JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 42, 727-742 (2026)
DOI: 10.6688/JISE.202605_42(3).0014

Using Multi-Source Integration and Information Retrieval Technology to Program Question-and-Answering Chatbots*

AN-CHI SHAU[1] AND SHANG-PIN MA[2,+]
[1]*Pennsylvania State University-University Park*
State College, PA 16801, USA
[2]*National Taiwan Ocean University*
Keelung, 202 Taiwan
E-mail: ajs10348@psu.edu; albert@ntou.edu.tw[+]

Despite rapid growth in the number of programming learners, the continuous influx of new information and a lack of human guidance has left many learners sifting through online resources for reliable content. Moreover, many learners struggle to frame questions that accurately capture their specific concerns. This study addressed the growing need for efficient learning resources by developing a microservice-based Chatbot to synthesize information from diverse sources for programming learners. The proposed MPAbot system utilizes keyword extraction and cross-referencing across multiple sourced posts to help users solve questions more effectively. MPAbot incorporates word embeddings, sentence similarity, Latent Dirichlet Allocation (LDA) topic modeling, and multi-criteria decision analysis to filter out redundant information, thereby reducing browsing time and enhancing learning efficiency. Besides, MPAbot also integrates generative AI to improve the accuracy of its answers to users' questions. Experimental results show that our GPT-ranked approach achieved a 52.43% higher mean rank score compared to purely GPT-generated responses in user evaluation tests, demonstrating the effectiveness of the proposed multi-source integration approach.

***Keywords*:** Chatbot, programming learning, data integration, information retrieval, microservices, generative AI

1. INTRODUCTION

The digital transformation in many industries has increased the demand for individuals with skills in information technology, particularly programming. Many academic departments have responded by making programming courses a mandatory requirement. Unfortunately, this shift has left numerous students struggling to gain proficiency in areas with which they have no prior experience. A lack of familiarity with the new concepts and conventions makes it challenging to identify search keywords appropriate to their needs [1], while a lack of human tutors exacerbates the situation. These challenges can seriously undermine one's motivation to learn [2].

Various solutions have been developed to manage the exponential growth of online information, including information retrieval systems [3], recommendation systems, and Chatbots [4]. Chatbot technology is widely utilized as virtual assistants (*e.g.*, Siri, Aisera, Google Assistant, IBM Watson), in language practice (*e.g.*, Andy English Bot, Mondly), and for information retrieval (*e.g.*, Kuki, ChatGPT, Google Gemini). Chatbots have also

Received October 31, 2024; revised December 25, 2024; accepted March 1, 2025.
Communicated by Chu-Ti Lin.
[+] Corresponding author.
[*] This research was sponsored by the Ministry of Science and Technology in Taiwan under Grants Nos. 110-2221-E-019-039-MY3 and 112-2221-E-019-021-MY3.

proven valuable in assisting students in locating relevant domain knowledge [5]. However, current Chatbots lack several critical features required for effective programming education. We have identified three primary shortcomings:

1) Lack of programming-specific functionality: Most existing Chatbots struggle with the complexities of coding questions. There is a distinct need for Chatbots tailored to the unique requirements of programming education.
2) Lack of credible, relevant responses: Chatbots often provide false positive responses when they do not understand a query instead of acknowledging their limitations and offering alternative solutions. This leaves the user with the frustrating task of determining the accuracy of responses. There is a need for robust systems to verify Chatbot responses.
3) Lack of up-to-date data: Chatbots predominantly rely on stored data, much of which is outdated. This is a serious issue in the realm of programming environments where software languages and best practices are constantly evolving. There is a need for systems that maintain and update information resources regularly.

The Multi-sourcing Programming Assistant bot (MPAbot) presented in this paper was designed to help newbie learners with fundamental problem-solving by synthesizing and making accessible knowledge from advanced developers across multiple programming community sources. By employing word embeddings, sentence similarity measures, Latent Dirichlet Allocation (LDA) topic modeling, and multi-criteria decision analysis, MPAbot helps to reduce browsing time, enhances response accuracy, and improves learning efficiency. Besides, MPAbot also integrates generative AI to improve the accuracy of its answers to users' questions. This innovative approach has the potential to transform how students and developers engage with programming resources to solve problems.
The remainder of this paper is structured as follows. Section 2 reviews the literature related to the topics addressed in this study. Section 3 presents an overview of the requirements and core design of the proposed MPAbot system. Section 4 outlines the evaluation experiments. Section 5 summarizes the benefits of the proposed system and suggests directions for future research.

2. BACKGROUND AND RELATED WORK

To provide context for our research, this section examines two key areas: the current landscape of developer communities and recent advances in AI-powered code generation. We analyze how these technologies have shaped modern programming education and identify opportunities for improvement through integrated solutions.

2.1 Online Communities for Developers

Developers are increasingly turning to online communities to connect, learn, and share knowledge. Below, we highlight some prominent platforms:

Stack Overflow was founded in 2008. Stack Overflow is a Q&A platform covering a vast array of programming topics specifically for developers. Users can search for information using tags, participate in discussions, and earn badges for active contributions. Many researchers are working to improve response accuracy on Stack Overflow. For instance, Suta

et al. [6] proposed using paragraph vectors and similarity scores to match questions with suitable answers. Zhang *et al.* [7] developed a Chatbot system for question retrieval to enhance search efficiency through user interactions.

