On the Variability of Software Engineering Needs for Deep Learning: Stages, Trends, and Application Types

Kai Gao, Zhixing Wang, Audris Mockus, *Member, IEEE*, and Minghui Zhou, *Member, IEEE*

Abstract—The wide use of Deep learning (DL) has not been followed by the corresponding advances in software engineering (SE) for DL. Research shows that developers writing DL software have specific development stages (i.e., SE4DL stages) and face new DL-specific problems. Despite substantial research, it is unclear how DL developers' SE needs for DL vary over stages, application types, or if they change over time. To help focus research and development efforts on DL-development challenges, we analyze 92,830 Stack Overflow (SO) questions and 227,756 READMEs of public repositories related to DL. Latent Dirichlet Allocation (LDA) reveals 27 topics for the SO questions where 19 (70.4%) topics mainly relate to a single SE4DL stage, and eight topics span multiple stages. Most questions concern *Data Preparation* and *Model Setup* stages. The relative rates of questions for 11 topics have increased, for eight topics decreased over time. Questions for the former 11 topics had a lower percentage of accepting an answer than the remaining questions. LDA on README files reveals 16 distinct application types for the 227k repositories. We apply the LDA model fitted on READMEs to the 92,830 SO questions and find that 27% of the questions are related to the 16 DL application types. The most asked question topic varies across application types, with half primarily relating to the second and third stages. Specifically, developers ask the most questions about topics primarily relating to *Data Preparation* (2nd) stage for four mature application types such as Image Segmentation, and topics primarily relating to *Model Setup* (3rd) stage for four application types concerning emerging methods such as Transfer Learning. Based on our findings, we distill several actionable insights for SE4DL research, practice, and education, such as better support for using trained models, application-type specific tools, and teaching materials.

Index Terms—Software engineering needs for deep learning, mining software repositories, topic modeling, stack overflow

1 Introduction

Deep learning (DL) has achieved tremendous success in different tasks such as image recognition [1] and object detection [2] owing to its strong representation capability and the explosive increase of data and computing power in recent years. Many DL frameworks (e.g., TensorFlow [3], Keras [4], and PyTorch [5]) are proposed to help developers quickly transfer their ideas into applications and are widely used by developers. Based on the architecture documentation of various DL frameworks, Han *et al.* [6] found that to build DL applications with DL frameworks, developers usually go through seven stages starting from *Preliminary Preparation*, to *Data Preparation*, and to *Model Setup*, *Model Training*, *Model Evaluation*, *Model Tuning*, and ending with *Model Prediction* as shown in Table 1. In this paper, we refer to the software development in the DL domain, including the process consisting of the seven stages, as: software engineering (SE) for deep learning (SE4DL).

Although DL frameworks facilitate SE4DL, SE4DL still poses unique problems to developers that differ from regular software engineering. In general, developers use DL frameworks to define DL model structure and run-time configurations, then feed large-scale training data to train (adjust the parameters of) the model [7], [8]. Developers usually set aside some data not used for training to evaluate and tune the model. The above process is usually experimental: it consists of adjusting the data, model structure, and run-time configurations in a trial-and-error manner. As a result of such iterations, DL developers face problems across SE4DL stages. Examples include managing large-scale datasets at *Data Preparation* stage, designing effective model structures at *Model Setup* stage, and specifying efficient run-time configurations at *Model Training* stage.

The SE research community has investigated SE4DL needs in some detail. For example, researchers have extensively analyzed challenges and faults in general without separating them into SE4DL stages [7], [8], [9], [10], [11], and investigated deployment challenges and faults at the model prediction stage [12], [13], [14]. Little work [6], [15],

- *Kai Gao is with the School of Software & Microelectrics, Peking University, Beijing 100871, China. E-mail: gaokai19@pku.edu.cn.*
- *Zhixing Wang is with the School of Information Science and Technology, University of Tokyo, Bunkyo City, Tokyo 113-8654, Japan. E-mail: zhixing0@protonmail.com.*
- *Audris Mockus is with the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996 USA. E-mail: audris@utk.edu.*
- *Minghui Zhou is with the School of Computer Science, Peking University, Beijing 100871, China. E-mail: zhmh@pku.edu.cn.*

Manuscript received 17 October 2021; revised 11 February 2022; accepted 20 March 2022. Date of publication 30 March 2022; date of current version 13 February 2023.
This work was supported in part by the National Key R&D Program of China under Grant 2018YFB1004201 and in part by the National Natural Science Foundation of China under Grant 61825201.
(Corresponding author: Minghui Zhou.)
Recommended for acceptance by D. Lo.
Digital Object Identifier no. 10.1109/TSE.2022.3163576

TABLE 1
Definition of SE4DL Stages [6]

Stage	Description
Preliminary Preparation	Set up environment for developing DL applications.
Data Preparation	Convert raw data into the format required by the model.
Model Setup	Create neural network models with APIs provided by DL frameworks.
Model Training	Select loss function and optimization method, and feed data to train models with acceleration devices.
Model Evaluation	Evaluate models trained at the previous stage.
Model Tuning	Fix strange evaluation results and improve model's performance.
Model Prediction	Use tuned model to make predictions on new data.

[16] investigated SE4DL stages. Alshagiti *et al.* [15] labeled the stages of 684 Stack Overflow (SO) questions to investigate the most challenging stages; Islam *et al.* [16] manually labeled the stages of 970 bugs collected from SO and GitHub to reveal bug-prone stages. These two papers were based on a small sample size and didn't reveal what kind of problems are at each stage. Although Han *et al.* [6] applied LDA on a large-scale dataset collected from SO questions and GitHub issues, they investigated stages under an improper assumption that an LDA topic exclusively belongs to one stage. To prioritize efforts for improving SE4DL, we need a better understanding of the problems developers face at each stage and among different types of DL applications. Furthermore, it is important to identify what problems are already solved, at least partially, and what the most recent challenges are. Specifically, we lack understandings of: (1) how problems faced by DL developers are distributed over SE4DL stages, and (2) how these problems vary over time and application types. Given the rapid development of DL and its wide adoption in distinct tasks, such understanding will promote SE4DL research, practice, and education to meet developers' needs in a more targeted way. It may help researchers, practitioners, and educators understand current urgent SE4DL problems and design automated tools to mitigate them, improve DL framework APIs and documentation, and design customized teaching materials for different application types.

To investigate the variability of SE4DL problems, we decide to use two data sources. First, to understand problems faced by DL developers, we analyze DL-related questions from Stack Overflow (SO). Second, to understand the variety of DL-related projects, we gather approximately all public Git repositories and analyze their README files. Specifically, we answer the following research questions:

RQ1 (Stage Variability). How are problems faced by DL developers distributed over SE4DL stages? LDA reveals 27 topics for 92,830 DL-related SO questions. For each of the 27 topics, we randomly sample 63-67 questions (to obtain a 90% confidence level) and manually label the SE4DL stages for them. We find that 19 topics concentrate on a single SE4DL stage, and eight topics span multiple stages. The 19 single-stage topics cover all seven SE4DL stages and the eight multiple-stage topics are mainly about framework APIs and application tasks. Overall, developers ask the most about the second (*Data Preparation*) and third (*Model Setup*) stages with 23.3% and 30.7% questions respectively, in contrast to the former study that found the first (*Preliminary Preparation*) and fourth (*Model Training*) stages to be the stages with the most questions [6].

RQ2 (Time Variability). How do these problems vary over time? We apply the Mann-Kendall trend test to identify the change of relative rate of questions for each question topic. We find the relative rates of questions for 11 topics have increased, for eight topics decreased over time. Questions for the 11 trending-up topics had a lower percentage of having an accepted answer than for the remaining topics. The topics that increase the most (indicated by Sen's slope) are *Code Error*, *Training Anomaly*, *Model Load*, and *Model Conversion*, and the topic that decreases the most is *Graph Session*. The topic developers ask the most questions about, *Installation Error* hasn't increased or decreased significantly over time.

RQ3 (Application Variability). How do these problems vary over application types? We apply LDA on README files of 227k repositories and identify 16 distinct application types. We apply the LDA model fitted on the 227k README files to relate the SO questions to application types. We find that the most asked question topic varies across application types, with half primarily relating to the second and third stages. Specifically, developers ask the most questions about topics primarily relating to *Data Preparation* (2nd) stage for four mature application types such as Image Segmentation and ask the most questions about topics primarily relating to *Model Setup* (3rd) stage for four application types concerning emerging methods such as Transfer Learning.

In addition to answering these RQs that reveal the varied and interconnected landscape of DL development stages, developer needs, and DL-applications types, our contributions also include methodological improvements. From a comprehensive set of topics based on a careful LDA analysis of SO questions and repository READMEs to the reuse of LDA topics derived from repository READMEs for classifying SO questions, they all bring robustness improvements for the notoriously hard topic analysis area. We believe that our approach of using all public data to investigate an area of software development could be applied not just on SE4DL problems but more generally. Finally, based on our findings, we distill several actionable insights for SE4DL research, practice, and education.

The rest of the paper is organized as follows. The data collection, data preprocessing, and topic modeling processes are described in Section 2. Sections 3, 4, and 5 present the methods and results for the three research questions respectively. Section 6 discusses the implications for SE4DL research, practice, and education. We discuss limitations in Section 7 and review the related work in Section 8. Finally, we conclude the paper in Section 9.

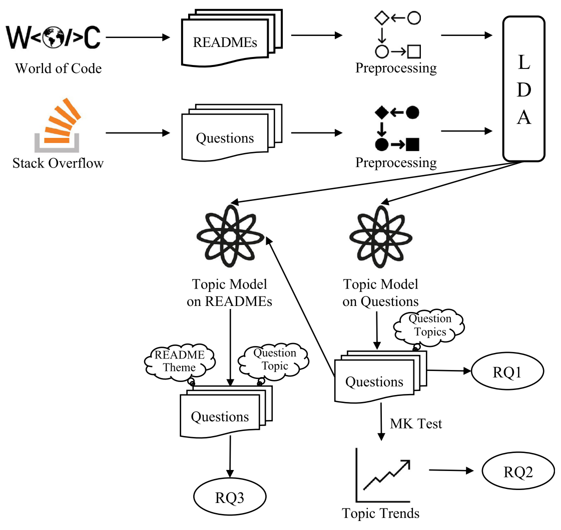


Fig. 1. Overview of methodology.

2 Data Preparation

This study follows the process depicted in Fig. 1. We select TensorFlow, Keras, and PyTorch for this study for three reasons:

1) They are under active development, allowing us to observe recent trends and to ensure the timeliness of the results;
2) The increasing numbers of downstream repositories and Stack Overflow questions better approximate current and emerging practical problems and application types;
3) They cover a broad range of DL framework implementations that represent past and emerging usage scenarios.

For these frameworks, we collect questions at Stack Overflow (SO) and README files of public repositories as described in Section 2.1, that represent the problems DL developers ask about and the application types DL developers work on. We then preprocess these artifacts (Section 2.2) and perform topic modeling with LDA on questions and READMEs, respectively (Section 2.3). Our data and scripts can be accessed at: https://github.com/KyleGau/SE4DL.

2.1 Data Collection

2.1.1 SO Data

SO is commonly used in research to understand problems faced by developers [8], [13], [17], [18], [19]. We downloaded a complete Stack Overflow Posts dataset from the official Stack Exchange Data Dump[1], which contains SO posts created from July 31, 2008, to March 1, 2021. It contains two types of posts: questions and answers. In this study, we focus on questions to gauge developers' needs. Each question may have one to five tags indicating related concepts and technologies. We regard a question as a DL-related question if at least one of its tags is "tensorflow", "keras", or "pytorch", resulting in 92,830 DL-related questions. The *Questions* column of Table 2 shows the number of questions related to each framework.

