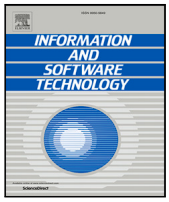
Information and Software Technology 166 (2024) 107367

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Understanding the implementation issues when using deep learning frameworks

Chao Liu, Runfeng Cai, Yiqun Zhou, Xin Chen *, Haibo Hu, Meng Yan

School of Big Data and Software Engineering, Chongqing University, 401331, Chongqing, China

ARTICLE INFO

Keywords:
Deep learning
Implementation issues
Stack overflow
Empirical study

ABSTRACT

Context: Deep Learning (DL) frameworks like TensorFlow can help developers implement DL applications (e.g., computer vision) faster and easier. When using DL frameworks, developers encountered a large number of questions and posted them on Stack Overflow (SO).

Objective: The goal of this paper is to conduct a comprehensive empirical study on the SO questions, summarize the implementation issues, and suggest future opportunities.

Methods: This paper focuses on three DL frameworks (i.e., TensorFlow, PyTorch, and Theano), groups 2,401 relevant SO questions into various implementation issues, and constructs a taxonomy. We also analyze the popularity and difficulty of these issues under the taxonomy.

Results: For the identified various implementation issues, we constructed a taxonomy consisting of seven major categories with 63 subcategories. Our analysis reveals that 91.7% of questions are related to the implementation categories of data processing, model setting, model training, and model prediction. Developers frequently address the remaining three categories (i.e., Model evaluation, runtime environment, and visualization), where runtime environment is the most difficult category. Based on empirical findings, we provide some suggestions for future research.

Conclusion: In this paper, we summarized the issues of DL implementation and proposed corresponding opportunities for future study. We expect this paper to help developers and researchers understand these issues and design better tools to improve the productivity of DL implementation.

1. Introduction

Deep Learning (DL) is represented in the form of a neural network and automatically learns the decision logic from training data [1–3]. The DL technique has been widely applied in several application domains [4–6] and significantly improved the performance of those application domains in comparison with the state of the art [7–11], including computer vision [12], natural language processing [13], machine translation [14], speech and audio processing [15], disease diagnosis [16], auto-driving [17]. Due to the advantages of the DL technique, DL has attracted considerable attention in both academia and industry for more than 15 years [1,4,7,18].

To help DL developers use DL techniques easier and better, many DL frameworks were proposed. Following Han et al. [19], we investigated three representative frameworks widely used for DL implementation and empirical study: Theano, one of the oldest frameworks [20]; TensorFlow, the most widely-deployed DL framework developed by Google [21]; PyTorch, one of the most popular frameworks presented by Meta in the recent years [22]. By using these DL frameworks, developers can devote themselves to their application development [23]. In this way, they can be free from the complex and tedious work of low-level developments [23], such as task scheduling (e.g., control flow and data transfer between tasks) and common algorithm implementations (e.g., batch normalization and average pooling).

Using these DL frameworks, developers can substantially accelerate their software development [19]. However, different from traditional software development, the bottlenecks of DL development for developers are building model structure, data processing, model optimization techniques, etc. [7]. Due to these bottlenecks, many developers are facing the challenges in improving the efficiency of DL implementation. Therefore, developers submitted a great number of specific implementation questions in Stack Overflow (SO), one of the most popular question-and-answer (Q&A) websites for developers, and expected solutions that are answered by other developers. Analyzing these questions can help DL developers point out their interests and requirements.

* Corresponding author.
E-mail addresses: liu.chao@cqu.edu.cn (C. Liu), crf@cqu.edu.cn (R. Cai), zhouyiqun@cqu.edu.cn (Y. Zhou), xinchen@cqu.edu.cn (X. Chen), haibo.hu@cqu.edu.cn (H. Hu), mengy@cqu.edu.cn (M. Yan).

https://doi.org/10.1016/j.infsof.2023.107367
Received 16 January 2023; Received in revised form 16 October 2023; Accepted 16 November 2023
Available online 20 November 2023

To characterize the challenges that developers encountered in developing DL applications, researchers conducted many empirical studies on the DL frameworks. Previously, researchers focused on the high-level topics of interest discussed in SO. For example, Zhang et al. [24] identified seven kinds of frequently asked DL questions in SO, including implementation, program crash, training anomaly, model deployment, comprehension, installation, and performance. Han et al. [25] leveraged the Latent Dirichlet Allocation (LDA) technique to cluster DL questions. In recent years, many in-depth empirical studies have been conducted on one of these high-level topics, such as programming bugs [26,27], technical debts [23,28], deployment [1], library dependency [19], computation performance [1,8], etc. However, the in-depth empirical study on DL implementation has not been investigated.

To fill the knowledge gap, this paper presents the first large-scale empirical study on analyzing DL implementation questions. It aims to summarize these specific questions with some implementation issues, and build a taxonomy based on the summarized issues. In this way, researchers and developers can have a comprehensive and in-depth understanding of the common implementation issues using DL frameworks. A better understanding can not only help developers accomplish their implementation tasks quickly, but also guide researchers in building better tools to improve developers' development productivity.

To reach this goal, we collected and manually inspected 2401 representative implementation questions from SO related to three widely adopted DL frameworks, including TensorFlow, PyTorch, and Theano. We chose to collect data from SO because SO is one of the most widely used data source for an empirical study on DL [19,27,29]. In this study, we investigated the following three aspects:

- **Taxonomy of Implementation Issues.** To identify specific implementation issues, we qualitatively extracted the issues behind each collected question. Finally, we build a comprehensive taxonomy consisting of seven categories (i.e., data processing, model setting, model training, model prediction, model evaluation, runtime environment, and visualization) with 63 subcategories. The resulting taxonomy indicates that developers suffered from a wide spectrum of implementation issues. Besides, we found that the first four categories accounted for 91.7% of all questions.
- **Popularity Analysis.** A popular implementation issue indicates that many developers suffered from this issue in practices [30,31]. To conduct the popularity analysis, we measured the popularity of a question using the viewcount recorded by SO and the score voted by different SO developers. A higher viewcount implies that many developers are interested in the implementation issue, while a larger score suggests the usefulness of a question with answers to address developers' issues. Generally, we found that the viewcount and score are highly correlated; model evaluation, runtime environment, and visualization are the most popular categories.
- **Difficulty Analysis.** A difficult implementation issue implies that this issue cannot be addressed in a short time and it will strongly hinder developers' implementation productivity [32,33]. To perform the difficulty analysis, we measured the difficulty of a question by the response time of an accepted answer following Chen et al. [31]. We found that the runtime environment is the most difficult implementation category, due to the complexity of manipulating computation devices, such as GPU.

The major contributions of this study are:

- grouping 2401 relevant questions into various implementation issues and building a comprehensive taxonomy on these issues to help developers and researchers understand what implementation issues occurred in practices.
- analyzing the popularity and difficulty of implementation issues under the taxonomy to help developers and researchers understand which issues are frequently occurred and which are difficult to address.
- providing some implications and future research directions for developers and researchers to understand how these implementation issues could be addressed to improve the DL implementation productivity.
- sharing our replication package with dataset and scripts in public [34] as an additional contribution to the research community for other researchers to replicate and build upon.

The remainder of this paper is organized as follows. Section 2 presents the methodology for this empirical study. Sections Section 3 provide the results for our empirical study, respectively. Subsequently, Section 4 discusses the implications and the threats to the validity of our study. Section 5 introduces the related work. Finally, Section 6 concludes this paper and gives directions for future work.

2. Methodology

Section 2.1 presents the study subject of this paper. Section 2.2 illustrates the overall workflow for data collection and analysis in this study. To characterize the implementation questions discussed in SO, we first described how we collected data in Section 2.3. Subsequently, Section 2.4 presented the procedures to construct the taxonomy. Besides, Section 2.5 showed the measurements for post popularity and difficulty.

2.1. Study subject

Developers frequently do not know how to implement a requirement for a DL application when coding with a DL framework, such as TensorFlow [19,24]. Following the definition described by Zhang et al. [24], this paper calls the summarization of this kind of question implementation issues. Different from the work by Zhang et al. [24], we aim to investigate what are the "implementation issues" instead of simply calling these questions with a category name. To reach the goal, we take the related questions asked in SO as our basic study subject and extract implementation issues by summarizing the semantics of different questions. Moreover, we regarded the extracted implementation issues as a secondary study subject and constructed a taxonomy based on it to further understand these issues.

2.2. Overall workflow

As shown in Fig. 1, this study first performs data collection in three steps, including downloading raw data from SO, extracting candidates related to DL implementation, and manually refining the candidates. Based on the collected data, we constructed a two-level taxonomy in four steps: (1) grouping specific implementation questions from all candidate data to understand the semantics expressed in these questions; (2) assigning implementation phase (e.g., model training) to each question to investigate which phase do these questions belong to; (3) summarizing implementation objects under different phases to better understand each implementation phase; (4) optimizing the constructed two-level taxonomy by manual inspections. Details are described in the following subsections.

2.3. Data collection

In this study, we collected raw data from SO because it is one of the most popular community-driven Q&A websites for DL developers. In SO, developers frequently posted their questions and quickly received answers from others, where the developers range from novices to experts. In this way, SO recorded a set of non-trivial implementation issues. Generally, we extracted 3842 candidate posts and finally obtained 2401 refined posts. Details can be found as follows.

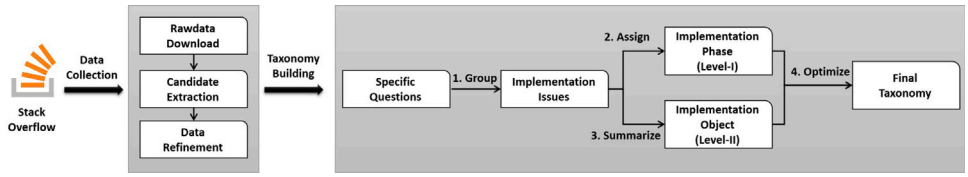


Fig. 1. Process of data collection and analysis.

2.3.1. Raw data download

We downloaded the entire post dataset [35] from the official Stack Exchange Data Dump on September 6th, 2021. Afterwards, we filled these data into MongoDB [36], an open-source non-rational database, to facilitate data searching and filtering. This dataset contained 22m+ questions and 33m+ answers in total.

The data structure can be found in file readme.txt [37] on the official website. In this data, the relevant features are: *(1) Id:* the unique number assigned to each post; *(2) Title:* the title of a post question; *(3) Tags:* the tags assigned by question owner; *(4) CreationDate:* the date-time of a question or an answer was created by its owner; *(5) ClosedDate:* a question would be deleted if the post is a duplication of existing posts or the question is meaningless, which is decided by developers' voting; *(6) Viewcount:* the total number of times that a post was viewed by developers; *(7) Score:* the total scores assigned by different developers (equaling to upvotes minus downvotes), where a higher score indicates that the post is more interesting and useful for developers.

