Transformer-aware Sequence-to-Sequence Network for Personalized Tag Recommendation in Software Information Sites

Shubhi Bansal[1,a] (phd2001201007@iiti.ac.in), Jahnavi Sunchu[1,a] (jahnavisunchu@gmail.com), Shahid Shafi Dar[a] (phd2201201004@iiti.ac.in), Nagendra Kumar (nagendra@iiti.ac.in)

[1]These authors contributed equally to this work.
[a] Department of Computer Science and Engineering,
Indian Institute of Technology Indore,
Khandwa Road, Simrol,
Indore 453552, INDIA

Corresponding Author:
Nagendra Kumar
Department of Computer Science and Engineering,
Indian Institute of Technology Indore,
Khandwa Road, Simrol,
Indore 453552, INDIA
Email: nagendra@iiti.ac.in

Transformer-aware Sequence-to-Sequence Network for Personalized Tag Recommendation in Software Information Sites

Shubhi Bansal[1a], Jahnavi Sunchu[1a], Shahid Shafi Dar[a], Nagendra Kumar[a,*]

[a]*Department of Computer Science and Engineering, Indian Institute of Technology, Indore, India*

Abstract

Context: Automatic tag recommendation is crucial for content understanding and retrieval on software information sites. Existing approaches formulate tag recommendation as either multi-label classification problem or sentence matching. However, multi-label classification by treating tags as independent labels, neglects the semantic relationships and dependencies between them, leading to inconsistent recommendations. Sentence matching techniques, which rely on lexical similarity, fail to capture contextual information and broader semantic meaning. Moreover, several works leverage limited content sources such as titles, body, and code snippets of software objects, leading to data sparsity issues. Extant research provides generic tag suggestions, overlooking users' expertise and interests.

Objective: To address these limitations, we propose a novel tr**AN**sformer-based seque**N**ce-to-sequence framework for pers**O**nlaised **TA**g recommenda**TION**, dubbed as ANNOTATION.

Method: This approach enables the model to learn dependencies between tags and generate contextually relevant recommendations. To enhance the representation of software objects and mitigate data sparsity, we incorporate valuable information from associated comments, such as clarifications, usage examples, and bug reports. This additional conversational context provides insights into the problem, alternative solutions, and related concepts discussed by the community, resulting in more informed tag recommendations. Furthermore, we personalize tag suggestions by incorporating user profile descriptions and badges, which reflect users' expertise and interests within specific domains. This ensures that generated tags align with both the software object and the user's specific knowledge domain, contributing to a more tailored user experience.

Results: Extensive empirical and qualitative evaluations on datasets from Code Review and Stack Overflow demonstrate that our approach significantly outperforms state-of-the-art methods.

Conclusion: Our findings highlight the importance of considering tag dependencies, contextual information, and user preferences for accurate and personalized tag recommendation in software information sites.

Keywords: Tag Recommendation, Sequence-to-sequence, Software Tagging, Software Information Sites

[1]These authors contributed equally to this work.

1. Introduction

Community Question-and-Answer (CQA) forums, such as Stack Overflow[3] and Code Review[4] , are indispensable resources for software developers. CQA forums facilitate knowledge exchange and collaborative problem-solving by allowing developers to pose questions and receive solutions from peers, reuse code snippet Wang et al. (2018), stay informed about technological advancements (Brandt et al., 2009; Xia et al., 2013; Treude & Storey, 2010), expand their knowledge base (Guerrouj et al., 2015), and address technical challenges (Storey et al., 2010). With nearly 24 million questions and 33 million answers on Stack Overflow[5], users struggle to determine if their query has been previously asked or locate relevant answers. To combat information overload (Gottipati et al., 2011; Kononenko et al., 2012), these platforms enable users to manually assign relevant labels to questions, referred to as tags (Nie et al., 2020). Tags function as topic indicators by summarizing questions and assisting users in navigating, classifying, and organizing software objects (Al-Kofahi et al., 2010) to enhance the overall performance of platforms (Cai et al., 2016). However, manual tagging is time-consuming, erroneous and prone to biases due to users' varying preferences, educational backgrounds, and English proficiency (Wang et al., 2019a; Anuradha et al., 2021) leading to inconsistent tagging (Zhang et al., 2018a) and making information retrieval even more difficult. Despite Stack Overflow allowing upto five tags[6] per question, it has been observed that 87% questions have fewer than five tags and 38.38% have only one or two (Saha et al., 2013). This incompleteness stems from factors such as user negligence, inadequate understanding of questions, and informal tagging behaviors (Nie et al., 2014; Zhou et al., 2017), thereby limiting efficient knowledge sharing (Nie et al., 2014; Xu et al., 2022). To address these limitations, automated tagging systems are needed to revolutionize information accessibility within QA platforms, fostering a more efficient and collaborative learning environment for the community.

Bridging the Gaps: Challenges in Automatic Tag Recommendation

Previous studies have formulated tag recommendation as Multi-Label Classification (MLC) (Zhou et al., 2017; Li et al., 2019; Liu et al., 2018; He et al., 2022; Bansal et al., 2024a,b) and sentence matching (Zheng et al., 2020; Li et al., 2023a,b) problem. However, these methods have limitations. MLC-based approaches treat tags as independent labels, representing them as sparse one-hot vectors (Yu et al., 2023). This prevents MLC from leveraging

*Corresponding author.

Email addresses: phd2001201007@iiti.ac.in (Shubhi Bansal), jahnavisunchu@gmail.com (Jahnavi Sunchu), phd2201201004@iiti.ac.in (Shahid Shafi Dar), nagendra@iiti.ac.in (Nagendra Kumar)

[3]https://stackoverflow.com/

[4]https://codereview.stackexchange.com/

[5]https://data.stackexchange.com/

[6]https://meta.stackoverflow.com/questions/405446/why-is-maximum-number-of-tags-on-a-question-set-to-5

semantic information embedded in tags and co-occurrence patterns(Bansal et al., 2022; Yu et al., 2023). Additionally, MLC methods are confined to predefined sets of candidate tags, making them less adaptable to evolving content. Sentence matching methods assess lexical similarity between content of software objects and candidate tags. These methods struggle to capture contextual relevance, resulting in recommendations that may be broadly related to the topic but do not accurately reflect the core focus of the content. Sequence Generation (SG) presents a promising alternative. By framing tag recommendation as a conditional language modeling task, SG-based methods capture deeper semantic relationships between texts and tags. Considering tags as a sequence allows them to capture dependencies, ensuring each subsequent tag is contextually relevant to previously generated ones. SG-based approaches can combine words in innovative ways yielding tags that did not appear as complete sequences in the training data. This flexibility arises from SG's ability to learn a probability distribution over word sequences. For instance, consider the example post shown in Figure 1. The question involves game theory, algorithms, and recursion which are interconnected concepts. SG-based tag recommendation system would capture correlations between tags, allowing it to suggest all relevant tags (java, algorithm, game-theory, recursion) simultaneously, improving the question's searchability and organization. While the model's ability to adapt to new domains and generate more specific tags depends on training data and model architecture, this approach offers greater flexibility compared to MLC and sentence matching. However, the potential of SG for tag recommendation remains largely unexplored, raising a key question. **Challenge 1**: How to effectively formulate tag recommendation task to capture correlations between tags?

Existing research on software object tagging relies on titles, descriptions, and code snippets (Li et al., 2023b,a). The reliance on limited content poses the challenge of data sparsity (Djuana et al., 2014; Ifada & Nayak, 2021), hindering the accuracy of tag recommendations for newer or less popular topics (Djuana et al., 2014; Wang et al., 2019c). However, rich discussions within comment threads associated with software objects are a valuable source of information that remains largely untapped. Consider Figure 1, the question title is generic but comments add crucial context. Comments discuss the importance of unit tests, clarify assumptions about the game's rules, and suggest alternative strategies. This information can help in identifying more specific and relevant tags, such as dynamic-programming or tags related to specific game theory concepts. This illustrates how comments can expand the vocabulary used to describe software objects, capturing the diverse ways users discuss concepts and technologies. Furthermore, comments enrich contextual insights by revealing specific use cases, challenges encountered, and alternative solutions. Comments can also surface emerging libraries, frameworks, and tools not explicitly mentioned in posts and disambiguate ambiguous terms. Comments serve as a valuable source of community-driven insights, reflecting the collective knowledge and experience of users. By analyzing comments, tag recommendation systems can leverage this collective intelligence to surface the most appropriate tags. Despite these advantages, the potential of comments in tag recommendation remains largely unexplored. **Challenge 2**: How can we effectively alleviate data sparsity issue to improve tag recommendations for software objects?

While personalized tag recommendation using user information and historical data (Bansal

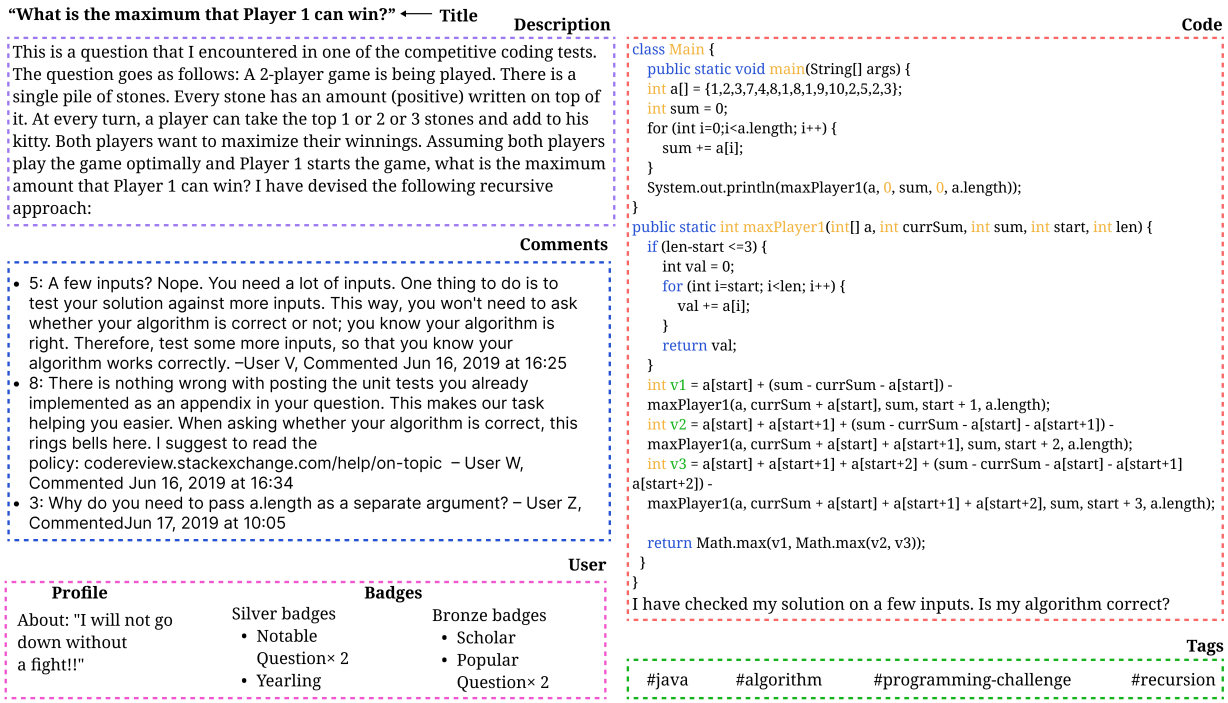


Figure 1: An Example Post from the Code Review Stack Exchange Dataset

et al., 2024a, 2022) has been explored in social media, this approach remains largely untapped for software information sites. Previous work in the domain of software objects tagging has focused primarily on historical posts (Wang et al., 2018; Liu et al., 2023), neglecting the rich insights available in user profiles and badges. However, these signals offer valuable clues about user expertise, interests, and community engagement that can be leveraged for personalized tag recommendation. Consider the user profile description in Figure 1 which hints at his competitive spirit. Badges such as "Notable Question" and "Scholar" suggest his interest in challenging problems and a desire for deeper understanding. The system can use this information and suggest tags related to advanced algorithmic techniques or optimization strategies, catering to the user's apparent interest in high-level problem-solving. Badges[7], earned through contributions or achievements, signify a user's expertise in specific areas while profile descriptions explicitly state a user's interests and domain knowledge. Badges awarded by the community reflect peer recognition and validate a user's knowledge. Mining keywords and phrases from a user's profile description can help in suggesting personalized tags that align with their preferences, job title or educational background (Lipczak et al., 2009). These signals provide social proof and community validation. **Challenge 3:** How to personalize tag recommendations for software objects?