1. https://archive.org/details/stackexchange

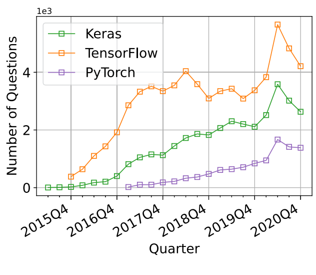
TABLE 2
Statistics of Collected Data

Framework	Questions	Repositories	READMEs	
			Raw	Preprocessed
TensorFlow	67,400	568,182	237,689	127,836
Keras	34,002	402,774	176,692	95,185
PyTorch	10,986	294,480	124,991	63,999
Total	92,830	998,514	429,204	227,756

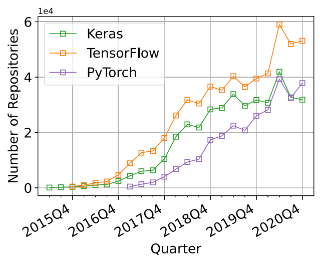
Fig. 2a shows the number of quarterly created questions related to each framework. All three frameworks are trending up over the years with minor differences. It suggests either more DL developers are using SO over time or existing developers encounter new problems. TensorFlow questions show a sharp increase before 2017Q2, followed by two peaks in 2018Q2 and 2020Q2. The peaks may be related to the releases of TensorFlow 1 and TensorFlow 2, which introduced many breaking changes and sparked many questions initially. Once the number of questions accumulates to a certain level, the number of new questions gradually decreases and stabilizes. Keras questions also show a peak in 2020Q2 partly because it comes packaged with TensorFlow 2 [20]. PyTorch questions keep increasing and stabilize after 2020Q2.

2.1.2 WoC Data

To investigate how problems faced by developers vary over application types (RQ3), we need to address two challenges: identifying application types and relating SO questions to application types. We apply LDA on README files of collected repositories to identify application types of the repositories. Then, we use the fitted LDA model on README files to infer the application types of SO questions. Repositories (many of which are on GitHub) have tags or labels and README files that represent a natural language description of the project. Both could be used to identify application types. There are, however, several advantages of using READMEs instead of repository labels: 1) It allows us to analyze more DL-related repositories to mitigate sample bias since the ratio of repositories with READMEs is much higher than that of repositories with labels (0.3% based on GHTorrent [21]'s latest dump (March 6, 2021)). 2) READMEs are more suitable to be fed to LDA to fit a model as LDA performs poorly on short text [22] and README files usually contain more words than labels.



(a) Trend of questions



(b) Trend of repositories

Fig. 2. The trend of quarterly created DL-related questions and repositories.

```
000005efe300482514d70d44c5fa922b34ff79a5;Rayhane-mamah_Tacotron-2;
1557284915;[redacted];05604b3f0632e98cc0eee3afef589dc5031f3a43;PY;Python;synthesiz
er.py;tacotron.utils.text.text_to_sequence;tacotron.utils.plot;ta
cotron.models.create_model;wave;datasets.audio;os;librosa.effects;
tensorflow;infolog.log;datetime.datetime;io;numpy
```

Fig. 3. A technical dependency record that imported TensorFlow (we hide author information for the sake of privacy).

We use WoC to collect public repository READMEs needed in this study. WoC[2] is an infrastructure for mining the universe of open source version control system (VCS) data. It collects Git objects [23] of open source repositories across code hosting platforms, curates the collected data by, for example, deforking repositories and parsing dependencies from each version of source code files (i.e., technical dependencies), and provides a variety of ways to query the data. We use WoC query[3] to identify all versions of all files that import packages associated with at least one of the frameworks we study (TensorFlow, Keras, and PyTorch). The format of a technical dependency record is: `commit; repository excluding forks;timestamp;author; blob;language used in WoC;language determined by ctags`[4]`;filename;modules separated by semicolon`. An example technical dependency record that imported TensorFlow is in Fig. 3. We regard a repository as a DL-related repository if it contains a blob that imports one of the three DL frameworks: TensorFlow (import name is `tensorflow`), Keras (import name is `keras`), or PyTorch (import name is `torch`). This results in 998,514 DL-related repositories as shown in the *Repositories* column of Table 2. The identified DL-related repositories cover a broad spectrum of programming languages such as Python, Java, and Go and spread multiple platforms such as GitHub, Bitbucket, and GitLab. In this study, we use the latest version of WoC, which was labeled as ***"T"*** and collected data up to February 2021.

Fig. 2b shows the number of quarterly created repositories importing the three frameworks. Like SO questions, the numbers of DL-related repositories all increase. Notably, the number of repositories importing TensorFlow grows faster than Keras between 2019Q3 and 2020Q1. It may be because that Keras comes packaged with TensorFlow 2 as `tensorflow.keras` [20] so that Keras users have to import TensorFlow to use Keras since TensorFlow 2. The number of repositories importing PyTorch keeps rapidly increasing, in line with the trend of PyTorch questions.

We also use WoC to overcome the time and space consumption challenges of obtaining these near 1M repositories' READMEs. We first retrieve the latest commit for each repository, then we obtain each repository's root folder structure from the tree object pointed by the latest commit. Finally, we check if "README.md" is contained in the root folder. If contained, we retrieve its content by its SHA-1 hash. Using this algorithm, we find 525,451 distinct repositories containing README.md in WoC "T" version.

2. https://worldofcode.org/
3. https://github.com/woc-hack/tutorial#activity-6-investigating-technical-dependencies
4. https://github.com/universal-ctags/ctags

2.2 Data Preprocessing

We preprocess collected SO questions and public repository READMEs to make the data suitable for LDA analysis.

2.2.1 SO Questions

As in prior work [6], [17], [18], [19], we preprocess SO questions' title and body: (1) remove code snippets marked with `<code></code>` or `<blockquote></blockquote>`; (2) remove HTML tags such as paragraph `<p></p>` and URLs `<a></a>`; (3) remove numbers, punctuation, and other non-alphabetic characters; (4) remove stop words such as 'a' with Mallet's English stoplist [24]. We also extend this stoplist with 'tensorflow', 'keras', and 'pytorch'; (5) bigram model is built using Gensim[5] since bigram model could improve the quality of text processing as reported by Tan *et al.* [25]; (6) Snowball stemmer provided by NLTK[6] is applied to reduce words to their stemmed representations, for example, "install", "installation", and "installing" are all stemmed to "instal".

2.2.2 Repository READMEs

README files contain not only information related to the functionality of a repository which we use to determine the application type of the repository's code, but also information related to installation, requirements, etc. [26], [27]. Since LDA is sensitive to input data, it's important to extract relevant information from README files. To accomplish that, Sharma *et al.* [26] extracted sections that are most similar to the repository description presented on its homepage. However, such description is optional and many repositories don't have one. We, therefore, conduct a preliminary study to investigate which section[7] of README describes the repository's functionality without referring to an external resource such as repository description. To accomplish that, we sample 384 READMEs (the number was chosen to be able to obtain a 95% confidence interval[8]). The first two authors independently checked which section contains the description of the repository's functionality separately. The Kappa value [28] between the two authors is 84%, which indicates an almost perfect agreement. We then held a meeting to resolve the inconsistencies. We find 8.1% (31) of READMEs to be in non-English languages. Of the remaining 353 READMEs, 14.7% (52) don't contain information about the repository's functionality, and 84.4% (298) READMEs have descriptive text on functionality in their first or second sections.

Based on the preliminary study, we exclude non-English READMEs with langid[9] and 429,204 READMEs remain as shown in the *Raw* column of Table 2. We then extract the first two sections of the remaining READMEs and preprocess the extracted text (we also refer to the extracted text as README in the following). As open source repositories serve other purposes besides software development [29], we remove READMEs that contain words explicitly

5. https://radimrehurek.com/gensim_3.8.3/
6. https://www.nltk.org
7. Following prior work, we define a section as the text in between two successive headers in a README file. And a header is included in the section behind it.
8. https://www.surveysystem.com/sscalce.htm
9. https://github.com/saffsd/langid.py

indicating that the repository is not for software development such as "tutorial" and "mooc". Then we preprocess the remaining READMEs: remove code snippets enclosed between '; remove URLs, numbers, punctuation, and other non-alphabetic characters; remove stop words, build bigram model, and stem words the same as Section 2.2.1. After removing empty preprocessed READMEs, 227,756 remain as shown in the *Preprocessed* column of Table 2.

2.3 Topic Modeling

Topic modeling is an unsupervised text-mining technique that automatically discovers hidden semantic structures (i.e., topics) in a text corpus. LDA (Latent Dirichlet Allocation) [30] is an extensively used topic modeling method in SE research community [15], [17], [18], [19], [26], [31], [32], [33], [34], [35]. We apply LDA on the preprocessed question corpus and README corpus, respectively, to identify question topics asked by DL developers and application types DL developers work on. We elaborate on how we fit LDA models and label LDA topics in the following.

2.3.1 *Fitting LDA Models*

LDA posits that each document in the corpus is modeled as a finite mixture over an underlying set of topics and a topic is modeled as a finite mixture over words in the corpus. Then it builds a model based on word frequencies and word co-occurrences to estimate the two distributions — document-topic distribution and topic-word distribution. We use Gensim's Python wrapper for Mallet LDA[10], which implements LDA with Gibbs sampling and is commonly used in previous work [17], [18], [19]. We set a constant random seed for the Gibbs sampler to eliminate the instability introduced by Gibbs sampling.

LDA requires multiple parameters to work well [35], [36], [37], [38]: a) topic number K; b) iteration number I in Gibbs sampling; c) the parameter $\vec{\alpha}$ for the prior distribution of document topics; d) the parameter $\vec{\beta}$ for the prior distribution of topic words. Earlier work [39] shows that an asymmetric (i.e., topics have different values) $\vec{\alpha}$ and a symmetric (i.e., all words share the same value) $\vec{\beta}$ could increase the robustness of LDA to variations in the number of topics and the highly skewed word frequency distributions. We, therefore, use Mallet's hyperparameter optimization to allow the model to learn asymmetric $\vec{\alpha}$ and symmetric $\vec{\beta}$ from the corpus[11]. We set `-optimize-interval` 10 iterations as suggested by Mallet.

Several heuristics have been proposed to tune LDA parameters such as Genetic Algorithms (GA) [36], Differential Evolution (DE) [37], and the iterated f-race procedure (irace) [35]. Several fitness functions are also proposed to measure how the LDA model fits the data such as perplexity [40], topic coherence [41], silhouette coefficient [36], and raw score $\mathcal{R}_n$ [37]. However, a recent study [38] reveals no heuristic and/or fitness function outperforms all the others. We use the heuristic GA, which searches for the optimal solution by simulating the natural evolutionary process and

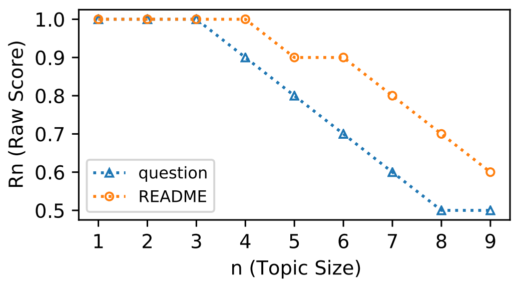


Fig. 4. Raw score $\mathcal{R}_n$ of tuned LDA for question corpus and README corpus.

fitness function, topic coherence (C_v) which measures the understandability of topics generated by LDA, to tune the remaining two parameters K and I for two reasons: 1) GA is widely used in SE community to tune LDA parameters on SO questions and repository READMEs [26], [33], [36], which are corpus also used in our study to identify question topics and application types. 2) C_v has been proved to be highly correlated with human judgment [42] and is widely used in recent SE studies [6], [18], [19], [43]. The tuning process is as follows: GA first generates p different parameter configurations (called population); for each parameter configuration, it runs LDA and computes the C_v score of the fitted LDA model; according to the p C_v values, GA generates new configurations and repeats the above step (begin a new generation); with the generations evolving, better and better parameter configurations emerge. We use Pyevolve[12] implementation of GA. We set the LDA parameter search space as $K \in [5, 50], I \in [500, 2000]$. We set both population size and generation size to 100 to ensure sufficient configurations are explored following prior work [26], [36].