2.3.2. Candidates extraction

From the entire candidates, we included 90,697 questions with tags marked as 'Tensorflow', 'PyTorch', or 'Theano' from the raw data at first following Han et al. [19], as all three DL libraries are typical and widely studied [25–27]. Note that it is possible that a post is not marked as these tags but is related to these three DL frameworks. However, the sampled posts cover a large number of representative posts, which can help us to understand the implementation issues. According to Treude et al. [38], the SO questions can be classified into ten categories, such as how-to, error, decision help, etc., where how-to is the most frequently asked question that requests implementation instructions. As we focused on the implementation issues, we extracted 14,203 "how-to" questions from candidates and excluded the other irrelevant categories. The other categories are filtered out because they belong to model configuration (i.e., environment), programming knowledge (decision help, conceptual, non-functional, and novice), implementation bugs (i.e., discrepancy and error), and others (i.e., review and noise), according to the description of Treude et al. [38]. Using keywords "how-to" is a simple and viable way to sample candidate questions relevant to implementation issues from the large-scale SO dataset. However, we found that other patterns (e.g., environment and decision help) may be described in the "how-to" pattern and belong to a programming issue. Meanwhile, some "how-to" questions may not be relevant to programming issues. To address this issue, we further refined the collected questions as described in Section 2.3.3.

To support the analysis of post difficulty, we obtained 4327 posts with accepted answers. In other words, we removed the posts without accepted answers as the difficulty (defined in Section 2.5.2) of a post is measured by the response time of the accepted answers following Chen et al. [39], where the answers contain implementation code. In addition, we discarded 485 closed posts depending on the appearance of the 'ClosedDate' in data, because they were duplicates of other questions or their topics were irrelevant to SO, which were determined by developers' votes [40]. In this way, we sampled 3842 candidate posts from SO.

Table 1
Number (percentage) of posts for three DL frameworks created in different years.

Year	TensorFlow	PyTorch	Theano	Total
2014	0 (00.0%)	0 (00.0%)	5 (00.2%)	5 (00.2%)
2015	15 (00.6%)	0 (00.0%)	8 (00.3%)	23 (01.0%)
2016	160 (06.7%)	0 (00.0%)	28 (01.2%)	188 (07.8%)
2017	316 (13.2%)	20 (00.8%)	6 (00.2%)	342 (14.2%)
2018	324 (13.5%)	56 (02.3%)	0 (00.0%)	380 (15.8%)
2019	463 (19.3%)	101 (04.2%)	0 (00.0%)	564 (23.5%)
2020	447 (18.6%)	151 (06.3%)	0 (00.0%)	598 (24.9%)
2021	172 (07.2%)	129 (05.4%)	0 (00.0%)	301 (12.5%)
Total	**1897 (79.0%)**	**457 (19.0%)**	**47 (02.0%)**	**2401 (100.0%)**

2.3.3. Data refinement

Moreover, we observed that many candidates were still irrelevant to implementation issues. These irrelevant questions may belong to other high-level topics, such as program crash (e.g., how to fix an error) and deployment (e.g., how to deploy or configure a model). To refine the candidates, the second and third authors inspected each candidate independently and excluded the posts not belonging to the implementation but the other six categories as defined by Zhang et al. [24], including program crash, training anomaly, model deployment, comprehension, installation, and performance. The agreement of the authors' assessment was measured using pairwise inter-rater reliability with Cohen's Kappa statistic [41]. The agreement rate was "moderate" (0.53). Before the formal data refinement, we conducted a pilot study with 100 randomly selected samples and found the observations. The agreement rate is not very high due to the small size of the sampling. The disagreements on 174 posts were resolved after open discussions between these two authors. In any case, if they did not reach a consensus, the first author was consulted as a tie-breaker. Specifically, the first author gave an independent comment for a case. Based on this consultation result, the three authors discussed the inconsistencies again and made the decision together. We spent two weeks accomplishing the data refinement process. Finally, we obtained 2401 refined posts and excluded 1441 noisy candidates.

2.3.4. Data characteristics

Table 1 showed the number and percentage of posts for three DL frameworks created in different years. From the table, we can find that the collected 2401 posts contained 1897 TensorFlow posts, 457 PyTorch posts, and 47 Theano posts, respectively. Although the number of Theano posts is small, the results drawn from the whole 2401 posts can help researchers and developers understand the common implementation issues shared by three DL frameworks. Moreover, during the last eight years, most posts were submitted during the years 2018–2020, accounting for 64.2% of total posts. Furthermore, we can observe that all the Theano posts were submitted before 2018. For the TensorFlow posts, the total number surged from 2015 and reached a peak in 2019. As for PyTorch, related posts were created from 2017 and the total number kept increasing.

2.4. Procedures to build taxonomy

As developers wrote their implementation questions in different ways, we aim to extract the core semantics from different questions and build a comprehensive taxonomy by summarizing their semantics. After

analyzing the syntax of the questions, we observed that the semantics of a question (e.g., how to restore a model after training) are commonly represented by three parts: *(1) Verb Phrase,* describing the core object and related operation involved in an implementation issue (e.g., restore model); *(2) Phase,* expressing the phase of DL implementation (e.g., model training) in an explicitly or implicitly way; *(3) Noisy Words,* representing the implementation issue in various forms. As the accepted answers of the extracted posts provide the implementation code, we can confirm that the posts are implementation issues instead of design problems.

Based on the observation, we first grouped specific questions in different implementation issues with a representative verb phrase (e.g., load model). We classified the SO posts using linguistic patterns based on our prior work [42]. We found that the core semantics of the post question can be represented by its important keywords, especially for the verb phrase. This is because the implementation questions commonly ask how to implement a programming object. Similarly, Xie et al. [43] also indicated the importance of the verb phrase in capturing the functionality in descriptions. We then construct a taxonomy with two levels: *Phase (Level-I),* relating an issue to a DL implementation phase (e.g., model training); *Object (Level-II),* summarizing the operational target for a set of implementation issues (e.g., model or layer). In this study, we choose to classify the SO posts based on the "phase" because (1) the DL phases (e.g., model training and prediction) are commonly discussed in studies [44,45], which can help developer understand the related tasks; and (2) we observed developers also referred to the "phase" in the SO posts (e.g., how to dense a dataset during model training).

Our study can be regarded as a special case of the constant comparison method, which aims to organize excerpts of raw data into groups according to attributes and organize those groups in a structured way to formulate a new theory [46]. In our study, we used the representative verb phrases as the incidents at first. This is because the verb phrase can be used for sentence reduction and text summarization [47]. For the domain of software engineering, the verb phrase can be used to summarize developers' intent, such as issue report [48], user queries [43], and programming requirement [42]. Second, we integrated the incidents with a two-level taxonomy (i.e., Phase and Object). The Object level aims to summarize the programming components for the implementation issues while the Phase describes when the implementation issues happened.

From the taxonomy, we can understand the implementation issues discussed by developers, the common objects involved in different issues, and the important implementation phases related to these issues. We built the taxonomy in the following steps.

2.4.1. Step-1: Grouping questions into implementation issues

We manually extracted verb phrases from 2401 questions in the form of $\langle verb, object \rangle$. As the verbs and objects could be represented in different ways but with the same semantics, we unified them and grouped the questions into implementation issues. Specifically, we split the extracted 2401 verb phrases into 100 sets, where each set contains 24 verb phrases while the last set has 25. The second and third authors unified the word choice of verb phrases set by set. For a set, the authors inspect each phrase in order and normalize them with the following rules: (1) for a phrase appeared before (e.g., load model), they then inspect the next; (2) for a new phrase (e.g., restore checkpoint or load checkpoint) with the same meaning of a previous one, they choose the frequently used one (i.e., restore checkpoint) as the representative; (3) if a phrase is too specialized with limited counts (e.g., pick output), they replaced it with a hypernym (i.e., get output); and (4) if the meaning of a new phrase is unclear, they reread the post question again and clarify the phrase (i.e., get output of RNN layer). After accomplishing one set, two authors checked their difference and unified an inconsistency with the same normalization rules. In the first 20 sets, it takes a long time for authors to reach a consensus due to the frequently occurring discrepancy between authors and incompatibility among sets. However, this situation was mitigated substantially from the later sets. And the discrepancy and incompatibility no longer appeared close to 50 sets.

Table 2
Definitions of seven DL implementation phases.

No.	Phase	Description
1	Data processing	Structuring data and manipulating structured data.
2	Model setting	Setting the network structure of the DL model.
3	Model training	Training a constructed DL model.
4	Model prediction	Testing a trained DL model on a prediction task.
5	Model evaluation	Measuring the performance of model training or prediction.
6	Runtime environment	Setting the devices (e.g., CPU or GPU) for running DL models.
7	Visualization	Presenting a visual representation of DL model or process.

2.4.2. Step-2: Assigning questions with implementation phase (Level-I)

The second and third authors inspected each question independently, and built the Level-I categories (i.e., Phase) by organizing the question through card sorting [49]. For the card sorting activity, the second and third authors sorted the questions into seven piles: data processing, model setting, model training, model prediction, model evaluation, runtime environment, and visualization. As shown in Table 2, we pre-defined the seven phases [50] based on the understanding of implementation issues grouped in Step-1. Specifically, each card has a title, i.e., the name of the category. The title and the keyword (i.e., concrete problem description) were carefully read by the authors to determine whether the keyword belongs to this title. The disagreements were resolved by open discussion, where the first author joined as a tie-breaker.

2.4.3. Step-3: Summarizing the objects for implementation issues (Level-II)

To bridge the implementation issues and phases, we constructed the Level-II categories (i.e., Object) to summarize the common objects involved in different implementation issues and regarded them as important elements for different phases (Level-I). In specific, for each phase in Level-I, we extracted the core object (e.g., "layer") in the related issues described with the verb phrases (e.g., "freeze layer" and "initialize LSTM layer"). Note that the verb phrase can be easily extracted from our investigated "how-to" implementation questions (e.g., "how to rotate an image for data augmentation") with multiple objects (i.e., "image" and "data"), the core object ("image") can be identified from the verb phrase ("rotate image").

2.4.4. Step-4: Optimizing the two-level taxonomy

Due to the difficulty of manual labeling, we estimate and optimize the taxonomy as follows. First, we assigned each question with the three-level elements: (1) the issue represented by the unified verb phrase, (2) the object summarized from the issues, and (3) the phase assigned for each issue. For example, we assigned the question *"how to plot training loss"* with a triplet $\langle$*plot training loss, model, visualization*$\rangle$. To optimize the taxonomy, the second and third authors checked whether a triplet can represent the semantics of the related question, respectively. We found that the agreement rate was "moderate" (0.57) in terms of Cohen's Kappa statistic [41]. They addressed the disagreements through open discussion with the help of the first author. Totally, we spent three weeks accomplishing the whole construction work of taxonomy.

