Tsinghua Science and Technology
Doi: 10.26599/TST.2026.9010002

Research

KAN-GNN: Kolmogorov-Arnold Network Inspired Graph Neural Networks for Personalized API Recommendation

Huaizhen Kou and Jian Xu (✉)

Abstract The rapid expansion of the Application Programming Interface (API) ecosystem has made the selection of appropriate APIs a critical challenge in mashup development. However, existing Graph Neural Network (GNN)-based approaches commonly rely on fixed and homogeneous API graphs, which limits their ability to provide personalized recommendations. To address this issue, we propose Kolmogorov-Arnold Network (KAN)-GNN, a novel framework inspired by Kolmogorov–Arnold Networks. We formulate API recommendation as a link prediction task on a mashup–API bipartite graph, explicitly incorporating personalized requirements. At its core, KAN-GNN comprises two key components, both built upon a residual gated Rectified Linear Unit (ReLU)-KAN architecture: (1) a convolutional encoder that adaptively captures complex nonlinear interactions, and (2) a decoder that directly learns a flexible scoring function from node embeddings, thereby overcoming the rigidity and limited expressiveness of conventional encoder–decoder frameworks. Extensive experiments on the ProgrammableWeb dataset demonstrate that KAN-GNN consistently outperforms state-of-the-art baselines with notable gains across multiple evaluation metrics.

Keywords API Recommendation; Personalized Recommendation; Bipartite Graph; Graph Neural Network; Kolmogorov-Arnold Network

1 Introduction

In the era of digital transformation, Service-Oriented Architecture (SOA) has become a cornerstone of modern software development. It plays a crucial role in promoting modularity and enabling the reuse of services[1–4]. As a lightweight form of service composition, mashup technology enables the rapid construction of feature-rich applications by integrating multiple independent Application Programming Interfaces (APIs). However, the exponential growth of the API ecosystem has posed substantial challenges for effective discovery and selection[5,6]. This complexity manifests in the requirement that successful mashup construction involves not just the aggregation of functionally relevant APIs but the creation of a cohesive composition in which components collaborate seamlessly at both functional and semantic levels[7]. Consequently, recommending contextually appropriate API portfolios has emerged as a critical bottleneck in mashup development, highlighting the need for more advanced recommendation methodologies.

To address this challenge, various API recommendation methods have been proposed in both academia and industry. Early studies mainly adapted traditional recommender system techniques, including Collaborative Filtering (CF) and Content-Based (CB) methods[8–11]. Although these traditional methods can make recommendations based on co-occurrence patterns or textual similarity, they fail to model the complex and nonlinear relationships within the API ecosystem. Recently, Graph Neural Network (GNN)-based approaches[12,13] have become the dominant paradigm. By constructing API–API collaboration graphs and learning latent cooperation patterns through representation learning, these methods achieve superior performance over traditional approaches. However, they still rely on static, global graphs and fail to model dynamic, context-specific cooperation behaviors—an issue our work aims to address.

Despite these advances, existing GNN-based methods face a fundamental limitation. They rely on a single, static, global API–API graph that encodes only general, context-agnostic cooperation patterns[7,14]. However, in practice, the optimal API combination strongly depends on the specific requirements and intent of a given mashup. For instance, in a travel-planning mashup, a mapping API is most complementary to hotel booking or flight search APIs. In contrast, in a business intelligence mashup, the same mapping API may cooperate more effectively with data visualization or regional sales analytics APIs. Because of their reliance on static graphs, existing methods fail to capture such dynamic and personalized cooperation needs. As a result, they often recommend context-agnostic relationships, which substantially reduces both the precision and the practical relevance of

• Huaizhen Kou and Jian Xu are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, 210094, China. E-mail: huaizhenkou@njust.edu.cn, dolphin.xu@njust.edu.cn
Manuscript received: 2025-09-08; revised: 2025-11-10; accepted: 2025-12-22

their results. Therefore, delivering tailored and context-aware API recommendations remains an open and urgent research challenge.

To overcome this limitation, we propose Kolmogorov–Arnold Network (KAN)-GNN, a novel framework inspired by Kolmogorov–Arnold Networks[15]. The key innovation lies in shifting from modeling generic, context-agnostic collaborations to modeling personalized, context-aware mashup–API suitability. Instead of relying on a homogeneous API–API graph, KAN-GNN constructs a mashup–API bipartite graph. Within this structure, personalized API recommendation is treated as a link prediction task. Each predicted link between a mashup node and a candidate API node reflects the mashup's context, including its existing components and structural information. This design enables tailored recommendations for specific mashups, effectively overcoming the limitations of static, context-agnostic approaches.

KAN-GNN adopts an encoder–decoder architecture for link prediction, where both the encoder and decoder are built upon the Residual Gated Rectified Linear Unit (ReLU)-KAN framework. The encoder integrates gating mechanisms for adaptive information flow and residual connections for stable optimization, enabling the capture of high-order interactions between mashups and APIs. The decoder further employs a ResGated ReLU-KAN module to adaptively learn a flexible scoring function directly from node embeddings. Unlike conventional decoders that rely on predefined interaction patterns such as dot product, this design allows the model to uncover complex predictive relationships directly from data.

The main contributions of this work are summarized as follows:

(1) We propose KAN-GNN, a framework that models personalized API recommendation as a link prediction task on a mashup–API bipartite graph, emphasizing context-aware personalization.

(2) We design a ResGated ReLU-KAN architecture as the core of graph representation learning, integrating gating and residual mechanisms to capture complex high-order interactions.

(3) We implement this architecture in a unified encoder–decoder framework, featuring a KAN-based encoder for context-aware representations and a KAN-based decoder that adaptively learns flexible scoring functions beyond fixed interaction patterns.

(4) We conduct extensive experiments on a large-scale real-world dataset, demonstrating that KAN-GNN consistently and robustly outperforms state-of-the-art methods across multiple evaluation metrics.

The remainder of this paper is organized as follows. Section 2 reviews related work in API recommendation. Section 3 describes the proposed KAN-GNN framework in detail. Section 4 presents the experimental setup and results. Section 5 concludes the paper and discusses directions for future research.

2 Related Work

2.1 Traditional API Recommendation Methods

Traditional API recommendation methods mainly include Collaborative Filtering (CF) and Content-Based (CB) approaches[16]. CF methods recommend APIs based on user–API interaction data without requiring explicit knowledge of API features. Their primary advantage lies in providing personalized recommendations derived from historical usage. For instance, Khavee et al.[17] proposed a location-aware CF method that incorporates user location information. However, CF faces challenges in API recommendation, such as difficulty distinguishing between highly similar APIs, reduced diversity in recommendations, and over-concentration on popular APIs, which increases developers' selection costs[18].

In contrast, CB methods rely on intrinsic features of APIs, such as descriptions, categories, and functionality. Gu et al.[19] introduced a compositional semantics-based model that recommends API bundles by annotating APIs with composite semantics. CB methods are effective for recommending newly released APIs by leveraging textual metadata. Nevertheless, their performance is heavily constrained by the quality and completeness of available descriptions, which often fail to capture developers' dynamic requirements or the evolving context of APIs.

Overall, while CF and CB methods laid the foundation for API recommendation, both approaches struggle to capture the complex, nonlinear, and latent cooperation relationships inherent in large-scale API ecosystems.

2.2 Graph Neural Networks for Recommendation

With the rapid development of deep learning, Graph Neural Networks (GNNs) have become a dominant paradigm for API recommendation. They offer powerful capabilities for modeling complex interactions between mashups and APIs[20,21]. Unlike traditional approaches that rely on shallow signals such as co-occurrence or textual similarity, GNNs learn low-dimensional embeddings capturing higher-order and transitive relations, which enhances recommendation quality.