We also devised a Chatbot system called PSAbot [5], which applies word embedding, LDA (Latent Dirichlet Allocation) topic modeling, and weighting functions to help retrieve appropriate posts from Stack Overflow for answering user queries. Building upon PSAbot's foundation, the proposed MPAbot system significantly advances through several key improvements: (1) expansion from Stack Overflow-only to multi-source data integration; (2) evolution from monolithic to microservices architecture for enhanced extensibility and scalability; (3) implementation of a specialized weighting algorithm optimized for multi-source characteristics; and (4) integration of generative AI to improve response accuracy. These architectural and algorithmic enhancements enable MPAbot to provide more comprehensive and accurate solutions while maintaining PSAbot's core strengths in semantic analysis and topic modeling.

Reddit was launched in 2005; Reddit is a social news aggregation and discussion platform. Developers can search for information and connect with peers through subreddits like "programming," "learn programming," and "coding." User-driven content makes Reddit a valuable resource for developers [8].

GitHub[1] was established in 2008. GitHub is an online platform for software development and version control using Git. Developers can create repositories, collaborate on projects, and communicate through features like starring, following, commenting, and forking. GitHub hosts a massive community with over a hundred million users.

CodeProject[2] was created in 1999. CodeProject is a knowledge-sharing and collaboration portal for the software developer community. It offers a wealth of articles, tutorials, and resources covering a broad range of programming languages and topics.

2.2 AI Code Generation

AI-powered code generation has recently emerged as a powerful tool for developers. By automating repetitive coding tasks, AI can improve the efficiency and accessibility of software development, allowing developers to focus on the more complex and creative aspects of their work. The ability of AI to generate common code patterns and templates is particularly beneficial in helping novice programmers learn proper coding and improve their skills. Below are some prominent AI code-generation tools:

(1) GitHub Copilot[3] is powered by OpenAI Codex and public repositories, uploading accurate and verified code suggestions directly into the user's editor. It supports multiple programming languages and offers a user-friendly interface. Nonetheless, a heavy reliance on existing code makes it less practical to generate entirely new code structures.
(2) AlphaCode[4] was developed by DeepMind, a subsidiary of Google, as an AI code generator trained on competitive programming problems. When applied to unknown solutions in Codeforces competitions, the ability of AlphaCode to make real-time decisions enabled it to beat 54% of human competitors.

[1] https://github.com/
[2] https://www.codeproject.com/
[3] https://github.com/features/copilot
[4] https://alphacode.deepmind.com/

(3) ChatGPT[5], developed by OpenAI, is trained using the Reinforcement Learning from Human Feedback (RLHF) method. ChatGPT can adapt its responses to prompts and handle a wide range of topics. Nonetheless, the use of internet data for training increases the likelihood of generating unreliable or inaccurate responses.

3. MPABOT: APPROACH

This section details the MPAbot system's operational concepts, architecture, and core components, highlighting how various technologies work together to provide accurate and relevant responses to user queries.

3.1 Operational Concepts

MPAbot was designed to assist users in finding answers across multiple programming-related forums, with the aim of filtering out false positives and minimizing the time spent browsing. Fig. 1 illustrates the operational concept of MPAbot.

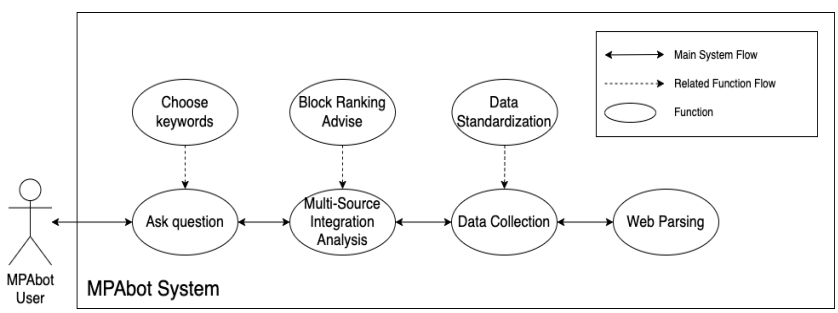

Fig. 1. Operational concept of the proposed MPAbot system.

Note that community discussion webpages typically present a single question post followed by multiple answer posts. In this study, we differentiate between these two types of posts using the term "question" for the initial question and "block" for each answer. The term "document" encompasses the original question and all corresponding blocks within the same website. Users seeking answers require the ability to extract relevant information from community discussion webpages. In this study, we used block analysis in conjunction with multi-criteria decision analysis (MCDA) to analyze information from multiple sources and fine-tune the criteria for response accuracy. We then use LLM-assisted content ranking to re-rank the recommended blocks to strengthen the relationship between blocks and each user query.

Fig. 2 illustrates the process for each operation. First, MPAbot gathers key information from the user, such as the operating system and language versions. It then analyzes data from multiple sources, including Stack Overflow, Reddit, and CodeProject. Finally, MPAbot provides block recommendations along with links to the original web page as a reference.