2.5. Feature measurement

Besides, we aimed to investigate the most popular and difficult implementation posts under our constructed taxonomy, respectively. To investigate these two aspects, we measured the popularity and difficulty with the following metrics.

2.5.1. Popularity

We measured the popularity of a post according to two metrics provided by SO. One is the 'viewcount', the total number of times that a post was viewed by developers [29,31]. A higher viewcount implies that more developers are interested in such implementation issues. Meanwhile, the second metric is the 'score' voted by different developers [31,51]. A higher score suggests the usefulness of posts to address developers' implementation issues. Besides, we observe that the trend of the score is similar to that of viewcount. To confirm this observation, we estimated the correlation between viewcount and score for all pairs of implementation posts. The statistical test showed the correlation coefficient was 0.91 with p-value < 0.05. This statistical result indicated that these two popularity metrics are strongly correlated in statistical significance. In other words, a frequently viewed post would obtain a higher score, and the viewcount is enough to measure the popularity of implementation posts. Therefore, we analyze the popularity of our taxonomy in terms of viewcount.

2.5.2. Difficulty

Following Chen et al. [39], we measured the difficulty of a post with the widely adopted metric, namely response time. It equals the duration time between the raised question and the accepted answer [33]. This metric is useful because researchers observed that SO developers would like to answer a new question in a very short time to gain more reputations, so that the response time can effectively reflect the difficulty of a post in general [31–33]. Although we used the widely adopted metric, it is worth noting that precisely measuring the difficulty is challenging. This is because the response time and acceptance time could be impacted by many factors. For example, when a question is regarding a new/small field, the response time could be long. The reason may not be that the question is difficult but most of the users of SO are not familiar with the field. The acceptance time of a question could also be long because of for example forgetting to accept the answer, which does not mean the question is a difficult one.

3. Results

This section provides our constructed taxonomy in Section 3.1 and describes seven subcategories in Sections 3.2–3.8, respectively. Section 3.9 and Section 3.10 analyzed the popularity and difficulty of different implementation categories, respectively.

3.1. Taxonomy of implementation issues

Table 3 illustrated the count and percentage of related posts on our constructed hierarchical taxonomy of implementation issues using DL frameworks. The taxonomy was organized into two-level categories with seven major categories (Level-I). These major categories contained 63 subcategories (Level-II), such as tensor, trained model, etc. From the table, we can notice that the top three submitted posts related to data processing, model setting, and model training, which accounted for 86.4% of the total posts. We discussed and exemplified each subcategory in the following subsections.

To analyze the difference between different DL frameworks, we measured the difference by computing the Pearson correlation for the post counts in Level-II (Object), as shown in Table 4. In the table, a significant correlation level (i.e., p-value < 0.05) is marked by a star. From the table, we can notice that TensorFlow shows similar count distribution with PyTorch (coefficient = 0.94, p-value < 0.05) and Theano (coefficient = 0.87, p-value < 0.05), respectively. Moreover, we investigated the correlation between TensorFlow and the total counts of three frameworks, and the results show that the correlation is strong (coefficient = 1.00) and significant (p-value < 0.05). These statistical results suggest that the findings generated from the total counts could be generalizable for the three DL frameworks investigated in our study. Therefore, we did not investigate the characteristics of the taxonomy from each framework, respectively.

> **Finding 1:** *We constructed a taxonomy of 7 categories with 63 subcategories, indicating the diversity of implementation issues. Data processing, model setting, and model training were the most frequently discussed categories. The findings generated from the taxonomy are generalizable for each DL framework.*

3.2. Data processing

Data Processing (D) refers to the manipulation of ten data types: data, tensor, image, variable, matrix, dataset, batch, vector, placeholder, and label. In this study, we come up with such classification by extracting the word from what the post questions describe. For example, if a question describes an object as a variable (or tensor), we record it as a variable (tensor). Similarly, many questions tell an object as "data", e.g., "reshape data", which could be any of the other data types. Note that the questions describing an object as a specific data type involve data manipulations that are related to that type and distinct from the rest of the types, e.g., "get covariance matrix", while the questions tell an object as "data" involve general manipulations that are not for a specific type. Since the use of more abstract types will result in the loss of much information about the post questions, we use data types as subcategories, and regard both data and other types as the same classification level, and a post will not be classified into multiple types. Then, the number of data processing posts is the sum of the numbers of the ten different recorded types. Other phases also applies this classification method.

Note that the dataset indicates the data that could be used for model training, prediction, or evaluation. As illustrated in Table 3, most of the processing objects were data, tensor, and image, which were related to 426, 387, and 105 posts, respectively. Together, these three data types accounted for 81.6% of the total posts in the data processing.

For the ten data types described above, the key difference is their processing methods. To understand these methods, we categorized related implementation issues into five groups as shown in Table 5. In this table, we did not distinguish the data types and called them all "data". From the table, we can observe that: 169 posts focused on data creation and initialization *(a)*, e.g., creating sparse data or freezing elements in data during creation, where the row "create data" means the requirement of data creation (e.g., variable or tensor); the row "create (random/sparse / ...) data" indicates some special requirements of data creation (e.g., the data needs to be random or sparse). 135 posts addressed the process of data reading (i.e., "read data") and saving (i.e., "save data") *(b)*. 108 posts are about getting and checking the status of data *(c)*, e.g., getting the shape of data and checking whether data contains zero values. Different from "read data" which loads existing data stored from disk to memory, "get data" means the operation of accessing one data in memory (e.g., getting one variable from a programming context). Moreover, 445 posts suffered from issues in data transforming and data type conversion *(d)*. And 268 posts devoted to making some basic computations or operations on data *(e)*, such as addition or sampling.

> **Finding 2:** *Among the 1125 data processing issues, 81.7% of the processing objects are general data, tensor, and image; 63.4% of the posts raised implementation questions on data transformation, conversion, computation, and operation.*

3.3. Model setting

Model Setting (M) considers how to set the structure of a DL software. Table 6 presented the 11 kinds of model settings. From the table, we can notice that 519 posts talked about implementation issues on

Table 3
Count and percentage of posts for the taxonomy.

Level-I (Phase)	Level-II (Object)	TensorFlow	PyTorch	Theano	Total
(D) Data Processing	1. Data	328 (13.7%)	91 (03.8%)	7 (00.3%)	426 (17.7%)
	2. Tensor	279 (11.6%)	102 (04.2%)	6 (00.2%)	387 (16.1%)
	3. Image	77 (03.2%)	28 (01.2%)	0 (00.0%)	105 (04.4%)
	4. Variable	60 (02.5%)	10 (00.4%)	3 (00.1%)	73 (03.0%)
	5. Matrix	25 (01.0%)	14 (00.6%)	4 (00.2%)	43 (01.8%)
	6. Dataset	26 (01.1%)	4 (00.2%)	0 (00.0%)	30 (01.2%)
	7. Batch	18 (00.7%)	9 (00.4%)	1 (00.0%)	28 (01.2%)
	8. Vector	10 (00.4%)	6 (00.2%)	1 (00.0%)	17 (00.7%)
	9. Placeholder	15 (00.6%)	0 (00.0%)	0 (00.0%)	15 (00.6%)
	10. Label	1 (00.0%)	0 (00.0%)	0 (00.0%)	1 (00.0%)
	Total	**839 (34.9%)**	**264 (11.0%)**	**22 (00.9%)**	**1125 (46.9%)**
(M) Model Setting	1. Layer	256 (10.7%)	42 (01.7%)	7 (00.3%)	305 (12.7%)
	2. Model	134 (05.6%)	25 (01.0%)	2 (00.1%)	161 (06.7%)
	3. Weight	41 (01.7%)	11 (00.5%)	3 (00.1%)	55 (02.3%)
	3. Trained model	40 (01.7%)	13 (00.5%)	0 (00.0%)	53 (02.2%)
	4. Model graph	24 (01.0%)	1 (00.0%)	0 (00.0%)	25 (01.0%)
	5. Activation	10 (00.4%)	1 (00.0%)	2 (00.1%)	13 (00.5%)
	7. Saved model file	11 (00.5%)	0 (00.0%)	0 (00.0%)	11 (00.5%)
	8. Trained weight	4 (00.2%)	2 (00.1%)	0 (00.0%)	6 (00.2%)
	9. Filter	5 (00.2%)	0 (00.0%)	0 (00.0%)	5 (00.2%)
	10. Operation	5 (00.2%)	0 (00.0%)	0 (00.0%)	5 (00.2%)
	11. Summary	4 (00.2%)	0 (00.0%)	0 (00.0%)	4 (00.2%)
	Total	**534 (22.2%)**	**95 (04.0%)**	**14 (00.6%)**	**643 (26.8%)**
(T) Model Training	1. Loss	67 (02.8%)	14 (00.6%)	0 (00.0%)	81 (03.4%)
	2. Model	47 (02.0%)	11 (00.5%)	1 (00.0%)	59 (02.5%)
	3. Gradient	36 (01.5%)	8 (00.3%)	2 (00.1%)	46 (01.9%)
	4. Trained model	15 (00.6%)	1 (00.0%)	1 (00.0%)	17 (00.7%)
	5. Weight	11 (00.5%)	5 (00.2%)	0 (00.0%)	16 (00.7%)
	6. Variable	9 (00.4%)	4 (00.2%)	0 (00.0%)	13 (00.5%)
	7. Checkpoint	12 (00.5%)	0 (00.0%)	0 (00.0%)	12 (00.5%)
	8. Learning rate	6 (00.2%)	5 (00.2%)	0 (00.0%)	11 (00.5%)
	9. Statement	9 (00.4%)	1 (00.0%)	1 (00.0%)	11 (00.5%)
	10. Batch	7 (00.3%)	0 (00.0%)	0 (00.0%)	7 (00.3%)
	11. Training progress	6 (00.2%)	1 (00.0%)	0 (00.0%)	7 (00.3%)
	12. Layer	5 (00.2%)	1 (00.0%)	0 (00.0%)	6 (00.2%)
	13. Training data	5 (00.2%)	0 (00.0%)	0 (00.0%)	5 (00.2%)
	14. Training step	4 (00.2%)	0 (00.0%)	0 (00.0%)	4 (00.2%)
	15. Optimizer	3 (00.1%)	1 (00.0%)	0 (00.0%)	4 (00.2%)
	16. Session	3 (00.1%)	0 (00.0%)	0 (00.0%)	3 (00.1%)
	17. Training history	2 (00.1%)	0 (00.0%)	0 (00.0%)	2 (00.1%)
	18. Derivative	1 (00.0%)	1 (00.0%)	0 (00.0%)	2 (00.1%)
	Total	**248 (10.3%)**	**53 (02.2%)**	**5 (00.2%)**	**306 (12.7%)**
(P) Model Prediction	1. Model	72 (03.0%)	6 (00.2%)	1 (00.0%)	79 (03.3%)
	2. Prediction result	27 (01.1%)	9 (00.4%)	1 (00.0%)	37 (01.5%)
	3. Layer	8 (00.3%)	0 (00.0%)	0 (00.0%)	8 (00.3%)
	4. Weight	3 (00.1%)	0 (00.0%)	0 (00.0%)	3 (00.1%)
	5. Threshold	1 (00.0%)	0 (00.0%)	0 (00.0%)	1 (00.0%)
	Total	**111 (04.6%)**	**15 (00.6%)**	**2 (00.1%)**	**128 (05.3%)**
(E) Model Evaluation	1. Accuracy	12 (00.5%)	4 (00.2%)	0 (00.0%)	16 (00.7%)
	2. Other metrics	13 (00.5%)	3 (00.1%)	0 (00.0%)	16 (00.7%)
	3. Result	9 (00.4%)	1 (00.0%)	1 (00.0%)	11 (00.5%)
	4. Time	3 (00.1%)	2 (00.1%)	0 (00.0%)	5 (00.2%)
	5. Evaluation method	2 (00.1%)	2 (00.1%)	0 (00.0%)	4 (00.2%)
	6. Training	1 (00.0%)	0 (00.0%)	0 (00.0%)	1 (00.0%)
	Total	**40 (01.7%)**	**12 (00.5%)**	**1 (00.0%)**	**53 (02.2%)**
(R) Runtime Environment	1. GPU	12 (00.5%)	5 (00.2%)	0 (00.0%)	17 (00.7%)
	2. GPU memory	8 (00.3%)	0 (00.0%)	1 (00.0%)	9 (00.4%)
	3. GPU/CPU	3 (00.1%)	1 (00.0%)	0 (00.0%)	4 (00.2%)
	4. TPU	1 (00.0%)	1 (00.0%)	0 (00.0%)	2 (00.1%)
	5. CPU	1 (00.0%)	1 (00.0%)	0 (00.0%)	2 (00.1%)
	6. RAM	1 (00.0%)	0 (00.0%)	0 (00.0%)	1 (00.0%)
	Total	**26 (01.1%)**	**8 (00.3%)**	**1 (00.0%)**	**35 (01.5%)**
(V) Visualization	1. Data	38 (01.6%)	0 (00.0%)	2 (00.1%)	40 (01.7%)
	2. Model	26 (01.1%)	5 (00.2%)	0 (00.0%)	31 (01.3%)
	3. Evalution	9 (00.4%)	4 (00.2%)	0 (00.0%)	13 (00.5%)
	4. Layer	8 (00.3%)	0 (00.0%)	0 (00.0%)	8 (00.3%)
	5. Progress	8 (00.3%)	0 (00.0%)	0 (00.0%)	8 (00.3%)
	6. Result	6 (00.2%)	1 (00.0%)	0 (00.0%)	7 (00.3%)
	7. Tensorboard	4 (00.2%)	0 (00.0%)	0 (00.0%)	4 (00.2%)
	Total	**99 (04.1%)**	**10 (00.4%)**	**2 (00.1%)**	**111 (04.6%)**

