ShellFusion: Answer Generation for Shell Programming Tasks via Knowledge Fusion

Neng Zhang[1,*], Chao Liu[2], Xin Xia[3], Christoph Treude[4], Ying Zou[5], David Lo[6], Zibin Zheng[1]
[1]School of Software Engineering, Sun Yat-sen University, China
[2]School of Big Data and Software Engineering, Chongqing University, China
[3]Software Engineering Application Technology Lab, Huawei, China
[4]School of Computing and Information Systems, University of Melbourne, Australia
[5]Department of Electrical and Computer Engineering, Queen's University, Canada
[6]School of Information Systems, Singapore Management University, Singapore
{zhangn279,zhzibin}@mail.sysu.edu.cn,liu.chao@cqu.edu.cn,xin.xia@acm.org,christoph.treude@unimelb.edu.au
ying.zou@queensu.ca,davidlo@smu.edu.sg

ABSTRACT

Shell commands are widely used for accomplishing tasks, such as network management and file manipulation, in Unix and Linux platforms. There are a large number of shell commands available. For example, 50,000+ commands are documented in the Ubuntu Manual Pages (MPs). Quite often, programmers feel frustrated when searching and orchestrating appropriate shell commands to accomplish specific tasks. To address the challenge, the shell programming community calls for easy-to-use tutorials for shell commands. However, existing tutorials (e.g., TLDR) only cover a limited number of frequently used commands for shell beginners and provide limited support for users to search for commands by a task.

We propose an approach, i.e., ShellFusion, to automatically generate comprehensive answers (including relevant shell commands, scripts, and explanations) for shell programming tasks. Our approach integrates knowledge mined from Q&A posts in Stack Exchange, Ubuntu MPs, and TLDR tutorials. For a query that describes a shell programming task, ShellFusion recommends a list of relevant shell commands. Specifically, ShellFusion retrieves the top-*n* Q&A posts with questions similar to the query and detects shell commands with options (e.g., `ls -t`) from the accepted answers of the retrieved posts. Next, ShellFusion filters out irrelevant commands with descriptions in MP and TLDR that share little semantics with the query, and further ranks the candidate commands based on their similarities with the query and the retrieved posts. To help users understand how to achieve the task using a recommended command, ShellFusion generates a comprehensive answer for each command by synthesizing knowledge from Q&A posts, MPs, and TLDR. Our evaluation of 434 shell programming tasks shows that ShellFusion significantly outperforms Magnum (the state-of-the-art natural language-to-Bash command approach) by at least 179.6% in terms of MRR@K and MAP@K. A user study conducted with 20 shell programmers further shows that ShellFusion can help users address programming tasks more efficiently and accurately, compared with Magnum and DeepAns (a recent answer recommendation baseline).

CCS CONCEPTS

• **Software and its engineering** → *Recommender systems.*

KEYWORDS

Shell Programming, Answer Generation, Knowledge Fusion

ACM Reference Format:
Neng Zhang[1,*], Chao Liu[2], Xin Xia[3], Christoph Treude[4], Ying Zou[5], David Lo[6], Zibin Zheng[1]. 2022. ShellFusion: Answer Generation for Shell Programming Tasks via Knowledge Fusion. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510003.3510131

1 INTRODUCTION

In Unix and Linux platforms, shell programming is widely used to accomplish many tasks such as network management and file manipulation. A large number of shell commands have been developed to support shell programming. For example, Ubuntu, a mainstream branch of Linux, has more than 50,000 commands documented in its official Manual Pages (MPs) [12]. It is extremely difficult for users to know and remember all the commands [21].

A great amount of manual effort has been devoted to creating easy-to-use tutorials for shell commands that are frequently used by programmers [2, 3, 11, 34, 36]. The TLDR (stands for "Too Long, Don't Read") project [11] is one of the popular tutorials. Currently, TLDR contains approximately 2,000 commands. For each command, TLDR records a summary of the command's functionality and several example tasks with the corresponding scripts (Section 2.1.5). Although the existing tutorials can help users learn and use shell commands to some extent, they only cover a limited number of frequently used commands with a few examples for shell beginners, and have no effective mechanisms for users to search commands by a task. Recently, several approaches are proposed to translate natural language tasks to Bash commands [15, 29] using a manually created dataset, NL2Bash. NL2Bash contains approximately 9,305 task-command pairs that cover 102 commonly

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9221-1/22/05...$15.00
https://doi.org/10.1145/3510003.3510131

used commands. The approaches cannot address many tasks that require commands outside the dataset.

Q&A communities, e.g., Stack Overflow (SO) [9], have accumulated millions of questions and answers about various problems and become a knowledge repository for users [49]. There are many Q&A posts related to shell programming in four popular Q&A communities, i.e., SO, Ask Ubuntu (AU) [1], Unix & Linux (UL) [13], and Super User (SU) [10]. Such posts can be leveraged to design automatic approaches for addressing shell programming tasks without being limited by the manually created task-command dataset.

In this paper, we propose an approach, namely ShellFusion, to automatically generate answers for shell programming tasks by integrating knowledge mined from Q&A posts, Ubuntu MPs, and TLDR tutorials. We first collect shell-related questions from four Q&A communities: AU, UL, SU, and SO, and extract knowledge of shell commands from Ubuntu MPs and TLDR tutorials. We then index the questions using Lucene [30], an efficient text search engine that implements the ranking function BM25. We also build a word Inverse Document Frequency (IDF) [16] vocabulary and a word embedding model using the questions. Given a query that describes a shell programming task, ShellFusion determines a list of relevant shell commands to recommend. Specifically, it retrieves the top-*n* questions similar to the query using Lucene search and a word embedding model-based method. It then detects shell commands with options (e.g., `ls -t`) from the accepted answers of the similar questions and filters out irrelevant commands which have descriptions in MP and TLDR that share little semantics with the query. Next, ShellFusion ranks the candidate commands based on their similarities with the query and the similar questions. To help users understand how to address the query using a recommended command, ShellFusion generates a comprehensive answer for the command by synthesizing 1) the official MP summary, 2) the most similar TLDR task-script pair, 3) the top-3 similar questions with accepted scripts that use the command, and 4) the explanations about the options of the command used in the scripts.

To evaluate ShellFusion, we create 434 shell programming tasks from the titles of shell-related questions. We compare ShellFusion with Magnum [15], the state-of-the-art approach for translating natural language tasks to Bash commands. ShellFusion significantly outperforms Magnum by at least 179.6% in terms of the MRR@K and MAP@K metrics. In addition to the widely used Q&A posts, ShellFusion integrates two specialized information sources: the official MPs and the unofficial TLDR tutorials. We evaluate the contributions of the two information sources by testing three variants of ShellFusion (Section 3.3). The results show that both MPs and TLDR contribute to ShellFusion, and TLDR has more contributions than MPs. We further conduct a user study with 20 shell programmers on ten programming tasks. The results show that ShellFusion can help users address shell programming tasks more efficiently and accurately, compared with Magnum and DeepAns [22] (an answer recommendation model that uses a deep learning technique).

The main contributions of this paper are outlined below:

- We propose ShellFusion to generate comprehensive answers for shell programming tasks by integrating knowledge mined from Q&A posts, MPs, and TLDR. To the best of our knowledge, ShellFusion is the first work on both command recommendation and answer generation for shell programming tasks using a knowledge fusion method.
- Our work advances software engineering research by fusing and filtering knowledge from Q&A sites based on both formal and informal software documentation.
- We propose a method for identifying shell commands with options from Q&A posts.
- Our quantitative evaluation and user study demonstrate the effectiveness and practicality of ShellFusion, compared with the state-of-the-art baselines.
- We release the source code of ShellFusion and the experiment dataset on GitHub [7] to help other researchers replicate and extend our study.

The rest of the paper is structured as follows. Section 2 describes the details of ShellFusion. Section 3 describes the experimental setup for evaluating ShellFusion. Section 4 presents the experiment results. Section 5 describes the user study. Section 6 discusses the threats to validity of this work. Section 7 reviews the related work. Section 8 concludes the paper and discusses future work.

2 APPROACH

Fig. 1 shows an overview of ShellFusion, consisting of two components: ① *Offline Processing* that extracts shell-related questions from Q&A communities and mines knowledge of shell commands from Ubuntu MPs and TLDR tutorials. We index the questions using Lucene and build a word IDF vocabulary and a word embedding model from the questions; and ② *Online Answer Generation* that generates answers for a query describing a shell programming task. We retrieve the top-*n* questions similar to the query and synthesize multi-source knowledge of candidate shell commands detected from the accepted answers of the similar questions.