A Multifaceted Approach: Overview of the Proposed Method

In this paper, we introduce a novel tr**AN**sformer-based sequ**N**ce-to-sequence farme-

[7]https://stackoverflow.com/help/badges

work for pers**O**nalized **TA**g recommenda**TION** in software information sites i.e., ANNOTATION. To tackle **Challenge 1**, we devise a sequence-to-sequence framework based on transformer-based encoder-decoder and generate tags sequentially. To address **Challenge 2**, we leverage comment threads associated with the target post since comments provide supplementary knowledge to enrich representation of software objects. To tackle **Challenge 3**, we leverage user information such as profile descriptions and badges to personalize tag suggestions. We employ a transformer model pre-trained for code understanding and software engineering tasks to encode all input features simultaneously, rather than encoding them separately and fusing them later. The simultaneous encoding allows the model to learn complex relationships between different modalities, such as how user expertise might influence the interpretation of comments. The cross-attention mechanism allows the decoder to focus on relevant parts of the encoded input sequence when generating each tag. A generative pre-trained language model serves as the decoder, generating tags sequentially. The autoregressive approach captures dependencies between tags. Experiments conducted on two real-world datasets demonstrate that the proposed model outperforms state-of-the-art methods. Ablation studies corroborate that incorporating user information and associated comments significantly improves the personalization and relevance of tag recommendations.

Key Contributions

Our key contributions are enlisted below.

- We introduce a multifaceted approach for tag recommendation in the software domain that leverages associated comment threads and user information besides textual content to enhance tag relevance and personalization. By leveraging diverse information, our method captures inherent semantics and contextual nuances of software objects.

- We propose a novel sequence generation method for tag recommendation in the software domain, utilizing a transformer-based encoder-decoder architecture to capture tag correlations. This work represents a novel exploration of using a generative pre-trained language model as the decoder, enabling the generation of contextually relevant tag sequences beyond the constraints of predefined candidate tags list.

- Unlike previous works in this domain, we incorporate user metadata to recommend personalized tags for software objects. By leveraging user-centric information, our model can capture individual interests and expertise, generating tags that are contextually relevant and tailored to user's preferences and background.

- To alleviate data sparsity, we enrich the context of software objects by incorporating associated comment threads. This leverages the rich contextual information and community-driven knowledge embedded in comments, generating tag recommendations that align with broader user discussions.

- Experiments conducted on two real-world datasets demonstrate that the proposed model yields significant improvements over state-of-the-art methods in terms of quantitative metrics and qualitative assessment.

The subsequent sections of this paper are structured as follows: Section 2 provides a comprehensive review of related literature. Section 3 delineates the problem definition and its mathematical formulation. The proposed methodology is elucidated in Section 4, followed by a presentation of the experimental evaluations in Section 5. Section 6 concludes the paper with key findings.

2. Related Works

The concept of automatic tag recommendation in software engineering was first proposed by (Al-Kofahi et al., 2010). In this section, we review the evolution of tag recommendation research. Subsequently, we delve into the emergence of sequence generation for tag recommendation, highlighting its origins and potential advantages.

2.1. Tag Recommendation for Software Information Sites

In this section, we explore traditional and neural network-based approaches specifically designed for software information sites.

2.1.1. Traditional Tag Recommendation

TagCombine (Xia et al., 2013) is a composite tag recommendation method that combines three ranking components: multi-label ranking, similarity-based ranking, and tag-term-based ranking. While it was a pioneering approach, its reliance on one-versus-rest Naive Bayes classifiers for multi-label ranking limited its scalability to larger datasets due to the computational burden of training numerous models. EnTagRec (Wang et al., 2014) addressed this scalability issue by combining Bayesian and frequentist inference techniques. It outperformed Tagcombine in recall and precision, demonstrating the effectiveness of considering all tags together in a mixture model. Additionally, EnTagRec incorporated a parts-of-speech (POS) tagger and spreading activation algorithm for refining tag suggestions. Building upon EnTagRec, EnTagRec++ (Wang et al., 2018) further improved the performance of tag recommendation by incorporating the user's past tagging behavioral information and initial tag selections. (Nie et al., 2014) devised an unsupervised approach to recommend tags through two components automatically. The first component constructs an adaptive probabilistic hypergraph integrating QA content, tag-sharing information, and user connections to identify a semantically similar question space and potential tag candidates. The second component heuristically filters tag candidates based on tag informativeness, stability, and question closeness, effectively removing subjective, ambiguous, and generic tags. (Wu et al., 2016) utilized a supervised random walk framework to leverage similarity with similar questions and tags, weighted by tag importance, to calculate question-tag similarity, and address the sparsity issue in question-tag relations. Zhou *et al.* (Zhou et al., 2017) proposed TagMulRec to address the rapid growth of tags by reducing inappropriate and redundant tags. It first constructs indices for software object descriptions, retrieves semantically similar objects to form candidate sets, and then ranks tags within these sets using a multi-classification algorithm.

To date, the progress that has been made in this field has been restricted to phrase selection from limited candidates or word-level hashtag discovery using topic models. Unlike previous works that consider hashtags inseparable, our work annotates hashtags to software objects with a novel sequence generation framework by viewing the recommended hashtags as a short sequence of words.

2.1.2. Neural Network-based Tag Recommendation

(Zhou et al., 2019) investigated the effectiveness of deep learning versus traditional approaches for tag recommendation in software information sites. The authors implemented four deep learning approaches (TagCNN, TagRNN, TagHAN, TagRCNN) and compared them with three traditional approaches (EnTagRec, TagMulRec, FastTagRec). Extensive experiments on over 11 million software objects revealed that TagCNN and TagRCNN outperformed traditional methods, while TagRNN and TagHAN did not. (Li et al., 2019) proposed TagDeepRec, a multi-label ranking method for tag recommendation that leverages an attention-based Bi-LSTM to learn semantic relationships between software objects and their associated tags. The model utilizes word2vec embeddings for text representations and outputs confidence probabilities for each candidate tag, facilitating the recommendation of top-k most relevant tags. (Wang et al., 2019a) proposed SOTagRec, a tag recommendation method combining a Convolutional Neural Network (CNN) and Collaborative Filtering (CF). CNN leverages deep semantic features from question context, while CF relies on traditional bag-of-words representation. A combined model integrates both probabilities to calculate the final probability for each tag. (Zheng et al., 2020) proposed a deep semantic matching framework for tag recommendation dubbed TagMatchRec. It utilizes attentive deep supervision to capture multi-granularity semantic information. It employs a multi-layer architecture with attention mechanism to learn features at different abstraction levels, providing a pairwise matching solution that considers tag semantics. (Ashvanth et al., 2022) proposed a hybrid tag recommender dubbed KMetaTagger that employed Term Frequency-Inverse Document Frequency (TF-IDF) for identifying informative terms and Latent Semantic Indexing (LSI) for topic modeling. It utilizes a heterogeneous information network and quantifies metadata generation through exponentially aggregating real-world knowledge, which is then classified using Gated Recurrent Units (GRU). The Color Harmony algorithm is used to refine tag recommendations from an initial solution set. (Wang et al., 2023) incorporated commonsense knowledge to enhance semantic understanding of text and leverages interclass correlations to address the long-tail effect of tags. The framework can seamlessly integrate with existing classification-based models (FastText, XML-CNN, AttentionXML, BERT) without significant parameter increase, enhancing performance and mitigating the long-tail effect. (Liu et al., 2023) proposed MLP4STR, a novel sequential tag recommendation algorithm that addresses the sequential nature of tag recommendation and models dynamic user preferences based on historical post and tag information. It leverages BERT for textual feature extraction and aligns text and tag information through a pure MLP-based sequence modeling approach. This model captures user preferences across feature dimensions to fuse them with current post features for accurate tag recommendation. The method utilizes a cross-feature-aligned pure MLP to learn user sequences. (Lu et al., 2024)

enhanced post representations by retrieving information from external knowledge sources and leverage cross-modal context-aware attention to use the main modality (description) for targeted feature extraction across submodalities (title and code). It employs a gate mechanism for fine-grained feature selection during fusion before recommending tags. (Hamidi Rad et al., 2024) proposed an interactive approach utilizing user feedback to improve tag suggestions. The authors used contextualized embeddings for questions and tags generated by a transformer-based language model. User feedback, collected through an interactive interface, is incorporated into the model as positive and negative signals for iterative refinement of tag recommendations. (Xu et al., 2024) constructed a Tri-Relational Question-Tag Graph to model question-tag relationships and learn informative node features for questions and tags. A specific Multi-Faceted Question GNN is designed to capture diverse semantics from relevant tags.

2.2. Genesis of Sequence Generation for Tag Recommendation

Automatically annotating tags for posts on social media and software information sites has garnered significant research attention. Existing approaches can be broadly categorized into extractive, classification, and generative methods.

Extractive methods (Zhang et al., 2016, 2018b) focus on identifying important words within the post itself but struggle to recommend relevant hashtags that are not directly stated in the text. Classification methods (Huang et al., 2016; Zhang et al., 2017; Kavuk & Tosun, 2020) rely on a predefined list of candidate hashtags, limiting their adaptability to the dynamic nature of social media. These methods typically employ either binary classifiers for each potential tag or multi-tag classifier algorithms. Most existing approaches (Xia et al., 2013; Li et al., 2019; Sahu et al., 2019; He et al., 2022; Wang et al., 2023) for tag recommendation typically frame the task as Multi-Label Classification (MLC) problem. While MLC models outperform topic and extractive models, they generally underperform Sequence Generation (SG) models. A key limitation of MLC-based approaches is their reliance on a predefined candidate list from which hashtags are recommended that often yields suboptimal results due to the imbalanced and dynamic nature of the hashtag space. Furthermore, these models frequently fail to capture the inherent correlations among hashtags, where the presence of one hashtag can significantly influence the likelihood of others. This limitation is further exacerbated by the dynamic nature of hashtags, as user-assigned hashtags appear neither in target posts nor in the candidate list. The vast and constantly evolving vocabulary of hashtags, stemming from the freedom afforded to users on software information sites and microblogging platforms, coupled with the rapid creation of new hashtags due to the diverse and evolving nature of topics, makes it challenging for classification models to capture the complex relationships in hashtag usage effectively. Additionally, the constant emergence of new trends in hashtag usage further challenges these models. Despite their relative success, the inherent limitations of classification methods necessitate exploring alternative approaches that can better adapt to the dynamic and evolving nature of hashtags.

MLC-based approaches, while effective for individual tag prediction, often overlook the semantic relationships between tags, leading to suboptimal recommendations. Some ap-

proaches (Zheng et al., 2020; Li et al., 2023b) have attempted to address this by formulating the task as sentence matching to consider the semantic similarity between software objects and tags. However, this formulation may not fully capture the complex interdependencies between multiple tags and the nuanced meaning of the underlying text.

Generative methods (Wang et al., 2019b,c) provide an adaptable approach by generating hashtags from scratch, eliminating the need for a candidate list. (Tang et al., 2019) introduced a unified encoder-decoder framework to generate tags sequentially for textual content found on information sites. The encoder, utilizing RNNs with attention, captures nuanced meanings in the text. The decoder, employing a prediction path, identifies relationships between tags. A shared embedding layer unifies the representation of text and tags. Finally, an indicator function assesses the probability of using words directly from the text as tags, effectively addressing the issue of content-tag overlap. However, previous generative models have often overlooked the broader context of relevant posts and conversations, which can significantly influence tag usage, especially in dynamic software information sites. Our proposed model addresses this limitation by explicitly incorporating associated comments into a simplified yet effective architecture, potentially improving tag recommendation accuracy and relevance.

3. Problem Definition

To formally define this problem, consider a set of posts on Software Question and Answering (SQA) sites denoted by $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$. Each post ($s_i$) is characterized by a tuple $(tle_i, des_i, cd_i, cm_i, u_i)$ representing its title, description, code snippets, comments, and user metadata, respectively. The user metadata for any user (u_i) encompasses user ID (u_{id_i}), user profile description (u_{des_i}), and badges they have earned (u_{bdg_i}). In addition, we define a vocabulary of candidate tags $\mathcal{V}_T = \{t_1, t_2, \ldots, t_L\}$. The objective is to learn a function $f(.)$ that maps the post's content and user metadata to generate a sequence of relevant tags:

$$f : \mathcal{TLE} \times \mathcal{DES} \times \mathcal{CD} \times \mathcal{CM} \times \mathcal{UID} \times \mathcal{UP} \times \mathcal{UB} \rightarrow \mathcal{T}^* \tag{1}$$

where $\mathcal{TLE}, \mathcal{DES}$, $\mathcal{CD}$, $\mathcal{CM}$, $\mathcal{UID}$, $\mathcal{UP}$, $\mathcal{UB}$ represent the spaces of titles, descriptions, code snippets, comments, user IDs, profile descriptions, and badges, respectively. The output space $\mathcal{T}^*$ is the set of all possible tag sequences that can be formed from $\mathcal{V}_\mathcal{T}$. This function $f(.)$ should not only capture the essence of the post's content but also leverage co-occurrence patterns and user-specific information to predict relevant tags accurately. The system should be able to suggest tags not explicitly present in the predefined vocabulary $\mathcal{V}_T$, allowing for the discovery of novel and meaningful tags. To achieve this, we frame the problem as an optimization problem seeking the function $f(.)$ that maximizes the conditional probability of the ground-truth tag sequence T_i given the input features of each post:

$$\max_f \sum_{i=1}^{N} \log P(\hat{T}_i | tle_i, des_i, cd_i, cm_i, u_{id_i}, u_{bdg_i}, u_{des_i}) \tag{2}$$

where $\hat{T}_i$ is the model's generated tag sequence for post (s_i). This formulation treats tag recommendation as a sequence generation task, aiming to predict the most likely sequence of tags that describe a given SQA post.