The optimal parameter configuration for question corpus and README corpus is $K = 27, I = 772$ and $K = 26, I = 891$ respectively, and the corresponding C_v values are 0.62 and 0.59. We then assess the stability of the two tuned LDA models as found by Agrawal *et al.* [37] that LDA suffers from order effects [44]. We use the metric, raw score $\mathcal{R}_n$ proposed by [37] to measure LDA stability. $\mathcal{R}_n$ denotes the median number overlaps of topic size with n words across multiple LDA runs. We set $1 \leq n \leq 9$ following prior work. For each corpus (question corpus and README corpus), we calculate $\mathcal{R}_i, i \in [1, 9]$ as follows. We run LDA 10 times with corresponding optimal K and I, each time shuffling the corpus, then we calculate $\mathcal{R}_i$ in the 10 runs. We repeat the above process 10 times to avoid any sampling bias and choose the median of the 10 $\mathcal{R}_i$ scores. The results are shown in Fig. 4. Overall, all $\mathcal{R}_i$ scores are no less than 50% for each corpus. Specifically, when reporting topics of up to nine words, in half cases, all the topics can be found in models generated using different input orderings, which is considered stable according to [37]. Therefore, we use the top nine words of each topic to label topics as described in Section 2.3.2.

2.3.2 *Labeling LDA Topics*

LDA generates topics represented as a probability distribution over words but topics' actual meaning is subject to interpretation. Often the most probable words for a topic

10. https://radimrehurek.com/gensim_3.8.3/models/wrappers/ldamallet.html
11. http://mallet.cs.umass.edu/topics.php
12. https://github.com/BubaVV/Pyevolve

TABLE 3
Stages, Names, Question Count, Percentage of Questions Having an Accepted Answer (*% acpt)*, Adjusted p-Values, Trend, and Sen's Slope for 27 Question Topics Sorted by Stages and Question Count

Stages	Topic Name	Count	% acpt	Adjusted P-value	Trend	Sen's Slope
Preliminary Preparation	Installation Error	**6556**	29.9	1.0	–	-9.03e-05
	Build Error	3299	30.1	0.00029	↓	-1.93e-04
Data Preparation	Tensor Operation	4163	**51.3**	1.5e-06	↓	-2.79e-04
	Image Preprocessing	3500	37.7	4.8e-07	↑	3.00e-04
	Data Type	3376	**42.3**	1.0	–	7.90e-05
	Data Load	3278	35.4	1.0	–	-6.46e-05
	Data Batch	2623	38.4	1.0	–	-2.69e-06
Model Setup	Model Load	**5225**	35.8	2.5e-09	↑	**4.28e-04**
	Graph Session	4021	40.5	0.0	↓	-1.16e-03
	Layer Operation	3459	**43.9**	1.0	–	4.72e-05
	Tensor Shape	3455	**45.3**	0.0032	↑	1.54e-04
	Probability	3373	37.9	0.0061	↓	-1.86e-04
	LSTM	2426	35.9	0.00040	↓	-2.55e-04
	Embedding	2218	31.3	1.0	–	3.25e-05
Model Training	Loss Function	4333	38.4	0.034	↑	1.49e-04
	Device Use	3963	28.3	2.3e-06	↓	-2.90e-04
Model Evaluation	Evaluation Metrics	3398	35.9	5.2e-10	↑	3.41e-04
Model Tuning	Training Anomaly	3724	33.3	3.2e-12	↑	**4.69e-04**
Model Prediction	Model Conversion	2601	25.7	1.4e-13	↑	**4.20e-04**
Multiple-Stage Topics	Code Error	**5818**	34.4	7.1e-12	↑	**5.75e-04**
	API Usage	**4473**	40.5	9.2e-10	↓	-4.70e-04
	Review	3822	42.0	0.0096	↓	-1.34e-04
	API Misuse	2608	37.0	0.00012	↑	1.74e-04
	Classification	2198	39.3	1.0	–	7.00e-05
	Reinforcement Learning	2150	38.6	0.16	–	-9.37e-05
	Object Detection API	1976	25.1	0.0012	↑	3.59e-04
	Error Traceback	794	24.1	0.15	↑	5.69e-05

The Adjusted P-value *column presents the p-values adjusted by Holm–Bonferroni method [45]. '↑', '↓', and '–' in the* Trend *column denote increasing, decreasing, and unchanging trend respectively.*

are often used to assign a subjective label. In our approach, which is much more effort-intensive but also much more likely to lead to meaningful labels, we use full documents (SO questions and README files) to give label names. Specifically, to label these topics, we follow the general procedure of open card sort, which is frequently used to label LDA topics in SE research (e.g., [17], [18], [19]). In open card sort, there are no predefined topic names. Instead, they come up during the labeling process. We first assign each document to the topic with the highest probability in its topic distribution. Then the first two authors, each has three or four years of DL experience respectively, manually check each topic's top nine words and read through 30 randomly selected documents assigned to that topic. Then, they give a topic name that best explains the words and documents of that topic. The process is iterative, where the authors individually perform labeling, jointly unify topic names, discuss conflicts, and refine topic names until they agree on topic names. An arbitrator, who has five years of DL experience and is skilled at all the three frameworks, is invited to review the topic names. The arbitrator is someone external to the project. He agreed with most (49/53) topic names and provided better phrasing suggestions for the remaining four topics. These suggestions are discussed and integrated into the final topic names. For example, one question topic was initially labeled as *Dataset*, after checking its top nine words and the 30 randomly selected documents assigned to it, he suggested that *Data Load* is clearer. After a discussion, we adopted his suggestion.

3 RQ1: How are Problems Faced by DL Developers Distributed Over SE4DL Stages?

3.1 Methods

LDA reveals 27 topics for the 92,830 SO questions shown in the *Topic Name* column of Table 3. The number of questions assigned to each topic is shown in the *Count* column. We also calculate the percentage of questions having an accepted answer for each topic shown in the *% acpt* column. We use the seven DL development stages proposed previously [6] as shown in Table 1. The stages were derived by analyzing the architecture documentation of several DL frameworks. To relate question topics to SE4DL stages, we manually label 1797 randomly sampled questions. Specifically, we determine the sample size based on the 90% confidence level, resulting in 63 to 67 questions for each question topic. Then, the first two authors manually label the SE4DL stages of 1797 questions independently following the definition of stages as described in Table 1. The Kappa value between the two authors is 82%, reaching an almost perfect agreement. The inconsistencies are resolved through discussion. If more than two-thirds of sampled questions of a topic relate to the same stage, we refer to that topic as a single-

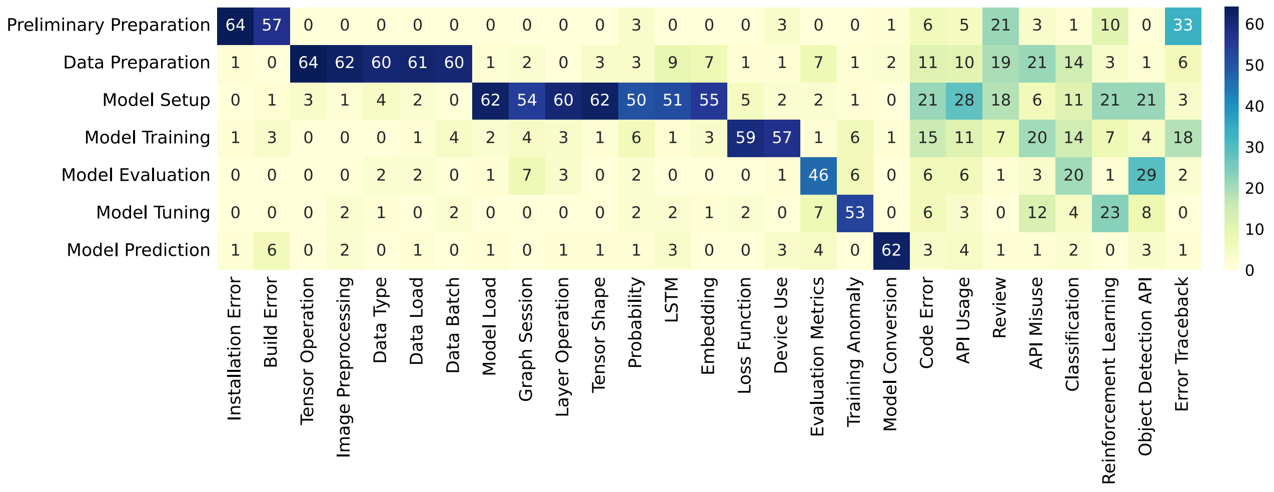


Fig. 5. The occurrence of sampled questions at each SE4DL stage for each question topic. The horizontal axis represents question topics and the vertical axis represents SE4DL stages. Single-stage topics are arranged by the stage and question count. Multiple-stage topics are arranged by question count.

stage topic primarily relating to the stage. Otherwise, we refer to the topic as a multiple-stage topic. The threshold we choose, 2/3, is a commonly used supermajority threshold for voting in various government and social organizations [46]. We denote the 27 question topics as $t_1, t_2, ..., t_{27}$ respectively and the seven stages as $s_1, ..., s_7$ respectively. We denote $ratio(t_i)$ as the ratio of questions assigned to t_i over all (92,830) questions and $ratio(t_i, s_j)$ as the ratio of questions relating to s_j in the sampled questions of t_i, then we estimate the ratio of questions relating to s_i as $ratio(s_i) = \sum_{k=1}^{27} ratio(t_k) * ratio(t_k, s_i)$.

3.2 Results

Fig. 5 shows the occurrence of the 1797 sampled questions at each SE4DL stage. All the 27 question topics have different distributions over stages, and none are exclusive to one stage as was assumed in prior work [6]. In total 19 question topics primarily relate to a single SE4DL stage (but occasionally they relate to other stages), and eight topics span multiple SE4DL stages. Among the 19 single-stage topics, the primary stages of 13 topics account for over 90% of the sampled questions. For the remaining 6 topics, after excluding the questions relating to their primary stages, none of the rest stages dominate (i.e., account for over 2/3 (i.e., supermajority) of) the remaining questions.

The 19 single-stage topics have their primary SE4DL stages ranging from the first to the last. *Installation Error* and *Build Error* are topics primarily relating to the 1st stage, *Preliminary Preparation*. The *Installation Error* topic takes the most questions (6,556, 7.1%) in all topics. And both topics have relatively low *% acpt* with only 29.9% and 30.1% questions having an accepted answer respectively, indicating that developers fail to get good answers, possibly suggesting that their presumably novice questions may be poorly formulated [47]. Overall developers ask ~13.3% questions about the 1st stage. DL frameworks may try to ease the procedure of setting up environment and help developers focus on actually using DL frameworks.