2.1 Offline Processing

2.1.1 Shell-Related Question Collection. There are four main Q&A communities in Stack Exchange that contain posts related to shell programming, i.e., AU, UL, SU, and SO. We download the official data dumps of the communities released on March 1, 2021 [8]. The Q&A posts of each community are stored in an XML file, *Posts.xml*. The questions in AU and UL are all relevant to shell programming. We extract 371,801 and 198,939 shell-related questions from AU and UL, respectively. The questions in SU and SO are not all relevant to shell programming. We manually examine the technical tags assigned to the questions of both communities and determine 252 shell-related tags that contain 'shell', 'bash', 'sh', 'unix', 'linux', or 'ubuntu'. Using the tags, we collect 87,587 and 539,171 shell-related questions from SU and SO, respectively.

A question may have several answers provided by different developers worldwide. One of the answers can be accepted by the asker of the question. As confirmed by the asker, the shell commands mentioned in the accepted answer could be used to address the question. ShellFusion extracts candidate shell commands for a query from the accepted answers of questions similar to the query. Table 1 lists the statistics of our collected shell-related questions. In total, there are 537,129 questions with accepted answers.

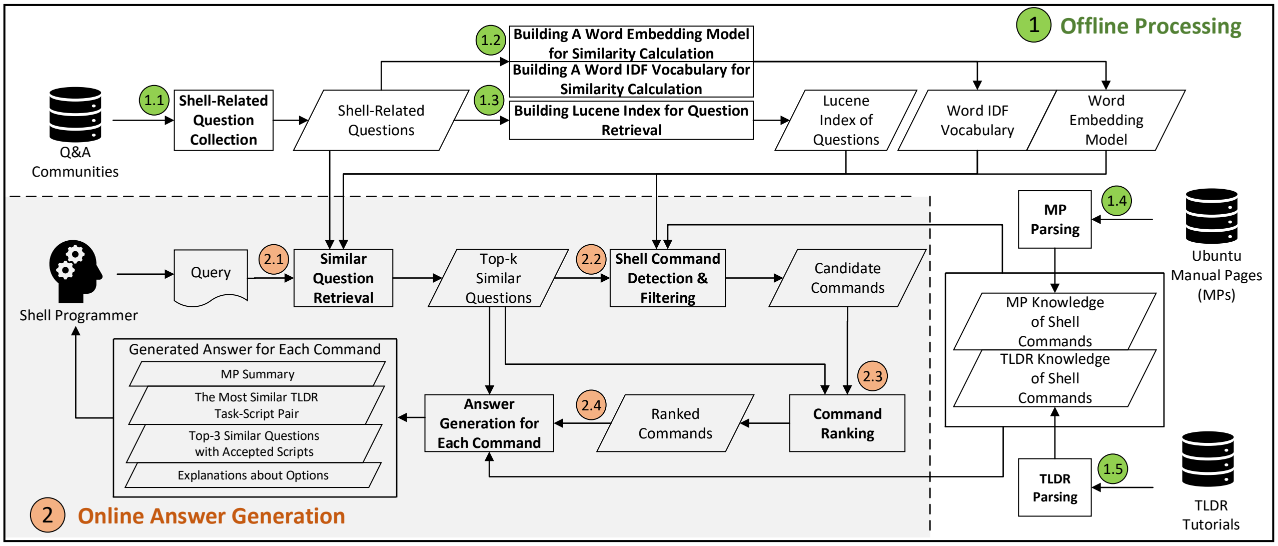

1 Offline Processing
Q&A Communities
1.1
Shell-Related Question Collection
Shell-Related Questions
1.2
Building A Word Embedding Model for Similarity Calculation
Building A Word IDF Vocabulary for Similarity Calculation
1.3
Building Lucene Index for Question Retrieval
Lucene Index of Questions
Word IDF Vocabulary
Word Embedding Model
MP Parsing
1.4
Ubuntu Manual Pages (MPs)
MP Knowledge of Shell Commands
TLDR Knowledge of Shell Commands
TLDR Parsing
1.5
TLDR Tutorials
Shell Programmer
Query
2.1
Similar Question Retrieval
Top-k Similar Questions
2.2
Shell Command Detection & Filtering
Candidate Commands
2.3
Command Ranking
Ranked Commands
2.4
Answer Generation for Each Command
Generated Answer for Each Command
MP Summary
The Most Similar TLDR Task-Script Pair
Top-3 Similar Questions with Accepted Scripts
Explanations about Options
2 Online Answer Generation

Figure 1: Approach overview of ShellFusion

Table 1: Statistics of shell-related questions

Q&A community	# Shell-related questions	# Shell-related questions with accepted answers
Ask Ubuntu (AU)	371,801	122,729
Unix & Linux (UL)	198,939	95,343
Stack Overflow (SO)	539,171	281,134
Super User (SU)	87,587	37,923
Total	**1,197,498**	**537,129**

2.1.2 *Building A Word Embedding Model and A Word IDF Vocabulary for Similarity Calculation.*

Using the shell-related questions, we build a word IDF vocabulary and a word embedding model, which are subsequently used to calculate the similarity between a question (or a shell command or a TLDR task) and a query. The word embedding model is the basic model for calculating word similarities, while the word IDF vocabulary is used to weight the importance of words. The IDF of a word is the inverse of the number of questions that contain the word. The more questions in which a word appears, the lower the word's IDF is, meaning that the less likely the word carries important information.

We build a text corpus by processing the questions using three steps: 1) removing long code snippets enclosed in HTML tag ⟨*pre*⟩; 2) removing stopwords (e.g., 'a' and 'the') based on the stopword list in the NLTK toolkit [17]; and 3) reducing each word to its root form (aka. stemming) by applying the Porter stemmer in NLTK. We use the corpus to train a word embedding model by applying the word2vec [32] module (with default parameter settings) in Gensim [5]. We compute the IDF value of each word in the corpus.

2.1.3 *Building Lucene Index for Question Retrieval.*

We create a document for each shell-related question by collecting the title and tags. We exclude the body of a question as when examining a question returned for a query, users often judge its relevance based on the title and tags before looking into the long text in the body. We index the question documents using Lucene. The index is used to improve the efficiency of retrieving similar questions for a query.

2.1.4 *MP Parsing.*

Ubuntu is a widely used open-source Linux system. To generate answers for shell programming tasks in Ubuntu, we need to acquire knowledge about shell commands available in Ubuntu, which are documented in the Ubuntu MPs [12]. We collect the MPs of Ubuntu 20.04 LTS (Focal Fossa) released on February 4, 2021. The MPs are grouped into nine sections, numbered from 1 to 9[1]. In the current ShellFusion, we focus on the shell commands and system administration commands in the sections 1 and 8, as they are more likely to be used by programmers. Note that system administration commands are considered as a special kind of shell commands. Each MP is a structured HTML page describing a command, e.g., the summary, synopsis, and options. The following shows a part of the information of command `ls`.

- **Summary**: list directory contents
- **Synopsis**: `ls` [OPTION]... [FILE]...
- **Options**:

  `-l`: use a long listing format

  `-h`, `--human-readable`: with `-l` and `-s`, print sizes like 1K 234M 2G etc.

  `-t`: sort by modification time, newest first

We parse the MPs using the `lxml` module in Python and obtain the summary and options of each command. In total, we obtain 50,841 commands, including 44,423 and 6,418 commands from the sections 1 and 8, respectively.

2.1.5 *TLDR Parsing.*

Ubuntu MPs are generally too lengthy to read and rarely provide examples to demonstrate how to use a command. It is often difficult for users to find appropriate commands with options for a task from the large number of commands and options described in the MPs. Existing work has been devoted to creating easy-to-use tutorials for frequently used commands [2, 3, 11, 34, 40]. The TLDR project [11] hosted on GitHub is one of the most popular tutorials. We download the TLDR repository from

[1] https://help.ubuntu.com/community/man

GitHub. The commands related to Linux are contained in two folders: */pages/common* that contains the commands common to multiple operating systems, e.g., Linux and Windows; and */pages/linux* that contains the commands specific to Linux. For each command, TLDR maintains a summary and several representative task-script pairs in a MARKDOWN file. The summary and a task-script pair of command `ls` are as follows.

- **Summary**: list directory contents
- **Task-Script Pairs**:

  Task 1: Long format list with size displayed using human readable units (KB, MB, GB)

  Script 1: `ls -lh`

We parse the MARKDOWN files and obtain the summaries and example task-script pairs of 1,797 commands, including 1,264 and 533 commands from */pages/common* and */pages/linux*, respectively.