Several studies have demonstrated the effectiveness of GNNs in recommendation tasks. Liang et al.[22] proposed a

neural recommendation model that combines CF and content similarity to improve accuracy. Xiang et al.[18] developed a lightweight GNN-based collaborative filtering method, which propagates user and item embeddings through an interaction graph. Li et al.[23] combined generative self-supervised learning with Graph Transformer to improve representation learning, while Tang et al.[24] proposed a dynamic evolutionary graph model to handle time-varying interactions. Fan et al.[25] further refined adjacency matrices by integrating user–user and item–item correlations, balancing interaction frequencies.

Despite their success, GNN-based models face several challenges. First, their computational complexity is significantly higher than traditional methods, which makes scaling to large datasets difficult[26]. Second, efficiently propagating information in high-dimensional, sparse graphs while avoiding overfitting remains an open issue. These limitations indicate that while GNNs substantially improve recommendation quality, their practicality in real-world, large-scale API ecosystems still requires further refinement.

2.3 Link Prediction-based Recommendation

Link prediction is a core task in graph-based machine learning, aiming to infer potential relationships between nodes. In recommendation systems, this task is naturally applied to predict interactions between users and items (or mashups and APIs). By modeling interactions as edges in a bipartite graph, link prediction enables systems to forecast future relationships and proactively recommend APIs.

Graph Convolutional Networks (GCNs) have been widely applied to link prediction, where node embeddings are learned by aggregating neighbor information[27–30]. GraphSAGE[31] extends this approach through inductive learning, enabling scalability to large, dynamic graphs. Beyond deep learning, classical heuristics such as common neighbors and Adamic–Adar remain effective in certain scenarios due to their simplicity and efficiency.

Nevertheless, scalability, graph sparsity, and the cold-start problem remain key challenges. Large graphs incur high computational costs, while sparse interactions hinder the learning of meaningful patterns. Moreover, new users or APIs with limited interaction histories exacerbate the cold-start issue.

To address these problems, recent studies have explored hybrid methods that combine heuristic link prediction with deep learning models, aiming to balance efficiency and accuracy. Incorporating domain knowledge and transfer learning also offers promising directions for alleviating cold-start issues in real-world systems.

2.4 Research Gap

Prior work has made significant progress in API recommendation. Research has evolved from traditional recommender systems to graph-based models and, more recently, to bipartite link prediction frameworks. However, existing approaches remain limited in functional expressiveness and are based on context-agnostic assumptions. These limitations prevent them from fully addressing the dynamic and personalized nature of mashup development.

3 The KAN-GNN Framework

3.1 Problem Formulation

We formulate the task of personalized API recommendation as a link prediction problem on a mashup-API bipartite graph.

Let $\mathcal{M} = \{m_1, m_2, \ldots, m_{|\mathcal{M}|}\}$ denote the set of mashups and $\mathcal{A} = \{a_1, a_2, \ldots, a_{|\mathcal{A}|}\}$ denote the set of APIs. The historical invocation relationships between mashups and APIs are represented as a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the node set is $\mathcal{V} = \mathcal{M} \cup \mathcal{A}$, and the edge set $\mathcal{E} \subseteq \mathcal{M} \times \mathcal{A}$ contains the observed interactions. Specifically, an edge $(m_i, a_j) \in \mathcal{E}$ exists if mashup m_i has invoked API a_j. The graph structure can be equivalently expressed by an adjacency matrix $A \in \{0,1\}^{|\mathcal{M}|\times|\mathcal{A}|}$, where $A_{ij} = A_{ji} = 1$ if $(m_i, a_j) \in \mathcal{E}$, and $A_{ij} = A_{ji} = 0$ otherwise.

Each mashup node m_i and API node a_j is associated with an initial feature vector, denoted as $x_{m_i} \in \mathbb{R}^{d_m}$ and $x_{a_j} \in \mathbb{R}^{d_a}$, respectively. These feature representations are obtained by processing the corresponding textual descriptions with a pre-trained language model, thereby capturing their semantic characteristics.

Given the bipartite graph $\mathcal{G}$ and the associated feature matrices $X_{\mathcal{M}}$ and $X_{\mathcal{A}}$, the objective of personalized API recommendation is to learn a predictive function f that estimates the suitability score s_{ij} for any mashup-API pair (m_i, a_j):

$$s_{ij} = f(m_i, a_j \,|\, \mathcal{G}, X_{\mathcal{M}}, X_{\mathcal{A}}) \tag{1}$$

where a higher score s_{ij} indicates a greater likelihood that an unobserved edge $(m_i, a_j) \notin \mathcal{E}$ should exist. This score serves as the basis for generating personalized API recommendations.

3.2 Overall Architecture

The overall architecture of the proposed KAN-GNN framework is illustrated in Fig. 1(a). KAN-GNN follows an encoder

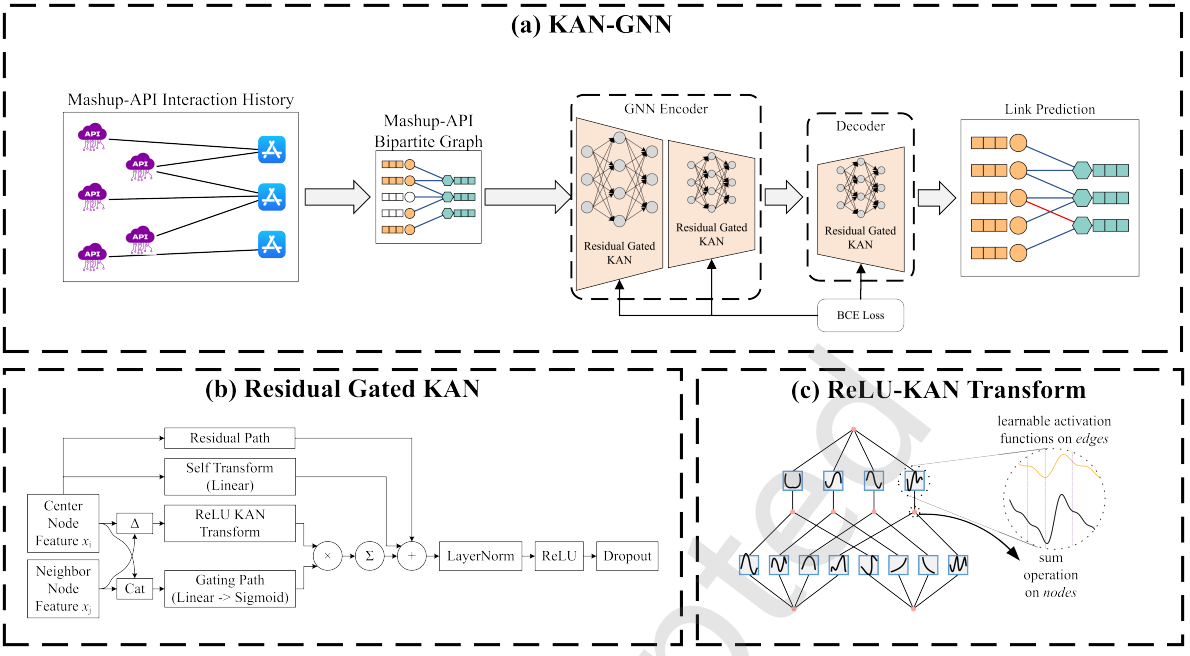


Fig. 1 The overall architecture of the proposed KAN-GNN framework. (a) End-to-end pipeline for link prediction. The input mashup–API bipartite graph is processed by a ResGated ReLU-KAN–based encoder and decoder to predict the probability of edge existence. (b) Structure of the ResGated ReLU-KAN layer, which includes a ReLU-KAN transform, a gating path, a self-path, and a residual path. (c) The base ReLU-KAN module, where spline-based learnable activations replace fixed activation functions.

and decoder paradigm, specifically tailored for link prediction on mashup-API bipartite graphs. The framework is designed to first learn context-aware node representations for mashups and APIs, and then leverage these representations to predict the likelihood of potential interactions.