how to manipulate a DL *layer (M1)*, *model (M2)*, and *trained model (M4)*. In this study, the model means the whole DL components and the layer indicates one component of the model. As the objects (layer, model, trained model) could be exchanged in practices (e.g., an LSTM

Table 4
Correlation of post counts between different DL frameworks, "Total" indicates the total counts for all frameworks.

Level-I (Phase)	TensorFlow vs. PyTorch	TensorFlow vs. Theano	TensorFlow vs. Total
(D) Data Processing	0.98* (p = 0.00)	0.85* (p = 0.00)	1.00* (p = 0.00)
(M) Model Setting	0.98* (p = 0.00)	0.86* (p = 0.00)	1.00* (p = 0.00)
(T) Model Training	0.92* (p = 0.00)	0.42 (p = 0.08)	1.00* (p = 0.00)
(P) Model Prediction	0.69 (p = 0.13)	0.22 (p = 0.68)	0.99* (p = 0.00)
(E) Model Evaluation	0.68 (p = 0.21)	0.84 (p = 0.08)	0.99* (p = 0.00)
(R) Runtime Environment	0.70 (p = 0.12)	0.39 (p = 0.45)	0.99* (p = 0.00)
(V) Visualization	0.18 (p = 0.69)	0.82* (p = 0.02)	0.99* (p = 0.00)
All	0.94* (p = 0.00)	0.87* (p = 0.00)	1.00* (p = 0.00)

* Denotes that the correlation level shows statistical significance (i.e., p-value < 0.05).

Table 5
Implementation issues for Data Processing (D) subcategories in terms of the verb phrases.

Summary of issues	Implementation issues	Count	Total
(a) Create/Initialize	1. create data	33	169
	2. create (random/sparse/mutable/constant/consecutive/adversarial) data	11	
	3. (initialize/assign/feed/change/label/remap/rename) data	102	
	4. (encode/decode/tokenize/parse) data	20	
	5. freeze (data/element)	3	
(b) Read/Save	1. read data	95	135
	2. save data	40	
(c) Get/Check	1. get data	43	108
	2. get (type/structure/shape/dimension/size/length/count/summary) of data	44	
	3. get (index/label/coordinate/decoder) of data	15	
	4. check (exist/in/empty/zero/invertible) for data	6	
(d) Transform/Convert	1. (combine/concatenate/stack/merge/unpack/unstack/map/cluster) data	56	445
	2. convert data type	127	
	3. (transform/transpose/rotate/expand/reverse/flatten/distort/shift) data	28	
	4. (mutate/pad/mask) data	24	
	5. (upscale/rescale/enlarge/upsampling/add-Gaussian-noise) data	7	
	6. (reshape/resize/repeat/vectorize) data	74	
	7. (augment/interpolate/principal-component-analysis) data	12	
	8. (slice/split/extract/crop/segment/interleave/subset) data	81	
	9. remove (element/duplicates) from data	8	
	10. (normalize/denormalize/standardize/clip) data	16	
	11. (append/insert/extend) data	12	
(e) Compute/Operate	1. (add/sum/increment/subtract/decay/multiply/divide) data	72	268
	2. (square/product/tensordot/mean-square/square-error) data	11	
	3. (average/mean/variance/covariance) data	11	
	4. (intersect/trace/softmax/factorize) data	9	
	5. compute (distance/entropy/similarity/difference/divergence)	16	
	6. compute (histogram-equalization/hessian-matrix/statistics)	11	
	7. (find/replace/sample/compare) data	69	
	8. (enqueue/synchronize/share) data	3	
	9. (iterate/index/sort/shuffle/filter/permutate/serialize) data	66	
Total	–		**1125**

can be used as a layer of another model) and these posts cover a large number of posts, we analyze these posts together. Subsequently, we summarized the specific objects (Table 7) and related operation requirements (Table 8), respectively.

Table 7 summarized the 46 kinds of specific objects (e.g., embedding and pooling) of *M1, M2, and M4*. We can notice that the frequently discussed specific models or layers are convolution network, long short-term memory (LSTM), embedding network, and dropout network. Meanwhile, Table 8 showed the key operations working on the model or layer listed in Table 7. The verb groups *(a–c)* are the same as the verbs listed in Table 5 for data processing. However, there were 97 special operations *(f)*, such as splitting a model, doing singular value decomposition (SVD) on a layer, selecting the best-trained model, etc.

When setting a model, some important components were frequently discussed in 95 posts, such as how to manipulate the model *weight (M3)* and the *trained weight (M8)*, how to create customized *activation function (M6)*, *filter (M9)*, *operation (M10)*. Meanwhile, for a constructed model or layer, 40 posts discussed the implementation issues on their inner components, including how to process the *model graph (M5)*, *saved model file (M7)*, and the *summary (M11)* of model structure using DL frameworks.

Finding 3: *Among the 643 model setting posts, 80.7% of the implementation posts were about setting model or its layer, involving 43 types of specific DL models with 37 kinds of operations; the remaining posts focused on setting inner components of a layer or model or handling the recording of the structure.*

3.4. Model training

Model Training (T) aims to build and optimize a constructed DL model. Table 9 presented 18 different types of objects during model training. We can find that similar to model setting many posts were relevant to DL *model (T2)*, *trained model (T4)*, and *layer (T12)* with a total of 82 implementation questions. Different from the model or layer listed in Table 7 for model setting, the model training only covered a limited number of kinds. Besides, a significant effort was spent on optimizing the inner components of a model or layer, which included 173 relevant posts, e.g., normalizing the *loss (T1)*, ascending the *gradient (T3)*, clipping the *weight (T5)*, updating the model *variable (T6)*, decaying the *learning rate (T8)*, setting the *optimizer (T15)*, and

Table 6
Implementation issues for Model Setting (M) subcategories in terms of the verb phrases, where details on M1, M2, M4 are summarized in Tables 7–8. Note that Model (M2) indicates the whole DL structure; Summary (M15) copes with the implementation issues on the recording of model setting; Trained Model (M5) cares about the structure changing while Trained Weight (M11) focused on the operations on model weights.

Level-II (Object)	Implementation Issues	Count	Total
(M1) Layer	–	305	305
(M2) Model	–	161	161
(M3) Weight	4. (get/copy/save/restore) weight 5. (share/embed/mask/merge) weight 6. (addition/multiply/product/average/decay) weight 7. (initialize/regularize/mutate) weight 8. (transform/convert/reset/change) weight	15 10 5 17 8	55
(M4) Trained Model	–	53	53
(M5) Model Graph	1. (create/save/restore) model graph 2. (convert/copy) model graph 3. (add/find) model graph 4. get (input/output) of model graph	15 4 2 4	25
(M6) Activation	1. create (Gaussian/linear/PReLU/ReLus) activation 2. (change/regularize) activation 3. get (type/shape/output) of activation	8 2 3	13
(M7) Saved Model File	1. (format/convert) saved model file 2. get (tag-name/content/activation) of saved model file	7 4	11
(M8) Trained Weight	1. (create/get/mask/restore) trained weight	6	6
(M9) Filter	1. (create/enlarge/delete) filter	5	5
(M10) Operation	1. (get/append/join/name) operation	5	5
(M11) Summary	1. (create/restore) summary	4	4
Total	**–**		**643**

Table 7
Object types of layer, model, and trained model in model setting.

No.	Object types	Count	No.	Object types	Count
1	Model	143	24	Deconvolution	3
2	Layer	88	25	Lambda	3
3	Convolution	43	26	Flatten	2
4	Trained model	42	27	Standardization	2
5	LSTM	27	28	Decoder	2
6	Embedding	22	29	DNN	2
7	Dropout	15	30	Inception	2
8	RNN	13	31	convLSTM	1
9	Input	13	32	Feedforward	1
10	Dense	12	33	Probabilistic	1
11	Pooling	11	34	Logistic	1
12	Normalization	8	35	Nonlinear	1
13	Fully connected	7	36	Gradient reversal	1
14	Output	7	37	Sharpening	1
15	Hidden	6	38	Reshaping	1
16	BERT	5	39	Attention	1
17	Softmax	4	40	Mask	1
18	Autoencoder	4	41	Sorting	1
19	CNN	4	42	Encoder/decoder	1
20	VGG	4	43	Encoder	1
21	Transformer	3	44	Regression	1
22	:inear	3	45	AlexNet	1
23	MLP	3	46	ResNet	1
–	–	–	–	**Total**	**519**

computing the *derivative (T18)*. Besides, 11 posts are about the issues of implementing control (i.e., looping or branching) *statement (T9)* during training.

