Navigating the Testing of Evolving Deep Learning Systems: An Exploratory Interview Study

Hanmo You
College of Intelligence and Computing
Tianjin University
Tianjin, China
youhanmo@tju.edu.cn

Zan Wang
College of Intelligence and Computing
Tianjin University
Tianjin, China
wangzan@tju.edu.cn

Bin Lin
Hangzhou Dianzi University
Hangzhou, China
b.lin@live.com

Junjie Chen†
College of Intelligence and Computing
Tianjin University
Tianjin, China
junjiechen@tju.edu.cn

***Abstract*—Deep Learning (DL) systems have been widely adopted across various industrial domains such as autonomous driving and intelligent healthcare. As with traditional software, DL systems also need to constantly evolve to meet ever-changing user requirements. However, ensuring the quality of these continuously evolving systems presents significant challenges, especially in the context of testing. Understanding how industry developers address these challenges and what extra obstacles they are facing could provide valuable insights for further safeguarding the quality of DL systems. To reach this goal, we conducted semi-structured interviews with 22 DL developers from diverse domains and backgrounds. More specifically, our study focuses on exploring the challenges developers encounter in testing evolving DL systems, the practical solutions they employ, and their expectations for extra support. Our results highlight the difficulties in testing evolving DL systems (e.g., regression faults, online-offline differences, and test data collection) and identify the best practices for DL developers to address these challenges. Additionally, we pinpoint potential future research directions to enhance testing effectiveness in evolving DL systems.**

***Index Terms*—Deep Learning, Software Evolution, Testing, Interview Study**

I. Introduction

Deep Learning (DL) systems have been widely integrated into products of different sectors, such as autonomous vehicles [1], medical diagnostics [2] and software engineering [3]–[5]. Like traditional software, DL systems also need to continuously evolve to adapt to new requirements and ever-growing demands of users. For example, during the COVID-19 pandemic, Apple updated Face ID, a built-in facial recognition system, which enables users to unlock devices and authenticate purchases while wearing masks, bringing huge convenience to users [6]. However, the evolution of DL systems sometimes may also pose challenges or side effects. For example, the chatbot of the parcel delivery firm DPD was found to swear at users after an update in January 2024, leading to substantial reputation damage [7]. Similarly, ChatGPT has faced criticism from users for providing incorrect or overly simplified answers after updates, resulting in accusations of it becoming "dumber" and "lazier". This has significantly impacted user experience and dented user confidence in OpenAI [8]. These cases underscore the potential risks associated with the evolution of DL systems. Therefore, it is particularly important to conduct rigorous testing during the system evolution.

The academic community has conducted extensive research on DL testing and various approaches have been proposed to reveal bugs of DL systems, such as test case generation [9], [10] and mutation testing [11], [12]. However, these studies often focus on testing a specific version of the DL system, neglecting its evolving nature, which encompasses continuous development, refinement, and enhancement over time. In practice, many aspects of DL systems may change along with the system evolution, such as training data or model structure. The complexity of these systems also increases, making it more challenging to effectively test the entire system [13]. Currently, there remains a notable gap in the knowledge of testing evolving DL systems. Understanding the difficulties faced by developers and the strategies they adopt to handle these difficulties can provide valuable insights for further improving the effectiveness of the testing process in DL systems.

In this study, we interviewed 22 DL system developers from different companies in different sectors and countries, aiming to explore various aspects of testing evolving DL systems, including 1) the specific challenges they face, 2) the effective solutions they adopt to address these challenges, and 3) what kind of additional support they yearn for. Our results lead to six different types of obstacles developers face (e.g., high costs for test case collection/annotation), 19 best practices for addressing these challenges (e.g., adopting large language models to assist in data labeling), and 16 types of concrete support developers would like to have (e.g., tools that can effectively detect errors in annotated data).

† Corresponding Author.

The main contributions of our paper are as follows.

- By consulting experts in the field, we summarize the challenges encountered in testing evolving DL systems.
- We extract the best practices of current DL system developers in testing evolving DL systems. These strategies serve as invaluable references for the industry, offering concrete guidelines and actionable insights that can benefit other DL developers in their endeavors.
- We highlight developers' expectations for extra support in their testing activities, indicating future research directions and potential collaboration opportunities.