2.2 Online Answer Generation

2.2.1 Similar Question Retrieval. Given a query describing a shell programming task, we first preprocess the query using stopword removal and stemming. Then, we need to retrieve a list of shell-related questions similar to the query. In Section 2.1.2, we build a word IDF vocabulary and a word embedding model for measuring the similarity between a question and the query. However, it is time-consuming to calculate the similarities of 537,129 shell-related questions using the word embedding model due to the required matrix representations of the questions [26]. To expediate the retrieval of similar questions, we use a two-phase method by combining Lucene and a language model-based method. In the first phase, we use the Lucene search engine to retrieve the top N questions similar to the query by leveraging the Lucene index built for all questions. In the second phase, for each of the N questions, we calculate an asymmetric IDF-weighted similarity of the question T to the query Q based on the words of the title and tags of T and the words of Q:

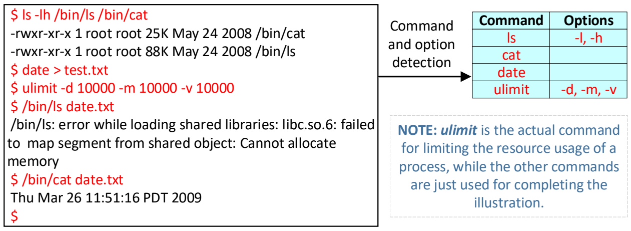
$$sim(T \rightarrow Q) = \frac{\sum_{w \in T} sim(w, Q) \times idf(w)}{\sum_{w \in T} idf(w)} \quad (1)$$

where $sim(w, Q)$ is the maximum value of $sim(w, w'), \forall w' \in Q$, and $sim(w, w')$ is the cosine similarity of the word embedding vectors of w and w'. Another asymmetric similarity $sim(Q \rightarrow T)$ is calculated by swapping T and Q in Eq. (1). Finally, we calculate the similarity between T and Q as the *harmonic mean* of the two asymmetric similarities:

$$sim(T, Q) = \frac{2 \times sim(T \rightarrow Q) \times sim(Q \rightarrow T)}{sim(T \rightarrow Q) + sim(Q \rightarrow T)} \quad (2)$$

After calculating the similarities of the N questions, we obtain the top-n similar questions for the query. In ShellFusion, we set n = 50 by default as too many questions introduce noise to the answer generation process, and set N = 1,000, so that the top-50 semantically similar questions can be covered by the N questions.

2.2.2 Shell Command Detection & Filtering. It is likely that the shell commands used in the accepted answer of a similar question are relevant to the query. A set of candidate commands can be obtained by detecting commands from the accepted answers of the top-n similar questions. For an accepted answer, we identify commands with options as follows.



```
$ ls -lh /bin/ls /bin/cat
-rwxr-xr-x 1 root root 25K May 24 2008 /bin/cat
-rwxr-xr-x 1 root root 88K May 24 2008 /bin/ls
$ date > test.txt
$ ulimit -d 10000 -m 10000 -v 10000
$ /bin/ls date.txt
/bin/ls: error while loading shared libraries: libc.so.6: failed
to  map segment from shared object: Cannot allocate
memory
$ /bin/cat date.txt
Thu Mar 26 11:51:16 PDT 2009
$
```

Command	Options
ls	-l, -h
cat	
date	
ulimit	-d, -m, -v

NOTE: ***ulimit*** is the actual command for limiting the resource usage of a process, while the other commands are just used for completing the illustration.

Figure 2: Commands and options detected from a code snippet that shows how to limit the memory usage of a process

- **Command detection.** We build a dictionary that stores all the commands extracted from Ubuntu MPs. We collect both short code snippets enclosed in HTML tag ⟨*code*⟩ and long code snippets enclosed in HTML tag ⟨*pre*⟩ in the accepted answer. By manually examining the code snippets (i.e., shell scripts) in the accepted answers of 100 sampled questions, we find that the commands in shell scripts often appear at the beginning of a script line or immediately behind six special characters: $, /, (, |, {, and =, e.g., '$ ls' and '/bin/ls' shown in Fig. 2. For each script, we replace the special characters with a whitespace and then split the script into a set of tokens by whitespaces. The tokens that match any command in the dictionary are marked as candidate commands. Fig. 2 shows four commands detected from a code snippet that illustrates how to limit the memory usage of a process[2].
- **Option detection.** We further detect the options of each candidate command used in the scripts. We treat each script line as a token sequence in which some tokens correspond to candidate commands. For the i-th command token, C_i, its associated options should exist in the token subsequence between C_i and the next command token or the end of the sequence. If a token in the subsequence matches any option extracted from C_i's MP, we mark it as an option of C_i. In practice, multiple options of a command can be compacted. In Fig. 2, the options `-l` and `-h` of command `ls` are compacted as `-lh`. For a token, t, which matches the regular expression `-[a-zA-Z]{2,}`, if t does not match any option of C_i, we divide t into multiple tokens that start with '-' and contain every single letter in t, and then check whether each of the tokens matches any option of C_i. We also find that some options of a command may not exist in its MP. For example, the options `-d`, `-m`, and `-v` of command `ulimit` shown in Fig. 2 are not included in the MP[3]. Therefore, we mark a token that does not match any option of C_i but matches the regular expression `-[a-zA-Z]` as a candidate option of C_i.

By collecting the commands with options detected from the accepted answers of the top-n similar questions, we obtain the entire set of candidate commands with associated options.

We observe that some detected candidate commands are irrelevant to the query. In Fig. 2, `ulimit` is the actual command for limiting the memory usage of a process, while the other commands,

[2] https://superuser.com/questions/134269

[3] http://manpages.ubuntu.com/manpages/focal/en/man1/ulimit.1posix.html

e.g., `ls`, are used for completing the illustration. We filter out the irrelevant commands by measuring their similarities with the query. For a command, C, we process its summary and the descriptions of its options in MP using stopword removal and stemming, resulting in a bag of words, C_{mp}. Similarly, we process the summary and example tasks of C in TLDR, resulting in a bag of words, C_{tldr}. We calculate two partial similarities between C and the query Q using Eq. (2) based on C_{mp} and C_{tldr}, respectively. Finally, we calculate the similarity between C and Q as:

$$sim(C, Q) = \alpha \times sim(C_{mp}, Q) + (1 - \alpha) \times sim(C_{tldr}, Q) \quad (3)$$

where $\alpha \in [0, 1]$ is a coefficient to balance the importance of the MP and TLDR descriptions of C. We distinguish C_{tldr} and C_{mp} for similarity measurement since a command in TLDR means that it is frequently used and could be considered for recommendation with priority. In this work, we set $\alpha = 0.5$ to assign equal weights to the MP and TLDR descriptions.

We refine the candidate commands by retaining the top-k commands with the maximum similarities.

2.2.3 Command Ranking. Intuitively, a command that is more frequently used in the accepted answers of similar questions is more likely to address the query. If a command can implement a question that is more similar to a query, it is more likely to address the query. We re-rank the top-k candidate commands by considering both the number of accepted answers that use the commands and the similarities of questions. For a command, C, we obtain the set of similar questions whose accepted answers mention the command, denoted as $T(C, Q)$. Then, we measure the likelihood that the command could be used to address the query as

$$likelihood(C \rightarrow Q) = min(\frac{\sum_{T \in T(C,Q)} sim(T, Q)}{n} \times log_2 n, 1.0) \quad (4)$$

where $sim(T, Q)$ is the similarity between a question, T, and the query Q, which is calculated using Eq. (2); and n is the number of questions in $T(C, Q)$. The equation has two parts: the first part calculates the average of the similarities of the questions whose accepted answers contain C; and the second part boosts the similarity based on the number of such questions. We add a logarithm transformation to avoid excessive boosting. The likelihood should not exceed 1 for calculating the recommendation score below.

We calculate the final recommendation score of C as the *harmonic mean* of its similarity with the query, $sim(C, Q)$, and its likelihood of achieving the query, $likelihood(C \rightarrow Q)$. A refined list of the candidate commands is produced by ranking them in a descending order according to the recommendation scores.

2.2.4 Answer Generation for Commands. To help users understand the recommended commands and quickly locate useful commands and scripts for the query, we generate a comprehensive answer for each candidate command by synthesizing the following aspects:

- **MP summary.** We present the official summary of the command in its MP to help users quickly check the command's functionality.
- **The most similar TLDR task-script pair.** If the command has a tutorial in TLDR, we calculate the similarity between each example task of the command and the query using Eq. (2). We then present the most similar task with the corresponding script, so that users can quickly find a useful script if the task matches the query.
- **Top-3 similar questions with accepted scripts.** We list the titles of the top-3 similar questions whose accepted answers contain the command. For each question, we also list the scripts that use the command in the accepted answer. Users could find useful scripts by comparing the question titles with the query. We only present the top-3 questions with accepted scripts is to prevent information overload.
- **Explanations about options.** There can be some options of the command used in the associated scripts. To help users understand the scripts, we present the official MP descriptions of the options. Since the command designer may miss options (Section 2.2.2), for the options without MP descriptions, we present the sentences that contain the options in the accepted answers of the top-3 similar questions.