Moreover, for the remaining posts, 28 of them cared about the training process. For example, developers wanted to limit the total number of *checkpoint (T7)*, switch *training progress (T11)* from model optimization to validation, early stop the *training step (T14)*, exit the training *session (T16)*, and save the *training history (T17)* for each epoch. Furthermore, 12 posts discussed implementation issues on manipulating *batch (T10)* and *training data (T13)*, such as how to sample or shuffle a batch during the model training, and how to reshape or add Gaussian noise to training data.

Finding 4: *For the 306 model training posts, 80.2% of posts were devoted to the implementation issues on the training model, layer, or inner parts of components; the remaining posts concentrated on the training progress and related data.*

3.5. Model prediction

Model Prediction (P) focuses on the testing tasks for a trained model. Table 10 showed that 90 related posts targeted the model objects. Specifically, 79 prediction tasks involved the issues on a *model (P1)*, such as the gated recurrent unit (GRU), deep neural network (DNN), and recurrent neural network (RNN) models. Eight prediction tasks worked on a model *layer (P3)*, e.g., input and embedding layer. Three posts discussed the issues on model *weights (P4)*, such as freezing weight during prediction, or performing prediction on a specific weight. On the other hand, 37 tasks were looking for the answers to check and manipulate the *prediction result (P2)*, e.g., getting, selecting, and aggregating the label of the prediction result. Meanwhile, one task aimed to lower the *threshold (P5)* to control the prediction result.

Finding 5: *For the 128 model prediction posts, 70.3% of them addressed the implementation issues on the testing model, layer, and weight; the remaining posts were related to the manipulation of prediction results.*

3.6. Model evaluation

Model Evaluation (E) endeavors to measure the performance of model training or prediction. As illustrated in Table 11, 53 evaluation tasks were grouped into six categories. From the table, we can notice that 37 posts addressed the implementation issues on the performance measurement in terms of *accuracy (E1)*, *other metrics (E2)*, and *time*

Table 8
Verb types of layer, model, and trained model in model setting.

Summary of verb	Verb types	Count	Total
(a) Create/Initialize	1. create	136	239
	3. (initialize/feed/change/name/rename)	98	
	4. decode	2	
	5. freeze weight	3	
(b) Read/Save	1. restore	50	76
	2. save	26	
(c) Get/Check	1. get	4	9
	2. get (shape/dimension)	5	
(d) Transform/Convert	1. (combine/concatenate/stack/merge/convert)	57	84
	3. (transpose/expand/flatten/reverse)	9	
	4. (pad/mask)	4	
	5. (scale/rescale)	3	
	6. reshape	1	
	8. split	2	
	9. remove	6	
	10. normalize	2	
(e) Compute/Operate	1. (add/sum/multiply)	6	14
	3. mean	1	
	7. find	1	
	8. share	6	
(f) Special Operation	1. (add/split/remove/change/prune/SVD layer)	21	97
	2. add (variable/constant/threshold/bias/filter/sliding-window)	7	
	3. (regularize/unfreeze) weight	4	
	4. get (feature/tokenizer)	4	
	5. change (axis/activation/class-count)	3	
	6. (select-best/copy)	3	
	7. get layer from model	8	
	8. get (name/scope)	3	
	9. get (input/output)	32	
	10. get (weight/bias/unit/encoder/stride/activation)	12	
Total	–	**519**	

(E4), such as computing accuracy for a training epoch, calculating a specific metric (e.g., F1-score and perplexity), and measuring the computation time during model preprocessing. To analyze the measured performance, 11 posts were about the issues on *results (E3)* for validation or prediction. In specific, developers addressed the issues on how to compute the confusion matrix, class probabilities, and error ratio; how to find errors during the model evaluation. Moreover, four posts cared about the *evaluation method (E5)*, including how to use cross-validation for evaluation, and how to parallelize the process to reduce evaluation time. Additionally, one evaluation post aims to determine the overfitting of a model training process (see Table 11).

Finding 6: *For the 53 model evaluation posts, 69.8% of them worked on performance measurement in terms of accuracy, other metrics, and time. The remaining posts were devoted to analyzing the prediction result, choosing the best evaluation method, and determining the overfitting of model training.*

3.7. Runtime environment

Runtime Environment (R) refers to the key devices for accomplishing the above implementation issues. We grouped relevant 35 posts into six parts as shown in Table 12. We can find that 17 posts worked on *GPU (R1)*, such as checking the available GPU, selecting one GPU, using multiple GPU for model training/testing, and prefetching data into GPU. Nine posts discussed the implementation issues on the *GPU memory (R2)*, including how to check, add, limit, and clear memory usage. Besides, four posts talked about the issues on the usage of TPU (R4) and CPU (R5), while four other posts aimed to switch the usage between *GPU/CPU (R3)*. In addition, one more post sought a way to free the memory of *RAM (R6)*.

Finding 7: *For the 35 posts on runtime environment, 74.3% of them addressed the implementation issues on the usage of GPU; the remaining posts involved the usage of other devices (i.e., CPU, TPU, and RAM), and the switch of GPU and CPU.*

3.8. Visualization

Visualization (V) aims to present a visual representation of the other implementation tasks for better understanding. Table 13 illustrated the seven visualization subcategories. The table shows that 40 posts asked the methods to visualize *data (V1)*, such as printing a tensor, variable, or image. 39 posts found the answers to display *model (V2)* or *layer (V4)*, e.g., displaying the weight and activation of a model, and printing the shape of an input layer. Additionally, eight posts addressed the issues on printing the *progress (V5)* of model training, validation, or evaluation. Meanwhile, 20 posts discussed the issues on visualization method for prediction *result (V6)* and *evaluation (V3)* outputs (e.g., confusion matrix and accuracy). Besides, four posts asked for ways to use *Tensorboard (V7)*, a visualization toolkit in the TensorFlow framework.

Finding 8: *For the 111 visualization posts, 42.3% of them addressed the implementation issues on visualizing data or prediction results; 35.1% of posts found a way to visually represent a model or layer; the remaining ones are related to the visualization issues of model evaluation, progress, and the tool Tensorboard.*

3.9. Popularity analysis

As described in Section 2.5.1, we measured the popularity of an implementation issue by the viewcount and score, where a higher viewcount indicates an issue is frequently inspected by many developers

Table 9
Implementation issues for Model Training (T) subcategories in terms of the verb phrase. Note that "Statement" indicates the usage of looping and branching statements during model training.

Level-II (Object)	Implementation issue	Count	Total
(T1) Loss	1. (compute/get/save/restore) loss	74	81
	2. (replace/remove) loss	3	
	3. (mask/normalize/regularize) loss	4	
(T2) Model	1. train one model	29	59
	2. train (RNN/UNet/CNN/LSTM/GAN/Tflean/Bert/DNN) model	18	
	3. train (multi-task/autoencoder/regression) model	5	
	4. change input for training epoch	1	
	5. save model during training	6	
(T3) Gradient	1. (compute/change/update) gradient	23	46
	2. (differentiate/accumulate) gradient	4	
	3. (get/find/track/save) gradient	13	
	4. (ascend/clip/invert/scale) gradient	5	
	5. stop using gradient	1	
(T4) Trained Model	1. (restore/save) trained model	15	17
	2. tune trained model	2	
(T5) Weight	1. (train/update) weight	6	16
	2. (freeze/decay/clip) weight	5	
	3. (get/save) weight	5	
(T6) Variable	1. (create/get/count/save/restore) variable	9	13
	2. (assign/train/update) variable	4	
(T7) Checkpoint	3. (save/restore) checkpoint	11	12
	4. limit checkpoint	1	
(T8) Learning Rate	1. (get/log/save) learning rate	4	11
	2. (change/decay) learning rate	7	
(T9) Statement	1. (create/delete) branch	9	11
	2. (create/skip) loop	2	
(T10) Batch	1. (get/sample) batch during training	3	7
	2. (feed/shuffle/change) batch during training	4	
(T11) Training Progress	1. (check/switch training/validation) mode	3	7
	2. get training step	1	
	3. change random number during training	1	
	4. (save/stop) training	2	
(T12) Layer	1. train (embedding/convolution/dropout) layer	3	6
	2. unfreeze layer	1	
	3. (reset/test) layer for training epoch	2	
(T13) Training Data	1. (load/sample/count/reshape) training data	4	5
	2. add Gaussian noise to training data	1	
(T14) Training Step	1. (initialize/get) training step	2	4
	2. (reduce/early-stop) training step	2	
(T15) Optimizer	1. (get/compute) optimizer	4	4
(T16) Session	1. (get/restore/exit) session	3	3
(T17) Training History	1. (get/save) training history for training epoch	2	2
(T18) Derivative	1. (get/compute) derivative	2	2
Total	–	**306**	

Table 10
Implementation issues for Model Prediction (P) subcategories in terms of the verb phrase.

Level-II (Object)	Implementation issue	Count	Total
(P1) Model	1. test one model	60	79
	2. test (GRU/DNN/RNN/LSTM/FeedForward/Resnet/CNN/BERT/Inception) model	17	
	3. feed data during testing	2	
(P2) Prediction Result	1. get prediction (result/label/index)	27	37
	2. (select/save) prediction result	4	
	3. (convert/mask/normalize/aggregate) prediction result	6	
(P3) Layer	1. test (one/LSTM) layer	4	8
	2. test (input/embedding) layer	4	
(P4) Weight	1. (load/freeze) weight during prediction	2	3
	2. perform prediction using weight	1	
(P5) Threshold	1. lower threshold for prediction	1	1
Total	–	**128**	

and a larger score means that more developers confirmed its usefulness. To understand the popularity of the implementation issues, we calculated the median values for different categories and subcategories, respectively.

Table 11
Implementation issues for Model Evaluation (E) subcategories in terms of the verb phrases.

Level-II (Object)	Implementation issue	Count	Total
(E1) Accuracy	1. compute accuracy	5	16
	2. compute accuracy for (one/epoch/training)	5	
	3. compute accuracy for (pixel-wise/in-class/multi-label)	6	
(E2) Other Metrics	1. compute custom metric	6	16
	2. compute (F1-score/R2-score/AUC/Recall/MAP)	6	
	3. compute perplexity	2	
	4. compute hamming loss	1	
	5. compute PSNR	1	
(E3) Result	1. compute confusion matrix	6	11
	2. compute class probabilities	1	
	3. compute error ratio	2	
	4. find error	2	
(E4) Time	1. compute preprocessing time	1	5
	2. compute (total/layer) training time	3	
	3. compare (training/testing) time	1	
(E5) Evaluation Method	1. use cross-validation	3	4
	2. parallelize evaluation	1	
(M6) Training	1. determine overfitting	1	1
Total	–	**53**	

Table 12
Implementation issues for Runtime Environment (R) subcategories in terms of the verb phrases.