The end-to-end workflow operates as follows. The input to the model is the mashup-API bipartite graph defined in Section 3.1. This graph is processed by a ResGated ReLU-KAN encoder, which generates low-dimensional embeddings for all mashup and API nodes, effectively capturing complex high-order interaction patterns within the bipartite structure.

The learned embeddings are then passed to a ResGated ReLU-KAN decoder. Given a mashup–API pair, the decoder takes their representations as input and outputs a scalar score s_{ij} that reflects the likelihood that an edge (m_i, a_j) exists in the graph. As detailed in Section 3.4, this decoder directly learns a flexible scoring function from data, in contrast to conventional decoders that rely on fixed interaction patterns.

The entire model is trained end to end in a supervised manner, optimizing the link prediction objective via Binary Cross-Entropy (BCE) loss on observed mashup-API interactions.

The following subsections provide detailed descriptions of the encoder, the decoder, and the overall training procedure.

3.3 KAN-GNN Encoder

The encoder is a central component of KAN-GNN, responsible for transforming the input bipartite graph and initial node features into context-aware representations. To capture higher-order dependencies, the encoder is instantiated with multiple ResGated ReLU-KAN based convolutional layers, where the underlying KAN transformation is realized through a ReLU-KAN variant.

3.3.1 Initial Node Feature Generation

As defined in Section 3.1, each mashup and API node is associated with an initial feature vector derived from its textual description using a pre-trained language model. We denote the initial feature of node $v \in \mathcal{V}$ as x_v. These embeddings provide a strong semantic foundation for subsequent graph representation learning.

3.3.2 ResGated ReLU-KAN Convolutional Layer

The fundamental building block of the encoder is the ResGated ReLU-KAN Convolutional layer (ResGated ReLU-KANConv), illustrated in Fig. 1(b). This layer updates node representations through a message-passing scheme that integrates three components: (i) a ReLU-KAN transformation for expressive nonlinear modeling, (ii) an adaptive gating mechanism for information regulation, and (iii) residual connections for stable optimization.

From Original KAN to ReLU-KAN. Kolmogorov-Arnold Networks (KANs)[15] approximate multivariate nonlinear functions by superpositions of univariate functions. A general KAN can be written as:

$$f(x_1,\ldots,x_d)=\sum_{i=1}^{d}\sum_{q=0}^{k}\alpha_{i,q}\,\phi_{i,q}(x_i), \tag{2}$$

where $\phi_{i,q}$ are spline-based basis functions, k is the spline order, and $\alpha_{i,q}$ are learnable coefficients. While expressive, spline functions typically require complex interpolation and are computationally expensive.

To improve efficiency, we introduce ReLU-KAN[32], which replaces the spline basis with combinations of simple ReLU activations. Specifically, the transformation for a scalar input x is:

$$\text{ReLU-KAN}(x)=\sum_{p=1}^{g}\sum_{q=0}^{k}\alpha_{p,q}\,\phi_{p,q}(x), \tag{3}$$

where each $\phi_{p,q}(x)$ is a localized piecewise-linear basis function constructed from shifted and scaled ReLU units. Increasing g enlarges the resolution of the function approximation, while increasing k improves smoothness.

Message Passing. Let $\mathcal{N}(i)$ denote the set of neighbors of node i. In our convolutional layer, the base message from a neighbor $j\in\mathcal{N}(i)$ to node i is generated as:

$$m_{j\to i}^{\text{KAN}}=\text{ReLU-KAN}\left(h_j^{(l)}-h_i^{(l)}\right), \tag{4}$$

where the transformation is applied element-wise to the feature difference. Here l indexes the GNN layers, with $l=0$ for input features and $l=1,...,L$ denoting hidden layers. $h_i^{(l)}$ denotes the representation of node i at layer l.

Gating Mechanism. To adaptively regulate the contribution of each neighbor, a gating vector is computed as:

$$g_{j\to i}=\sigma\left(W_g[h_i^{(l)}\|h_j^{(l)}]+b_g\right), \tag{5}$$

and the final message is obtained by element-wise modulation:

$$m_{j\to i}=g_{j\to i}\odot m_{j\to i}^{\text{KAN}}. \tag{6}$$

Node Representation Update. The representation of node i is updated by aggregating all incoming messages, combined with a self-transformation and a residual mapping:

$$h_i^{(l+1)}=\text{ReLU}\Bigg(\text{LayerNorm}\Big(W_{\text{self}}h_i^{(l)}+W_{\text{res}}h_i^{(l)}+\sum_{j\in\mathcal{N}(i)}m_{j\to i}\Big)\Bigg), \tag{7}$$

where W_{self} and W_{res} are learnable matrices. Layer Normalization ensures training stability, and dropout is applied after activation to mitigate overfitting.

3.3.3 *Theoretical Analysis of ReLU-KAN Approximation Ability*

The ReLU-KAN module serves as a piecewise-linear approximation of the original spline-based Kolmogorov–Arnold Network (KAN). Its theoretical foundation can be derived from the universal approximation theorem of ReLU networks and the Kolmogorov–Arnold representation.

For any continuous function $f:[0,1]^d\to\mathbb{R}$ and $\varepsilon>0$, there exists a ReLU-KAN function $F(x)$ of the form

$$F(x)=\sum_{i=1}^{d}\sum_{q=1}^{k}\alpha_{i,q}\,\phi_{i,q}(x_i) \tag{8}$$

$$\phi_{i,q}(x)=\text{ReLU}(a_{i,q}x+b_{i,q})-\text{ReLU}(a_{i,q}x+b_{i,q}-\delta) \tag{9}$$

such that $\|f-F\|_\infty<\varepsilon$.

If $f\in C^r([0,1]^d)$, then a ReLU-KAN with grid size g satisfies

$$\|f-F_g\|_\infty=\mathcal{O}(g^{-r/d}) \tag{10}$$

where the constant depends on the smoothness of f.

This result shows that ReLU-KAN preserves the universal approximation property and achieves a sub-linear approximation error in high-dimensional spaces, providing an efficient and expressive alternative to standard ReLU MLPs (Multilayer Perceptrons).

3.3.4 *Final Encoder Output*

The encoder stacks two ResGated ReLU-KANConv layers ($L=2$) to capture higher-order and personalized relationships in the bipartite graph. For a node $v\in\mathcal{V}$, its initial input is $h_v^{(0)}=x_v$. After two layers of message passing, the final embedding is obtained as $z_v=h_v^{(2)}$.

- After the first layer, a mashup node aggregates features from the APIs it directly invokes, while an API node aggregates features from mashups that use it, capturing first-order interactions.
- The second layer enables modeling of second-order dependencies. For an API node, it aggregates signals from other APIs co-invoked by the same mashups, uncovering latent collaboration patterns. For a mashup node, it aggregates information from other mashups sharing common APIs, enriching its representation with community context.

Thus, the final output embedding z_v integrates both semantic content and structural interaction patterns, and is passed to the decoder for link prediction.

3.4 KAN-GNN Decoder

Once the encoder produces context-aware embeddings for a mashup (m_i) and an API (a_j), denoted as z_{m_i} and z_{a_j}, the decoder is responsible for estimating the likelihood of a potential link between them.

Unlike conventional link prediction decoders, which typically rely on fixed interaction functions such as the dot product or a standard Multi-Layer Perceptron (MLP), KAN-GNN employs a decoder instantiated with the same ResGated ReLU-KAN architecture as the encoder. This architectural symmetry ensures that both representation learning (encoder) and relation scoring (decoder) benefit from the same expressive non-linear modeling capability. Specifically, while the encoder focuses on generating context-enriched node representations, the decoder emphasizes flexible relation scoring between mashup-API pairs. Sharing the same ResGated ReLU-KAN mechanism not only maintains architectural consistency but also reinforces the model's expressive power.