4. Methodology

This section delineates the methodology employed in this study. A high-level overview of our proposed method is shown in Figure 2. We further elaborate on different input feature types as follows:

4.0.1. Content-based Features

Content-based features are derived directly from the software object itself, encompassing title, description, and code as detailed below.

Title. The title of a software object is a concise, informative label that encapsulates its content. It serves as a distilled summary and the first point of contact for users, providing critical signals about the core focus.

Description. The textual description elaborates on title of the software object, providing explanations, background information, error messages, desired outcomes, and potential use cases. This richer context is particularly useful for identifying less prominent or niche tags that might not be immediately evident from the title alone.

Code Snippets. Code snippets are a defining feature of software objects, offering concrete examples of problems, solutions, or implementations. These snippets, while primarily written in diverse programming languages, contain rich semantic and structural information that can significantly enhance tag recommendations. Despite their potential value, existing approaches often neglect (Li et al., 2020) or oversimplify code snippet information (Gharibi et al., 2021; Zhou et al., 2017). However, the prevalence and centrality of code snippets necessitate developing effective methods for incorporating this information into the recommendation process. Our approach recognizes the unique characteristics of code snippets as structured language, distinct from natural language.

4.0.2. Interaction-based Features

Comments surrounding a software object offer valuable contextual information, explicitly mentioning relevant topics, technologies, or issues. These can clarify ambiguities in the original post, highlight potential solutions, and reflect sentiments and interests of the community. Each comment is associated with a score that reflects its importance, derived from factors such as user reputation, upvotes, or comment length. We sort comments by descending order of associated scores to prioritize the most relevant ones. Let $cm_i = [cm_{i1}, cm_{i1}, \ldots, cm_{iA},]$ be the set of comments for a software object s_i, where A_i is the number of comments. After sorting these comments based on their scores, the concatenated comment text $cm_{i_{concat}}$ is given by:

$$cm_{i_{concat}} = cm_{i1} \oplus cm_{i2} \oplus \cdots \oplus cm_{iA_i} \tag{3}$$

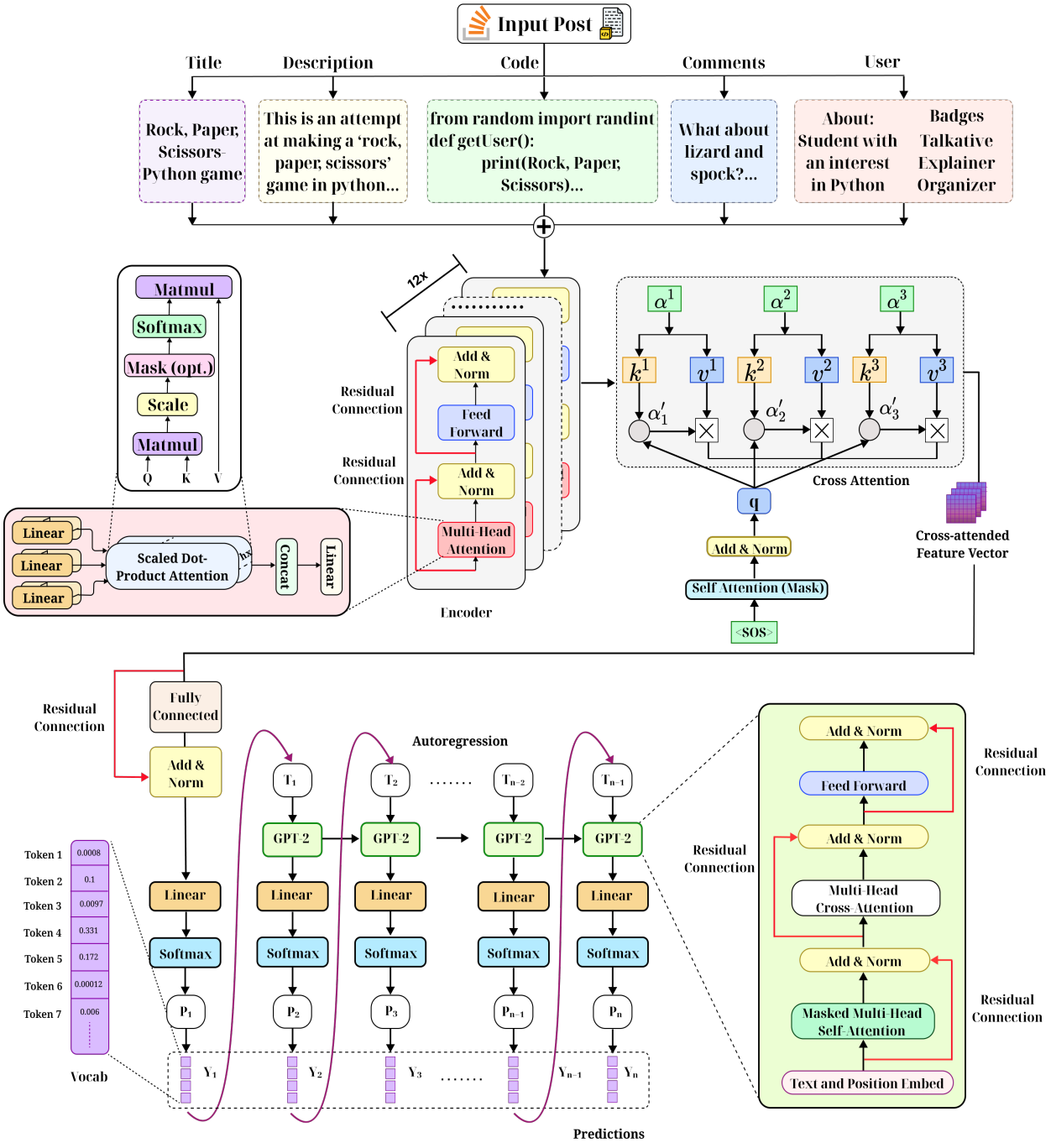


Figure 2: Overview of System Architecture: The input to the system is a software object that comprises title, description, code snippet, associated comments, user profile, user ID, and badges assigned to the corresponding user. This input is encoded into a feature vector representation and fed to a decoder consisting of GPT-2 blocks. The decoder generates a probability distribution over its vocabulary to select the most relevant tags. The final output is a recommended sequence of tags for the input software object.

Here, $\oplus$ denotes concatenation. This aggregated text provides insights beyond the original content. Analyzing it allows us to understand user intent and leverage community knowledge to improve tag recommendation. Incorporating this text enables our model to learn from collective tagging behavior, leading to more accurate and relevant recommendations.

4.0.3. User-centric Features

The author of a software object and their associated metadata provide valuable insights into their expertise and areas of interest.

User ID. The user ID, a unique identifier for each user enables tracking of past contributions, such as comments, ratings, or edits

User Profile Description. The user profile, including the "About Me" section, can reveal self-declared interests, skills, and experiences.

Badges. Badges on software information sites indicate a user's expertise and contributions within specific domains. Awarded for various achievements, such as asking well-received questions, providing accepted answers, or actively participating in the community, badges provide valuable insights into a user's knowledge and interests, aiding in generating relevant tags. For example, a user with a "C#" badge will likely post about C# development. By analyzing badges, the system can prioritize tags associated with their expertise and give more weight to tags suggested by these experienced users. Incorporating this knowledge allows the system to refine tag suggestions based on the user's demonstrated interests.

To harness the valuable information encoded in user badges, we first preprocess the badge data to ensure a concise and informative representation. Specifically, we deduplicate the list of badges associated with each user, retaining only the unique badges earned. Let $u_{bdg_i} = [b_{i1}, b_{i2}, \ldots, b_{iB_i}]$ be the initial set of badges for user u_i associated with the post s_i, where B_i is the total number of badges earned by that user. After deduplication, we obtain a refined set of unique badges. We then concatenate these unique badges into a single text string, $u_{bdgi_{concat}}$ to create a unified representation of the user's achievements and expertise given by:

$$u_{bdg_{i_{concat}}} = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{iB_i} \tag{4}$$

This consolidated badge representation, along with other user-centric features such as user ID and profile description serves as a comprehensive input to our model. By incorporating this diverse set of features, we empower the model to consider not only the content of the software object but also the expertise and interests of the user who created it, ultimately leading to more accurate and personalized tag recommendations.

4.1. Encoder

We formalize the task of generating relevant tags as a sequence generation problem using a seq2seq encoder-decoder framework. For the post encoder, we utilize CodeBERT (Feng et al., 2020), a pre-trained transformer model specifically designed for understanding and representing code. Unlike traditional language models trained solely on natural

language text, CodeBERT is trained on code and natural language pairs, enabling it to capture the unique characteristics and semantic relationships within code and its associated documentation. This makes it well-suited for processing the diverse modalities present in software objects, including code snippets, natural language descriptions, and user-related information. CodeBERT processes content-based information, conversation context, and user-centric features, transforming them into numerical representations. These features are first embedded into a high-dimensional vector space using a learned embedding function:

$$E_{s_i} = e(x_{s_i}) \tag{5}$$

where, x_{s_i} represents raw features (text, numerical values) of the i^{th} post (s_i) and $e(.)$ is the embedding function that maps the features into a continuous vector space suitable for CodeBERT.

To accommodate CodeBERT' maximum input sequence length of 512 tokens, we truncate or pad input sequences. This ensures efficient processing while preserving the most salient information. All input sequences (title, text, code, comments, user profile description, badges, and user ID) are concatenated and fed into CodeBERT:

$$\begin{aligned} S = [CLS] + tle_i + [SEP] + des_i + [SEP] + cd_i \\ + cm_{i_{concat}} + [SEP] + u_{id_i} + [SEP] \\ + u_{des_i} + [SEP] + u_{bdg_{i_{concat}}} + [SEP] \end{aligned} \tag{6}$$

where $[CLS]$ and $[SEP]$ are special tokens marking the beginning and separation of segments.

CodeBERT processes the input sequence S token by token, considering the bidirectional context. It leverages multiple layers of self-attention, allowing it to capture complex relationships and dependencies. The output is a sequence of hidden states:

$$H = [h_1, h_2, \ldots, h_L] \tag{7}$$

where L is the length of input sequence S, and each hidden state h_j captures rich contextual information of the corresponding token in S. The final hidden state h_L serves as a condensed representation of the software object, incorporating information from all input features. This representation is passed to the decoder for tag generation.

4.2. Decoder's Cross- Attention Mechanism

Cross-attention mechanism allows the decoder to focus on relevant parts of the encoder's output (hidden states H) at each decoding step t. This focus is guided by a query vector q_t derived from the decoder's current state and key/value vectors $K = [k_1, k_2, \ldots, k_L]$ and $V = [v_1, v_2, \ldots, v_L]$ derived from the encoder's output. Attention scores are calculated using scaled dot-product attention:

$$A_t = softmax(\frac{q_t K^T}{\sqrt{d_k}}) \tag{8}$$

where d_k is the dimension of key vectors. The query vector q_t represents what the decoder looks for at time step t. Key vectors K determine the similarity between the query and each position in the encoder's output. The value vector V contains information the decoder will focus on based on attention scores. The scaled dot product attention calculates the similarity between query and key vectors for each position in the encoder's output. The scaling factor $\sqrt{d_k}$ prevents attention scores from becoming too large, which could lead to instability during training. The softmax function normalizes attention scores into a probability distribution, where each score represents the relative importance of attending to the corresponding position in the encoder's output. These scores are used to compute a weighted sum of value vectors, producing a context vector c_t:

$$c_t = A_t V \tag{9}$$

Here, context vector c_t is a weighted sum of value vectors, with weights given by attention scores. It allows decoder to generate next tag in the sequence. By dynamically attending to different parts of the input, the decoder generates tags that consider the specific information present in the software object.