Level-II (Object)	Implementation issue	Count	Total
(R1) GPU	1. check (available/used) GPU	2	17
	2. (select/use) one GPU	7	
	3. use multiple GPUs	7	
	4. prefetch data to GPU	1	
(R2) GPU Memory	1. check memory usage	2	9
	2. use more memory	2	
	3. (limit/reduce/clear) memory	5	
(R3) GPU/CPU	4. check if using GPU or CPU	2	4
	5. (select/switch) GPU or CPU	2	
(R4) TPU	1. check TPU	1	2
	2. use TPU	1	
(R5) CPU	1. use CPU	2	2
(R6) RAM	1. free memory	1	1
Total	–	**35**	

From Table 14, we can notice that the median score is no more than 9 while the median viewcount shows a wide range from 69 to 7643.5. This case implies that many developers met implementation issues, they only viewed the post but not voted.

Table 14 shows that *visualization (V)* is the most popular category with median viewcount 1047, where most viewcounts come from the model (1817), evaluation (1535), Tensorboard (1301), and progress (1277). These popular posts indicate that many developers would like to visualize the model settings, prediction performance, and running progress, which can help implement DL models correctly. Runtime environment (R) ranks second in terms of popularity (median viewcount = 761), where many developers inspected the posts on switching GPU/CPU (7643.5). Meanwhile, model evaluation (E) is the third most popular category (median viewcount = 755). Many developers inspected the posts on the evaluation issues during model training (2015) and testing (1274), instead of the evaluation measurements and others.

Among the remaining category, model setting ranks the highest (median viewcount = 639). We can notice that most viewcounts come from the setting of operation (3978) and the recording of model setting (i.e., save model file), suggesting the lack of this knowledge for many developers. The other subcategories (e.g., model, layer, activation, and weight) draw a similar degree of popularity. Furthermore, although model training (T) shows lower popularity (median viewcount = 529), four subcategories involve many viewcounts including the derivative (2615), training step (2528), layer (2423), and learning rate (2353). This result implies that many developers suffered from issues in implementing the customized optimization method, instead of invoking a standard API in the library. For the model prediction with a median viewcount of 568, many views focused on the implementation issues about prediction result. Besides, data processing (D) ranks the last (median viewcount = 475). We can notice that many relevant posts care about the processing of images (765), indicating a large number of image-processing practitioners.

> **Finding 9:** *The most popular implementation issues are related to the Level-II categories model evaluation, runtime environment, and visualization in terms of the median viewcount; the posts score is strongly correlated with the viewcount, indicating a frequently viewed post is also useful for developers.*

3.10. Difficulty analysis

To analyze the difficulty of an implementation issue, we measured it by the response time (days) of an accepted answer as described in Section 2.5.2. Same to popularity analysis, we measured the difficulty of a category or subcategory in our taxonomy using the median values.

As illustrated in Table 14, the most difficult category is the runtime environment (R). The related implementation posts took about 0.33 median days to receive an accepted answer. The issues on TPU (7.14) and CPU (5.40) are the most difficult subcategories compared to GPU (0.53) and GPU memory (0.06), which implies that many developers lack such implementation experiences. Mode setting (M) is the second difficult category (0.15), where most of the difficulties come from the setting of the model summary. The third difficult category is visualization (V) with median days 0.14, where Tensorboard issues involved more time (18.32 days). For the model training (T), we can notice that the response time on issues of the derivative is long (179.25 days), this is because someone answered the question but the answer was not accepted, and no one answered for a long time afterwards. Considering its high popularity (median viewcount 2615), many developers suffered from this kind of difficult issue. While the responses in the SO are positive, there may be a need for sustained attention to these difficult issues

As to the other three categories, the overall difficulty degree is smaller with median days ≤0.10. Model prediction (P) shows higher difficulty, where the manipulating threshold took the second longest

Table 13
Implementation issues for Visualization (V) subcategories in terms of the verb phrases.

Level-II (Object)	Implementation issue	Count	Total
(V1) Data	1. print (data/tensor/variable) 2. display image	30 10	40
(V2) Model	1. display model 2. display (weight/filter/activation/learning-rate/gradient) of model 3. plot training loss 4. display regressed equation	8 11 11 1	31
(V3) Evaluation	1. display confusion matrix 2. print accuracy 3. plot (accuracy/ROC) curve	6 5 2	13
(V4) Layer	1. print shape of input layer 2. print output of (certain/hidden/output) layer	1 7	8
(V5) Progress	3. print (training/prediction/evaluation) progress 4. print checkpoint	7 1	8
(V6) Result	1. print prediction (result/label) 2. display tested image	6 1	7
(V7) Tensorboard	1. display tensorboard 2. reset tensorboard	3 1	4
Total	–	**111**	

time among all subcategories, due to this question has gone unanswered for a long time since it was posted. The difficulty of data processing (D) is close to (0.08), which demonstrates that issues in this category are relatively easy to address. The implementation issues on batch show a higher response time (0.15), suggesting that this data type needs more attention for developers. Moreover, the difficulty of model evaluation (E) ranks last but the evaluation during training (7.79) is not easy for developers.

To understand the relationship between the posts of popularity and difficulty, we conducted a correlation analysis on all posts. The statistical test showed that the Pearson correlation coefficient is 0.17 with p-value < 0.05, indicating a small correlation with high confidence. This statistical result indicates that a popular post is not necessarily difficult.

Finding 10: *The most difficult implementation issues are coming from the category of runtime environment, followed by model setting, visualization, and model training; the posts on setting derivative and manipulating threshold for model prediction require a median of more than four months to receive an accepted answer.*

4. Discussion

This section presents the implications and future directions to address the implementation issues in Section 4.1, and discusses the potential threats to validity in Section 4.2.

4.1. Implication

In this study, we investigated the taxonomy, popularity, and difficulty of DL implementation issues, respectively. Based on the findings, we discussed our insights and practical implications for developers, researchers, and practitioners as follows.

Generally, developers suffered from various DL issues when implementing DL models. There is a requirement of comprehensive handbook for developers to list the taxonomy of frequently occurred implementation issues with highlighted the difficulties and high-quality solutions. Otherwise, they have to repeat the search of Stack Overflow and try different possible solutions after reading many answers. Our constructed taxonomy could be used as a reference to build the handbook in the future.

In our taxonomy, data processing, model setting, and model training dominated the implementation issues. This is because developers do not know the original data structure and have to implement their customized algorithms to handle the data; they are unclear of the model structure in details and inexperience of building, recording, and restoring models; they are struggling in optimizing a model in terms of each specific objects (e.g., gradient, derivative, and optimizer) in their own ways. Besides, developers have difficulties in addressing the implementation issues on runtime environment due to their lack of experience in dealing with hardware (e.g., GPU, TPU, and CPU).

To address these issues, we first suggest researchers and practitioners providing more visualization tools to help developer easily understand the data, model, and running procedures. The popularity of visualization posts can confirm this requirement, although the total number is still small. Besides, to help developers avoid repeated implementations, a reusable high-quality toolkit with easy-to-use APIs is required. To mitigate the difficulties on runtime environment, the DL framework needs to incorporate better libraries to help developers control the environment easier.

4.2. Threats to validity

In this section, we discuss the threats to the validity of our empirical study.

4.2.1. Selection of DL frameworks

Our empirical study on implementation issues was based on three selected DL frameworks, which may lead to potential selection bias for this study. The validity of our findings may be threatened by including more DL frameworks, such as Keras [52] and DL4J [24]. We plan to investigate them in the near future. To mitigate the negative effect of the selection bias, we chose three representative DL frameworks as our study subjects, namely TensorFlow, PyTorch, and Theano. Moreover, these three frameworks are widely used in research studies and industrial applications [7,9].

4.2.2. Scope of data sampling

In this study, collecting data from other source, e.g., issue tracker on GitHub [27], may threaten the validity of our findings. We plan to investigate the differences of implementation issues requested in various data sources in the future. We filtered the collected data based on tags (i.e., "TensorFlow", "PyTorch", and "Theano") of posts. However, it is possible that a post is not marked as one of these tag but is related to the three DL frameworks. We noticed that related works commonly filtered the data on tags, due to the difficulty of filtering this kind of post [19,24,29]. Therefore, we followed the related works

Table 14
Median viewcount, score, response time (days) under the taxonomy.

Level-I (Phase)	Level-II (Object)	Viewcount	Score	Response
(D) Data Processing	1. Data	529.00	1.00	0.09
	2. Tensor	402.00	1.00	0.06
	3. Image	765.00	1.00	0.10
	4. Variable	650.00	1.00	0.06
	5. Matrix	350.00	1.00	0.06
	6. Dataset	436.00	0.00	0.09
	7. Batch	263.00	0.00	0.15
	8. Vector	466.00	2.00	0.04
	9. Label	368.00	0.00	0.01
	Overall	**475.00**	**1.00**	**0.08**
(M) Mode Setting	1. Layer	603.00	1.00	0.13
	2. Model	639.00	1.00	0.17
	3. Weight	861.00	1.00	0.14
	4. Trained Model	883.00	1.00	0.21
	5. Model Graph	608.00	1.00	0.42
	6. Activation	693.00	0.00	0.06
	7. Saved Model File	1672.00	2.00	0.28
	8. Trained Weight	1061.00	0.50	0.05
	9. Filter	277.00	1.00	0.05
	10. Operation	3978.00	2.00	0.04
	11. Summary	891.50	4.00	1.23
	Overall	**639.0**	**1.00**	**0.15**
(T) Model Training	1. Loss	322.00	1.00	0.10
	2. Model	283.00	1.00	0.11
	3. Gradient	864.50	1.50	0.20
	4. Trained Model	993.00	2.00	0.05
	5. Weight	713.50	1.50	0.26
	6. Variable	576.00	0.00	0.03
	7. Checkpoint	1493.00	2.50	1.00
	8. Learning Rate	2353.00	3.00	0.06
	9. Statement	119.00	1.00	0.07
	10. Batch	537.00	1.00	0.09
	11. Training Progress	879.00	4.00	0.03
	12. Layer	2423.00	2.00	1.17
	13. Training Data	368.00	1.00	0.06
	14. Training Step	2528.00	2.00	2.57
	15. Optimizer	261.50	0.50	0.27
	16. Session	1551.00	1.00	0.03
	17. Training History	141.50	1.50	0.34
	18. Derivative	2615.00	6.00	179.25
	Overall	**529.00**	**1.00**	**0.11**
(P) Model Prediction	1. Model	570.00	1.00	0.18
	2. Prediction Result	959.00	1.00	0.05
	3. Layer	147.00	1.00	0.20
	4. Weight	184.00	1.00	0.01
	5. Threshold	69.00	0.00	128.43
	Overall	**568.00**	**1.00**	**0.10**
(E) Model Evaluation	1. Accuracy	676.50	1.00	0.07
	2. Other Metrics	983.50	1.50	0.65
	3. Result	1274.00	1.00	0.01
	4. Time	177.00	0.00	0.04
	5. Evaluation Method	404.00	1.00	0.07
	6. Training	2015.00	9.00	7.79
	Overall	**755.00**	**1.00**	**0.05**
(R) Runtime Environment	1. GPU	312.00	1.00	0.53
	2. GPU Memory	2399.00	3.00	0.06
	3. GPU/CPU	7643.50	6.00	2.78
	4. TPU	1268.50	1.50	7.14
	5. CPU	4311.50	4.00	5.40
	6. RAM	456.00	0.00	1.80
	Overall	**761.00**	**1.00**	***0.33***
(V) Visualization	1. Data	714.50	1.00	0.16
	2. Model	1817.00	2.00	0.15
	3. Evaluation	1535.00	2.00	0.09
	4. Layer	302.50	1.00	0.08
	5. Progress	1277.50	1.00	0.02
	6. Result	995.00	0.00	0.27
	7. Tensorboard	1301.50	3.00	18.32
	Overall	***1047.00***	**1.00**	**0.14**