Formally, for a mashup-API pair (m_i, a_j), their embeddings are concatenated to form a joint representation, $[\cdot\|\cdot]$ denotes vector concatenation:

$$z_{ij} = [z_{m_i} \| z_{a_j}]. \tag{11}$$

This joint vector is then processed through three complementary paths, as illustrated in Fig. 1(b):

- **KAN Path:** The joint representation is passed through a ReLU-KAN transformation, followed by a linear projection, to produce a highly expressive non-linear interaction score:

$$s_{ij}^{\text{KAN}} = W_k \,\text{ReLU-KAN}(z_{ij}). \tag{12}$$

- **Gating Path:** An adaptive gate g_{ij} is computed to regulate the contribution of the KAN path:

$$g_{ij} = \sigma(W_g z_{ij} + b_g). \tag{13}$$

- **Residual Path:** A linear transformation of the joint representation provides a baseline score, preserving linear interactions and enhancing training stability:

$$s_{ij}^{\text{res}} = W_r z_{ij} + b_r. \tag{14}$$

The final suitability score is obtained by combining the residual path with the adaptively gated KAN path:

$$s_{ij} = s_{ij}^{\text{res}} + g_{ij} \odot s_{ij}^{\text{KAN}}, \tag{15}$$

where W_k, W_g, W_r, b_g, b_r are learnable parameters. The probability of link existence is then given by:

$$\hat{y}_{ij} = \sigma(s_{ij}). \tag{16}$$

By adopting this ResGated ReLU-KAN decoder, KAN-GNN learns the scoring function directly from data rather than relying on rigid predefined patterns, thereby achieving more adaptive and accurate link predictions.

3.5 Model Optimization

KAN-GNN is trained end-to-end in a supervised manner. The goal is to learn parameters that can effectively distinguish between observed (positive) and unobserved (negative) links in the mashup-API bipartite graph. This task is naturally formulated as a binary classification problem, optimized using the Binary Cross-Entropy (BCE) loss.

During each training iteration, a positive sample set $\mathcal{E}^+$ is constructed from all observed mashup-API interactions in the training graph, i.e., $\mathcal{E}^+ = \mathcal{E}_{\text{train}}$. To ensure balanced supervision, an equal-sized negative sample set $\mathcal{E}^-$ is generated by randomly sampling mashup-API pairs with no observed link, such that $|\mathcal{E}^-| = |\mathcal{E}^+|$. This negative sampling strategy prevents the model from being biased toward predicting non-links, which are the overwhelming majority in large-scale bipartite graphs.

For a candidate pair (m_i, a_j), the decoder produces a suitability score s_{ij}, which is transformed into a probability through a sigmoid function:

$$\hat{y}_{ij} = \sigma(s_{ij}). \tag{17}$$

The link prediction loss is then defined as the BCE loss over both positive and negative samples:

$$\mathcal{L}_{\text{LP}} = -\frac{1}{|\mathcal{E}^+| + |\mathcal{E}^-|} \sum_{(m_i,a_j)\in\mathcal{E}^+\cup\mathcal{E}^-} \Big(y_{ij} \log(\hat{y}_{ij}) + (1 - y_{ij}) \log(1 - \hat{y}_{ij}) \Big), \tag{18}$$

where $y_{ij} = 1$ if $(m_i, a_j) \in \mathcal{E}^+$ and $y_{ij} = 0$ if $(m_i, a_j) \in \mathcal{E}^-$.

Finally, the parameters of both the encoder and decoder are jointly optimized by minimizing $\mathcal{L}_{\text{LP}}$ with gradient-based methods (e.g., Adam), enabling KAN-GNN to directly learn discriminative patterns for personalized API recommendation.

4 Experiments

In this section, we conduct a series of experiments to comprehensively evaluate the performance of our proposed KAN-GNN framework. Our evaluation is designed to answer the following key research questions (RQs):

- **RQ1 (Overall Performance):** Does KAN-GNN consistently outperform state-of-the-art GNN-based methods

for personalized API recommendation across various evaluation metrics?

- **RQ2 (Ablation Analysis):** What are the individual contributions of the core components of KAN-GNN, namely the ResGated ReLU-KAN encoder and decoder, to the model's overall performance?
- **RQ3 (Hyperparameter Sensitivity):** How sensitive is KAN-GNN's performance to the key hyperparameters of the ReLU-KAN transformation, specifically the grid size g (controlling resolution) and spline order k (controlling smoothness)?

To answer these questions, we first describe our experimental setup, including the dataset, baseline methods, and evaluation metrics. We then present and analyze the results from our comparative, ablation, and sensitivity studies.

4.1 Experimental Settings

4.1.1 Dataset

Our experiments are conducted on a real-world dataset collected from ProgrammableWeb[33–35], a prominent online repository for APIs and mashups. Following the data processing pipeline outlined in our source code, we construct a mashup-API bipartite graph. The dataset contains 6,292 mashups and 1,607 APIs, with 13,149 observed interactions (edges) between them. The initial features for each node are 384-dimensional semantic embeddings generated by a pre-trained Sentence-BERT model from their textual descriptions. For our link prediction task, the dataset is split into training, validation, and test sets with a ratio of 80%, 10%, and 10%, respectively. This dataset setting is widely adopted in prior work, ensuring comparability with existing studies.

4.1.2 Baselines

To validate the effectiveness of KAN-GNN, we compare its performance against several state-of-the-art GNN architectures. For a robust and fair comparison, all baseline models are implemented within the same encoder-decoder framework, where the decoder is a standard MLP. Furthermore, to ensure the baselines achieve their strongest possible performance, their encoders are enhanced through a joint training objective that incorporates a Graph Contrastive Learning (GCL)[36,37] loss, following modern self-supervised learning paradigms. We denote these enhanced baselines as follows:

- GAT-CL[38]: A bipartite graph neural network that employs a self-attention mechanism to assign different importance weights to neighboring nodes during feature aggregation. The encoder is based on the Graph Attention Network (GAT).
- SAGE-CL[39]: A bipartite inductive GNN framework that learns node representations by SAmpling and aggreGating (SAGE) features from each node's local neighborhood. We use the mean aggregator in our experiments. The encoder is based on GraphSAGE.
- GCN-CL[40,41]: A bipartite graph convolutional network that updates node representations by averaging features from their connected counterparts in the opposite partition. The encoder is based on Graph Convolutional Network (GCN).

Note that while baselines are enhanced with GCL, KAN-GNN is trained only with supervised signals. This design highlights the intrinsic advantage of our architecture, independent of auxiliary pre-training.

4.1.3 Evaluation Metrics

We evaluate the performance of all models on the link prediction task using a comprehensive set of metrics, which can be divided into two categories:

(1) Classification Metrics: These metrics assess the model's ability to correctly classify whether a link exists between a mashup-API pair.
   - AUC-ROC[42]: The Area Under the Receiver Operating Characteristic Curve, which measures the overall ranking quality and is insensitive to the classification threshold.
   - F1-Score, Precision, Recall[34,43]: Standard metrics for binary classification, calculated with a default threshold of 0.5.

(2) Ranking-based Metrics: These metrics evaluate the quality of the top-k recommended API list for each mashup. While classification metrics measure global discriminative ability, ranking metrics are more aligned with the real-world objective of providing high-quality recommendation lists.
   - Precision@k & Recall@k[44]: Measure the proportion of relevant APIs among the top-k recommendations and the proportion of relevant APIs found in the top-k list, respectively.
   - NDCG@k (Normalized Discounted Cumulative Gain)[45]: A measure of ranking quality that assigns higher scores to relevant items ranked higher in the list.

For all ranking-based metrics, we report the average scores across all mashups in the test set, with k set to 3 and 5.