3 EXPERIMENTAL SETUP

This section introduces the experimental setup for evaluating ShellFusion. Our experimental environment is a server with an Intel-i7 CPU, an Nvidia Geforce GTX1060 GPU, 64G RAM, and Win 10 OS.

3.1 Prototype Implementation

As described in Sections 2.1.1, 2.1.4, and 2.1.5, we collect 537,129 shell-related questions with accepted answers from four Q&A communities, 50,841 shell commands from the MPs of Ubuntu 20.04 LTS, and 1,797 Linux commands from the TLDR tutorials. We perform the offline process of ShellFusion on the dataset (Section 2.1).

In the online process of ShellFusion, there is a key parameter, k, which is the number of candidate commands used to generate answers. To determine the proper setting of k, we conduct a pilot study to measure the performance of ShellFusion on 50 experimental queries using different k values (Section 4.1).

3.2 Query Selection

Similar to prior work [26, 35, 48, 50, 51], we create a number of experimental queries from shell-related questions without accepted answers that are not included in the question repository used for implementing ShellFusion. To ensure the quality and popularity of queries, we collect 14,383 questions according to the following criteria: the *score* should be at least 10; and the *viewcount* should be at least 1,000. We then randomly select 500 questions from the collected questions. The lead author of this paper and a PhD student (who is not a co-author of this paper) manually examine the titles of the 500 questions to filter out questions that do not aim to search shell commands for programming tasks. They first independently label the questions to be removed. Typical examples of the questions being removed are given below:

- A question seeks for explanations about commands or other resources, e.g., *What is the meaning of "chmod 666"?*
- A question seeks for comparison between commands or other resources, e.g., *Difference between cp -r and cp -a.*
- A question describes an error, e.g., *"ping: unknown host google.com" but IPs works fine.*

We measure the inter-rater agreement between the two labelers using Fleiss Kappa [20]. The Kappa value is 0.87, indicating an almost perfect agreement. After discussing the disagreements, both labelers reach a common decision on the removal of 29 questions. Furthermore, we find that there are questions describing analogous tasks. For example, both *How to install Google Chrome* and *How do I install iTunes on Ubuntu?* ask for the installation of applications. To better evaluate the performance of ShellFusion on various tasks, only one of the analogous questions should be retained. Both labelers further independently examine the remaining 471 questions and label the analogous questions to be removed. The Kappa value is 0.82, indicating an almost perfect agreement. They discuss the disagreements and reach a consensus on the removal of 37 questions. As a result, we obtain 434 queries by collecting the titles of the final 434 questions.

3.3 Baseline Approaches

ShellFusion recommends shell commands for a query and generates a detailed answer for each command by leveraging Q&A posts, MPs, and TLDR. Q&A posts are used to extract candidate commands, while MPs and TLDR are used to filter out irrelevant commands and generate comprehensive answers. After extensive literature search, there is no existing work that exploits Q&A posts to recommend shell commands. However, several approaches, e.g., Magnum and Tellina, are proposed to translate natural language to Bash commands [15, 29]. According to the NLC2CMD competition held at NeurIPS 2020, Magnum is the state-of-the-art approach. Moreover, some work generates or recommends answers for programming tasks by exploiting Q&A posts [22, 48]. DeepAns [22] is one of the state-of-the-art approaches. We compare ShellFusion with Magnum and DeepAns, which are briefly described below:

- **Magnum** [15] translates a natural language task to the corresponding Bash command. The translation model is an ensemble of five separately trained transformer [43] models consisting of six layers with Beam Search [41]. Magnum is trained using the NL2Bash dataset [29] that contains 9,305 task-command pairs. We implement Magnum using its source code and pre-trained model released on GitHub [6].
- **DeepAns** [22] recommends the most relevant answer for a query from the answers of similar questions in Q&A communities, e.g., AU and SO. It first boosts the query by appending a clarification question generated using a sequence-to-sequence model. The model is trained based on the clarification questions asked in the comments of Q&A posts. DeepAns then retrieves the top-5 similar questions and obtains the answers of the similar questions as candidate answers. Finally, DeepAns sorts the answers by the matching scores calculated using a neural network model. The neural network model is trained based on four kinds of automatically labeled Q&A pairs. We implement DeepAns using its source code released on GitHub [4].

We evaluate two aspects of ShellFusion: the recommended shell commands and the generated answers. For the evaluation of shell command recommendation, we use Magnum as a baseline but exclude DeepAns for the following reasons: DeepAns is a general approach proposed for recommending relevant answers for queries from the existing answers of questions in Q&A communities. DeepAns can apply to shell programming tasks but it does not identify shell commands relevant to the queries from the recommended answers. For the evaluation of answer generation, we conduct a user study by comparing ShellFusion with both Magnum and DeepAns.

As described previously, in ShellFusion, Q&A posts are the basic information source used to obtain candidate shell commands for queries, and the other two information sources, i.e., MPs and TLDR, are used to filter out irrelevant commands and producing comprehensive answers. To evaluate the contributions of MPs and TLDR, we implement three variants of ShellFusion as follows.

- **ShellFusion-QA** uses Q&A posts only. Candidate commands are ranked by the likelihood scores computed using Eq. (4).
- **ShellFusion-QA+MPs** uses Q&A posts and MPs. The MP knowledge of shell commands is used to filter out irrelevant candidate commands.
- **ShellFusion-QA+TLDR** uses Q&A posts and TLDR. The TLDR knowledge of shell commands is used to filter out irrelevant candidate commands.

3.4 Evaluation Metrics

We evaluate the shell commands recommended by ShellFusion, Magnum, and the variants of ShellFusion using two widely used metrics: Mean Reciprocal Rank at K (MRR@K) [38] and Mean Average Precision at K (MAP@K) [26]. For a query, Reciprocal Rank@K (RR@K) refers to the multiplicative inverse of the rank of the first relevant command in the top-K recommendation list. MRR@K averages the RR@K values for a set of queries. For a query, Average Precision@K (AP@K) calculates the average of the precision at the occurrence of every relevant command in the top-K recommendation list. MAP@K averages the AP@K values for a set of queries.

4 EXPERIMENT RESULTS

4.1 RQ1: What is the best setting of parameter k in ShellFusion?

Motivation. As shown in Fig. 2, ShellFusion may detect shell commands irrelevant to a query from the code snippets in the accepted answers of similar questions. We filter out the noises by retaining the top-k commands that have the maximum similarities with the query. The setting of k can affect the quality of recommended commands. A large k (e.g., 20) may not filter out some irrelevant commands which can be included or even ranked highly in the recommendation list. A small k (e.g., 1) may leave out relevant commands. We need to determine the best setting of k to ensure a good performance of ShellFusion.

Approach. We perform a sensitivity analysis of k by conducting a pilot study with 50 queries randomly selected from the experimental queries. We run ShellFusion on the queries with different settings of $k \in \{1, 3, 5, 10, 15, 20\}$. For each query, we collect the answers generated using different k values. We then recruit two PhD students who are familiar with Ubuntu and have more than five years of experience in shell programming to evaluate the relevance of recommended commands in the answers. The *relevance* of a command indicates to what extent the command could be used to

Table 2: Performance of ShellFusion with different *k* values

k	MRR@1	MRR@3	MRR@5	MAP@1	MAP@3	MAP@5
1	0.480	0.480	0.480	0.249	0.249	0.249
3	**0.560**	**0.670**	**0.670**	**0.278**	0.395	0.395
5	**0.560**	0.663	0.668	**0.278**	**0.398**	0.425
10	**0.560**	0.657	**0.670**	**0.278**	0.395	**0.432**
15	**0.560**	0.657	0.666	**0.278**	0.395	0.431
20	**0.560**	0.657	0.666	**0.278**	0.395	0.431

Table 3: Performance of ShellFusion and four baselines

	MRR@1	MRR@3	MRR@5	MAP@1	MAP@3	MAP@5
Magnum	0.194^{***}	0.194^{***}	0.194^{***}	0.103^{***}	0.108^{***}	0.108^{***}
ShellFusion-QA	0.364^{***}	0.476^{***}	0.500^{***}	0.195^{***}	0.272^{***}	0.301^{***}
ShellFusion-QA+MPs	0.371^{***}	0.508^{***}	0.523^{***}	0.164^{***}	0.284^{***}	0.324^{***}
ShellFusion-QA+TLDR	0.608	**0.703**	**0.709**	0.274	0.426	0.465
ShellFusion	**0.615**	0.702	0.707	**0.288**	**0.432**	**0.468**

$^{***}p<0.001$, $^{**}p<0.01$, $^{*}p<0.5$

address the corresponding query. Both PhD students first independently evaluate the relevance of each recommended command by five grades 0-4, where 0, 1, 2, 3, and 4 mean 'strongly irrelevant', 'irrelevant', 'neutral', 'relevant', and 'strongly relevant', respectively. If the synthesized knowledge in an answer is insufficient to judge the relevance of the recommended command in the answer, they can search for the command on the web. In total, both students independently evaluate 302 commands for the 50 queries. The Fleiss Kappa value is 0.78, which indicates a substantial agreement. The students reach a consensus by discussing the disagreements.