to filter data based on tags, which could be a threat to the validity of our findings. Besides, we sampled the "how to" posts as study subjects. This sampling method may introduce potential threats to the validity of our findings. However, the threats would be minor, because these "how to" questions are representative of implementation issues [38,39]. We excluded the posts without accepted answers to support the analysis of post difficulty. These posts can represent the frequently encountered important implementation issues, as SO has a very active community in which the questions posed receive an answer that is formally accepted by the question-asker [53,54]. However, our constructed taxonomy may not cover these unanswered implementation questions. In the near future, we plan to extend our work on more implementation posts.

4.2.3. Subjectivity of manual inspection

To refine the dataset, we excluded irrelevant data with manual inspections based on a pilot study with 100 randomly selected samples, which could threaten the validity of this study. Also, we manually labeled each post with an unified verb phrase and constructed the taxonomy of implementation issues. The subjective bias of the manual work could threaten the validity of empirical analysis. To mitigate this threat, two authors conducted the data refinement. Meanwhile, they also estimated and optimized the constructed taxonomy. Any disagreement during these two manual works was resolved by open discussion with another author as a tie-breaker. Besides, the inter-rater agreements were relatively high, which implies the reliability of the manual works.

5. Related work

In this section, we summarized the related work on empirical studies of DL frameworks.

5.1. High-level topics

Many researchers focused on the high-level topics on DL frameworks discussed in SO to understand existing issues [24,25,31,55]. Zhang et al. [24] manually inspected a sample of 715 questions in SO and identified seven kinds of frequently asked questions, including implementation, program crash, training anomaly, model deployment, comprehension, installation, and performance. They found that implementation, program crash, and model migration are the top frequently asked questions in SO. Later, Han et al. [25] leveraged a domain-specific workflow and LDA technique to analyze the high-level topics on three widely adopted DL frameworks (i.e., TensorFlow, PyTorch, and Theano) using dataset collected from SO and GitHub. They observed that model training and preliminary preparation are the most frequently discussed. Afterwards, many researchers conducted an in-depth analysis of one of these high-level topics. For example, Chen et al. [31] performed an in-depth analysis on the DL deployment by mining 3k relevant posts from SO. Besides, Yang et al. [55] investigated the self-claimed major reasons behind the uncertainties during DL implementations, such as vulnerabilities, failures, inconsistencies, misuse, etc. Other in-depth empirical studies on DL are summarized as below.

5.2. Quality assurance

As program crash is the top frequently asked questions for DL developers [24], many researchers performed an in-depth analysis on the quality assurance of DL frameworks [23,26–28,39]. Zhang et al. [26] studied the characteristics and root causes of defects in DL applications built on TensorFlow, and collected related program bugs from SO and GitHub. Islam et al. [27] performed a comprehensive study on DL bugs. They investigated the types of bugs, root causes of bugs, impacts of bugs, and bug-prone stage of the DL pipeline when using DL frameworks. Due to time pressure, market competition, and cost reduction, DL developers commonly implemented the core

components of DL projects only and left many technical debts to be done in the near future. Liu et al. [23] investigated the technical debt in DL implementation in GitHub. They found that many DL developers recorded their incomplete design, requirement, and algorithm as to-do tasks in the code comments. Liu et al. [28] explored how the technical debts in seven DL frameworks were introduced and removed. Furthermore, Chen et al. [39] presents a comprehensive study on the deployment faults of mobile DL apps by mining discussions from SO and GitHub.

5.3. Library dependency

DL frameworks depend on several libraries that come into a supply chain of DL frameworks, and the features of the supply chain affect how developers use them for DL implementation and deployment. Han et al. [19] investigated the dependency networks of libraries in DL frameworks. The study unveiled some commonalities among DL frameworks in terms of purpose, application domain, and dependency degrees, and discovered some discrepancies in the update behaviors, update reasons, and version distributions. Zhang et al. [56] conducted a large-scale and in-depth study on the API evolution in the DL framework TensorFlow. They investigated the key reasons for API changes by mining API documentation, commits and SO. Results showed that the APIs were changed mainly due to performance optimization, ease of use, functionality enhancement, etc. Tan et al. [57] presented an empirical study on the supply chain within the DL framework in terms of their structure, application domain, and evolutionary factors.

5.4. Computation capability

A DL model could be implemented and deployed using different DL frameworks. The differences in computing performance and energy will support the choice of DL framework. Bahrampour et al. [8] compared the capability of different DL frameworks. They found that Theano and Torch are easier to extend new DL components or models. Theano and Torch showed the best running speed on GPU and CPU, respectively. TensorFlow is very flexible to use. Shams et al. [58] analyzed the performance of DL frameworks over different hardware environments in terms of speed and scaling. They observed that the model performance could be strongly affected by the choice of DL framework and hardware. Liu et al. [59] observed that the DL performance could be highly impacted by the default settings optimized for a specific dataset in different DL frameworks. Guo et al. [1] compared the DL models implemented by using different DL frameworks TensorFlow, PyTorch, CNTK, and MXNET. Experimental results indicated that the prediction accuracy could be affected by different frameworks. The migration and quantization of DL models on different platforms suffered from compatibility and reliability issues. Shanbhag et al. [60] investigated the energy pattern for developing DL applications by mining 1361 posts from Stack Overflow, and provided a general energy guideline for DL developers.

5.5. Comparison between our study and related work

The related empirical studies on high-level topics (Section 5.1) manually or automatically build general classification for SO posts. Different from them, we focused on one of the important category (i.e., the DL implementation) and conducted an in-depth empirical study. Although there are many other in-depth studies described in Sections 5.2–5.4, they did not cover the implementation issues. Besides, many empirical studies (e.g., library dependency and computation capability) are commonly based on code analysis or program running, instead of analyzing the SO post as our study. In this study, we summarized a wide spectrum of implementation issues when using DL frameworks, indicating a requirement for more studies to help developers improve their development productivity.

6. Conclusion

In this paper, we have presented a comprehensive study of implementation issues for DL applications. By manual examination of 2401 real-world implementation posts extracted from SO, we have derived a taxonomy of implementation issues with 7 categories and 63 subcategories, indicating that the process of developing DL models stretches over a wide spectrum of issues. Moreover, we investigated their popularity and difficulty under our constructed taxonomy, which helps to facilitate understanding of frequently occurred issues and issues that are difficult to address in a short time. Finally, we have discussed insightful implications for developers and researchers based on our findings, and suggested some directions to address the implementation issues and improve the DL implementation productivity. In the near future, we will investigate the tools to address the implementation issues following our discussions.

CRediT authorship contribution statement

Chao Liu: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing. **Runfeng Cai:** Investigation, Resources, Data curation, Formal analysis. **Yiqun Zhou:** Investigation, Resources, Data curation. **Xin Chen:** Supervision, Project administration, Funding acquisition. **Haibo Hu:** Visualization, Validation. **Meng Yan:** Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared a link to my data in the manuscript.

Acknowledgments

This research/project is supported by: the National Key R&D Plan of Ministry of Science and Technology of China (2020YFC2007902); the National Nature Science Foundation of China (62202074, 62372071); China Postdoctoral Science Foundation (2022M710519); the Postdoc Foundation of Chongqing, China (2021LY23); the Venture & Innovation Support Program for Chongqing Overseas Returnees, China (cx2019106); and Chongqing Technology Innovation and Application Development Special Key Project, China (cstc2019jscx-fxydX0054).

References

[1] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, X. Li, An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms, in: International Conference on Automated Software Engineering (ASE), IEEE, 2019, pp. 810–822, http://dx.doi.org/10.1109/ASE.2019.00080.

[2] B.J. Erickson, P. Korfiatis, Z. Akkus, T. Kline, K. Philbrick, Toolkits and libraries for deep learning, J. Digit. Imaging (JDI) 30 (4) (2017) 400–405, http://dx.doi.org/10.1007/s10278-017-9965-6.

[3] G. Giray, A software engineering perspective on engineering machine learning systems: State of the art and challenges, J. Syst. Softw. (JSS) 180 (2021) 111031, http://dx.doi.org/10.1016/j.jss.2021.111031.

[4] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554, http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[5] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90, http://doi.acm.org/10.1145/3065386.

[6] C. Liu, X. Xia, D. Lo, C. Gao, X. Yang, J. Grundy, Opportunities and challenges in code search tools, ACM Comput. Surv. 54 (9) (2021) 1–40, http://dx.doi.org/10.1145/3480027.

[7] Y. Yang, X. Xia, D. Lo, J. Grundy, A survey on deep learning for software engineering, ACM Comput. Surv. 54 (10s) (2022) 1–73, http://dx.doi.org/10.1145/3505243.

[8] S. Bahrampour, N. Ramakrishnan, L. Schott, M. Shah, Comparative study of deep learning software frameworks, 2015, arXiv preprint arXiv:1511.06435. https://arxiv.org/pdf/1511.06435.

[9] C. Liu, C. Gao, X. Xia, D. Lo, J. Grundy, X. Yang, On the reproducibility and replicability of deep learning in software engineering, ACM Trans. Softw. Eng. Methodol. (TOSEM) 31 (1) (2021) 1–46, http://dx.doi.org/10.1145/3477535.

[10] N. Zhang, C. Liu, X. Xia, C. Treude, Y. Zou, D. Lo, Z. Zheng, ShellFusion: Answer generation for shell programming tasks via knowledge fusion, in: International Conference on Software Engineering (ICSE), 2022, pp. 1970–1981, http://dx.doi.org/10.1145/3510003.3510131.