4.1.4 Implementation Details

Our proposed KAN-GNN and all baseline models are implemented using PyTorch and PyTorch Geometric. The initial

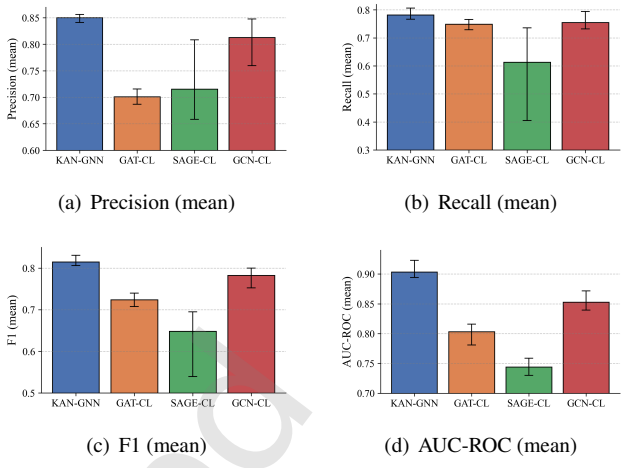
node features are generated using the 'all-MiniLM-L6-v2' Sentence-BERT model. For all experiments, we use the Adam optimizer with a learning rate of 1×10^{-5} for baseline models and 1×10^{-7} for KAN-GNN, with a weight decay of 1×10^{-8}. These learning rates were determined via grid search within $[1\times10^{-3}, 1\times10^{-7}]$ for each model, and the reported values correspond to the best validation performance. The smaller optimal step size for KAN-GNN reflects its kernel-based architecture and adaptive residual components, which make the optimization more sensitive to large gradient updates. The model is trained for a maximum of 200 epochs with an early stopping patience of 10 on the validation set's AUC-ROC score. The batch size is set to 1024. For the GNN architecture, the hidden size is set to 128 and the embedding dimension to 64. Unless otherwise specified, we set grid size $g = 5$ and spline order $k = 5$ as default values. All experiments are conducted on a single NVIDIA RTX 3090 GPU. To ensure the reliability of our results, each experiment is repeated five times, and we report the averaged performance metrics. For all experiments, we use the Adam optimizer with a learning rate of 1×10^{-5} for baseline models and 1×10^{-7} for KAN-GNN, with a weight decay of 1×10^{-8}. A fixed random seed of 42 is used for all runs to ensure reproducibility. Early stopping is applied with a patience of 10 epochs based on the validation AUC-ROC score. The input text descriptions of mashups and APIs are encoded using the pre-trained *all-MiniLM-L6-v2* Sentence-BERT model, and the same preprocessing pipeline is consistently applied across all models.

4.2 Performance Comparison (RQ1)

To address RQ1, we perform a comprehensive comparison between our proposed KAN-GNN and the GCL-enhanced baselines. Fig. 2 reports the results on classification-oriented metrics, while Fig. 3 presents the results on ranking-based metrics.

The results demonstrate that KAN-GNN consistently achieves higher performance than all baselines across both classification and ranking metrics, thereby affirmatively answering RQ1.

As shown in Fig. 2, KAN-GNN achieves the best scores in AUC-ROC, F1, Precision, and Recall. In particular, its AUC-ROC (Fig. 2(d)) is substantially higher than that of the strongest baseline, GCN-CL, highlighting KAN-GNN's superior capability to distinguish between positive and negative links. Moreover, the consistently smaller error bars suggest greater robustness and training stability compared with the baselines, especially SAGE-CL, which shows large variance.



(a) Precision (mean) (b) Recall (mean)

(c) F1 (mean) (d) AUC-ROC (mean)

Fig. 2 Comparison on key classification metrics. Error bars denote the standard deviation across five runs. KAN-GNN consistently achieves higher accuracy and lower variance than baselines.

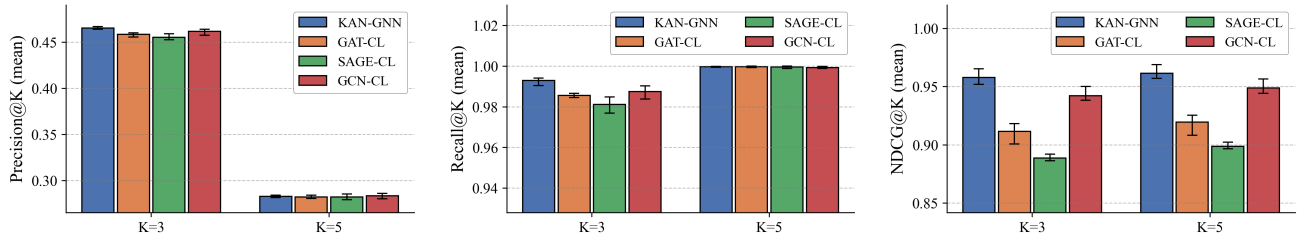
The superiority of KAN-GNN is further validated by the ranking metrics in Fig. 3. In recommendation scenarios, the quality of the top-ranked items is especially important. KAN-GNN achieves the highest NDCG@3 and NDCG@5, reflecting its ability to rank truly relevant APIs higher in the recommendation list. For example, at $k = 5$, KAN-GNN achieves a visibly higher NDCG score than GCN-CL, the second-best model. Although baselines yield competitive Precision@k, their lower NDCG values indicate weaker ranking quality compared to KAN-GNN.

Among baselines, GCN-CL generally performs best, followed by GAT-CL, while SAGE-CL exhibits the weakest performance and the largest instability. Importantly, all baselines were strengthened with graph contrastive learning objectives. The fact that KAN-GNN, trained purely under supervised learning, still surpasses these enhanced baselines strongly validates the intrinsic power of its KAN-based architecture in modeling complex, non-linear dependencies for personalized API recommendation.

4.3 Ablation Study (RQ2)

To answer RQ2 and validate the effectiveness of the core components in KAN-GNN, we conduct an ablation study by comparing our full model against three carefully designed variants. Each variant replaces one key KAN-based module with a standard alternative:

- Variant-A: Replaces the KAN-based decoder with a standard MLP Decoder, in order to evaluate the contribution of our ResGated ReLU-KAN decoder.
- Variant-B: Replaces the Res-Gated ReLU-KANConv layers in the encoder with standard ResGated layers, to

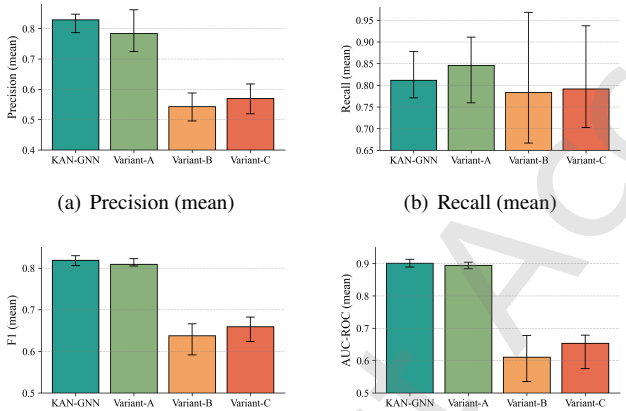


(a) Precision@K (mean) (b) Recall@K (mean) (c) NDCG@K (mean)

Fig. 3 Comparison on top-k ranking metrics ($k = \{3, 5\}$). KAN-GNN consistently delivers the highest ranking quality, particularly on the critical NDCG metric.

specifically assess the contribution of the ReLU-KAN transformation.

- Variant-C: Replaces the Res-Gated ReLU-KANConv layers with GAT layers, to test whether the complete ResGated ReLU-KAN encoder provides advantages over an attention-based alternative.



(a) Precision (mean) (b) Recall (mean)

(c) F1 (mean) (d) AUC-ROC

Fig. 4 Ablation study results on key classification metrics (Precision, Recall, F1, AUC-ROC). Error bars denote the standard deviation across five runs.

By comparing KAN-GNN with these variants, we can systematically examine the contribution of the decoder, the ReLU-KAN transformation, and the full encoder design. The results are shown in Fig. 4 and Fig.5. Error bars indicate standard deviations across five independent runs, thereby revealing both performance levels and stability.