Based on the relevance of recommended commands, we measure the average performance of ShellFusion. For every specific MRR@K or MAP@K metric, we compute the metric value of the top-*K* commands recommended for each query. Then, we compute the average of the metric values of the 50 queries.

Results. ShellFusion achieves better performance with the setting of *k* = 3, 5, or 10. Table 2 presents the performance of ShellFusion achieved using different *k* values. As *k* increases from 1 to 20, in terms of five specific metrics except MRR@5, the performance first increases until reaching a peak; thereafter it 1) becomes stable or 2) degrades slightly and then becomes stable. ShellFusion achieves the optimal performance on the maximum number of (i.e., four) metrics using *k* = 3 or 10. Specifically, when *k* = 3, the MRR@1, MRR@3, MRR@5, and MAP@1 values are optimal; when *k* = 10, the MRR@1, MRR@5, MAP@1, and MAP@5 values are optimal. Moreover, when *k* = 5, the MRR@1, MAP@1, and MAP@3 values are optimal, and the values of the other three metrics are very close to the optimal ones achieved using *k* = 3 or 10. **From the results, users can set *k* = 3, 5, or 10 based on their preferences with respect to the four metrics: MRR@3, MRR@5, MAP@3, and MAP@5.** In this work, we prefer MAP@K to MRR@K as MAP@K considers both scores and ranking of the relevant commands. We are also more interested in the top-3 recommended commands. Therefore, we set *k* = 5 for the subsequent experiments.

4.2 RQ2: How effective is ShellFusion in shell command recommendation?

Motivation. ShellFusion aims to recommend relevant shell commands with comprehensive knowledge for programming tasks. It is necessary to evaluate the effectiveness of ShellFusion in shell command recommendation. ShellFusion integrates the knowledge of shell commands mined from Q&A posts, MPs, and TLDR. Q&A posts are used to extract candidate shell commands, while both MPs and TLDR are used to filter out irrelevant commands and generate comprehensive answers for commands. We need to validate the contributions of MPs and TLDR to ShellFusion.

Approach. We implement Magnum and three variants of ShellFusion, i.e., ShellFusion-QA, ShellFusion-QA+MPs, and ShellFusion-QA+TLDR, as baselines (Section 3.3). We perform ShellFusion and the baselines on the 434 experimental queries. For each query, we collect the answers produced by the five approaches. The two PhD students involved in RQ1 further evaluate the relevance of recommended commands in the answers. Note that the answer (i.e., a command template) produced for a query by Magnum may contain multiple commands, as shown in Fig. 3. We ask the students to evaluate every command in the command templates produced by Magnum. Similarly, they first independently evaluate the relevance of each command by five grades 0-4. In total, both students independently evaluate 5,509 commands, and the Fleiss Kappa value is 0.83, indicating an almost perfect agreement. After discussing the disagreements, the students achieve a consensus.

We measure the average performance of the five approaches on all experimental queries. For Magnum, we measure its best performance of the command template produced for a query based on the optimal command list ranked in descending order by the relevance. In terms of every specific MRR@K and MAP@K metric, we also examine the statistical significance of the differences between the performance of ShellFusion and the baselines using the Wilcoxon signed-rank test [45].

Results. ShellFusion has demonstrated the effectiveness in shell command recommendation. Table 3 presents the performance of ShellFusion and four baselines, and the significant results of ShellFusion over the baselines. ShellFusion achieves the optimal performance on MRR@1, MAP@1, MAP@3, and MAP@5. The MRR@3 and MRR@5 values of ShellFusion are very close to the optimal ones achieved by ShellFusion-QA+TLDR. ShellFusion significantly outperforms Magnum with an improvement of at least 217.0% on MRR@K and at least 179.6% on MAP@K. The poor performance of Magnum is mainly due to the training dataset, NL2Bash, which covers only 102 commands. Magnum cannot deal with many tasks that require commands outside the dataset.

ShellFusion-QA+MPs improves ShellFusion-QA slightly on five metrics except MAP@1. The worse MAP@1 of ShellFusion-QA+MPs may be caused by the misleading similarities between some commands and queries calculated based on the MP descriptions of the commands and their options. ShellFusion-QA+TLDR achieves much better performance than ShellFusion-QA on all metrics. **The results indicate that both MPs and TLDR contribute to ShellFusion, and TLDR has more contributions than MPs.** Through our analysis, the results can be explained by the fact that unlike the official MPs provided by command designers, the unofficial TLDR contains practical knowledge about the usage of commands, which is more useful for users. Although MPs have limited contributions to recommending better commands, MPs are necessary to

Table 4: Time cost of ShellFusion, Magnum, and DeepAns

	Offline processing time	Online processing time
ShellFusion	3.2 hours	0.4 seconds per query
Magnum	–	0.01 seconds per query
DeepAns	12.0 hours	3.37 seconds per query

help users understand the generated answers by offering official explanations about the recommended commands and options, as confirmed by the participants of our user study (Section 5).

4.3 RQ3: How efficient is ShellFusion?

Motivation. As shown in Fig. 1, ShellFusion has five offline steps, e.g., word embedding model building, and MP and TLDR parsing. Although the steps require substantial time, they do not affect the efficiency of online answer generation. To accelerate the online process, we conduct two steps offline: 1) transforming the titles and tags of all questions in the repository into matrix representation based on the word embedding model; and 2) detecting commands with options from the code snippets in the accepted answers of all questions. Using the results, ShellFusion can efficiently calculate semantic similarities between a query and the N = 1,000 questions reduced by Lucene, and quickly obtain candidate commands from the top-50 similar questions.

During the online answer generation for a query, ShellFusion needs to filter out irrelevant candidate commands by calculating their similarities with the query, rank the refined candidate commands, and generate a detailed answer for each command by synthesizing knowledge from multiple sources. If ShellFusion cannot run with a reasonable runtime, users may not be willing to use it in practice. We need to validate the online efficiency of ShellFusion.

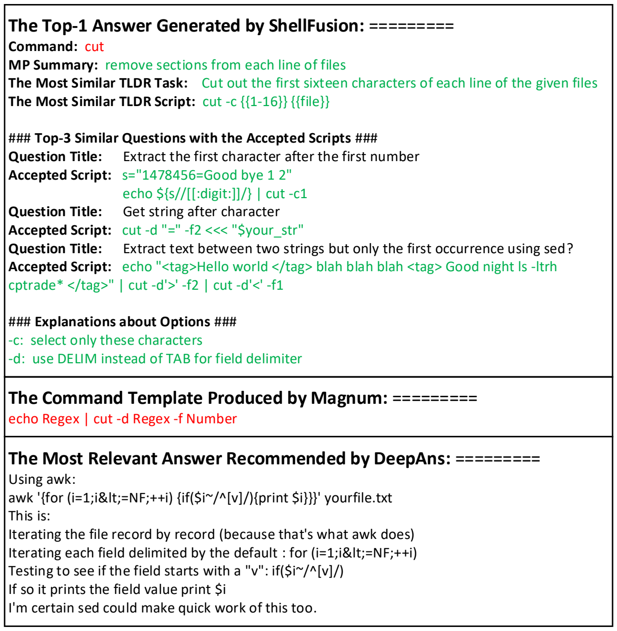
Approach. We record the time spent on the offline and online steps of ShellFusion, DeepAns, and Magnum.

Results. ShellFusion can rapidly respond to users after they submit a shell programming task. Table 4 presents the offline and online time cost of three approaches. We do not report the offline time of Magnum as we directly use its pre-trained model. The offline process of ShellFusion takes 3.2 hours, which should be acceptable in practice. In the online process, ShellFusion takes only 0.4 seconds to generate answers for a query. DeepAns has higher offline and online time cost than ShellFusion. Magnum can answer a query within 0.01 seconds, which is much faster than ShellFusion, however, the effectiveness of Magnum is poor (Table 3).

The efficiency of ShellFusion largely depends on the two-phase method used for similar question retrieval. After reducing the large repository to a set of N questions using the Lucene search engine, the top-n similar questions can be efficiently retrieved using the word embedding model-based method.