4.3. Decoder

We employ Generative Pretrained Language Model-2 (GPT-2) as decoder. GPT-2, with its extensive training on diverse text corpora, has demonstrated exceptional capabilities in generating contextually appropriate sequences. This makes it well-suited for tag generation, where understanding input text is crucial for producing meaningful tags. We tokenize tags using the GPT-2 tokenizer. Let the set of unique tags be denoted as $\mathcal{V}_T$. For each tag $t_i \in \mathcal{V}_T$, the tokenizer maps it to a corresponding token ID, denoted as $token\_id(t_i)$, through a vocabulary lookup:

$$token\_id(t_i) = GPT2\_Tokenizer.encode(t_i) \tag{10}$$

This process converts each tag into a numerical representation. The sequence of token IDs for a post's ground-truth tags $T = [t_1, t_2, \ldots, t_T]$ is used as the target sequence during training. To initiate generation, GPT-2 receives the final hidden state h_L from CodeBERT and beginning-of-sentence (BOS) token. BOS token denoted by $\langle s \rangle$ signals the start of tag sequence and serves as input for the first time step (t=1). GPT-2 uses masked self-attention, where it can only attend to tokens that have already been generated, preventing it from looking ahead in the output sequence.

In general, SG methods utilize the previous time step's output y_{t-1} as the model's input at the current time step t. However, during the early training phases, the model's predictions can be inaccurate. A series of incorrect predictions can negatively impact the model's learning process, leading to slower convergence and instability. To address this issue, we employ a teacher-forcing strategy to accelerate and stabilize the training of our sequence generation model. At each time step t, instead of using the model's predicted tag (y_{t-1}) from the previous step, we feed the ground-truth tag from the training data as input. This helps guide the model towards generating more accurate and relevant tag sequences. Let

the ground-truth tag sequence for a post as $T = [*t_1, *t_2, \ldots, t_T]$. Mathematically, during teacher-forcing, the input to GPT-2 at time step t is:

$$Input_t = \begin{cases} < s > & \text{if } t = 1 \\ t_t & \text{if } 1 < t \leq T \end{cases} \tag{11}$$

At each subsequent time step t, GPT-2 predicts the next tag (y_t) in the sequence, conditioned on previously generated tags (Y_t) and CodeBERT-encoded contextual information (h_L). For the first tag ($t = 1$), Y_t is the BOS token $\langle s \rangle$. Mathematically, the probability distribution over the vocabulary of candidate tags V_T at a time step t is given by:

$$P(y_t|Y_{<t}, h_L) = softmax(W_o * (W_h * h_L + W_y * Y_{<t} + b)) \tag{12}$$

Here, W_o, W_h, W_y are learnable weight matrices, b is a bias vector.

4.4. Loss Function

To optimize model's tag generation capabilities, we utilize the sparse categorical cross-entropy loss function. Let N be the number of training samples. For each training sample i and time step t, the loss function is given by:

$$L_i(t) = -\log P(y_t = token\_id(tag_{it})|Y_{<t}, h_L) \tag{13}$$

Here, tag_{it} represents the ground-truth tag at time step t for i^{th} post. The overall loss for a training sample i is the sum of losses at each time step, and the average loss over all training samples is used to update the model's parameters.

4.5. Decoding Strategy

During inference, we use the model's own predictions (y_{t-1}) as input at each step and employ a greedy search decoding strategy. At each time step t, the model selects the most probable tag y_t from the probability distribution $P(y_t|Y_t, h_L)$ as its prediction. Mathematically, this can be expressed as:

$$y_t = argmax_{v \in V_T} P(v|Y_{<t}, h_L) \tag{14}$$

The predicted tag is then appended to the sequence, and the process repeats until End-of-sequence (EOS) token is generated or the maximum sequence length is reached. The autoregressive nature of GPT-2, combined with teacher forcing during training and greedy search during inference, ensures that generated tags form a meaningful sequence. The model learns to predict the most relevant tag based on post context and previously generated tags. The resulting sequence of tags $Y = [y_1, y_2, \ldots, y_T]$ represents the model's prediction for the most relevant tags associated with the software object s_i.

5. Experimental Evaluations

In this section, we outline the experimental setup and then analyze the results obtained.

5.1. Experimental Setup

In this section, we shed light on datasets used for experimentation, followed by methods under comparison, and evaluation metrics to assess their performance.

5.1.1. Datasets

To evaluate the effectiveness of our proposed model dubbed ANNOTATION, we utilize two publicly available datasets commonly employed in previous research (Li et al., 2020; Liu et al., 2018): CodeReview and Stack Overflow. The datasets and code are available at https://github.com/shubhi1207/ANNOTATION.

Table 1: Dataset Statistics

Parameter	CodeReview	StackOverflow
Number of posts	13111	13295
Number of tags	402	549
Number of users	7306	11516
Avg no. of tags/post	3	2.3
Avg no. of comments/post	3.2	3.5
Max no. of tags/post	5	5

CodeReview. CodeReview is a question-and-answer site for peer programmer code reviews, providing feedback on best practices, design, security, performance, and correctness. We downloaded the CodeReview data dump from the Stack Exchange Data Download archive[8]. The downloaded data included badges, comments, post history, post links, posts, tags, users, and votes, all formatted in Extensible Markup Language (XML).

Stack Overflow. Stack Overflow (Anderson et al., 2012; Srba & Bielikova, 2016) is a widely used question-and-answer website for programmers. We constructed our training and test sets from a quarterly data dump of Stack Overflow, encompassing user-generated content from August 27, 2009, to October 1, 2015. This data, obtained from Stack Exchange Data Download archive, includes posts, tags, comments, and user information, consistent with the CodeReview dataset.

Data Preprocessing. To ensure data quality and consistency, both CodeReview and Stack Overflow datasets underwent preprocessing. This involved extracting code snippets from textual descriptions, cleaning posts by removing those with missing information and extraneous elements like HTML tags, and filtering out rare tags with a frequency below 10 to prevent overfitting. Following previous research (Li et al., 2020; Liu et al., 2018), we restricted the CodeReview dataset to posts created before December 31, 2016, for consistency and comparability. Table 1 presents descriptive statistics of both datasets after preprocessing.

[8]https://archive.org/download/stackexchange/

5.1.2. Compared Methods

We evaluate the effectiveness of our proposed method in comparison to several state-of-the-art approaches for automatic question tagging.

- *Fast Tag Recommendation for Software Information Sites (FastTagRec)* (Liu et al., 2018): FastTagRec is a scalable, automated tag recommendation method. It uses a neural network to infer tags for new software objects based on existing data, learning from descriptions and associated tags. It employs a single-hidden layer neural network with shared parameters and incorporates a bag of n-grams with a hashing trick for efficient handling of local word order information.

- *Tag recommendation with Deep learning and Collaborative filtering (TagDC)* (Li et al., 2020): TagDC tackles tag recommendation challenges such as synonymity and explosion. It combines deep learning and collaborative filtering to improve tag accuracy. TagDC-DL, a word-learning enhanced CNN capsule module, uses LSTM to build a multi-label classifier, analyzing the semantic meaning within software object descriptions. TagDC-CF leverages collaborative filtering to identify similar objects and refine tag suggestions.

- *An Integral Tag Recommendation Model for Textual Content (iTAG)* (Tang et al., 2019): The authors proposed a novel encoder-decoder framework that integrates sequential text modeling, tag correlation, and content-tag overlap. iTAG utilizes RNNs with attention for semantic encoding, a prediction path in the decoder to capture tag relationships, and a shared embedding layer with an indicator function to address vocabulary overlap.

- *Attention-based Multimodal Neural Network (AMNN)* (Yang et al., 2020): The authors reformulated tag recommendation as a sequence generation task using a sequence-to-sequence encoder-decoder architecture. The encoder extracts visual features using CNN and textual features using Bi-LSTM from microblogs, with attention mechanisms to identify salient information in each modality. These representations are fused and fed into GRU decoder, which generates a tag sequence based on predicted probabilities.

- *Pre-Trained Models for Stack Overflow Tag Recommendation (PTM4TAG)* (He et al., 2022): It conceptualizes tag recommendation as a multi-label classification problem. It leverages pre-trained language models with a triplet architecture to model title, description, and code components independently. CodeBERT demonstrated superior performance compared to other pre-trained models and a state-of-the-art convolutional neural network-based approach. An ablation study revealed the significance of post titles and showed optimal performance observed when all post components (title, description, and code) are considered together.

- *Code-Enhanced fine-grained semantic matching for Tag Recommendation (CETR)* (Li et al., 2023a): The researchers employed a multi-stage approach to model semantic relationships between code snippets, text descriptions, and tags. Code snippets are

encoded using GraphCodeBERT (Guo et al., 2020) to capture structural information, while text descriptions are encoded separately. The model uses a self-attention mechanism to learn semantic interaction between these representations, and a multi-level feature integration component to capture semantic information at various granularities. Finally, a pairwise sentence-matching mechanism predicts tag relevance.

- *Code-mixed Deep Representation learning via dual-interactive fusion for Tag Recommendation (CDR4TAG)* (Li et al., 2023b): CDR4Tag uses deep learning to fuse semantic relationships between software objects and tags. It utilizes GraphCodeBERT for code understanding and combines self-attention and multi-layer attention mechanisms to extract richer semantic information. Two specialized modules learn the similarity between tags and the object; and refine textual and code representations. By framing tag recommendation as a sentence-matching problem and leveraging code snippets, CDR4Tag enhances the representation of software objects and calculates matching probabilities between object-tag pairs.

5.1.3. Evaluation Metrics

For a test set ($\mathcal{TO}$) composed of n software objects (o_i), where $1 \leq i \leq n$, top-k recommended tags ($\hat{T}_{o_i}$) are evaluated against actual tags (T_{o_i}) of each software object (o_i). To generate top-k recommended tags, we calculate the following evaluation metrics as detailed below.

- Hit rate: Proportion of software objects for which at least one ground-truth tag is present within predicted tags.

$$\text{Hit rate} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{[T_{o_i} \cap \hat{T}_{o_i} \neq \emptyset]} \tag{15}$$

  Here, $\mathbb{1}$ is the indicator function.

- Precision: Proportion of relevant tags among predicted tags, averaged across all objects.

$$Precision = \frac{1}{n}\sum_{i=1}^{n} \frac{|T_{o_i} \cap \hat{T}_{o_i}|}{|\hat{T}_{o_i}|} \tag{16}$$

- Recall: Proportion of ground-truth tags present within predicted tags, averaged across all objects.

$$Recall = \frac{1}{n}\sum_{i=1}^{n} \frac{|T_{o_i} \cap \hat{T}_{o_i}|}{|T_{o_i}|} \tag{17}$$

- F1-score: Harmonic mean of precision and recall, providing a balanced measure of the model's performance.

$$F1 - score = \frac{2 \cdot P \cdot R}{P + R} \tag{18}$$

  F1@k score is a balanced measure that combines precision and recall into a single metric.

- Recall-Oriented Understudy for Gisting Evaluation (ROUGE): Evaluates n-gram overlap between generated tag sequence and ground-truth tag sequence

$$\text{ROUGE-N} = \frac{|N_{match}|}{|N_{ref}|} \tag{19}$$

where N_{match} is the number of matching N-grams between generated and ground-truth tags, and N_{ref} is the total number of N-grams in ground-truth tags, ground-truth tag sequence is the reference, and generated tag sequence is the evaluated output.

5.2. Experimental Results

To analyze our proposed model, we conducted quantitative analysis, ablation studies, qualitative analysis, and parameter sensitivity studies.

5.2.1. Quantitative Analysis

In this section, we discuss the performance comparison of our proposed model with various state-of-the-art methods on two different datasets.

Table 2: Effectiveness Comparison against SOTA on Code Review Dataset

Methods	**Hit rate**	**Precision**	**Recall**	**F1-score**	**ROUGE-1**	**ROUGE-L**
Classification/ Retrieval						
FastTagRec (Liu et al., 2018)	0.342	0.116	0.137	0.126	0.169	0.169
TagDC (Li et al., 2020)	0.421	0.141	0.166	0.153	0.206	0.205
PTM4TAG (He et al., 2022)	0.718	0.269	0.327	0.296	0.294	0.286
Sentence Matching						
CETR (He et al., 2022)	0.573	0.197	0.235	0.215	0.246	0.245
CDR4TAG (Li et al., 2023b)	0.665	0.256	0.301	0.276	0.282	0.278
Generation						
iTAG (Tang et al., 2019)	0.195	0.0653	0.087	0.075	0.177	0.178
AMNN (Yang et al., 2020)	0.309	0.166	0.197	0.180	0.133	0.133
ANNOTATION (Ours)	**0.920**	**0.487**	**0.590**	**0.534**	**0.544**	**0.540**

Effectiveness Comparison on Code Review Dataset. Table 2 presents the effectiveness comparison of our proposed method, ANNOTATION against state-of-the-art methods in terms of Hit rate@3, Recall@3, Precision@3 and F1-score@3 on the Code Review Stack Exchange dataset. The highlighted values indicate the best results. The proposed model, ANNOTATION significantly outperforms FastTagRec in tag recommendation for software objects, achieving relative improvements of 57.8%, 37.1%, 45.3%, 40.8%, 37.5%, and 37.1% in HR@3, P@3, R@3, F1@3, ROUGE-1 and ROUGE-L respectively. This substantial advantage stems from PM's ability to comprehend tag semantics and generate contextually relevant recommendations through its CodeBERT encoder and GPT-2 decoder architecture, leveraging the extensive vocabulary of GPT-2 to capture complex tag dependencies and their semantic interplay. In contrast, FastTagRec relies on a less sophisticated multi-label classification

approach, using a single-hidden layer neural network with word rank constraints and shared parameters to handle large tag output spaces. While FastTagRec incorporates a bag of n-grams and employs the hashing trick to capture partial word order information and improve efficiency, it fundamentally treats tags as independent labels, neglecting their semantic relationships and limiting its ability to generate novel tags beyond the pre-defined vocabulary.