The remainder of this paper is organized as follows. In Section II, we discuss the background of DL system evolution and provide an overview of related work pertaining to the development, evolution, and testing of DL systems. In Section III, we elaborate on the methodology, detailing the process of conducting the interviews and analyzing the collected data. Section IV presents our key findings. Subsequently, Section V discusses the implications and offers suggestions for both researchers and DL developers. Furthermore, Section VI addresses the limitations that apply to our study. Finally, Section VII concludes the paper.

II. Background and Related Work

A. Deep Learning Software Development

A typical model training process for a DL system contains the following steps [14]: First, developers collect the relevant training dataset, tailored to the specific requirements of the task at hand. Subsequently, they craft a program, which encapsulates stages including data preprocessing, algorithm selection, training mode configuration, and model architecture design. By executing this program and iteratively training the model with the dataset, the DL model progressively learns and optimizes its internal parameters, known as weights, until it meets the predetermined stopping criteria. In summary, the primary components of the DL system can be conceptually construed as the training dataset, the training program, and the model weights acquired from the training process.

Given the increasingly important role of DL software, researchers have conducted studies to investigate the challenges when developing DL systems. Hill et al. [15] studied the challenges faced by machine learning (ML) system developers through interviews and pointed out the unique challenge of establishing a repeatable process. Zhang et al. [16] surveyed 195 practitioners to understand the difficulties encountered by DL system developers during different stages of the development process, including resource management, requirement analysis, design, implementation, testing & debugging, deployment, and maintenance. Amershi et al. [17] conducted in-depth interviews with Microsoft's internal ML system development team to disclose high-level challenges encountered at each step of the ML development lifecycle, especially, they identified some unique challenges in ML systems, involving data management, model customization and reuse, as well as entangled AI components. Zhang et al. [18] mined StackOverflow and found the most prevalent questions faced by developers in their programming processes include program crashes, model migration, and implementation. They also identified five main root causes of these questions which are worth further attention of the research community.

B. Deep Learning Software Evolution

Like traditional software, DL systems also require continuous evolution to optimize existing functionalities or meet new requirements. Following the definition of software evolution by Eijkelenboom [19], software evolution refers to the cumulative effect of all changes made to a software system throughout its entire lifecycle. In the context of DL systems, evolution usually includes modifications at three levels: data, program, and weights. At the data level, modifications may involve updating the dataset (e.g., adding, deleting, or cleaning data) [20] and refining the features (e.g., enriching, removing, or replacing features) [21]. These changes are crucial for enhancing its predictive capabilities or improving its performance of new functionalities (e.g., new domains of data). At the program level, modifications include processes such as upgrading the model architecture (e.g., model updating [22] or pruning [23]), adjusting the training paradigm or updating dependent libraries and frameworks to their latest versions [24]. These changes are aimed at enhancing the model's predictive capabilities or reducing technical debt. Finally, at the weight level, modifications typically involve direct manipulations of the model's weights to achieve performance improvement [25] or defect fixing [26]. In practical scenarios, multiple aspects of a DL system are often modified simultaneously (e.g., pruning may affect both the model structures and the weights).

C. Deep Learning Software Testing

To ensure the quality and reliability of DL systems, many studies have focused on how to test them. The statistical nature of DL systems makes it challenging to conduct sufficient testing [27]. DL system testing has been explored at the program level [28], [29], model level [30], [31], and library level [32], [33]. Several testing frameworks have been proposed to uncover the vulnerabilities of DL systems. Xie et al. [10] proposed DeepHunter, which uses neuron coverage to guide the generation of test cases for neural networks. Sun et al. [34] proposed DeepConcolic, which leverages the execution of concrete inputs and symbolic analysis to synthesize new test inputs. Wang et al. [35] proposed RobOT, automatically generating test cases to improve model robustness. Ma et al. [12] and Humbatova et al. [36] proposed to design high-quality mutants to improve mutation testing for DL systems. Ma et al. [37] proposed DeepCT based on the insight of combinatorial testing and built an LP-constraint-solving-based test generator. Tian et al. [38] proposed DeepTest, utilizing metamorphic testing to create synthetic test images representing realistic weather or lighting conditions to trigger inaccurate behavior of self-driving cars. Different from these works, which aim to test the specific version of DL systems, we focus on investigating how developers test the evolving DL systems in practice.