[11] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, G. Jeong, Applying deep learning based automatic bug triager to industrial projects, in: Foundation of Software Engineering (FSE), 2017, pp. 926–931, http://dx.doi.org/10.1145/3106237.3117776.

[12] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, Deep learning for computer vision: A brief review, Comput. Intell. Neurosci. 2018 (2018) http://dx.doi.org/10.1155/2018/7068349.

[13] J. Hirschberg, C.D. Manning, Advances in natural language processing, Science 349 (6245) (2015) 261–266, http://dx.doi.org/10.1126/science.aaa8685.

[14] A. Lopez, Statistical machine translation, ACM Comput. Surv. 40 (3) (2008) 1–49, http://dx.doi.org/10.1145/1380584.1380586.

[15] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, IEEE Signal Process. Mag. 29 (6) (2012) 82–97, http://dx.doi.org/10.1109/MSP.2012.2205597.

[16] J. Petersen, P.F. Jäger, F. Isensee, S.A. Kohl, U. Neuberger, W. Wick, J. Debus, S. Heiland, M. Bendszus, P. Kickingereder, et al., Deep probabilistic modeling of glioma growth, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2019, pp. 806–814, http://dx.doi.org/10.1007/978-3-030-32245-8_89.

[17] C. Chen, A. Seff, A. Kornhauser, J. Xiao, Deepdriving: Learning affordance for direct perception in autonomous driving, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2722–2730, http://dx.doi.org/10.1109/ICCV.2015.312.

[18] Z. Deng, L. Xu, C. Liu, M. Yan, Z. Xu, Y. Lei, Fine-grained co-attentive representation learning for semantic code search, in: International Conference on Software Analysis, Evolution and Reengineering, IEEE, 2022, pp. 396–407, http://dx.doi.org/10.1109/SANER53432.2022.00055.

[19] J. Han, S. Deng, D. Lo, C. Zhi, J. Yin, X. Xia, An empirical study of the dependency networks of deep learning libraries, in: International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2020, pp. 868–878, http://dx.doi.org/10.1109/ICSME46990.2020.00116.

[20] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, Y. Bengio, Theano: A CPU and GPU math compiler in Python, in: Proc. 9th Python in Science Conf, Vol. 1, 2010, pp. 3–10.

[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., TensorFlow: a system for Large-Scale machine learning, in: Symposium on Operating Systems Design and Implementation (OSDI), 2016, pp. 265–283, https://dl.acm.org/doi/10.5555/3026877.3026899.

[22] N. Ketkar, J. Moolayil, Introduction to pytorch, in: Deep Learning with Python, Springer, 2021, pp. 27–91, http://dx.doi.org/10.1007/978-1-4842-5364-9_2.

[23] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, S. Li, Is using deep learning frameworks free? characterizing technical debt in deep learning frameworks, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIP), 2020, pp. 1–10, http://dx.doi.org/10.1145/3377815.3381377.

[24] T. Zhang, C. Gao, L. Ma, M. Lyu, M. Kim, An empirical study of common challenges in developing deep learning applications, in: International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2019, pp. 104–115, http://dx.doi.org/10.1109/ISSRE.2019.00020.

[25] J. Han, E. Shihab, Z. Wan, S. Deng, X. Xia, What do programmers discuss about deep learning frameworks, Empir. Softw. Eng. 25 (4) (2020) 2694–2747, http://dx.doi.org/10.1007/s10664-020-09819-6.

[26] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, L. Zhang, An empirical study on TensorFlow program bugs, in: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2018, pp. 129–140, http://dx.doi.org/10.1145/3213846.3213866.

[27] M.J. Islam, G. Nguyen, R. Pan, H. Rajan, A comprehensive study on deep learning bug characteristics, in: Foundation of Software Engineering (FSE), 2019, pp. 510–520, http://dx.doi.org/10.1145/3338906.3338955.

[28] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, S. Li, An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks, Empir. Softw. Eng. 26 (2) (2021) 1–36, http://dx.doi.org/10.1007/s10664-020-09917-5.

[29] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, J.-L. Sun, What security questions do developers ask? a large-scale study of stack overflow posts, J. Comput. Sci. Technol. (JCST) 31 (5) (2016) 910–924, http://dx.doi.org/10.1007/s11390-016-1672-0.

[30] M. Bagherzadeh, R. Khatchadourian, Going big: a large-scale study on what big data developers ask, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE), 2019, pp. 432–442, http://dx.doi.org/10.1145/3338906.3338939.

[31] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, X. Liu, A comprehensive study on challenges in deploying deep learning based software, in: Foundation of Software Engineering (FSE), 2020, pp. 750–762, http://dx.doi.org/10.1145/3368089.3409759.

[32] S. Ahmed, M. Bagherzadeh, What do concurrency developers ask about? a large-scale study using stack overflow, in: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2018, pp. 1–10, http://dx.doi.org/10.1145/3239235.3239524.

[33] M. Alshangiti, H. Sapkota, P.K. Murukannaiah, X. Liu, Q. Yu, Why is developing machine learning applications challenging? a study on stack overflow posts, in: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, 2019, pp. 1–11, http://dx.doi.org/10.1109/ESEM.2019.8870187.

[34] C. Liu, Replication package, 2022, https://github.com/liuchaoss/UnderstandDL.

[35] StackExchange, The archived dataset of Stack Overflow, 2021, https://archive.org/download/stackexchange/stackoverflow.com-Posts.7z.

[36] MongoDB, MongoDB official website, 2022, https://www.mongodb.com/.

[37] StackExchange, Data structure for Stack Overflow post, 2022, https://archive.org/download/stackexchange/readme.txt.

[38] C. Treude, O. Barzilay, M.-A. Storey, How do programmers ask and answer questions on the web? in: Proceedings of the 33rd International Conference on Software Engineering (ICSE), 2011, pp. 804–807, http://dx.doi.org/10.1145/1985793.1985907.

[39] Z. Chen, H. Yao, Y. Lou, Y. Cao, Y. Liu, H. Wang, X. Liu, An empirical study on deployment faults of deep learning based mobile applications, in: International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 674–685, http://dx.doi.org/10.1109/ICSE43902.2021.00068.

[40] StackOverflow, Meaning of the closed questions in Stack Overflow posts, 2022, https://stackoverflow.com/help/closed-questions.

[41] J. Cohen, A coefficient of agreement for nominal scales, Educ. Psychol. Meas. 20 (1) (1960) 37–46, http://dx.doi.org/10.1177/001316446002000104.

[42] C. Liu, X. Xia, D. Lo, Z. Liu, A.E. Hassan, S. Li, Codematcher: Searching code based on sequential semantics of important query words, ACM Trans. Softw. Eng. Methodol. (TOSEM) 31 (1) (2021) 1–37.

[43] W. Xie, X. Peng, M. Liu, C. Treude, Z. Xing, X. Zhang, W. Zhao, API method recommendation via explicit matching of functionality verb phrases, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pp. 1015–1026.

[44] Y. Yang, X. Xia, D. Lo, J. Grundy, A survey on deep learning for software engineering, ACM Comput. Surv. 54 (10s) (2022) 1–73.

[45] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M.P. Reyes, M.-L. Shyu, S.-C. Chen, S.S. Iyengar, A survey on deep learning: Algorithms, techniques, and applications, ACM Comput. Surv. 51 (5) (2018) 1–36.

[46] J.F. Dye, I.M. Schatz, B.A. Rosenberg, S.T. Coleman, Constant comparison method: A kaleidoscope of data, Qual. Rep. 4 (1/2) (2000) 1–9.

[47] H. Jing, Sentence reduction for automatic text summarization, in: Sixth Applied Natural Language Processing Conference, 2000, pp. 310–315.

[48] P. Sangaroonsilp, H.K. Dam, M. Choetkiertikul, C. Ragkhitwetsagul, A. Ghose, A taxonomy for mining and classifying privacy requirements in issue reports, Inf. Softw. Technol. 157 (2023) 107162.

[49] M. Kim, T. Zimmermann, R. DeLine, A. Begel, The emerging role of data scientists on software development teams, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE, 2016, pp. 96–107, http://dx.doi.org/10.1145/2884781.2884783.

[50] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, nature 521 (7553) (2015) 436–444.

[51] B. Kou, Y. Di, M. Chen, T. Zhang, SOSum: a dataset of stack overflow post summaries, in: Proceedings of the 19th International Conference on Mining Software Repositories (MSR), 2022, pp. 247–251, http://dx.doi.org/10.1145/3524842.3528487.

[52] X. Du, Y. Sui, Z. Liu, J. Ai, An empirical study of fault triggers in deep learning frameworks, IEEE Trans. Dependable Secure Comput. (2022).

[53] W. Zhu, H. Zhang, A.E. Hassan, M.W. Godfrey, An empirical study of question discussions on Stack Overflow, Empir. Softw. Eng. 27 (6) (2022) 148.

[54] A. Anderson, D. Huttenlocher, J. Kleinberg, J. Leskovec, Discovering value from community activity on focused question answering sites: a case study of stack overflow, in: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2012, pp. 850–858.

[55] C. Yang, P. Liang, L. Fu, Z. Li, Self-claimed assumptions in deep learning frameworks: An exploratory study, in: Evaluation and Assessment in Software Engineering (EASE), 2021, pp. 139–148, http://dx.doi.org/10.1145/3463274.3463333.

[56] Z. Zhang, Y. Yang, X. Xia, D. Lo, X. Ren, J. Grundy, Unveiling the mystery of API evolution in Deep Learning frameworks: a case study of TensorFlow 2, in: International Conference on Software Engineering (ICSE-SEIP), IEEE, 2021, pp. 238–247, http://dx.doi.org/10.1109/ICSE-SEIP52600.2021.00033.

[57] X. Tan, K. Gao, M. Zhou, L. Zhang, An exploratory study of deep learning supply chain, in: International Conference on Software Engineering (ICSE), 2022, pp. 86–98, http://dx.doi.org/10.1145/3510003.3510199.

[58] S. Shams, R. Platania, K. Lee, S.-J. Park, Evaluation of deep learning frameworks over different HPC architectures, in: International Conference on Distributed Computing Systems, IEEE, 2017, pp. 1389–1396, http://dx.doi.org/10.1109/ICDCS.2017.259.

[59] L. Liu, Y. Wu, W. Wei, W. Cao, S. Sahin, Q. Zhang, Benchmarking deep learning frameworks: Design considerations, metrics and beyond, in: International Conference on Distributed Computing Systems (ICDCS), IEEE, 2018, pp. 1258–1269, http://dx.doi.org/10.1109/ICDCS.2018.00125.

[60] S. Shanbhag, S. Chimalakonda, V.S. Sharma, V. Kaulgud, Towards a catalog of energy patterns in deep learning development, in: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022, 2022, pp. 150–159, http://dx.doi.org/10.1007/s10664-021-10099-x.