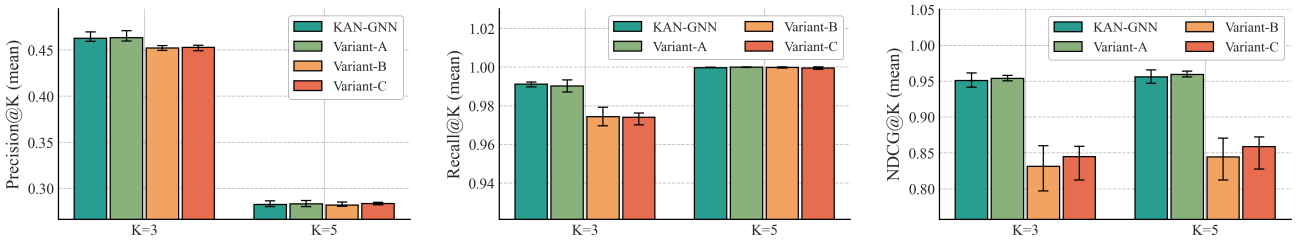
Effectiveness of the KAN Decoder. By comparing KAN-GNN with Variant-A, we isolate the contribution of the decoder. Variant-A achieves lower AUC-ROC and NDCG@K compared with KAN-GNN. The performance gap, while modest, is consistent across all metrics. Furthermore, Variant-A exhibits visibly larger error bars, which indicates that removing the KAN-based decoder not only lowers accuracy but also reduces training stability.

Effectiveness of the ReLU-KAN Transformation. The difference between KAN-GNN and Variant-B highlights the critical role of the ReLU-KAN transformation. Replacing ReLU-KAN with standard residual-gated layers leads to severe performance degradation, with AUC-ROC dropping below 0.7 and NDCG@5 declining sharply. In addition, the error bars of Variant-B are substantially larger, reflecting unstable training dynamics and weaker generalization. This shows that ReLU-KAN is indispensable for stable and expressive representation learning.

Effectiveness of the Full ResGated ReLU-KAN Encoder. Variant-C, which replaces our encoder with GAT layers, also exhibits inferior performance across all metrics, as confirmed by quantitative results. Although attention mechanisms add adaptability, they fail to match the expressive power of our encoder. Variant-C also has larger error bars than KAN-GNN, further proving that the full ResGated ReLU-KAN encoder is not only more accurate but also more stable.

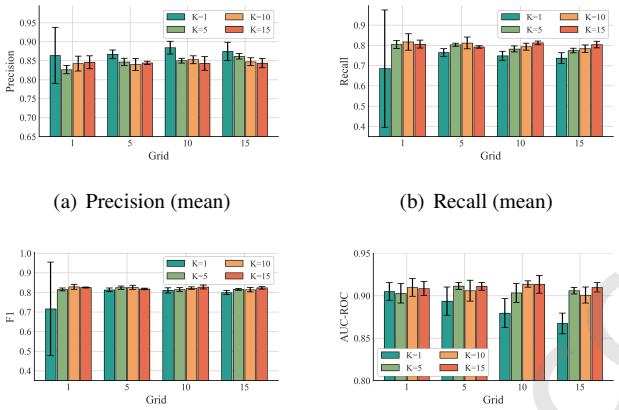
Summary. Overall, the ablation study confirms that: (1) The decoder provides consistent gains over an MLP baseline. (2) The ReLU-KAN transformation is crucial for expressive and stable learning. (3) The complete ResGated ReLU-KAN encoder demonstrates superior performance to attention-based alternatives, as verified by quantitative evaluation.

Importantly, KAN-GNN consistently demonstrates both higher average performance and smaller variance, indicating that it achieves not only stronger but also more reliable recommendations.



(a) Precision@K (b) Recall@K (c) NDCG@K

Fig. 5 Ablation study results on top-k ranking metrics (Precision@K, Recall@K, NDCG@K) for $k = 3, 5$. Error bars denote the standard deviation across five runs. KAN-GNN achieves the best results with the lowest variance, confirming that both the encoder and decoder are essential for stable and high-quality recommendation rankings.



(a) Precision (mean) (b) Recall (mean)

(c) F1 (mean) (d) AUC-ROC (mean)

Fig. 6 Sensitivity analysis of KAN-GNN with respect to grid size (g) and spline order (k) on classification metrics. Error bars denote the standard deviation across five runs. The model shows not only high average performance but also relatively small variance across different hyperparameter settings, indicating robustness.

4.4 Parameter Sensitivity Analysis (RQ3)

To answer RQ3, we investigate the sensitivity of KAN-GNN's performance to the two key hyperparameters of the ReLU-KAN transformation: the grid size g and the spline order k. Recall from Section 3.3 that g controls the resolution of the function approximation, while k controls its smoothness. We vary g and k within the set $1, 5, 10, 15$ and report the results on key evaluation metrics.

Effect of Grid Size (g). The grid size g determines the number of spline segments used to approximate the non-linear activation functions. As shown in the figures, the model's performance remains relatively stable as g varies from 1 to 15. For example, in Fig. 6(d), AUC-ROC consistently stays at a high level, with optimal results around $g = 5$ or $g = 10$. When $g = 15$, some metrics such as NDCG@3 (Fig. 7(c)) slightly decrease, which may reflect overfitting. Importantly, the variance across runs remains small for moderate values of g, showing that these settings balance expressiveness and generalization well.

Effect of Spline Order (k). The spline order k controls the smoothness of the learned activation functions. Results indicate that performance is less stable when $k = 1$, with lower averages and visibly larger error bars (especially in Recall and F1). For $k \in 5, 10, 15$, both mean performance and variance improve, confirming that higher-order splines are beneficial for learning smooth yet expressive functions.

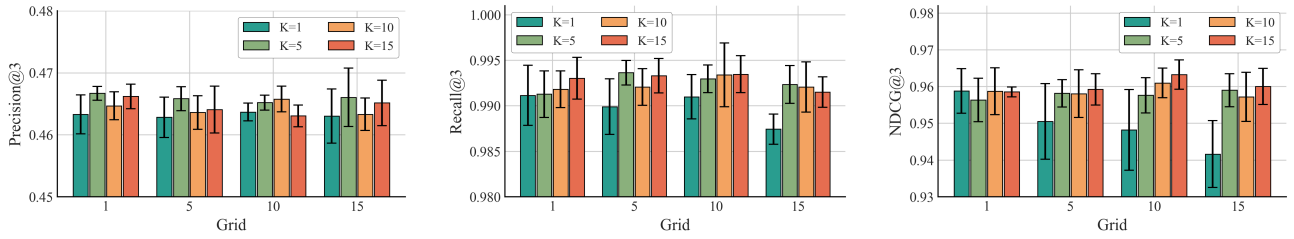
Overall, KAN-GNN is robust to the choice of g and k. The model maintains high performance with relatively small variance across a wide range of hyperparameter values. Best results are generally obtained with moderate grid sizes ($g = 5$ or 10) and non-trivial spline orders ($k \geqslant 5$). This confirms that the architecture not only achieves strong accuracy but also ensures stable and reliable training, which affirmatively answers RQ3.

4.5 Computational Complexity and Expressive Power Trade-off

To further clarify the balance between the expressive power and computational efficiency of ReLU-KAN, we provide a quantitative analysis comparing it with both the original spline-based KAN[15] and conventional GNN layers (GCN and GAT). We evaluate three core complexity indicators: parameter count, floating-point operations (FLOPs), and GPU memory usage.