ANNOTATION beats TagDC by 49.9%, 34.6%, 42.4%, 38.1%, 33.8%, and 33.5% in hit rate, precision, recall, F1-score, ROUGE-1 and ROUGE-L. This performance gap can be attributed to ANNOTATION leveraging a broader range of features, including the title, body, code snippets, comments, and user information (profile, badges, ID), providing a richer context for tag recommendation. In contrast, TagDC primarily focuses on textual features and collaborative filtering, potentially missing valuable information embedded in other modalities. ANNOTATION employs an encoder-decoder architecture with CodeBERT and GPT-2, generating tags sequentially and considering their interdependencies. This allows for more nuanced and contextually relevant recommendations. Conversely, TagDC-DL utilizes a multi-label classification approach, treating tags independently and potentially overlooking their semantic relationships. ANNOTATION's GPT-2 decoder enables tag generation from an extensive vocabulary, not limited to the training data. This allows ANNOTATION to suggest novel and creative tags that might not have been explicitly encountered during training. In contrast, TagDC is constrained by a pre-defined vocabulary based on the tags present in the training dataset. ANNOTATION is an end-to-end model that directly learns to generate tag sequences from the input features. TagDC, on the other hand, is a composite model where the collaborative filtering module post-processes the output of the deep learning module. This two-step approach might introduce additional errors and limitations compared to ANNOTATION's integrated learning process.

ANNOTATION demonstrates superior performance compared to AMNN across multiple evaluation metrics (HR@3: 61.1%, P@3: 32.1%, R@3: 39.3%, F1@3: 35.4%, ROUGE-1@3: 41.1% ROUGE-L@3: 40.7%). This suggests that ANNOTATION is more adept at capturing top-ranking relevant tags and modeling complex interactions between different modalities. While AMNN employs a sequence generation approach with BiLSTM as an encoder, self-attention, and GRU as a decoder, it is limited to the vocabulary of the training data. ANNOTATION's utilization of GPT-2 as a decoder gives it an edge over AMNN by generating novel tags and expanding its vocabulary beyond pre-defined sets.

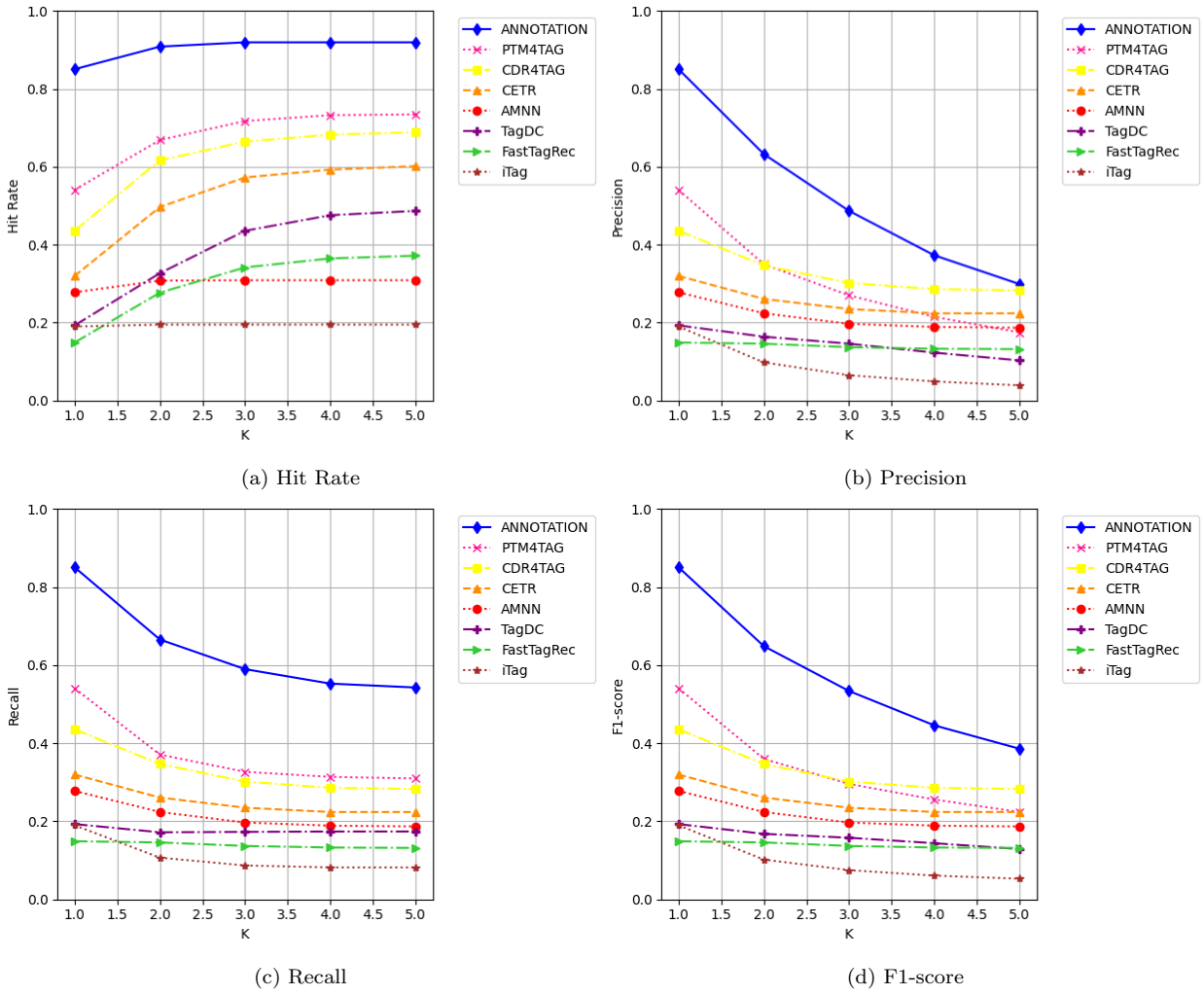
ANNOTATION demonstrates superior performance compared to PTM4TAG across all evaluation metrics (HR@3: 20.0%, P@3: 21.6%, R@3: 26.2%, F1@3: 23.7%, ROUGE-1@3: 24.0%, ROUGE-L@3: 25.4%). While both models leverage pre-trained language models, ANNOTATION incorporates a broader range of features, including comments and user metadata (profile description, badges). Including additional modalities enhances ANNOTATION's understanding of user preferences and contextual information, leading to personalized and accurate tag recommendations. Furthermore, ANNOTATION's utilization of GPT-2 as a decoder enables it to generate tags from an extensive vocabulary, surpassing the limitations imposed by PTM4TAG's reliance on a fixed set of pre-defined tags.

ANNOTATION demonstrates a consistent improvement over CETR across all evaluation metrics (HR@3: 34.7%, P@3: 29.0%, R@3: 35.5%, F1@3: 31.9%, ROUGE-1@3: 29.8%,

ROUGE-L@3: 29.5%). This enhanced performance can be attributed to ANNOTATION's broader input space, incorporating comments and user metadata alongside content and code features. While CETR focuses on code enhancement, it overlooks valuable semantic information embedded within comments and user profiles, potentially limiting its ability to generate contextually relevant tag recommendations. Furthermore, ANNOTATION leverages GPT-2's extensive vocabulary to generate novel tags not present in the training data, providing a significant advantage over CETR, which is constrained by the vocabulary of the dataset. This open vocabulary approach, combined with the integration of diverse information sources, allows ANNOTATION to achieve a more comprehensive understanding of both the software object and the user's needs, resulting in superior tag recommendation performance compared to CETR.

ANNOTATION significantly outperforms CDR4TAG in tag recommendation tasks, achieving improvements across all evaluation metrics (HR@3: 25.5%, P@3: 23.1%, R@3: 28.9%, F1@3: 25.8%, ROUGE-1@3: 26.2%, ROUGE-L@3: 26.2%). This superior performance is attributed to key differences in architecture and information utilization. ANNOTATION leverages CodeBERT to encode diverse features, including content, code, comments, and user information, capturing a broader context than CDR4TAG's focus on code and text using GraphCodeBERT and mixed attention networks. This enables ANNOTATION better to understand the intent and nuances of software objects. ANNOTATION incorporates user-centric information such as profile descriptions and badges, tailoring tag suggestions to individual expertise and interests, unlike CDR4TAG which lacks this personalization aspect. ANNOTATION utilizes GPT-2 for sequence generation, allowing it to propose novel tags not present in the training data, whereas CDR4TAG's sentence-matching approach restricts it to a fixed vocabulary. While CDR4TAG fuses text and code representations, ANNOTATION integrates comments and user information, enabling a more comprehensive understanding of the software object and the user's intent. This broader context results in more accurate and contextually relevant tag recommendations.

ANNOTATION far exceeds iTAG across all evaluation metrics (HR@3: 72.5%, P@3: 42.17%, R@3: 50.3%, F1@3: 45.9%, ROUGE-1@3: 0.367%, ROUGE-L@3: 36.2%). Even though, iTAG incorporates a sequence-generation approach with GRU as both the encoder and decoder, and utilizes a shared embedding layer between these components, it doesn't exploit the semantic relation between the tags and the post content, effectively. On the other hand, ANNOTATION incorporates advanced encoder-decoder framework based on the transformers architecture, making it more effective in understanding the semantic relation between the content and tags and hence, producing more topic-relevant tags. Figure 3 illustrates the performance of various tag recommendation methods across different evaluation metrics. The x-axis shows the number of recommended hashtags, ranging from 1 to 5 and the y-axis represents the corresponding performance values. The results demonstrate that ANNOTATION consistently outperforms state-of-the-art methods across all metrics, regardless of the number of tags recommended. Moreover, the increasing performance gap between ANNOTATION and state-of-the-art methods further emphasizes the significant advancements achieved by our proposed framework i.e., ANNOTATION. These empirical findings provide compelling evidence for the superiority and efficacy of ANNOTATION in



(a) Hit Rate

(b) Precision

(c) Recall

(d) F1-score

Figure 3: Effectiveness Comparison Curves on Code Review Dataset

the realm of tag recommendation in software information sites.

Effectiveness Comparison on Stack Overflow. To investigate the generalizability of ANNOTATION in recommending tags for different software information sites, we conducted experiments on another dataset sourced from Stack Overflow. As can be seen from Table 3, ANNOTATION consistently outperforms existing state-of-the-art methods showing a relative improvement of 63.7%, 30.3%, 44.7%, 36.2%, 48.7%, and 48.3% over FastTagRec; 53.2%, 26.9%, 40.7%, 32.5%, 41.1%, and 40.8% over TagDC; 68.2%, 32.1%, 46.3%, 38.0%, 49.9%, and 49.5% over iTAG; 64.6%, 26.9%, 40.5%, 32.4%, 47.7%, and 47.4% over AMNN; 31.5%, 17.3%, 26.5%, 21.0%, 30.9%, and 31.2% over PTM4TAG; 50.2%, 25.9%, 39.3%, 31.3%, 39.9%, and 39.6% over CETR; 41.5%, 21.9%, 34.2%, 26.8%, 35.3%, and 35.1% over CDR4TAG in HitRate@3, Precision@3, Recall@3, F1@3, ROUGE-1@3, and ROUGE-L@3 respectively. These consistent improvements across diverse datasets and evaluation metrics underscore ANNOTATION's effectiveness and generalizability for tag recommendation in

Table 3: Effectiveness Comparison against SOTA on Stack Overflow Dataset

Methods	Hit rate	Precision	Recall	F1-score	ROUGE-1	ROUGE-L
Classification/Retrieval						
FastTagRec (Liu et al., 2018)	0.150	0.053	0.070	0.060	0.076	0.076
TagDC (Li et al., 2020)	0.255	0.087	0.110	0.097	0.152	0.151
PTM4TAG (He et al., 2022)	0.472	0.183	0.252	0.212	0.254	0.247
Sentence Matching						
CETR (He et al., 2022)	0.285	0.097	0.124	0.109	0.164	0.163
CDR4TAG (Li et al., 2023b)	0.372	0.137	0.175	0.154	0.210	0.208
Generation						
iTAG (Tang et al., 2019)	0.105	0.035	0.054	0.042	0.064	0.064
AMNN (Yang et al., 2020)	0.141	0.087	0.112	0.098	0.086	0.086
ANNOTATION (Ours)	**0.787**	**0.356**	**0.517**	**0.422**	**0.563**	**0.559**

Table 4: Ablation Studies

Comments	User	Hit rate	Precision	Recall	F1-score	ROUGE-1	ROUGE-L
✗	✗	0.902	0.471	0.571	0.516	0.535	0.531
✗	✓	0.901	0.475	0.575	0.520	0.533	0.529
✓	✗	0.912	0.475	0.577	0.521	0.544	0.540
✓	✓	**0.920**	**0.487**	**0.590**	**0.534**	**0.544**	**0.540**

various software information sites.

