "Generative-Adversarial-Based Feature Compensation to Predict Quality of Service," *IEEE TSC*, vol. 17, no. 01, pp. 209–223, 2024.
[5] E. Tong *et al.*, "A Missing QoS Prediction Approach via Time-Aware Collaborative Filtering," *IEEE TSC*, vol. 15, no. 6, pp. 3115–3128, 2022.
[6] T. Liang *et al.*, "Recurrent Neural Network Based Collaborative Filtering for QoS Prediction in IoV," *IEEE TITS*, vol. 23, no. 3, pp. 2400–2410, 2022.
[7] Y. Zhang *et al.*, "Recurrent Tensor Factorization for Time-Aware Service Recommendation," *Appl. Soft Comput.*, vol. 85, no. C, 2019.
[8] I. Loshchilov *et al.*, "Decoupled Weight Decay Regularization," in *ICLR*. arXiv:1711.05101, 2019.
[9] Y. Zhang *et al.*, "WSPred: A Time-Aware Personalized QoS Prediction Framework for Web Services," in *IEEE ISSRE*, 2011, pp. 210–219.
[10] F. Ye *et al.*, "Outlier-Resilient Web Service QoS Prediction," in *The Web Conf.*, 2021, p. 3099–3110.
[11] W. Zhang *et al.*, "Temporal QoS-Aware Web Service Recommendation via Non-Negative Tensor Factorization," in *ACM WWW*, 2014, p. 585–596.
[12] X. Luo *et al.*, "Temporal Pattern-Aware QoS Prediction via Biased Non-Negative Latent Factorization of Tensors," *IEEE Trans. on Cybernetics*, vol. 50, no. 5, pp. 1798–1809, 2020.
[13] G. Zou *et al.*, "TRCF: Temporal Reinforced Collaborative Filtering for Time-Aware QoS Prediction," *IEEE TSC*, no. 01, pp. 1–14, 5555.
[14] "A two-dimensional time-aware cloud service recommendation approach with enhanced similarity and trust," *JPDC*, vol. 190, p. 104889, 2024.
[15] J. Zhou *et al.*, "Spatial Context-Aware Time-Series Forecasting for QoS Prediction," *IEEE TNSM*, vol. 20, no. 2, pp. 918–931, 2023.
[16] R. Xiong *et al.*, "Personalized LSTM Based Matrix Factorization for Online QoS Prediction," in *IEEE ICWS*, 2018, pp. 34–41.
[17] H. Wu *et al.*, "Effective Graph Modeling and Contrastive Learning for Time-Aware QoS Prediction," *IEEE TSC*, vol. 17, no. 6, pp. 3513–3526, 2024.
[18] B. Yu *et al.*, "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting," in *IJCAI*, 7 2018, pp. 3634–3640.
[19] J. Bi *et al.*, "Long-Term Water Quality Prediction With Transformer-Based Spatial-Temporal Graph Fusion," *IEEE TASE*, vol. 22, pp. 11 392–11 404, 2025.
[20] J. Zhu *et al.*, "Online QoS Prediction for Runtime Service Adaptation via Adaptive Matrix Factorization," *IEEE TPDS*, vol. 28, no. 10, pp. 2911–2924, 2017.