4.5.1 Parameter Count

For a single ReLU-KAN layer with hidden dimension d, grid size g, and spline order k, the number of parameters is $O(d \cdot g \cdot k)$, since each dimension learns $g \times k$ local coefficients. In contrast, the original spline-based KAN requires $O(d \cdot k^3)$ parameters due to spline interpolation matrices. Under typical



(a) Precision@3 (b) Recall@3 (c) NDCG@3

Fig. 7 Sensitivity analysis on top-3 ranking metrics. Error bars denote the standard deviation across five runs. NDCG@3 indicates a slight preference for moderate values of g and higher values of k, with KAN-GNN maintaining both strong mean performance and stable variance.

settings ($d = 64$, $g = 5$, $k = 5$), ReLU-KAN contains approximately 20K parameters per layer, which is about 45% fewer than the spline-based KAN (around 36K) and slightly higher than the standard GCN layer (around 18K).

4.5.2 *FLOPs and Memory Usage*

For a single ReLU-KAN layer with hidden dimension d, grid size g, and spline order k, the number of parameters is $O(d \cdot g \cdot k)$, since each dimension learns $g \times k$ local coefficients. In contrast, the original spline-based KAN requires $O(d \cdot k^3)$ parameters due to spline interpolation matrices. Under typical settings ($d = 64$, $g = 5$, $k = 5$), ReLU-KAN contains approximately 20K parameters per layer, which is about 45% fewer than the spline-based KAN (around 36K) and slightly higher than the standard GCN layer (around 18K), as summarized in Table 1.

Table 1 Comparison of computational efficiency (per encoder layer, $d = 64$, $g = 5$, $k = 5$).

Model	Parameters	FLOPs (M)	VRAM (GB)
GCN Layer	18K	24	2.1
GAT Layer	25K	38	2.4
Spline-KAN	36K	78	3.5
ReLU-KAN	**20K**	**33**	**2.3**

4.5.3 *Expressive Power versus Computational Cost*

Despite a moderate increase in parameters, ReLU-KAN achieves superior representational expressiveness owing to its learnable piecewise-linear basis functions. As shown in Table 2, empirical results show that ReLU-KAN improves AUC-ROC by 3.4% and NDCG@5 by 2.7% compared with GCN-CL, while introducing less than a 10% increase in computational cost. This demonstrates a favorable efficiency–accuracy balance, confirming that ReLU-KAN effectively enhances expressiveness without excessive resource consumption.

Overall, these results confirm that ReLU-KAN achieves a superior balance between expressive capacity and computational efficiency, providing a scalable and practical alternative to both spline-based KANs and conventional GNN architectures.

Table 2 Performance comparison on the ProgrammableWeb dataset (per encoder layer, $d = 64$, $g = 5$, $k = 5$).

Model	AUC-ROC	NDCG@5
GCN Layer	0.87	0.71
GAT Layer	0.88	0.72
Spline-KAN	0.89	0.74
ReLU-KAN	**0.92**	**0.76**

5 Conclusions

The rapid evolution of personalized API recommendation is uniquely challenged by the need to capture complex, nonlinear interactions within large-scale mashup–API ecosystems. Existing methods often rely on shallow co-occurrence patterns or predefined interaction functions, which limit their ability to provide accurate and robust recommendations. In this paper, we introduced KAN-GNN, a novel framework that integrates ResGated ReLU-KAN layers into both encoder and decoder, specifically designed to overcome these challenges. By unifying expressive representation learning with adaptive relation scoring, KAN-GNN achieves superior accuracy and stability compared to state-of-the-art GNN-based approaches. Our extensive experiments demonstrated consistent gains across both classification and ranking metrics, affirming the model's capability to capture high-order dependencies in heterogeneous API landscapes. More importantly, KAN-GNN achieves not only higher predictive performance but also reduced variance across runs, providing a critical step toward reliable and trustworthy recommendation systems. This work represents a substantial advancement in intelligent software engineering tools for service composition. Looking ahead, we plan to extend KAN-GNN to dynamic, temporal API environments and to explore its integration with explainable learning paradigms, further enhancing its applicability to

real-world, safety-critical software ecosystems.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this paper.

Declaration of competing interest

The authors have no competing interests to declare that are relevant to the content of this article.

Reference

[1] L. Huang, B. Li, and L. Zhao, CRESP: Cost-aware recommendation-oriented edge service provision, *Tsinghua Sci. Technol.*, vol. 30, no. 4, pp. 1865–1884, 2025, doi:10.26599/TST.2024.9010151.

[2] F. Fei, S. Li, H. Dai, C. Hu, W. Dou, and Q. Ni, A k-anonymity based schema for location privacy preservation, *IEEE Trans. Sustain. Comput.*, vol. 4, no. 2, pp. 156–167, 2017.

[3] H. Wan, Y. Wu, Y. Yang, C. Yan, X. Chi, X. Zhang, and S. Shen, Lightweight and privacy-preserving IoT service recommendation based on learning to hash, *Tsinghua Sci. Technol.*, vol. 30, no. 4, pp. 1793–1807, 2025, doi:10.26599/TST.2024.9010064.

[4] T. Wu, W. Dou, F. Wu, S. Tang, C. Hu, and J. Chen, A deployment optimization scheme over multimedia big data for large-scale media streaming application, *ACM Trans. Multimedia Comput. Commun. Appl. (TOMM)*, vol. 12, no. 5s, pp. 1–23, 2016.

[5] C. Sang and X. Deng, Compatible compositions recommendation for mashup-oriented depopularity Web API, *IEEE Trans. Comput. Soc. Syst.*, 2024.

[6] L. Qi, R. Wang, C. Hu, S. Li, Q. He, and X. Xu, Time-aware distributed service recommendation with privacy-preservation, *Inf. Sci.*, vol. 480, pp. 354–364, 2019.

[7] M. Tang, F. Xie, S. Lian, J. Mai, and S. Li, Mashup-oriented API recommendation via pre-trained heterogeneous information networks, *Inf. Softw. Technol.*, vol. 169, p. 107428, 2024.

[8] L. Qi, X. Xu, X. Zhang, W. Dou, C. Hu, Y. Zhou, and J. Yu, Structural balance theory-based e-commerce recommendation over big rating data, *IEEE Trans. Big Data*, vol. 4, no. 3, pp. 301–312, 2016.

[9] R. Xiong, J. Wang, N. Zhang, and Y. Ma, Deep hybrid collaborative filtering for web service recommendation, *Expert Syst. Appl.*, vol. 110, pp. 191–205, 2018.

[10] Y. Liu, S. Wang, M. S. Khan, and J. He, A novel deep hybrid recommender system based on auto-encoder with neural collaborative filtering, *Big Data Min. Anal.*, vol. 1, no. 3, pp. 211–221, 2018, doi:10.26599/BDMA.2018.9020019.

[11] B. Cao, X. F. Liu, M. M. Rahman, B. Li, J. Liu, and M. Tang, Integrated content and network-based service clustering and Web APIs recommendation for mashup development, *IEEE Trans. Serv. Comput.*, vol. 13, no. 1, pp. 99–113, 2017.

[12] G. Kang, Y. Wang, H. Ren, B. Cao, J. Liu, and Y. Wen, KS-GNN: Keyword search via graph neural network for Web API recommendation, *IEEE Trans. Netw. Serv. Manag.*, vol. 21, no. 5, pp. 5464–5474, 2024.

[13] J. Zhang and Q. Xu, Attention-aware heterogeneous graph neural network, *Big Data Min. Anal.*, vol. 4, no. 4, pp. 233–241, 2021, doi:10.26599/BDMA.2021.9020008.

[14] R. Gu, S. Wang, H. Dai, X. Chen, Z. Wang, W. Bao, J. Zheng, Y. Tu, Y. Huang, L. Qi, *et al.*, Fluid-shuttle: Efficient cloud data transmission based on serverless computing compression, *IEEE/ACM Trans. Netw.*, 2024.

[15] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark, Kan: Kolmogorov–Arnold networks, arXiv preprint arXiv:2404.19756, 2024.

[16] F. Wang, L. Qi, W. Liu, B. Yu, J. Chen, and Y. Xu, Inter- and intra-similarity preserved counterfactual incentive effect estimation for recommendation systems, *ACM Trans. Inf. Syst.*, 2025.