5.2.2. Ablation Studies

We conducted ablation studies to assess the impact of different features on our proposed method, ANNOTATION's performance. We compare the following variants:

- ANNOTATION (w/o {comments + user}): This variant removes both comments and user-related attributes.
- ANNOTATION (w/o comments): This variant removes comments feature.
- ANNOTATION (w/o user): This variant removes user profile descriptions and badges.

Table 4 shows that ANNOTATION (w/o {comments+user}) shows a drop of 1.8%, 1.6%, 1.9%, and 1.8% in HR@3, P@3, R@3, F1@3 on CodeReview dataset respectively.

ANNOTATION (w/o comments) leads to a noticeable decrease of 1.9%, 1.2%, 1.5%, and 1.4% in HR@3, P@3, R@3, F1@3 on CodeReview dataset respectively. This demonstrates the effectiveness of comments in enriching post representations with community knowledge. Comments provide valuable insights into discussions, explanations, and details about the code or problem being addressed in a post, helping the model grasp nuances that lead to more accurate predictions. These clarify ambiguities, highlight specific issues, and offer alternative perspectives. Comments reveal the underlying intent behind code choices, design decisions, or error messages, which in turn, helps the model understand the problem's

Table 5: Performance Comparison with Decoding Strategies

Decoding Strategy	Hit rate	Precision	Recall	F1-score	ROUGE-1	ROUGE-L
$ANNOTATION_{Beam}$	0.856	0.428	0.525	0.471	0.524	0.290
$ANNOTATION_{Greedy}$	**0.920**	**0.487**	**0.590**	**0.534**	**0.544**	**0.540**

broader context. As users engage in discussions within comments, they naturally use relevant terms and phrases related to the problem domain. These terms act as strong indicators of the appropriate tags. Through back-and-forth conversations, users refine their understanding of the problem and its solutions. This iterative process leads to a clearer picture of the relevant concepts, guiding tag suggestions. Comments involve experienced users or domain experts who offer insights and solutions. Their contributions enrich the conversation with specialized knowledge, helping the model identify more accurate and specialized tags. The collective knowledge and experience reflected in comments can reveal connections between concepts or technologies, which might not be immediately apparent from the original post alone. Interaction-based comments are a goldmine of information for recommendation systems. They go beyond the surface-level details of a post to reveal the underlying context, community expertise, and evolving trends. By harnessing this conversational context, the model can generate more relevant and accurate tag suggestions, leading to a better user experience and improved knowledge sharing within software information sites.

ANNOTATION (w/o user) exhibited a decline of 0.8% to 1.3% in the same metrics, highlighting the importance of user-related information in understanding user needs and interests and recommending personalized tags. User profile descriptions and badges offer valuable insights into a user's interests, expertise, preferences, and community standing. Profile descriptions often contain keywords and phrases that directly reflect a user's interests and areas of expertise. By analyzing these descriptions, the recommendation system can identify topics and tags likely to resonate with the user. Badges earned by users indicate their proficiency and achievements in specific domains. This information can be leveraged to recommend tags relevant to the user's skill level, providing them with content that is neither too basic nor too advanced. Personalized recommendations leverage a user's past behavior, such as posting and tagging history. However, without such historical data, user profile descriptions and assigned badges can serve as valuable proxies for personalization. While not a direct record of past actions, profile descriptions, and badges reveal a user's interests and expertise. Even without historical data, profile descriptions and badges can serve as effective proxies for personalization, especially for new users who lack a history of interactions.

The ablation study corroborates that comments and user-related information are valuable features for tag recommendation. The consistent decline in performance when these features are removed indicates that the method benefits from combining multiple sources of information rather than relying heavily on any single feature.

Table 6: Performance Comparison with Different Embedding Procedures for Comments

Variant	Hit rate	Precision	Recall	F1-score	ROUGE-1	ROUGE-L
$ANNOTATION_{Average}$	0.910	0.477	0.579	0.523	0.540	0.536
$ANNOTATION_{Concatenate}$	0.906	0.473	0.574	0.519	0.545	0.543
$ANNOTATION_{Sorted}$	0.920	0.487	0.590	0.534	0.544	0.540

5.2.3. Decoding Strategies

The ablation study reveals intriguing insights into the performance differences between greedy and beam decoding strategies within the proposed tag recommendation framework.

- Greedy Decoding: This strategy selects the most probable tag at each step in the sequence generation process without considering future possibilities. It is a straightforward and computationally efficient approach.
- Beam Decoding: This strategy maintains a set of the most promising sequences (the "beam") at each step. It explores multiple potential paths, looking ahead to improve the overall quality of the generated tag sequence.

As can be seen from Table 5, ANNOTATION employing beam decoding shows a drop of 6.4%, 5.9%, 6.5%, 6.3%, 2.0%, and 25.0% in hit rate, precision, recall, F1-score, ROUGE-1, and ROUGE-L respectively, over greedy decoding. This performance difference stems from the fundamental difference between natural language generation and hashtag generation. While natural language generation prioritizes fluent and coherent sentences, where beam search excels by maximizing the probability of the entire sequence, hashtag generation emphasizes relevance over probability. The goal is to generate a set of appropriate hashtags that accurately reflect the content without the constraints of strict grammatical rules. Greedy search, focusing on local context, is better suited for this task. Tag recommendations for software information sites often require specificity and relevance to the immediate context, such as the title, body, code, comments, and user information. Greedy decoding focusing on the most probable tag at each step aligns well with this requirement. In contrast, the lookahead approach in beam decoding can lead to over-optimization, prioritizing sequences that appear promising in the short term but are not the most relevant or meaningful overall. This issue is particularly relevant when dealing with noisy and informal text data in software information sites.

Therefore, the superior performance of greedy decoding demonstrates that tag recommendation task for software information sites benefits from prioritizing local context and specificity over the global probability of the entire sequence.

5.2.4. Comments Embeddings Variants

In this section, we discuss the performance of our proposed model with various embedding procedures for comments associated with software objects. Three variants are examined which are detailed below.

Table 7: Performance Comparison with Encoder-Decoder Combinations

Encoder-Decoder	Hit rate	Precision	Recall	F1-score	ROUGE-1	ROUGE-L
BERT-GRU	0.332	0.174	0.208	0.191	0.163	0.162
BiLSTM-LSTM	0.192	0.158	0.189	0.172	0.128	0.129
CodeBERT-GRU	0.310	0.205	0.243	0.222	0.173	0.173
BERT-GPT2	0.886	0.467	0.563	0.511	0.530	0.526
ALBERT-GPT2	0.863	0.445	0.539	0.488	0.507	0.502
T5	0.867	0.466	0.561	0.509	0.447	0.443
BART	0.678	0.254	0.323	0.285	0.386	0.384
ANNOTATION (Ours)	**0.920**	**0.487**	**0.590**	**0.534**	**0.544**	**0.540**

- $ANNOTATION_{Average}$: In this variant, we average all comments associated with a software object after tokenizing the comments.
- $ANNOTATION_{Concatenate}$: Here, we concatenate all comments after tokenization, preserving their sequential order.
- $ANNOTATION_{Sorted}$: This variant sorts comments based on their associated score in descending order before concatenating their tokenized sentences. This score, reflecting a comment's relevance or importance, could be derived from factors such as user reputation, upvotes, or comment length.

Table 6 presents a comparison of different embedding procedures for incorporating comments in ANNOTATION, showcasing their impact on various evaluation metrics. As can be seen from Table 6, $ANNOTATION_{Sorted}$ achieves the best performance across all metrics, indicating that prioritizing comments based on their score enhances the model's ability to capture the most salient information from the discussion. This suggests that incorporating a measure of comment relevance can lead to more accurate and contextually appropriate tag suggestions. $ANNOTATION_{Average}$ and $ANNOTATION_{Concatenate}$ exhibit comparable performance with the latter slightly outperforming the former in terms of ROUGE-1 and ROUGE-L scores, suggesting that preserving the sequential order of comments might provide a slightly richer representation. Overall, the results highlight the importance of considering not only the content of comments but also their relative relevance when incorporating them into tag recommendation models.

5.2.5. Encoder-Decoder Combination Study

This section investigates the critical role of encoder-decoder architectures suitable for capturing intricate information within software objects and generating relevant tags. Table 7 provides a comprehensive performance comparison of various encoder-decoder combinations for tag recommendation in software information sites, highlighting the impact of model architecture. As evident from Table 7, ANNOTATION while leveraging CodeBERT-GPT2 as encoder decoder exhibits a relative improvement of 58.8%, 31.3%, 38.2%, 34.3%, 38.1%, 37.8% over BERT-GRU; 72.8%, 32.9%, 40.1%, 36.2%, 41.6%, 41.1% over BiLSTM-LSTM; in terms of hit rate, precision, recall, F1-score, ROUGE-1 and ROUGE-L, respectively in

the realm of tag recommendation in software information sites. These significant gains underscore the efficacy of the proposed architecture in capturing the complex relationships and contextual information necessary for accurate tag recommendation.

Encoder Analysis. We first analyze the impact of different encoders while keeping the decoder consistent (GPT-2). We observed relative improvements with BERT (3.4%, 2.0%, 2.7%, 2.3%, 1.4%, 1.4% in HR@3, P@3, R@3, F1@3, respectively) and ALBERT (6.2%, 4.1%, 5.2%, 4.6%, 3.7%, 3.8%). While ALBERT-GPT2 (ALBERT, a lighter and faster version of BERT) shows a slight improvement over BERT-GPT2, CodeBERT emerges as the most effective encoder due to its domain-specific pre-training. Our proposed model, CodeBERT-GPT2, consistently outperforms all other combinations across most metrics. This highlights the significant advantage of leveraging CodeBERT's domain-specific pre-training on code and software-related text. CodeBERT's specialized knowledge allows it to more effectively capture the nuances and semantics inherent in software information, leading to more accurate and relevant tag recommendations.

Decoder Analysis. ANNOTATION employing CodeBERT-GPT2 shows a substantial improvement over CodeBERT-GRU (61.0%, 28.2%, 34.8%, 31.2%, 37.1%, 36.7% in hit rate, precision, recall, F1-score, ROUGE-1, and ROUGE-L, respectively). This underscores GPT-2's strength in generating coherent and contextually relevant tag sequence.

Pretrained Encoder-Decoder Models. We evaluated our proposed method against prominent pre-trained encoder-decoder models for tag recommendation.

- Text To Text Transfer Transformer (T5): 5.3%, 2.1%, 2.9%, 2.5%, 9.7%, 9.7% in HR@3, P@3, R@3, F1@3, ROUGE-1@3, and ROUGE-L@3;
- Bidirectional AutoRegressive Transformer (BART): 24.2%, 23.3%, 26.7%, 24.9%, 15.8%, 16% in HR@3, P@3, R@3, F1@3, ROUGE-1@3, and ROUGE-L@3;

The results, presented in Table 7 reveal the significant impact of encoder-decoder architecture on tag recommendation performance. Notably, BART, with its robust sequence-to-sequence modeling capabilities, demonstrates a substantial improvement over T5. While pre-trained models yield competitive results, they fall short of the performance achieved by CodeBERT-GPT2. This highlights the importance of selecting an architecture that aligns with the specific task and data characteristics. In this context, CodeBERT's specialization in understanding code and software-related text proves to be a significant advantage.

5.2.6. Results for Problem Formulation Procedures

This section explores two different approaches to tag annotation for software and information sites: Multi-Label Classification (MLC) and Sequence Generation (SG).

- MLC: We extract features from a software object and use a fully connected layer with softmax activation to predict the probability of each tag being relevant.