[5] https://chatgpt.com/

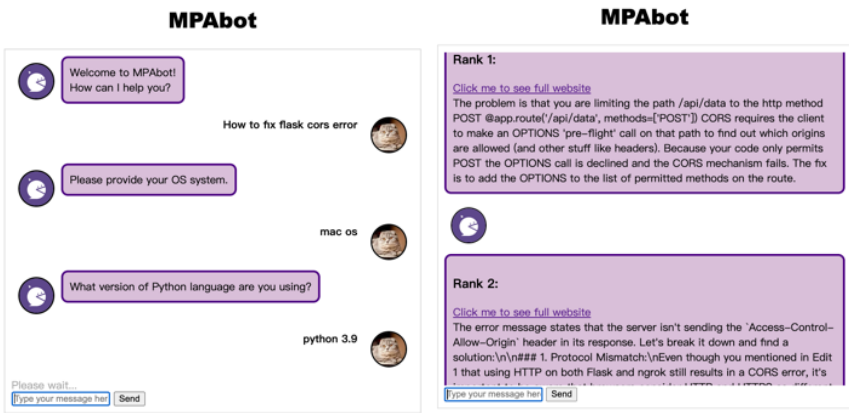

Fig. 2. Example MPAbot conversation.

3.2 System Architecture

The aim of MPAbot is to integrate data from multiple sources. The scalability and independence of the system were enhanced by adopting a microservices architecture consisting of seven services (MPAbot UI, the assistant bot, the MPAbot Tools service, the Data Controller service) and three Web Parsing services. The microservices architecture of the MPAbot system is illustrated in Fig. 3.

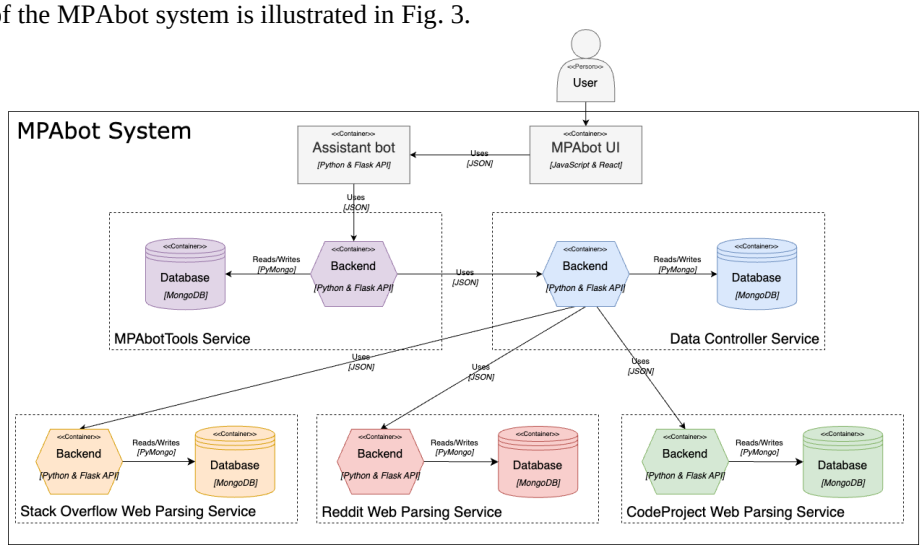

Fig. 3. Microservices architecture of MPAbot.

The MPAbot UI manages user interactions in collaboration with the Assistant Bot, which collects key information from users and simulates conversations to retrieve recommended answers from the various online communities. The MPAbot Tools service conducts multi-source block analysis, whereas the Data Controller service unifies the data collected by the individual web parsing services. We also designed a general web parsing service template for each source to facilitate the integration of new sources into MPAbot

by developers.

The underlying technology stack of MPAbot consists of multiple integrated components. The backend API is built using the Python Flask framework, providing RESTful endpoints for microservices communication. The frontend interface is developed using React.js with Bootstrap components, ensuring a responsive and user-friendly experience. The system employs Docker containerization technology to implement the microservices architecture, enabling scalable and isolated deployment of each service component. To enhance response ranking accuracy, MPAbot integrates GPT-3.5 generative AI technology, which helps optimize the ordering of retrieved answers through semantic analysis and context understanding. This technical architecture allows efficient multi-source data integration while maintaining system modularity and scalability [9].

3.3 Data Collection and Cleaning

The data collection process is performed by web parsing services. This study used the Google Search API to gather relevant discussion posts based on user queries from developer communities. The API makes it possible to set a date range to ensure access to up-to-date information. Note that we limited the number of retrieved responses to ten per user query. After obtaining web addresses, the website content undergoes parsing to facilitate further analysis. We also used external APIs, including the Stack Exchange API, to ensure stability and efficiency in the data collection process. Fig. 4 illustrates the web parsing process flow.