5 USER STUDY

In this section, we conduct a user study to validate the practical value of ShellFusion by checking whether the generated answers can help users find correct shell commands and scripts for programming tasks more efficiently and accurately.



```
The Top-1 Answer Generated by ShellFusion: =========
Command:  cut
MP Summary:  remove sections from each line of files
The Most Similar TLDR Task:   Cut out the first sixteen characters of each line of the given files
The Most Similar TLDR Script:  cut -c {{1-16}} {{file}}

### Top-3 Similar Questions with the Accepted Scripts ###
Question Title:     Extract the first character after the first number
Accepted Script:   s="1478456=Good bye 1 2"
                   echo ${s//[[:digit:]]/} | cut -c1
Question Title:     Get string after character
Accepted Script:   cut -d "=" -f2 <<< "$your_str"
Question Title:     Extract text between two strings but only the first occurrence using sed?
Accepted Script:   echo "<tag>Hello world </tag> blah blah blah <tag> Good night ls -ltrh
cptrade* </tag>" | cut -d'>' -f2 | cut -d'<' -f1

### Explanations about Options ###
-c:  select only these characters
-d:  use DELIM instead of TAB for field delimiter
```

```
The Command Template Produced by Magnum: =========
echo Regex | cut -d Regex -f Number
```

```
The Most Relevant Answer Recommended by DeepAns: =========
Using awk:
awk '{for (i=1;i&lt;=NF;++i) {if($i~/^[v]/){print $i}}}' yourfile.txt
This is:
Iterating the file record by record (because that's what awk does)
Iterating each field delimited by the default : for (i=1;i&lt;=NF;++i)
Testing to see if the field starts with a "v": if($i~/^[v]/)
If so it prints the field value print $i
I'm certain sed could make quick work of this too.
```

Figure 3: The answers produced by ShellFusion, Magnum, and DeepAns for the query Q7 listed in Table 5.

5.1 Study Design

We randomly select ten experimental queries. The two PhD students involved in RQ1 and RQ2 determine the ground-truth commands and scripts for the queries, as listed in Table 5. The queries have different levels of difficulties. Some queries, e.g., Q5, can be easily addressed using a common command, while other queries, e.g., Q1 and Q7, require uncommon or multiple commands. The diversity of queries can help improve the generalizability of our results. For a query that contains multiple commands in the ground-truth, some commands can be more relevant to the query than the others. For example, for Q7, cut is more relevant than echo. Similar to RQ1, both students evaluate the relevance of the ground-truth commands of each query by five grades 0-4. As listed in Table 5, the relevance scores of cut and echo of Q7 are 4 and 2, respectively.

We recruit 20 participants (4 faculty members, 2 postdocs, 11 PhDs, and 3 MScs) from the second and the third authors' organizations. They have different years of shell programming experience, varying from 1-8 years with an average (avg.) of 3.2 years. We divide them uniformly into four groups while ensuring that the members in different groups have comparative shell programming experience. The groups are assigned to different settings: 1) **ShellFusion**: using the answers generated by ShellFusion; 2) **Magnum**: using the answers produced by Magnum; 3) **DeepAns**: using the answers recommended by DeepAns; and 4) **Web Search (WS)**: using web search engines, e.g., Google, only. The numbers of years of experience in shell programming that the five participants of ShellFusion, Magnum, DeepAns, and WS have are [1, 2.5, 3, 4, 5] (avg.=3.1), [1, 2, 3, 3, 7] (avg.=3.2), [1, 2, 3, 3.5, 6.5] (avg.=3.2), and [1, 2, 2.5, 3, 8] (avg.=3.3), respectively. Fig. 3 shows the answers produced by ShellFusion, Magnum, and DeepAns for Q7. The participants assigned to the three approaches do not know the approach

Table 5: Ten queries for the user study and the ground-truth commands and scripts of the queries

ID	Query	Ground-truth commands with relevance scores	Ground-truth script
Q1	How to download an MP3 track from a YouTube video	youtube-dl(4)	youtube-dl -x –audio-format mp3 VideoURL
Q2	How to convert a video from mp4/flv to mpeg/mpg	ffmpeg(4)	ffmpeg -i Input.format Output.format
Q3	Create a single pdf from multiple text, images or pdf files	convert(3), pdfunite(4)	convert Text.txt Text.pdf convert Image.jpg Image.pdf pdfunite Text.pdf Image.pdf Out.pdf
Q4	How to recover deleted files?	sudo(2), photorec(4)	sudo photorec DeletedFilePartition
Q5	How to copy a file from remote server to local machine?	scp(4)	scp RemoteFile LocalFile
Q6	How to grep for same string but multiple files at the same time?	grep(4)	grep String File1 File2 ...
Q7	How to extract the first two characters of a string in shell scripting?	echo(2), cut(4)	echo String | cut -c 1-2
Q8	Bash How to find the largest file in a directory and its subdirectories?	du(4), sort(3), head(2)	du -a Directory | sort -nr | head
Q9	How to check which process is using most memory	ps(4), head(2)	ps aux –sort=-%mem | head
Q10	With the Linux "cat" command, how do I show only certain lines by number	cat(4), sed(4)	cat -n Text.txt | sed 'StartLine,EndLine!d'

that they are using and do not know the differences among the approaches.

We ask the participants to find shell command(s) and a script that uses the command(s) for addressing each of the ten queries. For the participants assigned to ShellFusion, Magnum, and DeepAns, they first browse the answers generated for a query and try to seek for useful commands and the corresponding script from the answers. If no desired command or script is found, they can resort to web search engines. The participants assigned to WS directly use web search engines to find useful commands and scripts for queries. After determining the commands and script for a query, the participants record the results and the time spent on finding the results. They are allowed to spend at most ten minutes on a query. We also ask the participants assigned to ShellFusion, Magnum, and DeepAns to evaluate the *usefulness* and *understandability* of the answers generated for each query by five grades 0-4.

- **Usefulness** measures to what extent the generated answers are useful for finding the commands and script of a query. The five grades are 0-'extremely useless', 1-'useless', 2-'neutral', 3-'useful', and 4-'very useful'. For example, the answers should be very useful if both the commands and script are included in the answers.
- **Understandability** measures whether the generated answers are easy to understand. The five grades are 0-'very poor', 1-'poor', 2-'neutral', 3-'good', and 4-'very good'. If the answers are well-organized and have explanations about the commands and options in the presented scripts, the answers should have a very good understandability.

The participants may answer the queries using three different methods: 1) by themselves if they already know the answers of a query based on their existing knowledge; 2) using the recommended answers; and 3) using web search if they fail to find useful commands and scripts in the recommended answers. We ask the participants to record the method used to answer each query.

5.2 Result Analysis

After the user study, we judge the *correctness* of the commands and scripts found by the participants. Given the commands found by a participant P for a query, we identify the correct commands CC included in the ground-truth commands GC. We then measure the correctness of the commands as the ratio of the sum of relevance of the correct commands to the sum of relevance of the ground-truth commands, i.e., $\frac{\sum_{C_i \in CC} rel(C_i)}{\sum_{C_j \in GC} rel(C_j)}$, where $rel(C)$ is the relevance of command C. Next, for each correct command C_i, we examine the script S provided by P and calculate the proportion of the options of C_i in S among the options of C_i in the ground-truth script. If the ground-truth script does not use any option of C_i, the proportion is 1. We finally measure the correctness of S as the ratio of the sum of proportion-weighted relevance of the correct commands to the sum of relevance of the ground-truth commands, i.e., $\frac{\sum_{C_i \in CC} rel(C_i) \times p(C_i)}{\sum_{C_j \in GC} rel(C_j)}$, where $p(C_i)$ is the proportion calculated for C_i. We find that some queries can be achieved using other commands different from the ground-truth commands. For example, for Q3, `pdftk` and `stapler` can be used to merge multiple PDFs into a single PDF. Both PhD students, who build the ground-truth, further evaluate the relevance of the commands that do not match the ground-truth. We then measure the correctness of the commands and scripts found by the participants that have not been measured in the first round.

We observe that there are a number of queries answered by the participants using the three methods mentioned previously in the four participant groups, as listed in Table 6. The '$m(n)$' value in each cell means that the answers to n of the m queries are correct (i.e., the correctness scores of the commands and scripts provided by the participants are 1). **ShellFusion helps the participants address the maximum number of queries (i.e., 25) correctly.** In the subsequent analysis, we exclude the results of the queries answered by the participants themselves because such results are not obtained using the experimental settings.

For each of the four groups, we compute the average correctness of the commands and scripts found by the participants for the ten queries. We also compute the average time spent on finding the results for the ten queries by each group. For the three groups except WS, we compute the average usefulness and understandability of the answers produced by the corresponding approach.

We further measure the correctness of the commands and scripts recommended for the ten queries by the three approaches. In terms of a query Q and an approach A, we first measure multiple correctness scores of the commands recommended for Q by A with respect to the ground-truth commands of Q and the other correct commands found for Q by each participant. We then determine the

Table 6: The numbers of queries answered by the participants of the four groups using the three methods. The '*m*(*n*)' in each cell means that the answers to *n* of the *m* queries are correct.