Developers ask ~23.3% (the second most) questions about the 2nd stage, *Data Preparation* with five topics primarily relating to it: *Tensor Operation*, *Image Preprocessing*, *Data Type*, *Data Load*, and *Data Batch*. This stage usually involves loading raw data into the program, performing data preprocessing and augmentation, converting data to correct format, and finally generating data batches for training and evaluation. Although DL frameworks provide functionalities to facilitate this procedure, developers face various problems using these functionalities. For example, in Question 49034250, a developer asked *"Does Keras flow_from_directory iterate through every sample in a directory?"* when using Keras's `ImageDataGenerator` module. *Tensor Operation* has the highest *% acpt*, and is the most common among the five topics primarily relating to the 2nd stage. The high *% acpt* suggests that such questions may be sufficiently well formulated to result in an acceptable answer. The relatively high number of such questions suggests that even such a simple topic as tensor operations is not completely obvious and well understood by DL developers or that its implementation in the considered frameworks may be problematic. Such problems are likely to be alleviated by improving DL frameworks' documentation.

Developers ask the most (30.7%) questions about the 3rd stage, *Model Setup* with seven topics primarily relating to it, including *Graph Session* which asks how to operate static computation graphs, *Model Load* which discusses how to correctly load pre-trained models, *Layer Operation* which asks how to create and link various neural layers, *Probability* which discusses issues on how to manipulate probability distributions and emerging probabilistic modeling, *Tensor Shape* which includes questions about how to correctly fit tensor shape into layers, and two kinds of neural layers *LSTM* and *Embedding*. Specifically, developers ask the most about *Model Load* topic in the seven topics with 5225 (5.6%) questions. Considering the high frequency of questions on the topic, better support for loading pre-trained and compatible models is urgently needed.

Developers ask ~14.4% questions about the 4th stage, *Model Training* with two topics, *Loss Function* and *Device Use*, primarily relating to it, indicating that developers mainly have problems with creating custom loss functions and configuring computing resources correctly and efficiently at this stage. The last three stages have only one topic primarily relating to them respectively. Specifically, *Evaluation Metrics* questions on how to evaluate DL models primarily relate to the 5th stage, *Training Anomaly* questions on how to fix abnormal training results primarily relate to the 6th stage (*Model Tuning*), and *Model Conversion* questions on how to correctly convert trained models for deployment primarily relate to the last stage (*Model Prediction*). Overall, developers ask ~6.9%, 6.7%, and 4.8% questions about the last three stages respectively.

The remaining eight question topics span multiple stages including five topics relating to framework APIs (*Code Error*, *API Usage*, *API Misuse*, *Object Detection API*, and *Error Traceback*), two topics relating to application tasks (*Classification* and *Reinforcement Learning*), and *Review*.

The five multiple-stage topics relating to framework APIs reflect the common needs for easier-to-use DL framework APIs across different stages. Specifically, questions in *Code Error* topic spread all stages and occur the most at *Model Setup* and *Model Training* stages. Developers usually provide code and error messages in such questions, e.g., *Keras 'InputLayer object has no attribute 'inbound_nodes' when converting to CoreML* (Question 48329150), indicating they fail to debug errors from the error messages. About half of sampled *API Usage* questions relate to *Model Setup* stage which discuss how to implement something using a specific API or errors when using a specific API. For example, developers are frequently confused about the difference between APIs in `torch.nn` and `torch.nn.functional` (e.g., Question 63826328) where they provide the same functionality but in different ways with the former in class-style and the latter in function-style. Besides, developers asking *API Usage* questions appear to be predominantly novices: among the 67 sampled questions, ten questions occurred when developers were running tutorial code and nine questions explicitly contain "I am new"-like phrases. This finding indicates that, perhaps not surprisingly, novices have the greatest challenges in understanding APIs from the documentation. Questions in *Object Detection API* topic discusses the use of TensorFlow Object Detection API[13] and mainly span *Model Setup* and *Model Evaluation* stages. Developers usually draw bounding boxes (or frames) in images to show the detected objects according to the coordinates produced by the model at *Model Evaluation* stage. But they face various questions in the procedure such as *How to output box coordinates produced from Tensorflow Object Detection API* (Question 48284800).

Two topics (*Classification* and *Reinforcement Learning*) spanning multiple stages relate to application tasks. For *Classification* topic, developers mainly have problems with the second to the fifth stage. Developers mainly ask questions about how to deal with imbalanced data at *Data Preparation* and *Model Setup* stage. At *Model Training* stage, developers ask about the use and differences of various loss functions such as categorical cross entropy and binary cross entropy. At *Model Evaluation* stage, developers ask about how to interpret model output such as *How to set different thresholds for each class in multi-label classification* in Question 62439043. Questions of *Reinforcement Learning* topic mainly span *Preliminary Preparation*, *Model Setup*, and *Model Tuning* stages. For *Review* topic, developers mainly seek practices and suggestions about the first three stages when applying DL in practice.

Unlike the finding reported by prior work [6] that developers ask the most questions about *Preliminary Preparation* and *Model Training* stages, we find that developers ask the most questions about *Data Preparation* and *Model Setup* stages. Besides, [6] reported no topic in *Model Tuning* stage, while we obtain a topic, *Training Anomaly*, primarily relating to this stage. Two reasons may attribute to such differences. One, prior work assigned each topic to a single stage, while we find that a topic may span multiple stages; Two, prior data was collected before April 2018 while our data is collected before March 2021. Over three years some changes may have taken place in the DL domain with the advent and improvement of supporting tools and theories. Therefore, developers' questions about SE4DL stages may have changed markedly. We, therefore, further investigate the time variability of SE4DL needs in RQ2.

Summary for RQ1: None of the 27 question topics revealed by LDA for SO questions are exclusive to one stage as was assumed in prior work. In total 19 topics primarily relate to a single SE4DL stage and eight topics span multiple stages. The 19 single-stage topics cover all seven SE4DL stages and the eight multiple-stage topics are mainly about framework APIs and application tasks. Overall, developers ask the most about the second (*Data Preparation*) and third stages (*Model Setup*) with 23.3% and 30.7% questions respectively, in contrast to the former study that found the first (*Preliminary Preparation*) and fourth (*Model Training*) stages to be the stages with the most questions [6].

4 RQ2: How do These Problems Vary Over Time?

4.1 Methods

For the 27 SO question topics identified in Section 3.2, we calculate each topic's relative rate over time where the total number of questions assigned to each question topic is compared to the total number of DL-related questions for each month. We then use Mann-Kendall trend test (MK test) [48] to identify the trend, i.e., the change of relative rate, of the 27 question topics at 0.05 significance level following prior work [18]. MK test is a non-parametric test used to identify monotonic trend in a series and is not affected by the length of series. We also use Theil-Sen's slope estimator (Sen's slope) [48] to measure the magnitude of monotonic trend, which is often used together with MK test. Since we perform 27 MK tests, we adjust the p-values using the Holm–Bonferroni method [45] to control the family-wise error rate, which has been widely used in SE studies [49], [50], [51], [52].

4.2 Results

The *Trend* column in Table 3 presents the trend of question topics identified by MK test based on the adjusted p-values shown in the *Adjusted P-value* column. There are three kinds

13. https://github.com/tensorflow/models/tree/master/research/object_detection

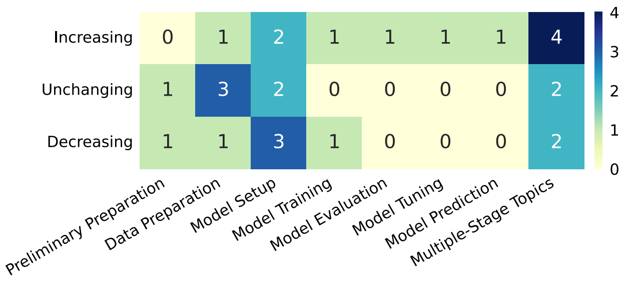


Fig. 6. Trend of single-stage topics grouped by stages and multiple-stage topics. The horizontal axis represents SE4DL stages and multiple-stage topics and the vertical axis represents trend.

of trends: increasing, decreasing, and unchanging (neither decreasing nor increasing at 0.05 significance level). The *Sen's Slope* column in Table 3 shows the measured magnitude of the trend for each topic. Fig. 6 presents the trend distribution of question topics. As Fig. 6 shows, topics primarily relating to the first three stages mainly (11 out of 14) show unchanging or decreasing trends, and the three topics primarily relating to the last three stages all show increasing trends. Overall, eleven, eight, and eight question topics increased, decreased, and didn't change over time respectively.

Increasing Trend. Seven among eleven increasing topics are single-stage topics including *Image Preprocessing* primarily relating to *Data Preparation* stage, *Tensor Shape* and *Model Load* at *Model Load* stage, *Loss Function* at *Model Training* stage, and the rest three primarily relating to the last three stages. *Training Anomaly* has the second-highest increasing rate and has 33.3% questions with an accepted answer. This suggests that problems encountered during the tuning stage are becoming relatively more common and are less likely to receive an answer, possibly because that developers don't know what context would be helpful to fix these problems. *Model Load* increases at the third-highest rate and developers ask the second most (5,225, 5.6%) questions about it, suggesting that developers' increasing needs for using pre-trained models are not well met by existing DL frameworks. Developers mainly have two kinds of problems about using pre-trained models. On the one hand, as revealed by [8], developers often face inconsistent behavior after loading pre-trained models due to the difference in frameworks, platforms, or framework versions. On the other hand, developers struggle with current procedure of loading pre-trained models. For example, a developer asked *How to read keras model weights without a model* because loading a pre-trained model assumes that its model architecture exists but the developer didn't know the architecture. In this case, DL frameworks may provide more flexible support on loading pre-trained models to ease the procedure. *Model Conversion* has the fourth-largest increase with only 25.7% questions having an accepted answer (the 3rd lowest), indicating developers are increasingly using or have more issues with model conversion technique and also have difficulty obtaining solutions on SO. As revealed in [13], developers' demand to deploy DL software to specific platforms for prediction is increasing. Although some tools such as TFLite, CoreML, and ONNX are rolled out to facilitate the deployment process, the model conversion support across platforms and frameworks appears to be incomplete [13], which possibly results in the increasing relative rate of *Model Conversion* questions. An abstract of model format conformed by different frameworks and platforms may alleviate this problem.

The remaining four topics with increasing rates represent half of the multiple-stage topics and are all concerned with framework APIs, including *Code Error*, *API Misuse*, *Object Detection API*, and *Error Traceback*. The *Code Error* topic has the highest Sen's slope of 5.75e-04. We find the rapid increase periods of *Code Error* questions overlap with the major release time of TensorFlow. Specifically, a rapid increase occurred in the first half of 2017 which rises from 4.6% to 6.6% (177 to 502 in absolute count) in terms of half-yearly ratios. TensorFlow 1.0 was released in February 2017 [53]. The second rapid increase occurred in the second half of 2019 with the rate increasing from 6.5% to 7.9% (649 to 768 in absolute count). TensorFlow 2.0 was released in September 2019 [54]. According to Tables 1 and 2 in [55], DL frameworks release frequently with many breaking changes that affect many projects, which undoubtedly increases the cost of mastering DL frameworks and results in many questions related to API errors and misuses. It appears DL frameworks need to improve their backward compatibility. The *Object Detection API* topic has the fifth largest increase with only 25.1% questions having an accepted answer (the 2nd least), suggesting developers have substantial difficulties in using TensorFlow Object Detection API and also face difficulties obtaining a solution on SO. Developers may lack adequate documentation when using these APIs, which may explain the increase and low *% acpt* of this topic. For example, in Question 49148962, a developer asked for *Tensorflow object detection config files documentation* and complained *I could not find any documentation or tutorial on the options for these config files though*. TensorFlow Object Detection API only provides documentation in the form of markdown files in its repository[14], which may be harder for developers to find and use. Therefore, TensorFlow may consider improving the readability and usability of the Object Detection API documentation.