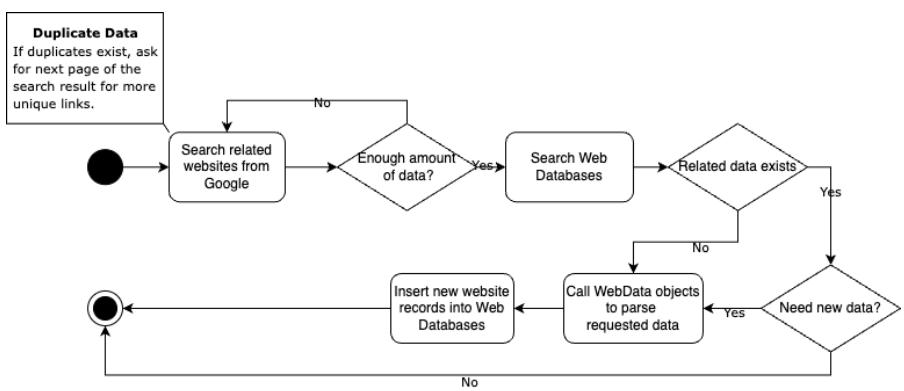

Fig. 4. Web parsing process flow.

The Data Controller standardizes each data source into a unified format. The unifying process includes concatenating sub-blocks, such as comments in CodeProject, that include meaningful information. Since the platforms have various formats when parsing from websites, an adaption preprocess reduces the running time of analysis. It also calculates post-information-based criteria, including the original web score, webpage traffic rate, and the reputation of the post author, and performs normalization to ensure consistency in scale. MPAbot Tools then applies natural language processing techniques to minimize noise and remove redundant tokens, as detailed in the following section.

The databases are categorized into three layers: raw, unified, and analyzed. The re-

lationship between each layer is displayed in Fig. 3. First, when a parsing service parses a webpage, the content is stored directly in its own raw database. Next, the unified database only exists in the data collector service, which stores the unified data. This unified data represents a standardized format for question-and-answer blocks collected from all three sources, with the data structure primarily modeled after Stack Overflow's format while retaining only the essential fields needed for subsequent processing. The data from Reddit and CodeProject are then mapped to this standardized structure, ensuring consistent data representation across all sources. Last, the analyzed database is located in the MPAbot Tools service, which stores each block's rankings and reduced terms. Each ranking maps to a user question and can be reused when MPAbot receives a similar question. The database hierarchy also provides a caching mechanism to enhance MPAbot's response time.

3.4 Multi-source Block Analysis

Multi-source block analysis is implemented in two phases: text analysis and rank recommendation. Text analysis involves natural language preprocessing and LDA topic modeling, during which we calculate two content-based criteria: block score and question similarity. The rank recommendation involves multi-criteria decision analysis (MCDA) [10, 11] and LLM ranking assistance. The blocks are ranked by combining the three post-information-based criteria with the two content-based criteria, after which the top 10 answers are extracted as responses. The overall process is illustrated in Fig. 5.

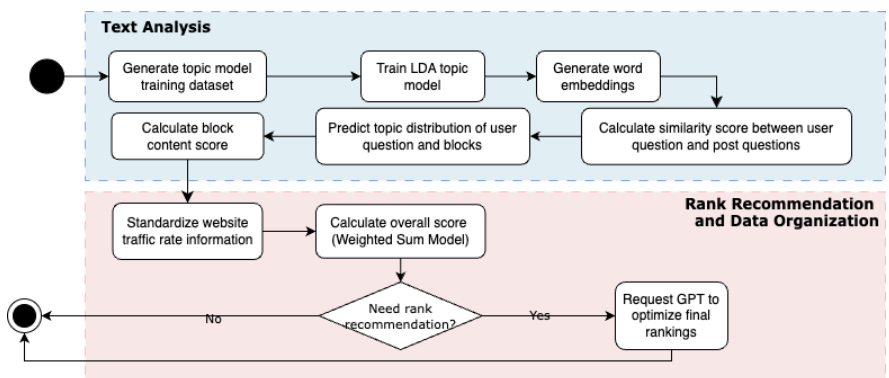

Fig. 5. Schematic diagram showing the multi-source block analysis process flow.

3.5 NLP Data Preprocessing

A multi-step data cleaning process is used to ensure the quality of the training data. The overall process is illustrated in Fig. 6. First, HTML markup and code sections are removed to extract plain text. The spaCy library [12] is then used for tokenization, wherein the text is broken down into individual words and punctuation marks. SpaCy is more user-friendly and efficient than NLTK, while remaining up-to-date with the latest advances in NLP technology.

Lemmatization is performed using spaCy to ensure that processing focuses on meaningful words and morphological variations by reducing inflected words to their base form. Common stop words are then filtered out to retain only common nouns (singular and plural)

and proper nouns based on part-of-speech (POS) tags. The initial filtering function of the Google Search API is used to retrieve discussion posts of high similarity, thereby allowing the exclusion of high-frequency verbs, which might otherwise hinder topic differentiation. We maintain a whitelist of specific terms that might be initially classified as stop words but are important in the context of programming contexts to prevent their accidental removal.

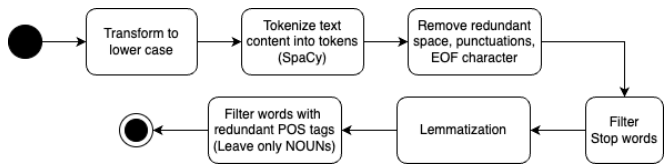

Fig. 6. Schematic illustration showing natural language processing flow.