Some researchers have focused on the challenges of testing DL systems. Riccio et al. [39] conducted a comprehensive literature review to reveal the concerns regarding testing ML systems, with a particular emphasis on crucial tasks such as test case generation. Similarly, Tambon et al. [40] conducted another literature review that not only analyzes testing aspects but also explores the challenges in model verification. Marijan et al. [41] analyzed the limitations of current testing approaches, with a focus on the absence of test oracles, large input space, and high test effort for white box testing. While these studies have presented an overview of challenges in DL system testing, they focus more on academic issues and less on the practical needs of the industry. Additionally, these studies often fail to consider the critical context of testing the evolving DL systems. This underpins the importance of our research, which focuses on industry demands and aims to bridge the gap between academia and industry.

III. Methodology

In this section, we describe our research questions, the participant recruitment process, the interview design, and the data analysis protocols.

A. Research Questions

To gain deeper insights on how developers can effectively test deep learning systems as they constantly evolve, we propose the following research questions.

- RQ1: What specific **challenges** do developers encounter when testing evolving deep learning systems?
- RQ2: What **solutions** do developers employ to addressing these challenges?
- RQ3: What kind of additional **support** do developers need to facilitate their testing activities in evolving deep learning systems?

B. Semi-Structured Interviews

Given the oversight in the existing literature regarding testing evolving DL systems, we found it challenging to build our study on a broad body of existing knowledge. Therefore, we opted for semi-structured interviews, which offer great flexibility to elicit unexpected insights [42], [43].

1) Interview Participant Recruitment: To recruit participants for this study, we used convenience sampling [44] by reaching out to developers within our network via email and other communication channels. Our selection criteria include that 1) the participants must have at least three months of experience in developing DL software, 2) the participants need to be currently employed as part of a development team for DL systems, and 3) since not all companies, especially small companies, have specified DL systems testers. The participants should have a reasonable understanding of the testing process in their subjects. We sent a total of 77 invitation letters, 55 of which were either ignored or outright rejected due to privacy and confidentiality concerns. Ultimately, we successfully recruited 22 participants from 21 companies, working for varying industry sectors, and located in four different countries.

The demographic information of the recruited participants can be seen in Table I. Our interview participants are located in China, Japan, the USA, and one other country the participant preferred not to disclose. Participants possess a wide range of work experience, ranging from 3 months to 12 years. Given that DL is an emerging field, 12 of 22 participants have a development experience of two to four years. However, we made every effort to include senior developers and managed to recruit six participants with more than five years of professional experience in DL software development. Moreover, our participants work in different types of industry sectors, such as Computer Vision (CV) and Natural Language Processing (NLP), tackling a variety of tasks including autonomous driving and text summarizing. They also work with different types of models, ranging from conventional DL models to Large Language Models (LLMs). The diversity of our participants in gender, location, experience, and job type could bring us comprehensive insights into the questions we aim to answer.

2) Interview Process: In this interview study, our interview questions are guided by our research questions, namely, we focus on the challenges developers face, the solutions they adopt to address the challenges, and the extra support they need to better fulfill the testing requirements. To ensure that our interview questions could lead to desired insights, we conducted pilot interviews with three participants and refined our interview questions. The three participants involved in the pilot interviews were selected from the larger group of 22 interviewees. The list of the interview questions can be found in our replication package [45], [46].

Currently, our department has not established an ethics committee for Software Engineering research. To avoid ethical issues, before the interviews, we asked the recruited developers for their consent to participate in the interview and permission to record the session. In addition, we carefully followed related work [17], [43], [47], [48] and adopted several measures to prevent ethical issues, i.e., using consent forms, anonymizing data, and thoroughly explaining the goal/process/potential outcome of our study to participants. We also informed them that they had the right to withdraw from the interview at any time and that their responses would be deleted.