Decreasing Trend. The decreasing-rate topics include six single-stage topics and two multiple-stage topics. Notably, half (3/6) of decreasing-rate single-stage topics primarily relate to *Model Setup* stage. These decreasing-rate topics may indicate that developers' needs may have been well met over time. Not surprisingly, substantial efforts have been devoted to improving DL frameworks. Studies reported by [7], [8] show that the static computation graph adopted by TensorFlow 1 was a major root cause of common programming issues. TensorFlow 2 introduced dynamic computation graph (i.e., eager execution) as a default option which may help set up and debug models [54]. The impact of this improvement is supported by our analysis as well. For example, *Graph Session* questions have the largest decrease.

Unchanging Trend. The topics exhibiting no trend are mainly (6 of 8) related to the first three stages, with three topics at the *Data Preparation* stage. The lack of change for some of the topics may be not the direct fault of the lack of improvements in frameworks but may be attributed to the third-party libraries. The topic that developers ask the most

14. https://github.com/tensorflow/models/tree/master/research/object_detection/g3doc

questions about, *Installation Error*, is often resulted from intricate third-party dependencies of DL frameworks. Even when an error occurs, developers may not be able to diagnose the causes from the traces reported by the framework. For example, some *Installation Error* questions are caused by version incompatibilities between dependencies such as the incompatibility between CUDA v9 and Ubuntu 14.04 (Question 54021556). *Data Type* topic involves conversion and operation among different data types supported by DL frameworks and other libraries such as Numpy. But developers, especially beginners, usually lack a clear understanding of the subtle differences between different data types and the data type requirements of operators. For instance, a typical error occurs when developers use the image transformations provided by PyTorch's `torchvision` library. Most transformations accept input of both PIL `Image` type and PyTorch `Tensor` type and return the same type as input. But some only accept one of the two types, e.g., `Normalize`. Therefore, developers should convert PyTorch tensors to and from PIL images with `ToPILImage` and `ToTensor` according to the transformation requirements. Otherwise, an error would occur. For example, a developer encountered a TypeError because he fed `Resize`'s output which is PIL `Image` type directly to `Normalize` which only accepts input of `Tensor` type. A possible solution is to improve the compatibility between DL frameworks and third-party libraries.

If comparing all the topics, questions for the increasing-rate topics are less likely to receive an accepted answer than for the unchanging- and decreasing-rate topics indicated by *% acpt*. In particular, the mean/median *% acpt* for the increasing-rate topics is 35.3%/35.8%, for decreasing-rate is 38.7%/38.6%, and for the unchanging-rate is 36.4%/38.4%. On the other hand, topics with a higher fraction of questions having an accepted answer tend to have decreasing or unchanging rate. For example, out of the top ten topics with the highest *% acpt*, eight show decreasing or unchanging trends.

Summary for RQ2: Among the 27 question topics, the relative rates of questions for 11 topics increased, for eight topics decreased, and for the remaining topics didn't change over time. Questions for the 11 trending up topics are less likely to receive an accepted answer than questions of the remaining topics. The topics that have the largest increases (as indicated by Sen's slope) are *Code Error*, *Training Anomaly*, *Model Load*, and *Model Conversion*, and the topic that decreases the most is *Graph Session*. The topic that developers ask the most questions about, *Installation Error* hasn't increased or decreased significantly over time.

5 RQ3: How do These Problems Vary Over Application Types?

5.1 Methods

As discussed in Section 2.1.2, there are two challenges to answer RQ3: identifying application types and relating SO questions to application types. In the following, we elaborate on how we tackle the two challenges.

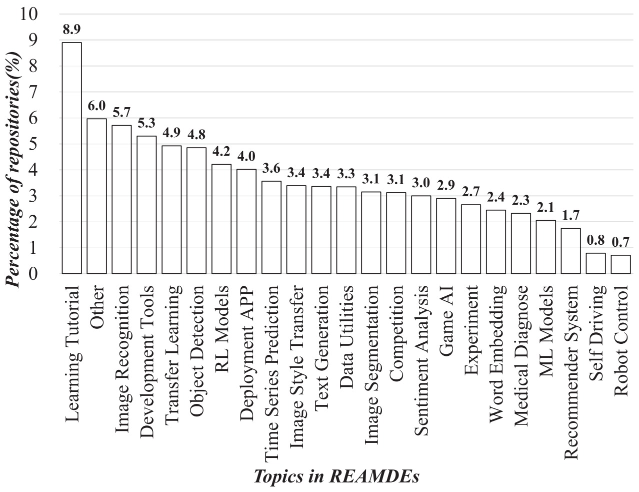


Fig. 7. Distribution of labeled README themes. RL stands for reinforcement learning and ML stands for machine learning.

5.1.1 *Identifying Application Types*

We identify DL application types by performing LDA on README files as described in Section 2.3, which results in 26 themes[15] (i.e., application types) for 227k repositories. However, four themes could not be labeled because the associated README files don't contain information on the repository's functionality. For example, one such theme's top nine words are *instal python run environ requir packag depend pip numpi* and 30 randomly sampled READMEs from this theme contain only environment information. We thus can not infer what the functionality of the software in these repositories is. The presence of these four functionality-unrelated themes is consistent with Sharma *et al.*'s results where 16 out of 49 themes could not be labeled according to functionality [26].

The remaining 22 themes do have information reflecting the functionality of the underlying software and are shown in Fig. 7. Although we have removed READMEs of some non-software development repositories depending on certain words as described in Section 2.2.2, some READMEs of repositories that are irrelevant to software development remain in the sample as some keywords may refer to software development or other activities. For example, "learning" can be used to express "learning tutorial", but also to express "reinforcement learning". As a result, we find two themes that appear to be irrelevant to software development, i.e., *Learning Tutorial* including repositories with various code examples, and *Competition* including repositories for Kaggle competitions. Most of the remaining 20 themes have self-explanatory names with a few below that require more explanations. *Other* theme includes repositories of DL in other disciplines such as material design[16]. *Development Tools* and *Data Utilities* include various packages and scripts to enhance DL frameworks and prepare data. *ML models* repositories implement various simple ML algorithms such as linear and logistic regressions. The remaining 16 themes, representing 51.1% (116,362/227,756) repositories, include specific DL methods like *Transfer Learning* and application

15. To avoid confusion with SO question topics, we refer to LDA topics obtained from the README corpus as "themes."

16. https://github.com/DesignInformaticsLab/fracture_network

tasks like *Image Recognition*. We, therefore, focus our analysis of application types on these 16 themes and investigate the question topic distribution for these application types represented by each of the 16 themes.

5.1.2 Relating SO Questions to Application Types

We apply the LDA model fitted on READMEs (LDA_r) to the 92,830 SO questions to relate SO questions to application types. Specifically, LDA_r infers the README theme probability distribution for each SO question, and we relate each SO question to the README theme that has the highest probability. Only 24,837 questions (27% of all 92,830 questions) could be related to the 16 application types, possibly because DL-related questions on SO do not always contain sufficient descriptions of the application type context.

To validate the LDA inference performance of each application type, for each application type, we randomly select 20 SO questions related to it. Then the first two authors manually check whether these questions are truly related to the application type. After that, the two authors compare their results and resolve the conflict. We measure the LDA inference performance for an application type with the ratio of questions that are truly related to it in its 20 sampled questions. Overall, the LDA inference performance is above 60% for all application types, with the minimum of 65% (only for *Sentiment Analysis*), the maximum of 95%, and the average of 81%. We find that application types with more general words in their top nine words are more likely to have relatively low LDA inference performance. For example, only 65% (13 out of 20) questions are correctly related to the application type, *Sentiment Analysis*, whose top nine words are *project analysi sentiment final cs notebook report provid file*. In contrast, 95%(19 out of 20) questions are correctly related to the application type, *Object Detection*, whose top nine words are *detect object face video recognit imag project emot frame*. It is not surprising to observe such differences since LDA is based on word frequencies and word co-occurrences and general words are more likely to lead to grouping of unrelated questions.

To conclude, each of the 24,837 SO questions is related to a question topic and an application type. For each application type, we thus calculate the distribution of questions related to it over question topics.

5.2 Results

Each column in Fig. 8 shows the question topic distribution for an application type. The horizontal axis represents application types and the vertical axis represents question topics. The cell with the darkest color in each column indicates the most common question topics (i.e., with the highest ratio of questions) for that application type, which we call primary question topic. Overall, 12 application types' primary question topics cover 10 unique single-stage topics and the remaining four application types' primary question topics cover three multiple-stage topics. Below we use italic and sans serif to distinguish between question topics (in *italic*) and application types (in sans serif).

As we can observe from the distribution of primary question topics, the primary question topics for each application type are different. Four application types (Image Segmentation,

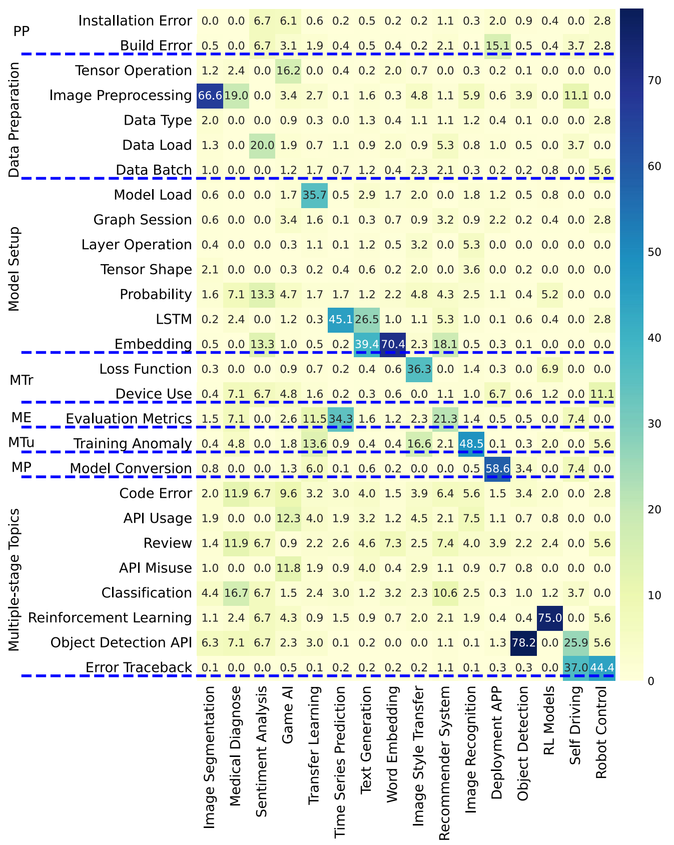


Fig. 8. Question topic distribution for the 16 application types. The horizontal axis represents application types and the vertical axis represents question topics. Normalization is done column-wise. We also separate question topics by stages with blue dashed line. PP, Mtr, ME, MTu, and MP stand for *Preliminary Preparation*, *Model Training*, *Model Evaluation*, *Model Tuning*, and *Model Prediction* respectively.