3.6 LDA Model Training

Gensim [13] is an open-source library for topic modeling, which includes algorithms for unsupervised machine learning based on plain text, including Latent Semantic Indexing (LSI), Latent Dirichlet Allocation (LDA), and Word2Vec. The LDA algorithm [14] is available in batch and online variants, both of which demonstrate strong performance. This study selected online LDA, which excels in handling smaller batches and achieves convergence more rapidly than batch LDA.

The LDA model was trained on the block contents of ten documents. The Gensim API was used to create a dictionary for mapping tokens to their IDs. The dictionary was subsequently used to convert the dataset into a list of (token_id, token_counts) pairs in the bag-of-words (BoW) format. The model extracted three topics from the corpus during training.

3.7 Word Embedding

TensorFlow Hub is a repository of pre-trained machine learning models, which provides a powerful token-based word embedding model trained on the vast English Wikipedia corpus [15]. This model goes beyond simple word representation by offering both word and sentence embeddings. Word embeddings provide several advantages, including (1) the ability to represent words as dense vectors rather than sparse one-hot encodings with numerous empty values; (2) the ability to overcome the limitations of limited training data; and (3) the ability to capture the semantic features of words.

Word embedding was used to calculate the degree of similarity between the user question and post questions, and topic vectors were used to strengthen the relationship between questions and answers to facilitate the extraction of relevant keywords from blocks. This process results in four key vector representations: (1) user question sentence vector *q_vec*; (2) question title sentence vectors *t_vec*; (3) block terms word vectors *w_vec*; and (4) topic vectors.

3.8 Calculating Content-Based Criteria

Our development process assumed that all documents related to a given topic are likely to be highly relevant to each other. Therefore, it should be possible to quantify the

relationship between a user question and a block by calculating the probability that they both belong to the same topic. Based on this assumption, we defined a **block score** as the probability that the block belongs to the same topic as the user query, which can be represented in vector notation as follows,

$$block\ score = \vec{q} \cdot \vec{b_i} \tag{1}$$

where *q* represents the topic distribution of the user query and *b* represents the topic distribution of each block.

We then leverage the well-established cosine similarity method [16, 17] to assess the degree of similarity between the user question and the question title. The user query and blocks are transformed into word embedding vectors, after which the similarity is calculated as follows,

$$question\ similarity = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|}. \tag{2}$$

3.9 MCDA Block Rank Recommendation

The five criteria are used to calculate a ranking score using the weighted sum model in which the weighting of each source can be modified. Based on statistical analysis of a weighting table for Stack Overflow, Reddit, and CodeProject, we selected Stack Overflow as the baseline (see Table 1).

Table 1. Weighting table of the three data sources.

Source	Web Score	Traffic Flow	Author Reputation	Block Score	Question Similarity
Stack Overflow	0.2	0.1	0.1	0.5	0.1
Reddit	0.1	0.2	0.1	0.5	0.1
CodeProject	0	0	0.1	0.5	0.4

3.10 Improving Ranking Using LLM

The development of Large Language Models (LLMs) has significantly enhanced the effectiveness of recommendation and question-answer systems [19]. LLMs are more effective when supplemented with an additional database rather than relying solely on pretrained knowledge, a process known as Retrieval-Augmented Generation (RAG) [18].

```
[
  {
    "role": "system",
    "content": "You are a professional programmer. You will be provided with mul
tiple unstructured documents about {q}, and your task is to optimize the rank of
these documents."
  },
  {
    "role": "user",
    "content": "My question is {q}. Can you suggest a better rank for these docu
ments?: {documents}"
  }
]
```

Fig. 7. Example ChatGPT prompt.

We aimed to enhance the efficacy of MPAbot responses by employing Retrieval-Augmented Generation (RAG) in conjunction with GPT as a ranking adjustment. ChatGPT was tasked with re-ranking our top 20 ranked blocks (real-time data) and providing a summary of each. Based on our internal experiments, the number of retrieved blocks varies significantly across different questions, ranging from approximately 100 to 3000 blocks. Since blocks ranked beyond the top 20 typically showed minimal relevance to the queries, we limited our ranking adjustment process to only the top 20 blocks. Figure 7 presents an example prompt for a given task. With the assistance of ChatGPT, we addressed the issue of shorter content receiving lower scores due to an insufficient number of key tokens. This optimized ranking scheme achieves a more balanced evaluation by ensuring that concise but relevant content is appropriately weighted, thereby improving the overall quality and diversity of recommendations in the final output.

Through our internal experiments, we found that GPT-4 frequently generated structurally inconsistent results, inserted extraneous information, and often engaged in unnecessary follow-up questioning, likely due to its more sophisticated natural language capabilities. Therefore, we opted to use GPT-3.5, which proved more reliable in adhering to our specific task requirements and producing direct, consistent outputs without excessive interaction.

4. EXPERIMENTS

The experiments conducted to evaluate the proposed method focused on two key factors: parameter tuning and performance. Parameter tuning involved adjusting the number of posts, POS tags, and LDA model hyperparameters when applied to a dataset containing 10 documents and 327 unique tokens. Model performance was assessed using two evaluation metrics: perplexity [21] and U_mass coherence [22]. Perplexity measures the model's effectiveness in interpreting topics within samples, where lower perplexity indicates better performance in capturing the underlying topical structures. U_mass coherence evaluates the quality of topics, with higher coherence scores indicating that topics are more coherent and meaningful.