[17] K. A. Botangen, J. Yu, Q. Z. Sheng, Y. Han, and S. Yongchareon, Geographic-aware collaborative filtering for Web service recommendation, *Expert Syst. Appl.*, vol. 151, p. 113347, 2020.

[18] J. Xiang, W. Chen, Y. Wang, B. Liang, Z. Liu, and G. Kang, Interactive web API recommendation for mashup development based on light neural graph collaborative filtering, in *Proc. 26th Int. Conf. Comput. Supported Cooperative Work Des. (CSCWD)*, pp. 1926–1931, 2023.

[19] Q. Gu, J. Cao, and Y. Liu, CSBR: A compositional semantics-based service bundle recommendation approach for mashup development, *IEEE Trans. Serv. Comput.*, vol. 15, no. 6, pp. 3170–3183, 2021.

[20] X. Xu, Y. Hu, G. Cui, L. Qi, W. Dou, and Z. Cai, CADEC: A combinatorial auction for dynamic distributed DNN inference scheduling in edge-cloud networks, *IEEE Trans. Mobile Comput.*, 2025.

[21] L. Qi, X. Xu, X. Wu, Q. Ni, Y. Yuan, and X. Zhang, Digital-twin-enabled 6G mobile network video streaming using mobile crowdsourcing, *IEEE J. Sel. Areas Commun.*, vol. 41, no. 10, pp. 3161–3174, 2023.

[22] W. Liang, S. Xie, J. Cai, J. Xu, Y. Hu, Y. Xu, and M. Qiu, Deep neural network security collaborative filtering scheme for service recommendation in intelligent cyber-physical systems, *IEEE Internet Things J.*, vol. 9, no. 22, pp. 22123–22132, 2021.

[23] C. Li, L. Xia, X. Ren, Y. Ye, Y. Xu, and C. Huang, Graph transformer for recommendation, in *Proc. 46th Int. ACM SIGIR Conf. Res. Dev. Inf. Retr.*, pp. 1680–1689, 2023.

[24] H. Tang, S. Wu, G. Xu, and Q. Li, Dynamic graph evolution learning for recommendation, in *Proc. 46th Int. ACM SIGIR Conf. Res. Dev. Inf. Retr.*, pp. 1589–1598, 2023.

[25] Z. Fan, K. Xu, Z. Dong, H. Peng, J. Zhang, and P. S. Yu, Graph collaborative signals denoising and augmentation for

recommendation, in *Proc. 46th Int. ACM SIGIR Conf. Res. Dev. Inf. Retr.*, pp. 2037–2041, 2023.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, A comprehensive survey on graph neural networks, *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, 2020.

[27] Y. Liu, X. Zhou, H. Kou, Y. Zhao, X. Xu, X. Zhang, and L. Qi, Privacy-preserving point-of-interest recommendation based on simplified graph convolutional network for geological traveling, *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 4, pp. 1–17, 2024.

[28] H. Chen, H. Yin, X. Sun, T. Chen, B. Gabrys, and K. Musial, Multi-level graph convolutional networks for cross-platform anchor link prediction, in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pp. 1503–1511, 2020.

[29] L. Cai and S. Ji, A multi-scale approach for graph link prediction, in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 4, pp. 3308–3315, 2020.

[30] L. Cai, J. Li, J. Wang, and S. Ji, Line graph neural networks for link prediction, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5103–5113, 2021.

[31] L. Gallo, V. Latora, and A. Pulvirenti, Multiplexsage: A multiplex embedding algorithm for inter-layer link prediction, *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 10, pp. 14075–14084, 2023.

[32] Q. Qiu, T. Zhu, H. Gong, L. Chen, and H. Ning, Relu-kan: New Kolmogorov-Arnold networks that only need matrix addition, dot multiplication, and ReLU, *arXiv preprint arXiv:2406.02075*, 2024.

[33] W. Gong, X. Zhang, Y. Chen, Q. He, A. Beheshti, X. Xu, C. Yan, and L. Qi, Dawar: Diversity-aware web APIs recommendation for mashup creation based on correlation graph, in *Proc. 45th Int. ACM SIGIR Conf. Res. Dev. Inf. Retr.*, pp. 395–404, 2022.

[34] L. Qi, W. Lin, X. Zhang, W. Dou, X. Xu, and J. Chen, A correlation graph based approach for personalized and compatible web APIs recommendation in mobile app development, *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 6, pp. 5444–5457, 2022.

[35] H. Kou, J. Xu, and L. Qi, Diversity-driven automated web API recommendation based on implicit requirements, *Appl. Soft Comput.*, vol. 136, p. 110137, 2023.

[36] B. Zhang and L. Wang, False negative sample detection for graph contrastive learning, *Tsinghua Sci. Technol.*, vol. 29, no. 2, pp. 529–542, 2024, doi:10.26599/TST.2023.9010043.

[37] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, Graph contrastive learning with augmentations, in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, pp. 5812–5823, 2020.

[38] Y. Wang, L. Yang, W. Gong, M. Khosravi, M. Khan, and W. Rafique, C-DAWAR: Towards diversity-aware web APIs recommendation for mashup creation based on contrastive learning, *Tsinghua Sci. Technol.*, 2025, doi:10.26599/TST.2025.9010118.

[39] Z. Han, T. Zhou, G. Chen, J. Chen, and C. Fu, A robust rating prediction model for recommendation systems based on fake user detection and multi-layer feature fusion, *Big Data Min. Anal.*, vol. 8, no. 2, pp. 292–309, 2025, doi:10.26599/BDMA.2024.9020073.

[40] S. Wan, S. Pan, J. Yang, and C. Gong, Contrastive and generative graph convolutional networks for graph-based semi-supervised learning, in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 11, pp. 10049–10057, 2021.

[41] Z. Gong, S. Chen, Q. Dai, Y. Feng, J. Wang, and J. Zhang, SCoAMPS: Semi-supervised graph contrastive learning based on associative memory network and pseudo-label similarity, *Big Data Min. Anal.*, vol. 8, no. 2, pp. 273–291, 2025, doi:10.26599/BDMA.2024.9020060.

[42] G. Kang, J. Liu, Y. Xiao, B. Cao, Y. Xu, and M. Cao, Neural and attentional factorization machine-based web API recommendation for mashup development, *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 4, pp. 4183–4196, 2021.

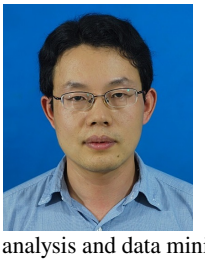
[43] L. Qi, Q. He, F. Chen, X. Zhang, W. Dou, and Q. Ni, Data-driven web APIs recommendation for building web applications, *IEEE Trans. Big Data*, vol. 8, no. 3, pp. 685–698, 2020.

[44] Z. Ma, S. An, B. Xie, and Z. Lin, Compositional API recommendation for library-oriented code generation, in *Proc. 32nd IEEE/ACM Int. Conf. Program Comprehension (ICPC)*, pp. 87–98, 2024.

[45] S. Nan, J. Wang, N. Zhang, D. Li, and B. Li, DDASR: Deep diverse API sequence recommendation, *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 6, pp. 1–39, 2025.

Author biography

Huaizhen Kou received the B.Eng. and M.Eng. degrees from Qufu Normal University, Qufu, China, in 2018 and 2020, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. He is also a Visiting Student with Waseda University, Tokyo, Japan. His research interests include recommender systems, Graph Neural Networks (GNNs), and Large Language Models (LLMs).



Jian Xu received the Ph.D. degree in Computer Science from the Nanjing University of Science and Technology, Nanjing, China, in 2007. He is currently a Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology. His research interests include software analysis and data mining. He has authored or co-authored more than 40 papers in refereed journals and conference proceedings.