Medical Diagnosis, Sentiment Analysis, and Game AI) raise the most common questions related to topic *Image Preprocessing*, *Data Load*, and *Game AI*, which primarily relate to *Data Preparation* stage. These four application types are pervasive in daily life with mature solutions. Specifically, both Image Segmentation and Medical Diagnosis applications concern the most about question topic *Image Preprocessing*. Image segmentation is "the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics" [56]. Therefore, when performing image segmentation, developers need to ensure that pixel labels and images align. For example, a developer encountered a problem with images and labels rotated at different angles due to improper image preprocessing operations in Question 58846552. In this case, dedicated image preprocessing packages that help automatically align pixel labels and images could alleviate developers' problems with preprocessing image segmentation datasets. For Medical Diagnosis applications, developers usually deal with medical images to perform tasks like pneumonia detection and tumor segmentation. Medical images often come from different proprietary systems and may need other knowledge of the clinical data. Therefore, tools that process various formats of medical images with clinical knowledge may be beneficial. Sentiment Analysis mostly concerns *Data Load* questions suggesting a potential lack of standard ways or lack of clear documentation on how to associate text corpus with sentiment labels. For example, in Question 64986037, a developer failed to use the code provided in the official tutorial to load a larger dataset. Therefore, DL frameworks could provide more examples to show the complex use of dataload-related APIs in their tutorials.

The application types (Transfer Learning, Time Series Prediction, Text Generation, and Word Embedding) mostly concern question topics of *Model Load*, *LSTM*, and *Embedding*, primarily relating to *Model Setup* stage. These four application types concern emerging DL methods. Specifically, *Model Load* is the primary question topic of Transfer Learning with 35.7% questions. Transfer learning is an emerging DL method that applies knowledge gained from solving one problem to a different but related problem [57] and is an effective way to speed up training and improve the performance of DL models, especially when the training data is limited [58]. As shown in Fig. 7, transfer learning is widely adopted by DL developers to train models with 4.9% (of 227,756) repositories. Our finding suggests further improvement on current support on loading pre-trained models is necessary and urgent. *Embedding* is the primary question topic of both Text Generation and Word Embedding with 39.4% and 70.4% questions respectively. Embedding is usually used to densely represent text data and is widely used in many natural language processing tasks such as text generation [59]. As embedding has become pervasive and fundamental, many models are proposed to train better embeddings such as BERT[17]. But our finding indicates that developers have problems understanding and implementing these embedding models in practice.

Only one application type's primary question topic primarily relates to *Model Training* stage, i.e., Image Style Transfer whose primary question topic is *Loss Function* with 36.3% questions. It is possibly due to Generative Adversarial Networks (GAN), the method widely used in this application type. GAN involves a contest between two submodels, a generator model for generating new examples and a discriminator model for classifying whether generated examples are real or fake. It generally needs to combine two loss functions, one for generator and the other for discriminator, which adds complexity in implementing loss functions, e.g., *how to assign weights to these two loss functions using Keras* (Question 54068352). Besides, achieving equilibrium between the generator and discriminator also leads to difficulties tuning GAN, illustrated by 16.6% *Training Anomaly* questions. *Training Anomaly*, which primarily relates to *Model Tuning* stage, is the primary question topic of Image Recognition with 48.5% questions. A possible explanation is that many novices to DL usually get started from image recognition tasks such as the well-known handwritten digit recognition task, resulting in many questions about how to fix abnormal training results. Therefore, summarizing common model tuning practices may be helpful. Finally, *Model conversion*, which primarily relates to *Model Prediction* stage, is the primary question topic of Deployment APP with 58.6% questions, indicating that converting trained models to the format supported by the deployment environment is the biggest challenge when deploying DL software. As shown in Fig. 7, 4.0% repositories concern this application type, suggesting the popularity of deploying DL software and the urgency of better support for model conversions.

The remaining four application types' primary question topics cover three multiple-stage topics. Particularly, *Object Detection API* is the primary question topic in questions related to Object Detection with 78.2% questions. Object detection is a computer vision task of detecting instances of objects of a certain class within images or videos [60]. It is more and more used in many cases such as Tesla's Autopilot AI [61]. Many tools are proposed to help developers build object detection models such as TensorFlow Object Detection API and are widely used by developers. But as revealed in Section 4.2, developers sometimes suffer from using the documentation.

Summary for RQ3: We identify 16 application types related to software development in the 227k repositories. The most asked question topic varies across application types, with half primarily relating to the second and third stages. Specifically, developers ask the most questions about topics primarily relating to *Data Preparation* (2nd) stage for four mature application types such as Image Segmentation and ask the most questions about topics primarily relating to *Model Setup* (3rd) stage for four application types concerning emerging methods such as Transfer Learning.

6 Implications

Our results show how SE needs for DL vary across stages, time, and application types. In the following, we discuss implications for SE4DL research, practice, and education.

SE4DL Research & Practice. (i) Reduce the rate of preliminary preparation problems. DL frameworks have complex dependencies, which makes it difficult to install and build them successfully, thus unable to proceed any further. Developers ask the most questions about *Installation Error* and this topic is stable over time. Besides, only 29.9% *Installation Error* and 30.1% *Build Error* questions have an accepted answer. Although docker technology allows developers to package their code conveniently, it has several limitations. On the one hand, as revealed by Haque *et al.* [18], docker brings new challenges to developers. On the other hand, current pre-built docker images provided by DL frameworks usually contain complete functionalities and don't support functionality customization. As evidenced by the *Build Error* questions, developers sometimes need to customize the functionalities of DL frameworks for various reasons, e.g., reducing binary size [62] and adding custom ops [63]. Therefore, many developers still choose to install and build frameworks locally. Specifically, only 3.6% (36,140 out of 998,514) collected repositories contain a "Dockerfile". One possible avenue for further research may be to perform an in-depth analysis of the influence of docker on reducing installation and build errors. Moreover, frameworks like Keras that act as interface of other DL frameworks may add additional difficulties to the installation. One possible

17. https://github.com/google-research/bert

solution to alleviate such problems is to provide a dedicated page like TensorFlow [64] and PyTorch [65] to collect common install and build errors and corresponding solutions.

(ii) Improve the compatibility of DL Framework APIs. The variety of data and data handling libraries such as Numpy, Pandas, and Gym, makes it often necessary for developers to convert data types between third-party libraries and DL frameworks. The relative rate of *Data Type* questions doesn't change over time, indicating that the data type compatibility between DL frameworks and third-party libraries is an ongoing and not-completely-addressed issue. At the same time, *Code Error* questions are growing the fastest and their burst of increases overlap with the major release time of TensorFlow. This suggests that the backward compatibility of DL frameworks may need to be improved to mitigate the influence of breaking changes. Research on tools that would automatically generate reports of how DL framework APIs are used in practice could be used to generate better test suites for the frameworks. Such tools could help DL API maintainers better understand how frequently users use their APIs, thus estimating the impact of introducing an API change.

(iii) Provide better support for using pre-trained models. The increasing rate of *Model Load* and *Model Conversion* questions that dominate *Transfer Learning* and *Deployment APP* applications respectively suggests that better support for loading and converting trained models to a form where they can be used for prediction would be beneficial. Model weights tend to be saved as key-value pairs where the keys are layer names and the values are layer weights. However, existing support on loading models appears to be rudimentary. For example, to load saved model weights in PyTorch, developers need to create an instance of the same model first, then load pre-trained weights using `load_state_dict` method, which is inconvenient and unnecessarily limits the flexibility of loading weights. To make matters even more complicated, the model formats supported by different frameworks and deployment platforms are not easily convertible as demonstrated by the rapidly increasing rate of *Model Conversion* questions. Developers find it hard to get answers to their questions as well, with only 25.7% *Model Conversion* questions having an accepted answer.

(iv) Provide application-type specific tools. Our results show that different topics dominate different application types with sometimes not immediately obvious associations. Application-type specific tools might be able to better satisfy developers' unique needs in some of these applications. For example, based on our findings, integrating image preprocessing packages that automatically align images and pixel labels for image segmentation applications might be beneficial.

(v) Design shape correction tools. Tensor Shape topic exhibits an increasing trend and has the highest *% acpt*. Such questions are typically raised by developers who do not completely understand the meaning of each dimension of the neural network layer's input and output. For example, developers are confused with the input shape of `torch.nn.Conv1d` when applying it on text input (e.g., Question 62372938). Since some of the dimension errors occur at the time of output, a massive amount of computational time may be spent before the error manifests itself. Although existing DL frameworks could print model architecture with each layer's output shape such as `print(model)` in PyTorch and `model.summary()` in Keras, they don't check whether the input shape satisfies the layer's requirement. Therefore, on the one hand, frameworks could provide meaningful information about the expected dimension and the mismatch in the error backtrace. On the other hand, validation tools might be designed to examine whether the model on developers' data induces shape errors and provide suggestions to correct the errors by analyzing the data flow in the model. Though several work [66], [67] has designed tools to detect shape errors for TensorFlow, similar tools for Keras and PyTorch are lacking and could be designed.

(vi) Improve documentation. As shown in our results, many developers have difficulty understanding and using DL framework APIs. For instance, *API Usage*, *API Misuse*, and *Object Detection API* topics account for 9.8% questions in total, and *API Misuse* and *Object Detection API* topics both show an increasing trend. Hence, the DL framework documentation should be improved. On the one hand, comparisons between similar APIs and best practices of using an API could be provided in the documentation to guide developers to efficiently use suitable APIs. On the other hand, as discussed in Sections 3.2 and 5.2, developers sometimes fail to learn from official tutorials (e.g., Question 59290830, 64986037), suggesting that the usefulness [68] of relevant documentation should be improved. For example, rather than using ready-to-use datasets, use raw data to demonstrate the usage of data-related APIs. In addition, as revealed in Section 4.2, TensorFlow Object Detection API organizes documentation as multiple markdown files, which causes some findability issues (e.g., Question 49148962). Therefore, the usability [68] of TensorFlow Object Detection API documentation could be improved.

SE4DL Education. (i) Design teaching materials in a more targeted way. The relationship between question topics and SE4DL stages and application types may provide a checklist for SE4DL educators to help them design more targeted teaching materials and tailor the curriculum towards the specific application type if the course concerns it. They may consider ensuring that topics found in RQ1 are in their teaching materials. RQ2 reveals that the questions for the 11 topics that are becoming more frequent had a lower percentage of having an accepted answer. First, the difficulty of getting an answer may be due to the difficulty of providing full relevant information [47]: a task difficult for newcomers to SO [69]. Therefore, educators may consider providing targeted training materials on how to ask questions on SO so that they are more likely to receive an accepted answer. Sometimes it may be difficult to provide relevant information even for experienced SO users for problems such as installation or mismatch of model formats. Possibly a tool could be written that automatically collects the necessary information so that it can be submitted with the question. Finally, for some of the topics that developers may not receive sufficient training, more teaching efforts may remedy that. Answers to RQ3 may be used to target teaching materials for specific application types and focus on primary pitfalls developers experience there. For example, a SE4DL educator may emphasize how to preprocess image data when teaching the medical diagnosis domain.