4.1 LDA Hyperparameters

According to the official Gensim LDA documentation, hyperparameters can have a significant impact on performance. In this section, we outline the outcomes of fine-tuning the LDA topic model by varying several critical parameters (Eta, Alpha, and topic number) with the aim of determining the optimal configuration for our data [19].

4.2 Eta & Alpha

The hyperparameter tuning process for Eta and Alpha consisted of multiple stages. In the initial stage, we independently tested eight values ranging from 0.0001 to 1000 for both parameters, which led us to identify four promising values (0.01, 0.1, 1, 10) for further analysis. Subsequently, we conducted pairwise experiments with these values, creating sixteen Eta-Alpha combinations (see Table 2). Results where Eta equals 0.01 were excluded due to their overly high perplexity scores.

Table 2. Hyperparameter-tuning results: Eta and Alpha.

Eta	Alpha	Coherence	Perplexity	Eta	Alpha	Coherence	Perplexity
0.01	0.01	2.34	182.01	1	0.01	2.64	39.02
0.01	0.1	3.71	7453.06	1	0.1	0.60	38.71
0.01	1	1.76	6619.68	1	1	0.64	39.70
0.01	10	1.49	7141.60	1	10	2.11	42.05
Avg.		*2.325*	*5349.088*	*Avg.*		*1.498*	*39.870*
0.1	0.01	0.73	43.78	10	0.01	0.72	45.34
0.1	0.1	1.90	46.63	10	0.1	2.04	45.54
0.1	**1**	**5.92**	**46.27**	10	1	5.60	46.54
0.1	10	4.54	51.50	10	10	1.51	48.80
Avg.		*3.270*	*47.045*	*Avg.*		*2.468*	*46.555*

According to the definitions, higher Eta indicates more terms in one topic, while higher Alpha implies less distinction between topics. Although different combinations of Eta and Alpha values led to varying term distributions and topic distinctions, all remaining parameter combinations produced acceptable clustering results. Therefore, we prioritized coherence scores as our primary selection criterion, ultimately determining (Eta, Alpha) = (0.1, 1) as the optimal LDA configuration.

4.3 Topic Number

The topic number is another key LDA hyperparameter that affects model performance. We varied the topic number from 2 to 9 and evaluated their performance using both perplexity and coherence metrics (see Table 3). While coherence was our primary criterion for Eta and Alpha tuning, we found that perplexity served as a more reliable indicator for determining the optimal number of topics in our specific use case. This is because perplexity directly measures how well the model generalizes to unseen data, which is crucial for a Q&A system handling diverse programming queries. The model with three topics achieved the lowest perplexity (44.190), indicating this configuration best captured the underlying structure of programming-related discussions. Although models with more topics showed different coherence scores, the perplexity scores suggested that increasing the number of topics beyond three did not improve the model's generalization capability.

Table 3. Comparison of model train on various topic numbers.

Topic #	Perplexity	Coherence
2	45.159	–0.751
3	**44.190**	**–3.191**
4	47.205	–0.688
5	47.044	–0.737
6	47.684	–1.839
7	47.131	–3.338
8	48.834	–2.947
9	49.718	–4.466
avg.	*47.121*	*–2.245*

4.4 MPAbot Efficacy

This experiment evaluated the effectiveness of our multi-source block analysis scheme by comparing three approaches: MPAbot-ranked, GPT-ranked, and GPT-generated. The MPAbot-ranked method was the original response from the multi-source block analysis without LLM ranking assistance. The GPT-ranked method re-ranked the top 20 blocks based on the MPAbot rankings. The GPT-generated response was produced by prompting GPT directly. We instructed the model to generate two different answers to provide diverse perspectives, ensuring a comprehensive comparison of the three methods.

The test data was sourced from the Python.org Programming FAQ website. We randomly selected 20 questions (please refer to the Appendix section) to construct the test dataset and generate answers using the three methods. To gain a thorough understanding of the results, we employed questionnaires to collect user ratings and the Kruskal-Wallis test (H-test) [20] to check for significant differences in performance between each method.

1) User Ranking

We invited six graduate students who majored in computer science to participate in a double-blind test, where they were tasked with evaluating and scoring the answers based solely on their quality, without knowing the source of the responses or any contextual information about the ranking method used. For each question, we provided six answers: MPAbot-ranked (R1, R2: the top two results from MPAbot's ranking), GPT-adjusted (R1, R2: the top two results after GPT's adjustment), and GPT-generated (A1, A2: two answers generated by GPT). The answers were presented in random order and anonymized to prevent bias. Users evaluated the answers based on relevance and accuracy, assigning scores from 5 (most preferred) to 1 (least preferred). This approach ensured a fair comparison between MPAbot and GPT responses. We then analyzed the results to determine the best-performing method. The outcomes are presented in Table 4.

Table 4. Efficiency test: Average of user ratings for each question and method.