	# Queries answered by the participants themselves	**# Queries answered using the recommended answers**	**# Queries answered using Web Search**
ShellFusion	3(2)	**27(25)**	20(13)
Magnum	0(0)	**12(6)**	38(26)
DeepAns	3(1)	**5(4)**	42(31)
WS	5(4)	–	45(29)

correctness of the commands recommended for Q by A as the maximum correctness score. Similarly, we determine the correctness of the scripts recommended for Q by A as the maximum correctness score measured based on the ground-truth scripts of Q and the other correct scripts found for Q by the participants. Finally, we compute the average correctness of the commands and scripts recommended for the ten queries by A.

For each of the seven evaluated aspects, including the time for answering the queries, the correctness of the commands and scripts found by the participants, the usefulness and understandability of the approaches, and the correctness of the commands and scripts recommended by the approaches, in addition to the averages (i.e., means) of the measured results, we compute the standard deviations of the results for each of the four groups. Moreover, we measure the statistical significance of the differences between the results achieved by the participants assigned to ShellFusion and those achieved by the participants assigned to the other three groups.

Results. ShellFusion has demonstrated its practicality in helping users address shell programming tasks more efficiently and accurately. Table 7 presents the analysis results of the user study. In terms of the correctness of the commands and scripts found by the participants, ShellFusion achieves the best performance. The participants assigned to ShellFusion successfully find 91.3% of the correct commands and 88.6% of the correct scripts for the ten queries, while spending the least amount of time, i.e., an average of 137.3 seconds, on a query. Compared with WS, ShellFusion improves the correctness of commands and scripts by 21.2% and 25.0%, respectively, and reduces the average amount of time that participants need by 44.4%. ShellFusion improves the second-best approach, DeepAns, by 7.4% and 9.8% in terms of the correctness of commands and scripts found by the participants, respectively, and reduces the average amount of time that the participants need by 30.7%. This can be explained by the fact that ShellFusion recommends more correct commands and scripts than DeepAns and Magnum for the queries, as listed in Table 7. The usefulness and understandability scores of ShellFusion are high (i.e., 3.553 and 3.766) and are much higher than those of Magnum and DeepAns. Furthermore, the advantage of ShellFusion over DeepAns and Magnum is significant in terms of the usefulness and understandability of the produced answers and the correctness of the recommended commands and scripts in most cases. All these indicate that **the answers generated by ShellFusion are more helpful for finding correct commands and scripts for queries and easier to understand.**

The standard deviations of the results obtained by the participants assigned to ShellFusion are all smaller than those of the results obtained by the participants assigned to the other groups. This further indicates that **shell programmers can achieve more stable performance using ShellFusion than using Magnum, DeepAns, or WS.**

The participants provide some comments about ShellFusion. They express that most of the recommended commands are helpful for addressing the queries and the presented knowledge of the commands and options are useful to help users quickly judge usefulness of the commands and scripts. However, the participants point out that ShellFusion only recommends a single command in a generated answer, which may not be suitable for the tasks that need to be addressed by combining multiple commands.

6 THREATS TO VALIDITY

Threats to internal validity relate to the errors in the implementation of ShellFusion and the baselines, and the bias of participants in the manual evaluation of recommended commands and generated answers. For ShellFusion and its variants, we double-check the code and make sure that the experimental queries are not included in the question repository. For Magnum and DeepAns, we carefully implement them using their published code. When evaluating the relevance of the commands recommended for queries, we recruit two PhD students with relatively high (i.e., more than five years of) experience in shell programming to reduce a single person's subjective judgment. In our user study, the participants' carefulness and lack of knowledge about shell commands may affect the evaluation results. To reduce this threat, we recruit participants interested in the user study and make sure the participants in different groups have comparative shell programming experience.

Threats to external validity relate to the generalizability of experiment results. We implement ShellFusion based on 537,129 shell-related questions with accepted answers from four Q&A sites. We create 434 experimental queries from other shell-related questions with a relatively high viewcount and score. The large and high-quality dataset can improve the generalizability of our quantitative results. As the user study requires significant manual effort, we use ten queries. The queries have different levels of difficulties, which can improve the generalizability of our user study results.

Threats to construct validity relate to the suitability of evaluation metrics. We use MRR@K and MAP@K for evaluating the top-k recommended shell commands as the two metrics are widely used for evaluating ranking problems in information retrieval [26, 38, 46, 48, 52]. The relevance, usefulness, and understandability metrics are also widely used for evaluating recommendation and summarization tasks in software engineering [19, 37, 39, 48].

7 RELATED WORK

Shell programming tutorials and tools. It is impossible for shell programmers to remember thousands of shell commands with various options. A good way to help users learn and use the commands is by explaining them with examples. Towards this direction, some work has been done [27, 34, 36, 40]. For example, Sobell [40] provides a practical guide to utilities, basic commands,

Table 7: The means and standard deviations of the user study results with respect to seven evaluated aspects, as well as the statistical significance of the differences between the results achieved by the participants assigned to ShellFusion and those achieved by the participants assigned to Magnum, DeepAns, and WS. The first and the second values in each cell are the mean and the standard deviation, respectively. Note that the results of the queries answered by the participants themselves (as listed in Table 6) are excluded.

	Time in second	Correctness of the commands found by the participants	Correctness of the scripts found by the participants	Usefulness	Understandability	Correctness of the commands recommended by the approach	Correctness of the scripts recommended by the approach
ShellFusion	**(137.3, 97.6)**	**(0.913, 0.209)**	**(0.886, 0.260)**	**(3.553, 1.048)**	**(3.766, 0.554)**	**(0.910, 0.188)**	**(0.893, 0.215)**
Magnum	(160.9, 137.0)	(0.844, 0.303)	(0.762, 0.367)	(1.500***, 1.565)	(1.120***, 1.227)	(0.478*, 0.482)	(0.311**, 0.418)
DeepAns	(198.0, 169.5)	(0.850, 0.319)	(0.807, 0.360)	(1.979***, 1.345)	(2.383***, 1.043)	(0.600, 0.437)	(0.400*, 0.437)
WS	(246.8**, 194.2)	(0.753, 0.392)	(0.709*, 0.417)	–	–	–	–

$^{***}p<0.001$, $^{**}p<0.01$, $^{*}p<0.05$

and shell programming in Linux. Negus and Casen [34] build a toolbox for 1,000+ Ubuntu commands related to file manipulation, multimedia playing, etc. Lau et al. [27] describe an approach, SMARTshell, that can assist users in learning shell scripts by demonstration. There are some other tutorials built by communities, e.g., TLDR [11] and Cheat [3]. These tutorials are helpful, however, they require a great amount of manual effort and contain only a few examples for a limited number of basic or frequently used commands. They do not provide a mechanism to search for commands by a task.

Several approaches and tools are proposed to support shell programming. Chakraborti et al. [18] present a platform, UbuntuWorld 1.0 LTS, for developing an automated technical support agent in Ubuntu. Lin et al. [29] build a dataset, NL2Bash that contains 9,305 task-command pairs, for the problem of mapping a task to the corresponding Bash command. They present three baselines for addressing the problem by leveraging three neural machine translation models: Seq2Seq [42], CopyNet [23], and Tellina [28]. During the NLC2CMD competition held at NeurIPS 2020 [15], six approaches, e.g., Magnum, are developed for translating natural language to Bash commands using the NL2Bash dataset. Agarwal et al. [14] present the design and implementation of a platform, CLAI, which aims to bring the power of AI, e.g., the approaches developed in the NLC2CMD challenge, to the command line interface. However, these approaches and tools are limited by the NL2Bash dataset which only covers 102 commands and 206 options.

Different from the existing tutorials and tools that are limited by manually created datasets, we develop ShellFusion that integrates Q&A posts, MPs, and TLDR to recommend shell commands and generate comprehensive answers for programming tasks.

Mining Q&A communities. The rich data in Q&A communities has been widely explored by researchers to understand software development practices and build tools for software development tasks. Wan et al. [44] analyze the discussion topics of blockchain in SO using a balanced LDA model. Nashehi et al. [33] study the factors that make an effective code example through a qualitative analysis of SO posts. Rahman et al. [38] propose an API recommendation approach, RACK, by mining keyword-API mappings from SO posts. Nie et al. [35] propose a code search approach by expanding queries with keywords extracted from SO posts. Gao et al. [22] propose DeepAns to recommend relevant answers for queries by searching Q&A pairs. Our ShellFusion also uses Q&A posts, but we focus on generating answers for shell programming tasks.