7 Limitations

Internal Validity concerns the soundness and accuracy of the methods used to perform our study. Specifically, the manual procedure used to label question topics and README themes may be subjective. To minimize subjective effects, two authors labeled separately and resolved inconsistencies through discussions. Moreover, a third person has inspected the named question topics and README themes. The Kappa values of labeling README sections in Section 2.2.2 and SE4DL stages in Section 3.1 were both above 0.80, which is considered almost perfect agreement [28], suggesting high reliability of the procedure. The method used to identify DL application types relied on considering the text of the first two sections of README files. README files have been used to classify repositories previously [26], [27]. We chose the first two sections to locate relevant information from README files based on the findings of our preliminary study. We discovered that in over 80% of cases of a random sample, functionality-relevant information was within the first two sections. For comparison, we also ran LDA on the entire text of READMEs, but the results were far worse, only having a coherence score of 0.47. The third limitation is the accuracy of the estimated question ratios for SE4DL stages. We estimate the ratios as described in 3.1. To assess the accuracy of the estimates, we also manually labeled 383 randomly sampled questions. Our finding indicates that errors in the estimation are within 1%. The fourth limitation relates to the way how LDA parameters were selected. To address it, we did parameter tuning using Mallet's hyperparameter optimization for $\vec{\alpha}$ and $\vec{\beta}$ and also used an approach described in [26], [33], [36], GA, to tune K and I. As is widely done in recent research [6], [18], [19], we used coherence score to evaluate how LDA fit. We also evaluate the LDA stability with widely used metric — raw score $\mathcal{R}_n$. Reusing the LDA model fitted on READMEs to make inferences on a different corpus (SO questions) may introduce vocabulary incompatibility issues. To minimize the impact of these potential issues, we use `–use-pipe-from` option suggested in Mallet documentation [70] to align tokens in SO questions with README corpus's vocabulary and validate the inference results as described in Section 5.1.

External Validity concerns the threats to generalize our findings. Similar to previous studies [15], [17], [18], [19], [31], [32], [33], [34], we use SO questions to identify practical problems. As a result, we may ignore problems reported in other platforms besides SO such as GitHub issues. As discussed in a prior study [6], *"GitHub provides more developer perspectives, while Stack Overflow provides more of a user's perspective"*. In this study, we aim to investigate problems faced by developers when developing DL applications (i.e., user's perspective). Therefore, we study SO posts instead of GitHub issues. Considering that developers who use SO appear to vary in experience and background, and that a search engine query often links to SO [71], we believe that SO questions should approximate developers' practical problems regarding which they are willing to attempt to crowdsource an answer. We identify DL-related SO questions based on tags that are similar to tags used in previous work [10]. We use tags representing the three most popular DL frameworks (in terms of GitHub stars). We can not, therefore, extrapolate our results to other frameworks. However, we carefully make a comparison between the three frameworks under study and four other frameworks (i.e., Theano, Caffe, MxNet, and CNTK) which once attracted attention from industry and academia in Appendix. Compared with the other four frameworks, the three frameworks selected for this study are actively developed, have increasing downstream repositories and SO questions, and cover different DL framework implementations. We also run LDA on all SO questions related to the seven frameworks, which identifies the same 27 question topics. Therefore, we believe the three selected frameworks are representative and influential. Some of the questions that discuss the three frameworks may not have the tags we used for filtering. To capture these untagged questions, future work may consider applying content-based filtering techniques as in [18]. We also identify DL-related repositories based on the three popular DL frameworks. Some of the DL-related repositories may use other frameworks. But the three frameworks we choose are used in almost one million repositories, so we believe our dataset represents a significant part of all open source DL development.

Construct Validity is the degree to which our metrics of the relative number of questions and proportion of answered questions measure the relative number of problems and difficulty of getting answers. For example, even a stable relative rate for a topic represents increasing number of questions. The question intensity or relative frequency may also reflect the changing or growing of the population of developers. From our perspective, we wanted to demonstrate the relative importance of a problem, so the exact reason why certain stages, topics, or themes have more questions is not essential. What matters is that if addressed through improvement in the frameworks, better training, or improved tools, it will bring benefits. The reasons why some questions do not get answers may vary as well. For example, a question may be harder, or be badly formulated, or lack context, or be hard to specify the full context (as in installation problems), or there simply may be no experts to answer it. As with the number of questions, these reasons may be important and may require different interventions. For example, to train developers how to ask questions, how to determine and provide relevant context, how to incentivize experts who answer questions better, etc. From the perspective of our research, however, unanswered questions indicate unresolved problems and the lower the percentage of accepted answers, the bigger that the problem is.

8 Related Work

SE4DL has unique problems that differ from those encountered in other domains' software development and has attracted several empirical studies to characterize SE4DL needs. Specifically, many studies focus on SE4DL challenges and faults, but they do not investigate how they vary among SE4DL stages. The study of Zhang *et al.* [7] investigated DL software bugs. The authors manually analyzed 175 TensorFlow program bugs collected from SO and GitHub and abstracted four symptoms such as *Error* and *Low Effectiveness* and seven root causes such as *Incorrect Model Parameter or Structure* and *Unaligned Tensor*. Islam *et al.* [16] and Humbatova *et al.* [10] studied more DL frameworks for a more

TABLE 4
Summary of Related Work on SE4DL Stages

Paper	Artifacts	Findings about SE4DL Stages	Findings about Trend
Alshangiti *et al.* [15]	SO questions	The *data pre-processing and manipulation* stage and the *model deployment and environment setup* stage are the most challenging.	N.A.
Islam *et al.* [16]	SO questions and GitHub commits	The stages with the most bugs are data preparation, model training, and model setup.	Structural logic bugs are increasing and data bugs are decreasing.
Han *et al.* [6]	SO questions and GitHub issues	*Model Training* and *Preliminary Preparation* are the most frequently discussed stages and *Model Tuning* stage has not been discussed	The impact trend of stages on TensorFlow and Theano are relatively flat and on PyTorch fluctuates intensely; The top 3 LDA topics with largest increases or decreases are always different on the three studied DL frameworks.

comprehensive understanding of DL software bug symptoms and root causes. Islam *et al.* also analyzed fix patterns and challenges of these bugs in their follow-up work [9]. Zhang *et al.* [11] studied the program failures of DL jobs running on DL platforms and found that nearly half of the failures occur in the interaction with the platform rather than in the execution of code logic. Zhang *et al.* [8] manually inspected 715 DL-related SO questions and identified seven kinds of questions such as *program crash*, *model migration and deployment*, and *implementation*. Other empirical studies of SE4DL focused on the model deployment task at *Model Prediction* stage. Guo *et al.* [12] investigated the performance gap when deploying trained models to mobile devices and web browsers and found that model deployment suffered from compatibility and reliability issues. Chen *et al.* [13] manually analyzed 769 SO posts and built taxonomies consisting of 72 challenges when deploying DL software to server/cloud, mobile, and browser. They further analyzed the symptoms and fix-strategies of deployment faults of mobile DL apps [14].

Work in [6], [15], [16] investigated SE4DL stages. We summarize the findings of these three papers in Table 4. The *Artifacts* column shows the data source used. The third and fourth columns show the findings concerning SE4DL stages and problem trends. *N.A.* means no findings. Alshangiti *et al.* [15] analyzed 684 machine learning (ML) related SO questions and revealed the stages with the highest percentage of questions without an accepted answer. Islam *et al.* [16] manually labeled the stages of 970 bugs collected from SO questions and GitHub commits and revealed stages with the most bugs and the annual trend of bugs. Han *et al.* [6] applied LDA on large-scale SO questions and GitHub issues of three DL frameworks, namely, Tensorflow, PyTorch, and Theano; and derived a total of 75 topics in the six corpora. The authors then aggregated LDA topics into 20 topic categories across all stages and reported the question topic distribution over stages. They also reported the impact (the averaged probability of a topic in the topic probability distribution of all questions) trend of stages in the six corpora and the impact trend of particular topics and topic categories.

In comparison to these three studies, our study has made several advances. In particular, unlike our study, the work described in [15], [16] was based on a much smaller dataset and didn't associate problems with DL stages. The work described in [6] investigated stages under an improper assumption that the question topics exclusively belong to a single stage. We found most topics to occur in most stages. Furthermore, the authors only presented the trends for only a few topics. Finally, none of the three studies investigate how problems faced by developers vary over DL application types.

In this study, we perform topic modeling with LDA on large-scale SO questions and README files, in order to reveal the varied and interconnected landscape of DL development stages, developer needs, and DL application types. In particular, we studied how problems faced by DL developers are distributed over SE4DL stages and vary over time and application types.

9 Conclusion

Software development of DL applications presents unique problems and is rapidly spreading and evolving. This paper aims to better understand SE4DL needs by identifying how the problems faced by DL developers vary over DL development stages, time, and application types. Our approach is to leverage approximately all DL-related SO questions and public DL software projects. In total, we analyzed 92,830 SO questions and 227,756 READMEs of repositories related to DL. We also describe the process we used to obtain the distribution of SO question topics. It not only helps reproduce our results, but also supports investigation in the software development of other domains. This overview approach analyzed nearly all actual projects and questions, hence can help better prioritize relevant training, tools creations, and further research efforts in the domain of concern. We find the distribution of topics uneven over DL stages, time, and application types. The most frequent question topics for an application type or a development stage are often not intuitive beforehand. We believe that our detailed description of the changing landscape of SE4DL needs over DL stages, time, and application types would help inform new ways to improve SE4DL.

Acknowledgments

We would like to thank Zhehao Zhao for his valuable feedback. We also sincerely thank the reviewers for their great suggestions.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[2] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[3] "Tensorflow," Accessed: Aug. 30, 2021. [Online]. Available: https://www.tensorflow.org/

[4] "Keras: The Python deep learning API," Accessed: Jul. 27, 2021. [Online]. Available: https://keras.io/

[5] "PyTorch," Accessed: Aug. 30, 2021. [Online]. Available: https://pytorch.org/

[6] J. Han, E. Shihab, Z. Wan, S. Deng, and X. Xia, "What do programmers discuss about deep learning frameworks," *Empirical Softw. Eng.*, vol. 25, no. 4, pp. 2694–2747, 2020.

[7] Y. Zhang, Y. Chen, S. Cheung, Y. Xiong, and L. Zhang, "An empirical study on tensorflow program bugs," in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2018, pp. 129–140.

[8] T. Zhang, C. Gao, L. Ma, M. R. Lyu, and M. Kim, "An empirical study of common challenges in developing deep learning applications," in *Proc. 30th IEEE Int. Symp. Softw. Rel. Eng.*, 2019, pp. 104–115.

[9] M. J. Islam, R. Pan, G. Nguyen, and H. Rajan, "Repairing deep neural networks: Fix patterns and challenges," in *Proc. 42nd Int. Conf. Softw. Eng.*, 2020, pp. 1135–1146.

[10] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proc. 42nd Int. Conf. Softw. Eng.*, 2020, pp. 1110–1121.

[11] R. Zhang, W. Xiao, H. Zhang, Y. Liu, H. Lin, and M. Yang, "An empirical study on program failures of deep learning jobs," in *Proc. 42nd Int. Conf. Softw. Eng.*, 2020, pp. 1159–1170.

[12] Q. Guo *et al.*, "An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2019, pp. 810–822.

[13] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, "A comprehensive study on challenges in deploying deep learning based software," in *Proc. ESEC/FSE 28th ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 750–762.

[14] Z. Chen *et al.*, "An empirical study on deployment faults of deep learning based mobile applications," in *Proc. 43rd IEEE/ACM Int. Conf. Softw. Eng.*, 2021, pp. 674–685.

[15] M. Alshangiti, H. Sapkota, P. K. Murukannaiah, X. Liu, and Q. Yu, "Why is developing machine learning applications challenging? A study on stack overflow posts," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2019, pp. 1–11.

[16] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proc. ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 510–520.

[17] M. Bagherzadeh and R. Khatchadourian, "Going big: A large-scale study on what big data developers ask," in *Proc. ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 432–442.

[18] M. U. Haque, L. H. Iwaya, and M. A. Babar, "Challenges in docker development: A large-scale study using stack overflow," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2020, pp. 7:1–7:11.