#	MPAbot Ranked			GPT Ranked			GPT Generated		
	R1	R2	Max	R1	R2	Max	A1	A2	Max
1	2.5	4.5	*4.5*	4.0	4.2	*4.2*	4.2	3.5	*4.2*
2	3.8	4.7	*4.7*	4.2	3.7	*4.2*	4.2	3.5	*4.2*
3	3.0	4.7	*4.7*	4.5	4.7	*4.7*	4.2	3.2	*4.2*
4	4.8	4.2	*4.8*	4.3	4.3	*4.3*	4.3	3.8	*4.3*
5	4.3	4.5	*4.5*	4.0	4.3	*4.3*	4.5	3.8	*4.5*
6	3.0	2.7	*3.0*	4.3	4.7	*4.7*	4.5	3.5	*4.5*
7	3.3	2.2	*3.3*	5.0	3.8	*5.0*	4.5	3.5	*4.5*
8	1.3	1.3	*1.3*	2.5	2.3	*2.5*	4.2	3.5	*4.2*
9	2.0	2.5	*2.5*	3.7	2.8	*3.7*	4.5	3.5	*4.5*
10	4.0	3.5	*4.0*	4.0	4.7	*4.7*	4.0	3.5	*4.0*
11	1.8	4.2	*4.2*	4.7	2.7	*4.7*	3.5	3.0	*3.5*
12	4.5	4.5	*4.5*	4.0	4.0	*4.0*	3.8	3.2	*3.8*
13	3.8	2.7	*3.8*	4.7	2.5	*4.7*	3.7	2.8	*3.7*
14	2.8	3.0	*3.0*	4.3	3.5	*4.3*	3.7	2.5	*3.7*
15	3.7	4.2	*4.2*	3.7	4.7	*4.7*	3.8	3.0	*3.8*
16	4.0	4.3	*4.3*	4.7	3.3	*4.7*	3.7	2.8	*3.7*
17	4.3	4.5	*4.5*	4.3	4.7	*4.7*	4.0	3.7	*4.0*
18	4.5	3.7	*4.5*	4.8	4.2	*4.8*	4.5	4.0	*4.5*
19	2.7	4.7	*4.7*	4.8	4.8	*4.8*	4.7	4.3	*4.7*
20	3.7	4.0	*4.0*	4.0	3.8	*4.0*	4.3	3.7	*4.3*
Avg.	3.4	3.7	*4.0*	4.2	3.9	*4.4*	4.1	3.4	*4.1*

2) H-Test

The performance of the various methods was assessed statistically by applying the Kruskal-Wallis H-test[6] to the user ratings. Overall, the MPAbot-ranked and GPT-ranked responses were consistently superior to the GPT-generated responses. Thus, we selected the highest score from each question as the representative value when applying the H-test to the three groups. The results revealed a significant difference between the groups, with mean rank scores of 27.7 for MPAbot-ranked, 38.52 for GPT-ranked, and 25.27 for GPT-generated. Both MPAbot-ranked and GPT-ranked surpass GPT-generate results; this indicates that users receive more satisfactory responses while using MPAbot. Furthermore, with the assistance of GPT on ranking adjustment, MPAbot excels in pure GPT with a 52.43% higher mean rank score.

In this analysis, the GPT-ranked method significantly outperformed the GPT-generated method. The MPAbot-ranked method performed slightly better than the GPT-generated. These findings indicate that the proposed method enhanced response quality. They also demonstrate the benefits of integrating multi-source analysis with automated ranking to improve performance.

4.5 Threats to Validity

We identify several potential threats to the validity of our study across three main categories:

1. Internal validity threats primarily stem from our experimental setup: (1) the limited number of participants (six users) in the evaluation may not represent all skill levels, and (2) users might be influenced by learning effects during the assessment process.
2. External validity concerns include: (1) the selected data sources might not comprehensively cover all programming resources; and (2) our focus on Python-related questions may not reflect system performance with other programming languages.
3. Construct validity issues related to the experimental approach: (1) the chosen metrics (perplexity and U_mass coherence) might not capture all aspects of topic model quality; and (2) user rating subjectivity could affect evaluation results.

To mitigate these threats, we implemented randomized question ordering, anonymous answer presentation, and statistical validation through H-test. Future work should address these limitations through larger-scale evaluations and broader programming language coverage.

5. CONCLUSIONS AND FUTURE WORK

This paper presents a novel Chatbot system designed to improve the learning experience for programmers and learners seeking answers within developer communities. The proposed MPAbot system combines LDA topic modeling with word embeddings and similarity measures to rank answer blocks. This approach provides users with a set of ranked recommendations on a single webpage, eliminating the need to navigate across multiple pages. The ability to deliver relevant information of high accuracy efficiently could significantly simplify the learning process for users.

[6] https://www.statskingdom.com/kruskal-wallis-calculator.html

We have identified several key improvements to enhance the effectiveness of the MPAbot system. First, developing an intuitive, user-friendly interface would increase accessibility and reduce the learning curve for users of all skill levels. Second, enhancing response times is essential for ensuring a seamless user experience, particularly for programmers who need prompt answers to maintain workflow continuity. Additionally, future work should address current limitations through (1) conducting larger-scale evaluations with more diverse user groups; (2) expanding programming language coverage beyond Python; and (3) exploring dynamic content updates to maintain system relevance as programming communities evolve. These improvements would strengthen MPAbot's capability as a robust learning tool while ensuring its adaptability to users' changing needs.