API recommendation. API recommendation is a research hotspot in software engineering [24, 26, 31, 38]. For example, McMillan et al. [31] propose Portfolio to recommend C/C++ functions for queries from a large archive of C/C++ source code. Huang et al. [26] propose BIKER for API recommendation by leveraging SO posts and API documentation. Gu et al. [24] develop an RNN-based approach, DeepAPI, that can generate API sequences for queries by mining code repositories. Moreover, there are many approaches proposed for recommending web APIs [25, 46, 47] which are widely used in distributed software development. In contrast, we propose an approach that integrates knowledge from multiple information sources to recommend shell commands.

8 CONCLUSION AND FUTURE WORK

In this paper, we propose ShellFusion to automatically generate answers for shell programming tasks. We integrate shell-related knowledge mined from Q&A posts, Ubuntu MPs, and TLDR tutorials. Given a query, ShellFusion extracts candidate commands with options from the accepted answers of questions similar to the query. It then filters out irrelevant commands and ranks the refined commands by considering their similarities with the query and the similar questions with accepted answers that use the commands. Finally, ShellFusion generates an answer for each command by synthesizing valuable knowledge mined from the three information sources. Evaluation results on 434 experimental queries confirm the effectiveness of ShellFusion. A user study shows that ShellFusion can help users find useful commands and scripts for shell programming tasks efficiently and accurately. In future work, we will improve ShellFusion to recommend collaborative commands with usage templates for complex tasks that involve multiple commands. Moreover, we plan to implement an easy-to-use interface for ShellFusion (e.g., using Docker).

ACKNOWLEDGMENTS

This research/project is supported by the National Natural Science Foundation of China (No. 62032025), and the National Research Foundation, Singapore, under its Industry Alignment Fund - Pre-positioning (IAF-PP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

REFERENCES

[1] 2022. Ask Ubuntu. https://askubuntu.com

[2] 2022. Bro pages. http://bropages.org/

[3] 2022. Cheat. https://github.com/cheat/cheat

[4] 2022. DeepAns. https://github.com/beyondacm/DeepAns

[5] 2022. Gensim. https://radimrehurek.com/gensim

[6] 2022. Magnum-NLC2CMD. https://github.com/magnumresearchgroup/Magnum-NLC2CMD

[7] 2022. ShellFusion. https://github.com/nengz/ShellFusion

[8] 2022. Stack Exchange Data Dump. https://archive.org/download/stackexchange

[9] 2022. Stack Overflow. https://stackoverflow.com

[10] 2022. Super User. https://superuser.com

[11] 2022. TLDR. https://github.com/tldr-pages/tldr

[12] 2022. Ubuntu Manual Pages. http://manpages.ubuntu.com

[13] 2022. Unix & Linux. https://unix.stackexchange.com

[14] Mayank Agarwal, Jorge J Barroso, Tathagata Chakraborti, Eli M Dow, Kshitij Fadnis, Borja Godoy, Madhavan Pallan, and Kartik Talamadupula. 2020. Project clai: Instrumenting the command line as a new environment for ai agents. *arXiv preprint arXiv:2002.00762* (2020).

[15] Mayank Agarwal, Tathagata Chakraborti, Quchen Fu, David Gros, Xi Victoria Lin, Jaron Maene, Kartik Talamadupula, Zhongwei Teng, and Jules White. 2021. NeurIPS 2020 NLC2CMD Competition: Translating Natural Language to Bash Commands. *arXiv preprint arXiv:2103.02523* (2021).

[16] Akiko Aizawa. 2003. An information-theoretic perspective of tf-idf measures. *Information Processing & Management* 39, 1 (2003), 45–65.

[17] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* O'Reilly Media, Inc.

[18] Tathagata Chakraborti, Kartik Talamadupula, Kshitij P Fadnis, Murray Campbell, and Subbarao Kambhampati. 2017. UbuntuWorld 1.0 LTS—A Platform for Automated Problem Solving & Troubleshooting in the Ubuntu OS. In *Twenty-Ninth IAAI Conference.*

[19] Andrea Di Sorbo, Sebastiano Panichella, Carol V Alexandru, Junji Shimagaki, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. 2016. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* 499–510.

[20] Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological Bulletin* 76, 5 (1971), 378.

[21] Ishaan Gandhi and Anshula Gandhi. 2020. Lightening the Cognitive Load of Shell Programming. (2020).

[22] Zhipeng Gao, Xin Xia, David Lo, and John Grundy. 2020. Technical Q8A Site Answer Recommendation via Question Boosting. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 1 (2020), 1–34.

[23] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393* (2016).

[24] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* 631–642.

[25] Yan Hu, Qimin Peng, Xiaohui Hu, and Rong Yang. 2014. Time aware and data sparsity tolerant web service recommendation based on improved collaborative filtering. *IEEE Transactions on Services Computing* 8, 5 (2014), 782–794.

[26] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API method recommendation without worrying about the task-API knowledge gap. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 293–304.

[27] Tessa Lau, Lawrence Bergman, Vittorio Castelli, and Daniel Oblinger. 2004. Programming shell scripts by demonstration. In *Workshop on Supervisory Control of Learning and Adaptive Systems, AAAI,* Vol. 4.

[28] Xi Victoria Lin, Chenglong Wang, Deric Pang, Kevin Vu, and Michael D Ernst. 2017. Program synthesis from natural language using recurrent neural networks. *University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, Tech. Rep. UW-CSE-17-03-01* (2017).

[29] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. 2018. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. *arXiv preprint arXiv:1802.08979* (2018).

[30] Erik Linstead, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. 2009. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery* 18, 2 (2009), 300–336.

[31] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Qing Xie, and Chen Fu. 2011. Portfolio: finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering.* 111–120.

[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems.* 3111–3119.

[33] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What makes a good code example?: A study of programming Q&A in StackOverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM).* IEEE, 25–34.

[34] Christopher Negus. 2013. *Ubuntu Linux Toolbox: 1000+ Commands for Power Users.* John Wiley & Sons.

[35] Liming Nie, He Jiang, Zhilei Ren, Zeyi Sun, and Xiaochen Li. 2016. Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing* 9, 5 (2016), 771–783.

[36] Ellie Quigley. 2005. *Unix Shells by example.* Prentice Hall Professional Technical Reference.

[37] Dragomir Radev and Weiguo Fan. 2000. Automatic summarization of search engine hit lists. In *ACL-2000 Workshop on Recent Advances in Natural Language Processing and Information Retrieval.* 99–109.

[38] Mohammad Masudur Rahman, Chanchal K Roy, and David Lo. 2016. Rack: Automatic api recommendation using crowdsourced knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 349–359.

[39] Simone Scalabrino, Gabriele Bavota, Christopher Vendome, Mario Linares-Vásquez, Denys Poshyvanyk, and Rocco Oliveto. 2017. Automatically assessing code understandability: How far are we?. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 417–427.

[40] Mark G Sobell. 2013. *A practical guide to Linux commands, editors, and shell programming.* Prentice Hall.

[41] Volker Steinbiss, Bach-Hiep Tran, and Hermann Ney. 1994. Improvements in beam search. In *Third international conference on spoken language processing.*

[42] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems.* 3104–3112.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems.* 5998–6008.

[44] Zhiyuan Wan, Xin Xia, and Ahmed E Hassan. 2019. What is discussed about blockchain? a case study on the use of balanced lda and the reference architecture of a domain to capture online discussions about blockchain platforms across the stack exchange communities. *IEEE Transactions on Software Engineering* (2019).

[45] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in statistics.* Springer, 196–202.

[46] Fang Xie, Jian Wang, Ruibin Xiong, Neng Zhang, Yutao Ma, and Keqing He. 2019. An integrated service recommendation approach for service-based system development. *Expert Systems With Applications* 123 (2019), 178–194.

[47] Ruibin Xiong, Jian Wang, Neng Zhang, and Yutao Ma. 2018. Deep hybrid collaborative filtering for web service recommendation. *Expert systems with Applications* 110 (2018), 191–205.

[48] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. 2017. AnswerBot: Automated generation of answer summary to developers' technical questions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 706–716.

[49] Deheng Ye, Zhenchang Xing, and Nachiket Kapre. 2017. The structure and dynamics of knowledge network in domain-specific Q&A sites: a case study of stack overflow. *Empirical Software Engineering* 22 (2017), 375–406.

[50] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering.* ACM, 404–415.

[51] Neng Zhang, Qiao Huang, Xin Xia, Ying Zou, David Lo, and Zhenchang Xing. 2020. Chatbot4QR: Interactive Query Refinement for Technical Question Retrieval. *IEEE Transactions on Software Engineering* (2020).

[52] Neng Zhang, Jian Wang, Yutao Ma, Keqing He, Zheng Li, and Xiaoqing Frank Liu. 2018. Web service discovery based on goal-oriented query expansion. *Journal of Systems and Software* 142 (2018), 73–91.