[19] A. Abdellatif, D. Costa, K. Badran, R. Abdalkareem, and E. Shihab, "Challenges in chatbot development: A study of stack overflow posts," in *Proc. 17th Int. Conf. Mining Softw. Repositories*, 2020, pp. 174–185.

[20] "About keras: Installation & compatibility," Accessed: Aug. 01, 2021. [Online]. Available: https://keras.io/about/#installation-amp-compatibility

[21] G. Gousios and D. Spinellis, "GHTorrent: Github's data from a firehose," in *Proc. 9th IEEE Work. Conf. Mining Softw. Repositories*, 2012, pp. 12–21.

[22] L. Hong and B. D. Davison, "Empirical study of topic modeling in twitter," in *Proc. 1st Workshop Social Media Analytics*, 2010, pp. 80–88.

[23] "Git objects," Accessed: Jul. 30, 2021. [Online]. Available: https://git-scm.com/book/en/v2/Git-Internals-Git-Objects

[24] "Mallet/en.txt," Accessed: Aug. 01, 2021. [Online]. Available: https://github.com/mimno/Mallet/blob/master/stoplists/en.txt

[25] C. Tan, Y. Wang, and C. Lee, "The use of bigrams to enhance text categorization," *Informat. Process. Manage.*, vol. 38, no. 4, pp. 529–546, 2002.

[26] A. Sharma, F. Thung, P. S. Kochhar, A. Sulistya, and D. Lo, "Cataloging github repositories," in *Proc. 21st Int. Conf. Eval. Assessment Softw. Eng.*, 2017, pp. 314–319.

[27] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the content of Github README files," *Empirical Softw. Eng.*, vol. 24, no. 3, pp. 1296–1327, 2019.

[28] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychol. Bull.*, vol. 76, no. 5, 1971, Art. no. 378.

[29] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. Germán, and D. E. Damian, "The promises and perils of mining Github," in *Proc. 11th Work. Conf. Mining Softw. Repositories, Proc.*, 2014, pp. 92–101.

[30] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," in *Proc. Adv. Neural Informat. Process. Syst.*, 2001, pp. 601–608. [Online]. Available: https://proceedings.neurips.cc/paper/2001/hash/296472c9542ad4d4788d543508116cbc-Abstract.html

[31] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 112–121.

[32] C. Rosen and E. Shihab, "What are mobile developers asking about? A large scale study using stack overflow," *Empirical Softw. Eng.*, vol. 21, no. 3, pp. 1192–1223, 2016.

[33] X. Yang, D. Lo, X. Xia, Z. Wan, and J. Sun, "What security questions do developers ask? A large-scale study of stack overflow posts," *J. Comput. Sci. Technol.*, vol. 31, no. 5, pp. 910–924, 2016.

[34] S. Ahmed and M. Bagherzadeh, "What do concurrency developers ask about?: A large-scale study using stack overflow," in *Proc. 12th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2018, pp. 30:1–30:10.

[35] C. Treude and M. Wagner, "Predicting good configurations for github and stack overflow topic models," in *Proc. 16th Int. Conf. Mining Softw. Repositories*, 2019, pp. 84–95.

[36] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms," in *Proc. 35th Int. Conf. Softw. Eng.*, 2013, pp. 522–531.

[37] A. Agrawal, W. Fu, and T. Menzies, "What is wrong with topic modeling? and how to fix it using search-based software engineering," *Informat. Softw. Technol.*, vol. 98, pp. 74–88, 2018.

[38] A. Panichella, "A systematic comparison of search-based approaches for LDA hyperparameter tuning," *Informat. Softw. Technol.*, vol. 130, 2021, Art. no. 106411.

[39] H. M. Wallach, D. M. Mimno, and A. McCallum, "Rethinking LDA: Why priors matter," in *Proc. Adv. Neural Informat. Process. Syst. 22, 23rd Annu. Conf. Neural Informat. Process. Syst.*, 2009, pp. 1973–1981. [Online]. Available: https://proceedings.neurips.cc/paper/2009/hash/0d0871f0806eae32d30983b62252da50-Abstract.html

[40] M. D. Hoffman, D. M. Blei, and F. R. Bach, "Online learning for latent dirichlet allocation," in *Proc. Adv. Neural Informat. Process. Syst. 23, 24th Annu. Conf. Neural Informat. Process. Syst.*, 2010, pp. 856–864. [Online]. Available: https://proceedings.neurips.cc/paper/2010/hash/71f6278d140af599e06ad9bf1ba03cb0-Abstract.html

[41] D. M. Mimno, H. M. Wallach, E. M. Talley, M. Leenders, and A. McCallum , "Optimizing semantic coherence in topic models," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2011, pp. 262–272. [Online]. Available: https://aclanthology.org/D11–1024/

[42] M. Röder, A. Both, and A. Hinneburg, "Exploring the space of topic coherence measures," in *Proc. 8th Eighth ACM Int. Conf. Web Search Data Mining*, 2015, pp. 399–408.

[43] Z. Wan, X. Xia, and A. E. Hassan, "What do programmers discuss 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 stack exchange communities," *IEEE Trans. Softw. Eng.*, vol. 47, no. 7, pp. 1331–1349, Jul. 2021.

[44] J. H. Gennari, P. Langley, and D. H. Fisher, "Models of incremental concept formation," *Artif. Intell.*, vol. 40, no. 1–3, pp. 11–61, 1989.

[45] S. Holm, "A simple sequentially rejective multiple test procedure," *Scand. J. Statist.*, vol. 6, no. 2, pp. 65–70, 1979.

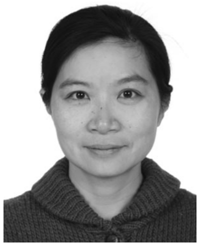
[46] Wikipedia contributors, "Supermajority—Wikipedia, the free encyclopedia," 2021. Accessed: Jan. 10, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Supermajority&oldid=1061442111

[47] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A. Schneider, "Answering questions about unanswered questions of stack overflow," in *Proc. 10th Work. Conf. Mining Softw. Repositories*, 2013, pp. 97–100.
[48] M. Hussain and I. Mahmud, "pymannkendall: A python package for non parametric mann kendall family of trend tests," *J. Open Source Softw.*, vol. 4, no. 39, 2019, Art. no. 1556.
[49] C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk, "Machine learning-based detection of open source license exceptions," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng.*, 2017, pp. 118–129.
[50] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, and M. Lanza, "Pattern-based mining of opinions in Q&A websites," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.*, 2019, pp. 548–559.
[51] A. Danilova, A. Naiakshina, S. Horstmann, and M. Smith, "Do you really code? Designing and evaluating screening questions for online surveys with programmers," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.*, 2021, pp. 537–548.
[52] J. Jiang, Q. Wu, J. Cao, X. Xia, and L. Zhang, "Recommending tags for pull requests in Github," *Informat. Softw. Technol.*, vol. 129, 2021, Art no. 106394.
[53] "Google developers blog: Announcing tensorflow 1.0," Accessed: Aug. 03, 2021. [Online]. Available: https://developers.googleblog.com/2017/02/announcing-tensorflow-10.html
[54] "Tensorflow 2.0 is now available!," Accessed: Jul. 27, 2021. [Online]. Available: https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html
[55] Z. Zhang, H. Zhu, M. Wen, Y. Tao, Y. Liu, and Y. Xiong, "How do Python framework APIs evolve? an exploratory study," in *Proc. 27th IEEE Int. Conf. Softw. Anal. Evol. Reengineering*, 2020, pp. 81–92.
[56] "Image segmentation - Wikipedia," Accessed: Aug. 30, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Image_segmentation
[57] "Transfer learning - Wikipedia," Accessed: Aug. 30, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Transfer_learning
[58] "A gentle introduction to transfer learning for deep learning," 2021. [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/
[59] "Word embedding - wikipedia," Accessed: Sep. 29, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Word_embedding
[60] "Object detection—Papers with code," Accessed: Aug. 30, 2021. https://paperswithcode.com/task/object-detection
[61] "Artificial intelligence & autopilot—Tesla," Accessed: Aug. 30, 2021. https://www.tesla.com/AI
[62] The TensorFlow teams, "Reduce tensorflow lite binary size," 2021, Accessed: Jan. 18, 2022. [Online]. Available: https://www.tensorflow.org/lite/guide/reduce_binary_size
[63] The TensorFlow teams, "Build tensorflow lite for android," 2021, Accessed: Jan. 18, 2022. [Online]. Available: https://www.tensorflow.org/lite/guide/build_android#build_tensorflow_lite_locally
[64] "Build and install error messages," Accessed: Aug. 03, 2021. https://www.tensorflow.org/install/errors
[65] The PyTorch teams, "Windows FAQ — PyTorch master documentation," 2021, Accessed: Jan. 20, 2022. [Online]. Available: https://pytorch.org/docs/master/notes/windows.html#installation
[66] S. Lagouvardos, J. Dolby, N. Grech, A. Antoniadis, and Y. Smaragdakis, "Static analysis of shape in tensorflow programs," in *Proc. 34th Eur. Conf. Object-Oriented Program.*, 2020, pp. 15:1–15:29. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/13172
[67] S. Verma and Z. Su, "Shapeflow: Dynamic shape interpreter for tensorflow," 2020, *arXiv:abs/2011.13452*.
[68] E. Aghajani *et al.*, "Software documentation issues unveiled," in *Proc. 41st Int. Conf. Softw. Eng.*, 2019, pp. 1199–1210.
[69] D. Ford, K. Lustig, J. Banks, and C. Parnin, ""We don't do that here": How collaborative editing with mentors improves engagement in social Q&A communities," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2018, Art. no. 608.
[70] "Topic modeling," Accessed: Aug. 30, 2021. [Online]. Available: http://mallet.cs.umass.edu/topics.php
[71] D. Yang, A. Hussain, and C. V. Lopes, "From query to usable code: An analysis of stack overflow code snippets," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, 2016, pp. 391–402.

Kai Gao received the BS degree from Peking University, in 2019. He is currently working toward the PhD degree with the school of software and microelectronics, Peking University. His research interests include mining software repositories, open source software ecosystems and supply chains. For more information, please visit gaokai19@pku.edu.cn.

Zhixing Wang received the BS degree in the computer science from the University of Edinburgh. He is currently working toward the master's degree with the School of Information Science and Technology, University of Tokyo. His research interests include computational models for cognition, sense of agency and body ownership. For more information, please visit zhixing0@protonmail.com.

Audris Mockus (Member, IEEE) received the BS degree in applied mathematics from the Moscow Institute of Physics and Technology in 1988, the MS degree in 1991, and the PhD degree in statistics from Carnegie Mellon University in 1994. He is a harlan mills chair professor of digital archeology with the Department of Electrical Engineering and Computer Science, University of Tennessee. He also continues to work part-time with Avaya Labs Research. Previously, he was with the Software Production Research Department, Bell Labs. He studies software developers' culture and behavior through the recovery, documentation, and analysis of digital remains. These digital traces reflect projections of collective and individual activity. He reconstructs the reality from these projections by designing data mining methods to summarize and augment these digital traces, interactive visualization techniques to inspect, present, and control the behavior of teams and individuals, and statistical models and optimization techniques to understand the nature of individual and collective behavior. He is a ACM. For more information, please visit audris@utk.edu.



Minghui Zhou (Member, IEEE) received the BS, MS, and PhD degrees in computer science from the National University of Defense Technology, in 1995, 1999, and 2002, respectively. She is a professor with the School of Computer Science with Peking University. She is interested in software digital sociology, i.e., understanding the relationships among people, project culture, and software product through mining the repositories of software projects. She is a member of the ACM. For more information, please visit zhmh@pku.edu.cn.