Table 8: Performance Comparison with Different Problem Formulation Procedures

Formulation	Hit rate	Precision	Recall	F1-score	ROUGE-1	ROUGE-L
$ANNOTATION_{MLC}$	0.421	0.141	0.165	0.152	0.205	0.204
$ANNOTATION_{SG}$	**0.920**	**0.487**	**0.590**	**0.534**	**0.544**	**0.540**

- SG: The features are encoded using CodeBERT and fed to GPT-2 that generates hashtags sequentially for the software object.

As shown in Table 8, SG significantly outperforms MLC across all metrics. SG, unlike MLC, explicitly models relationships between tags. While MLC treats each tag independently, SG generates them sequentially, considering the context of software information and previously generated tags. This allows SG to capture the inherent structure and dependencies within tags, leading to more coherent and relevant recommendations. Furthermore, SG offers two key advantages over MLC:

- Generating Tags Beyond Predefined Lists: SG can generate new and unseen tag combinations, crucial in a dynamic field such as software development, whereas MLC is limited to a fixed vocabulary.
- Enhanced Contextualization: SG leverages the context of previously generated tags to produce more relevant recommendations.

In summary, SG formulation provides a more powerful and flexible approach to tag recommendation. By capturing the complex relationships between tags and generating more informative and contextually relevant recommendations, SG aligns better with the evolving nature of software information and user needs.

5.2.7. Qualitative Assessment

This section presents a qualitative analysis of our proposed tag recommendation model, highlighting its effectiveness in generating high-quality tags for software information sites. We evaluate the model's performance on a specific post sourced from testing set of Stack Overflow dataset. We first compare its generated tags to ground-truth tags and those predicted by other state-of-the-art methods followed by micro-analysis of tags generated owing to inclusion of specific features. The post shown in Figure 4 requests code review and optimization for a C program that calculates the number of odd and even Fibonacci numbers within a given range. The user explicitly seeks advice on improving code efficiency and utilizing mathematical formulas to enhance performance.

Our model demonstrates a strong ability to understand the core concepts of the post, accurately identifying the programming language (#C), the problem domain (#fibonacci-sequence), and the nature of the task (#programming-challenge). The generated tags are highly relevant and effectively capture the essence of the post's content.

PTM4TAG introduces one irrelevant tags (#java) and one relevant tag (#algorithm) while correctly identifying #performance, thereby, shows a tendency to overgeneralize.

A program to find out the number of odd and even Fibonacci numbers between given range

I built a program in C that can tell how many odd and even Fibonacci numbers there are between a given range.
Input Specification:
First line of the input contains **T**, representing the number of test cases ($1 \le T \le 50$). Each test case contains two integers **N** and **M** ($1 \le N \le M \le 1018$) and ($|N-M| \le 105$), where **N** is the **N**th Fibonacci number and **M** is the **M**th Fibonacci number of the sequence.
Output Specification- Case T:
Odd = total number of odd Fibonacchi numbers between **N** and **M**
Even = total number of even Fibonacchi numbers between **N** and **M**

The full code

```
int main()
{
   int T, i, j, k;
   scanf("%d", &T);
   for (i=0; i<T; i++)
   {
      int N, M, val;
      scanf("%d%d", &N,&M);
      val = abs(N-M)+1;
      int n,m;
      if (M>N) { n=N; m=M; }
      if (N>M) { n=M; m=N; }
      int b_tri, e_tri, even=0, odd=0;
      if (n%3==1) { b_tri=3; even++;
odd+=2; }
      if (n%3==2) { b_tri=2; odd+=2; }
      if (n%3==0) { b_tri=1; odd++;  }
      if (m%3==1) { e_tri=1; even++; }
      if (m%3==2) { e_tri=2; odd++;
even++; }
      if (m%3==0) { e_tri=3; odd+=2;
even++; }
      val-=(e_tri+b_tri);
      val/=3;
      for (j=0; j<val; j++)
      {
         even++;
         odd+=2;
      }
      printf("Case %d:\nOdd =
%d\nEven = %d\n", i+1, odd, even);
   }
   return 0;
}
```

Logic

- **1st** Fibonacci number is 0 (**even**).
  2nd Fibonacci number is 1 (**odd**).
  3rd Fibonacci number is 1 (even + odd = **odd**).
- **4th** Fibonacci number is 2 (odd + odd = **even**).
  5th Fibonacci number is 3 (odd + even = **odd**).
  6th Fibonacci number is 5 (even + odd = **odd**).
- **7th** Fibonacci number is 8 (**even**).
  8th Fibonacci number is 13 (**odd**).
  9th Fibonacci number is 21 (**odd**).

I'd like advice on:

- Using mathematical formulas instead of loops to calculate this
- Making the code more efficient

- 2: Given two Fibonacci numbers N and M (N < M), you aim to compute (1) the number of all odd/ even integers between N and M (includin g N and M), or (2) the number of all odd/ even Fibonacci numbers (including N and M); please specify. – coderodde, Commented Aug 7, 2017 at 15:35

- @coderodde Question edited. – User X, Commented Aug 8, 2017 at 4:42

- I read the output specification, I do not assume that what you have implemented and what you should have implemented are the same thing. – User Y, Commented Aug 8, 2017 at 14:21

User

About :
Aspiring to be a world-class programmer. I like to solve olympiad-level math problems when I'm in the mood for it and I also dabble in coding from time to time. Writing is something I have a knack for. I can write well enough in English, despite being a non-native English speaker. Reaching the summit of success is my ultimate goal.

Badges

Silver badges	Bronze badges
• Good Question	• Peer Pressure
• Notable Question × 3	• Nice Question
• Yearling × 2	• Commentator

Ground-truth hashtags

#performance #c #programming-challenge #fibonacci-sequence

Predictions

Correct | Incorrect but relevant | Incorrect and irrelevant

ANNOTATION: #c #programming-challenge #fibonacci-sequence

PTM4TAG : #java #performance #c# #algorithm #programming-challenge

FastTagRec: #c# #sieve-of-eratosthenes #excel #opencv #factory-method

AMNN: #java #performance #programming-challenge

CETR: #c# #java #python #performance #javascript

CDR4TAG: #c++ #c# #performance #c #java

TagDC: #python #java #c# #performance #algorithm

iTAG: #c#

Figure 4: Example Post Depicting Tags Recommended by Different Methods

FastagRec generates completely off-topic tags, indicating a failure to understand the post's subject matter. AMNN misses key tags (#c, #fibonacci-sequence) while correctly identifying #performance. It shows limited understanding of the specific problem domain. CETR overgeneralizes by including irrelevant language tags (#java, #python, #javascript). CDR4TAG includes irrelevant language tags (#c++, #java) while correctly identifying #performance. TagDC's predictions are similar to CETR. TagDC overgeneralizes with irrelevant language tags (#python, #java) and a generic #algorithm tag.

Our proposed model outperforms other methods in terms of relevance and conciseness, avoiding the inclusion of irrelevant or overly generic tags. This suggests that our multi-modal encoder-decoder architecture effectively captures the essence of the post by leveraging diverse information sources, including code snippets, comments, and user information.

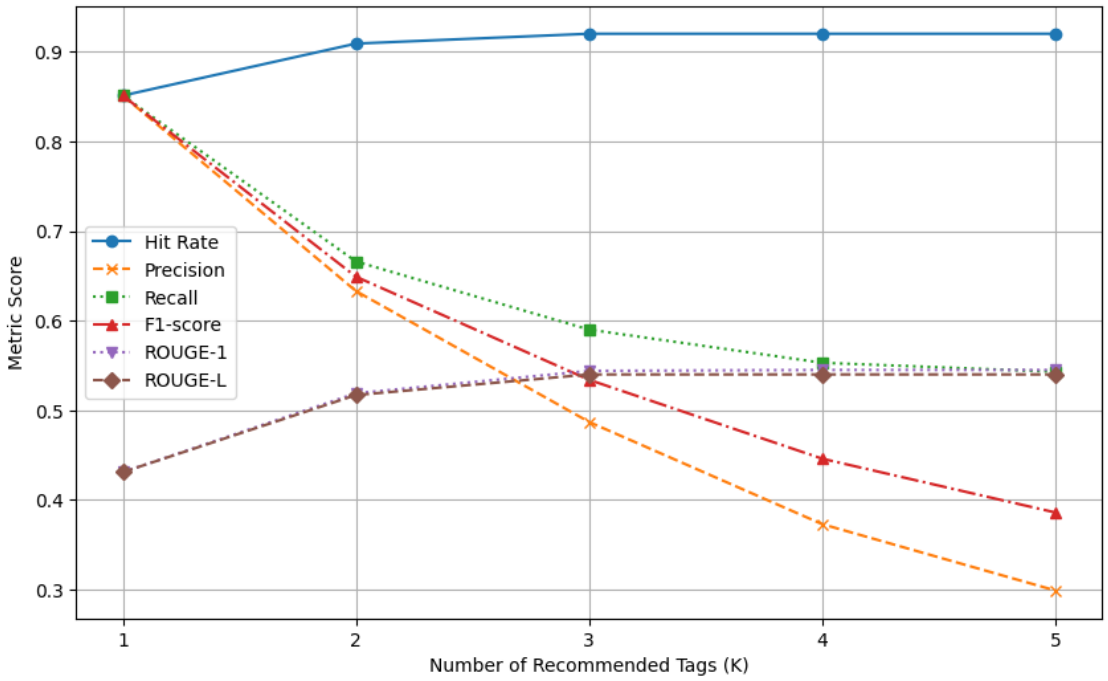


Figure 5: Effect of Number of Recommended Tags

5.2.8. Implementation Details

The proposed model was implemented and evaluated on a Linux server with GPU P100. We employed Adam optimizer with a learning rate 1e-5 to facilitate model convergence and optimization. The training process utilized a batch size of 8 and an 70:10:20 train, validation, and test split. The input sequences were either truncated or padded to maintain uniform lengths for efficient processing. The maximum sequence lengths were set to 8 for title, 80 for text, 100 for code, 80 for comments, 50 for the user profile description, 25 for badges assigned to users, and 9 for user ID, respectively. The software environment leveraged PyTorch 2.1.2, TensorFlow 2.15.0, Keras 2.15.1, and transformers 4.31.0. We used rouge[9] to compute ROUGE scores.

5.2.9. Parameter Sensitivity Studies

We study the effects of two parameters in our method, i.e., the number of recommended tags (K) and sequence length (S).

- $Top-k$: The parameter K determines the number of top-ranked tags to be recommended. To assess the impact of this parameter on performance, we evaluated ANNOTATION with K ranging from 1 to 5. As depicted in Figure 5, ANNOTATION exhibits optimal performance at K=3, achieving the highest F1-score. This suggests that recommending three tags strikes an effective balance between precision and recall, providing users with a concise yet informative set of recommendations.

[9]https://huggingface.co/spaces/evaluate-metric/rouge

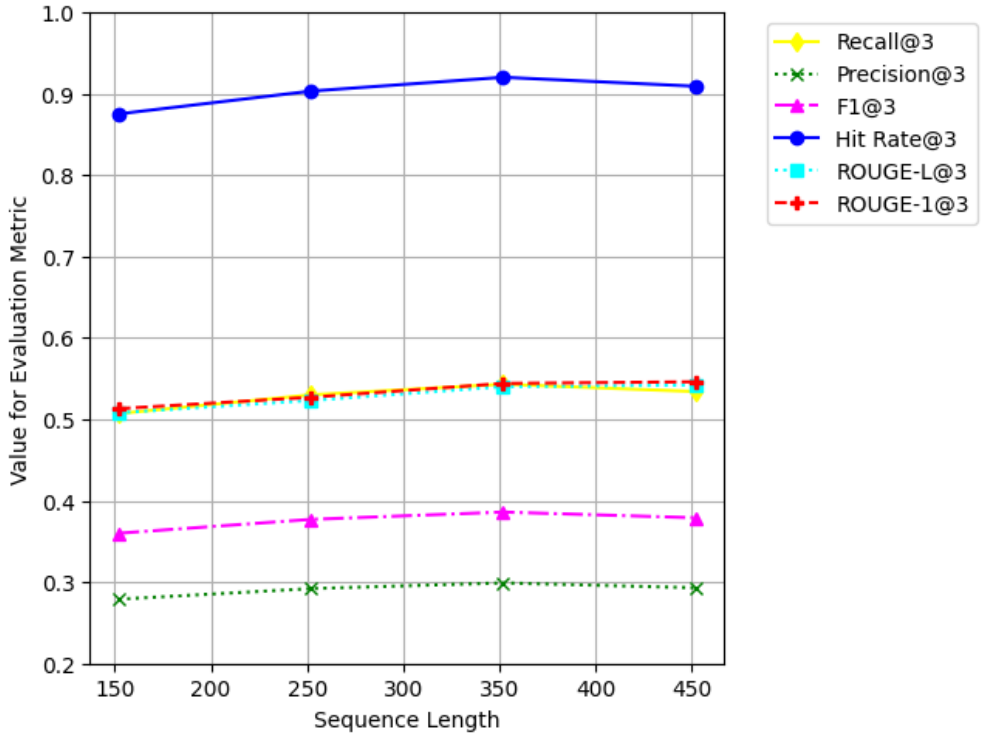


Figure 6: Effect of Sequence Length

- Sequence length: The sequence length parameter (S) determines the maximum number of tokens considered from each input modality (software object description, code, comments, user's "about me" section). We varied S within the range of 152 to 452. As shown in Figure 6, the optimal performance was achieved at a sequence length of 352. This finding suggests that capturing a sufficient amount of contextual information from each modality is crucial to achieve optimal performance. However, excessively long sequences may introduce noise or increase computational complexity, highlighting the importance of selecting an appropriate sequence length to balance performance and efficiency.