REFERENCES

1. T. Thindwa, W. D. Chawinga, and G. Dube, "Information-seeking behaviour of security studies students: A case study," *South African Journal of Information Management*, Vol. 21, 2019, pp. 1-10.
2. J. Xiao, "Tutors' influence on distance language students' learning motivation: voices from learners and tutors," *Distance Education*, Vol. 33, 2012, pp. 365-380.
3. A. Grzywaczewski and R. Iqbal, "Task-specific information retrieval systems for software engineers," *Journal of Computer and System Sciences*, Vol. 78, 2012, pp. 1204-1218.
4. V. Subramanian, N. Ramachandra, and N. Dubash, "TutorBot: contextual learning guide for software engineers," in *Proceedings of IEEE/ACM International Workshop on Bots in Software Engineering*, 2019, pp. 16-17.
5. A.-C. Shau, Y.-C. Liang, W.-J. Hsieh, X.-L. Lin, and S.-P. Ma, "PSAbot: A chatbot system for the analysis of posts on stack overflow," in *Proceedings of IEEE International Conference on Software Engineering Education and Training*, 2023, pp. 137-141.
6. P. Suta, P. Mongkolnam, C. C. Fung, and J. H. Chan, "Matching question and answer using similarity: an experiment with stack overflow," in *Proceedings of IEEE International WIE Conference on Electrical and Computer Engineering*, 2018, pp. 51-54.
7. N. Zhang, Q. Huang, X. Xia, Y. Zou, D. Lo, and Z. Xing, "Chatbot4qr: Interactive query refinement for technical question retrieval," *IEEE Transactions on Software Engineering*, Vol. 48, 2020, pp. 1185-1211.
8. J. Sawicki, M. Ganzha, M. Paprzycki, and A. Bădică, "Exploring usability of reddit in data science and knowledge processing," *arXiv Preprint*, 2021, arXiv:2110.02158.
9. S.-P. Ma, C.-Y. Li, W.-T. Lee, and S.-J. Lee, "Microservice migration using strangler fig pattern and domain-driven design," *Journal of Information Science and Engineering*, Vol. 38, 2022, pp. 1285-1303.
10. S. Greco, J. Figueira, and M. Ehrgott, *Multiple Criteria Decision Analysis*, Springer, 2016.
11. V. Belton and T. Stewart, *Multiple Criteria Decision Analysis: An Integrated Approach*, Springer Science & Business Media, 2012.
12. D. Altinok, *Mastering spaCy: An End-to-End Practical Guide to Implementing NLP Applications using the Python Ecosystem*," Packt Publishing Ltd., UK, 2021.

13. R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," in *Proceedings of the LREC Workshop on New Challenges for NLP Frameworks*, 2010, pp. 46-50.
14. M. Hoffman, F. Bach, and D. Blei, "Online learning for latent dirichlet allocation," *Advances in Neural Information Processing Systems*, Vol. 23, 2010, pp. 1-9.
15. T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv Preprint*, 2013, arXiv:1301.3781.
16. N. Dehak, R. Dehak, J. R. Glass, D. A. Reynolds, and P. Kenny, "Cosine similarity scoring without score normalization techniques," *Odyssey*, Vol. 15, 2010, pp. 1-5.
17. A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan, "Cosine similarity to determine similarity measure: Study case in online essay assessment," in *Proceedings of International Conference on Cyber and IT Service Management*, 2016, pp. 1-6.
18. P. Lewis, "Improving neural question answering with retrieval and generation," PhD Thesis, Department of Computer Science, University College London, 2022.
19. W. Zhao *et al.*, "A heuristic approach to determine an appropriate number of topics in topic modeling," *BMC Bioinformatics*, Vol. 16, 2015, pp. 1-10.
20. W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, Vol. 47, 1952, pp. 583-621.

APPENDIX: 20 Questions in the Experiment

1. How can I create a stand-alone binary from a Python script?
2. Is there a source code level debugger with breakpoints, single-stepping, *etc.*?
3. Are there coding standards or a style guide for Python programs?
4. How do I copy an object in Python?
5. Why are default values shared between objects?
6. What is the difference between arguments and parameters?
7. Is there an equivalent to Perl's chomp() for removing trailing newlines from strings?
8. Why does -22 // 10 return –3?
9. How do I use strings to call functions/methods?
10. My program is too slow. How do I speed it up?
11. What is the most efficient way to concatenate many strings together?
12. What is a negative index?
13. Why does a_tuple[i] += ['item'] raise an exception when the addition works?
14. I want to do a complicated sort: can you do a Schwartzian Transform in Python?
15. How do I get a list of all instances of a given class?
16. How do I create static class data and static class methods?
17. How can a subclass control what data is stored in an immutable instance?
18. When I edit an imported module and reimport it, the changes don't show up. Why does this happen?
19. How do I find the current module name?
20. How can I have modules that mutually import each other?

An-Chi Shau (邵安祺) received her 1st Master's degree from the Department of Computer Science and Engineering, National Taiwan Ocean University, Taiwan, in 2024. Her research interests include software engineering, chatbot architecture, and natural language processing.

Shang-Pin Ma (馬尚彬) received his Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2007. Dr. Ma is currently a Professor in the Department of Computer Science and Engineering at National Taiwan Ocean University. His research interests include service-oriented computing, software engineering, mobile computing, and chatbot architecture.