6. Conclusion

In this paper, we introduce a novel automatic tag recommendation system for software information sites. To this end, we devise an encoder-decoder architecture that leverages multiple modalities namely, textual content (title, body, code snippets), associated comments, and user information (profile descriptions, badges) to refine the representation of software objects and recommend contextually relevant and personalized tags. The encoder, a transformer model pre-trained for code understanding and software engineering, effectively embeds the diverse input modalities. A generative pre-trained language model serves as the decoder, generating tags sequentially conditioned on the encoded context and previously generated tags. Notably, our approach is the first to utilize a generative pre-trained

language model as the decoder for sequential tag recommendation in the software domain. Extensive experiments conducted on two real-world datasets demonstrate that our proposed model outperforms state-of-the-art methods, including multi-label classification and sentence matching approaches. Our approach excels in capturing tag correlations and generating tags beyond predefined lists, showcasing the effectiveness of sequence generation. Furthermore, incorporating user information and associated comments enhances the personalization and helps mitigate data sparsity issues, thereby ensuring pertinent tag suggestions.

References

Al-Kofahi, J. M., Tamrawi, A., Nguyen, T. T., Nguyen, H. A., & Nguyen, T. N. (2010). Fuzzy set approach for automatic tagging in evolving software. In *2010 IEEE international conference on software maintenance* (pp. 1–10). IEEE.

Anderson, A., Huttenlocher, D., Kleinberg, J., & Leskovec, J. (2012). Discovering value from community activity on focused question answering sites: a case study of stack overflow. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 850–858).

Anuradha, A., Prasad, K. S. R., & Madala, S. R. (2021). Multi-label classification system that automatically tags users' questions to enhance user experience. *NVEO-NATURAL VOLATILES & ESSENTIAL OILS Journal— NVEO*, (pp. 1281–1288).

Ashvanth, R., Deepak, G., Sheeba Priyadarshini, J., & Santhanavijayan, A. (2022). Kmetatagger: A knowledge centric metadata driven hybrid tag recommendation model encompassing machine intelligence. In *International Conference on Intelligent Systems Design and Applications* (pp. 1–11). Springer.

Bansal, S., Gowda, K., & Kumar, N. (2022). A hybrid deep neural network for multimodal personalized hashtag recommendation. *IEEE transactions on computational social systems*, *10*, 2439–2459.

Bansal, S., Gowda, K., & Kumar, N. (2024a). Multilingual personalized hashtag recommendation for low resource indic languages using graph-based deep neural network. *Expert Systems with Applications*, *236*, 121188.

Bansal, S., Gowda, K., Rehman, M. Z. U., Raghaw, C. S., & Kumar, N. (2024b). A hybrid filtering for micro-video hashtag recommendation using graph-based deep neural network. *Engineering Applications of Artificial Intelligence*, *138*, 109417.

Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., & Klemmer, S. R. (2009). Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1589–1598).

Cai, X., Zhu, J., Shen, B., & Chen, Y. (2016). Greta: Graph-based tag assignment for github repositories. In *2016 IEEE 40th Annual computer software and applications conference (compsac)* (pp. 63–72). IEEE volume 1.

Djuana, E., Xu, Y., Li, Y., & Jøsang, A. (2014). A combined method for mitigating sparsity problem in tag recommendation. In *2014 47th Hawaii International Conference on System Sciences* (pp. 906–915). IEEE.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D. et al. (2020). Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020* (pp. 1536–1547).

Gharibi, R., Safdel, A., Fakhrahmad, S. M., & Sadreddini, M. H. (2021). A content-based model for tag recommendation in software information sites. *The Computer Journal*, *64*, 1680–1691.

Gottipati, S., Lo, D., & Jiang, J. (2011). Finding relevant answers in software forums. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)* (pp. 323–332). IEEE.

Guerrouj, L., Azad, S., & Rigby, P. C. (2015). The influence of app churn on app success and stackoverflow discussions. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 321–330). IEEE.

Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S. et al. (2020). Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*, .

Hamidi Rad, R., Cucerzan, S., Chandrasekaran, N., & Gamon, M. (2024). Interactive topic tagging in community question answering platforms. In *European Conference on Information Retrieval* (pp. 195–209). Springer.

He, J., Xu, B., Yang, Z., Han, D., Yang, C., & Lo, D. (2022). Ptm4tag: sharpening tag recommendation of stack overflow posts with pre-trained models. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension* (pp. 1–11).

Huang, H., Zhang, Q., Gong, Y., & Huang, X.-J. (2016). Hashtag recommendation using end-to-end memory networks with hierarchical attention. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers* (pp. 943–952).

Ifada, N., & Nayak, R. (2021). A new weighted-learning approach for exploiting data sparsity in tag-based item recommendation systems. *International Journal of Intelligent Engineering & Systems*, *14*.

Kavuk, E. M., & Tosun, A. (2020). Predicting stack overflow question tags: a multi-class, multi-label classification. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (pp. 489–493).

Kononenko, O., Dietrich, D., Sharma, R., & Holmes, R. (2012). Automatically locating relevant programming help online. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 127–134). IEEE.

Li, C., Xu, L., Yan, M., He, J., & Zhang, Z. (2019). Tagdeeprec: tag recommendation for software information sites using attention-based bi-lstm. In *International Conference on Knowledge Science, Engineering and Management* (pp. 11–24). Springer.

Li, C., Xu, L., Yan, M., & Lei, Y. (2020). Tagdc: A tag recommendation method for software information sites with a combination of deep learning and collaborative filtering. *Journal of Systems and Software*, *170*, 110783.

Li, L., Wang, P., Zheng, X., & Xie, Q. (2023a). Code-enhanced fine-grained semantic matching for tag recommendation in software information sites. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 1–5). IEEE.

Li, L., Wang, P., Zheng, X., Xie, Q., Tao, X., & Velásquez, J. D. (2023b). Dual-interactive fusion for code-mixed deep representation learning in tag recommendation. *Information Fusion*, *99*, 101862.

Lipczak, M., Hu, Y., Kollet, Y., & Milios, E. E. (2009). Tag sources for recommendation in collaborative tagging systems. In *DC@ PKDD/ECML*.

Liu, B., Xu, P., Lu, S., Wang, S., Sun, H., & Jing, L. (2023). Sequential tag recommendation. *arXiv preprint arXiv:2310.05423*, .

Liu, J., Zhou, P., Yang, Z., Liu, X., & Grundy, J. (2018). Fasttagrec: fast tag recommendation for software information sites. *Automated Software Engineering*, *25*, 675–701.

Lu, S., Xu, P., Liu, B., Sun, H., Jing, L., & Yu, J. (2024). Retrieval augmented cross-modal tag recommendation in software q&a sites. *arXiv preprint arXiv:2402.03635*, .

Nie, L., Li, Y., Feng, F., Song, X., Wang, M., & Wang, Y. (2020). Large-scale question tagging via joint question-topic embedding learning. *ACM Transactions on Information Systems (TOIS)*, *38*, 1–23.

Nie, L., Zhao, Y.-L., Wang, X., Shen, J., & Chua, T.-S. (2014). Learning to recommend descriptive tags for questions in social forums. *ACM Transactions on Information Systems (TOIS)*, *32*, 1–23.

Saha, A. K., Saha, R. K., & Schneider, K. A. (2013). A discriminative model approach for suggesting tags automatically for stack overflow questions. In *2013 10th Working Conference on Mining Software Repositories (MSR)* (pp. 73–76). IEEE.

Sahu, T. P., Thummalapudi, R. S., & Nagwani, N. K. (2019). Automatic question tagging using multi-label classification in community question answering sites. In *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)* (pp. 63–68). IEEE.

Srba, I., & Bielikova, M. (2016). A comprehensive survey and classification of approaches for community question answering. *ACM Transactions on the Web (TWEB)*, *10*, 1–63.

Storey, M.-A., Treude, C., Van Deursen, A., & Cheng, L.-T. (2010). The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research* (pp. 359–364).

Tang, S., Yao, Y., Zhang, S., Xu, F., Gu, T., Tong, H., Yan, X., & Lu, J. (2019). An integral tag recommendation model for textual content. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 5109–5116). volume 33.

Treude, C., & Storey, M.-A. (2010). Work item tagging: Communicating concerns in collaborative software development. *IEEE Transactions on Software Engineering*, *38*, 19–34.

Wang, H., Wang, B., Li, C., Xu, L., He, J., & Yang, M. (2019a). Sotagrec: A combined tag recommendation approach for stack overflow. In *Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence* (pp. 146–152).

Wang, L., Li, Y., & Jing, W. (2023). Keic: A tag recommendation framework with knowledge enhancement and interclass correlation. *Information Sciences*, *645*, 119330.

Wang, S., Lo, D., Vasilescu, B., & Serebrenik, A. (2014). Entagrec: An enhanced tag recommendation system for software information sites. In *2014 IEEE International Conference on Software Maintenance and Evolution* (pp. 291–300). doi:10.1109/ICSME.2014.51.

Wang, S., Lo, D., Vasilescu, B., & Serebrenik, A. (2018). Entagrec++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering*, *23*, 800–832.

Wang, Y., Li, J., Chan, H. P., King, I., Lyu, M. R., & Shi, S. (2019b). Topic-aware neural keyphrase generation for social media language. *arXiv preprint arXiv:1906.03889*, .

Wang, Y., Li, J., King, I., Lyu, M. R., & Shi, S. (2019c). Microblog hashtag generation via encoding conversation contexts. *arXiv preprint arXiv:1905.07584*, .

Wu, Y., Wu, W., Li, Z., & Zhou, M. (2016). Improving recommendation of tail tags for questions in community question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*. volume 30.

Xia, X., Lo, D., Wang, X., & Zhou, B. (2013). Tag recommendation in software information sites. In *2013 10th Working Conference on Mining Software Repositories (MSR)* (pp. 287–296). IEEE.

Xu, N., Hu, J., Fang, Q., Xue, D., Li, Y., & Qian, S. (2024). Tri-relational multi-faceted graph neural networks for automatic question tagging. *Neurocomputing*, *576*, 127250.

Xu, S., Pang, L., Shen, H., & Cheng, X. (2022). Match-prompt: Improving multi-task generalization ability for neural text matching via prompt learning. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (pp. 2290–2300).

Yang, Q., Wu, G., Li, Y., Li, R., Gu, X., Deng, H., & Wu, J. (2020). Amnn: Attention-based multimodal neural network model for hashtag recommendation. *IEEE Transactions on Computational Social Systems*, *7*, 768–779.

Yu, T., Yu, H., Liang, D., Mao, Y., Nie, S., Huang, P.-Y., Khabsa, M., Fung, P., & Wang, Y.-C. (2023). Generating hashtags for short-form videos with guided signals. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 9482–9495).

Zhang, J., Sun, H., Tian, Y., & Liu, X. (2018a). Semantically enhanced tag recommendation for software cqas via deep learning. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (pp. 294–295).

Zhang, Q., Wang, J., Huang, H., Huang, X., & Gong, Y. (2017). Hashtag recommendation for multimodal microblog using co-attention network. In *IJCAI* (pp. 3420–3426).

Zhang, Q., Wang, Y., Gong, Y., & Huang, X.-J. (2016). Keyphrase extraction using deep recurrent neural networks on twitter. In *Proceedings of the 2016 conference on empirical methods in natural language processing* (pp. 836–845).

Zhang, Y., Li, J., Song, Y., & Zhang, C. (2018b). Encoding conversation context for neural keyphrase extraction from microblog posts. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (pp. 1676–1686).

Zheng, X., Li, L., & Zhou, D. (2020). An attentive deep supervision based semantic matching framework for tag recommendation in software information sites. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 490–494). IEEE.

Zhou, P., Liu, J., Liu, X., Yang, Z., & Grundy, J. (2019). Is deep learning better than traditional approaches in tag recommendation for software information sites? *Information and software technology*, *109*, 1–13.

Zhou, P., Liu, J., Yang, Z., & Zhou, G. (2017). Scalable tag recommendation for software information sites. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 272–282). IEEE.