Each interview spanned from 45 to 90 minutes. The interviews were conducted online since the participants are located in different regions and countries. We used Tencent Meeting to record each interview for initial transcription.

The first author thoroughly corrected the typos or inaccuracies found in the automated transcripts. Additionally, to uphold the privacy of our participants, the first author anonymized the transcripts and eradicated personal information, including participant names, company names, and project details. In some cases, we deliberately abstained from attributing specific quotations, thereby preserving the anonymity of participants.

TABLE I: Details of Study Participants

Name	Age	Gender	Experience	Current Role	Company Size*	Domain	Task	Model
P1	20-29	Female	5.5 yrs	DL Software Developer	*P*	CV	Autonomous Driving	DL Model
P2	20-29	Male	4 yrs	Algorithm Engineer	5-10	CV	Medical Imaging Segmentation	DL Model
P3	20-29	Male	6 mths	Project Manager	∼100	NLP	Text Summarizing	LLM
P4	20-29	Male	2 yrs	Algorithm Engineer	3K+	NLP	Machine Translation	DL Model
P5	20-29	Male	2 yrs	Algorithm Engineer	11K+	CV	Warehouse logistics	DL Model
P6	30-39	Male	5 yrs	Database Algorithm Engineer	∼300	Tabular Data	Database Query Prediction	DL Model
P7	20-29	Female	2.5 yrs	AI System Testers§	∼10K+	CV	Image Generation	LLM
P8	20-29	Male	2 yrs	Audio Algorithm Engineer	∼3K+	Audio	Audio Recognition	DL Model
P9	20-29	Male	2 yrs	Algorithm Engineer	∼10K+	CV	Medical Imaging Segmentation	DL Model
P10	20-29	Male	1.5 yrs	Algorithm Engineer	∼11K+	NLP	Question Answering Robot	DL Model
P11	40+	Male	12 yrs	Tech Lead	*P*	Recommendation	User Item Recommendation	DL Model
P12	20-29	Male	2 yrs	Algorithm Engineer	∼15K	Recommendation	Advertising Recommendation	DL Model
P13	30-39	Male	10 yrs	Senior Software Developer	∼700	Tabular Data	Car Battery Monitoring	DL Model
P14	30-39	Male	11 yrs	CTO	400+	CV	Defect Detection	DL Model
P15	20-29	Male	3 yrs	Algorithm Engineer	∼6K	Recommendation	User Item Recommendation	DL Model
P16	20-29	Male	6 mths	Testing Engineer§	∼38K+	Code	Test Case Generation	LLM
P17	20-29	Female	3 mths	AI System Testers§	∼8K+	Recommendation	User Item Recommendation	DL Model
P18	20-29	Male	2 yrs	LLM Algorithm Engineer	∼200	NLP	Question Answering	LLM
P19	20-29	Male	3 yrs	Algorithm Engineer	*P*	Code	Code Generation	LLM
P20	20-29	Male	2 yrs	Algorithm Engineer	∼100	NLP	Text Summarizing	LLM
P21	20-29	Female	2 yrs	Staff Engineer	∼20	Tabular Data	Intelligent Healthcare	DL Model
P22	30-39	Male	10 yrs	Tech Lead for Testing§	300+	CV	Autonomous Driving	DL Model

* In the column of "Current Role", § indicates that the interview participants are experienced DL testing experts.
* In the column of "Company Size", *P* indicates that the interview participants chose not to disclose the number of employees due to privacy concerns.

C. Qualitative Data Analysis

We followed the coding guidelines established by Saldaña [49] and utilized the computer-assisted qualitative data analysis software `Atlas.ti` [50] to facilitate our coding process. We performed inductive coding [51] at first and then performed axial coding [52] to group the codes into hierarchical themes. Following existing work [53], the modifications made to the codebook were carefully documented. To ensure the consistency of our categorization, two authors independently coded the data and discussed it until a consensus was reached on the categorization. To assess the extent of saturation in our coding, following existing work [43], [54], we initially sampled six interviews and set a stopping criterion of two additional interviews with a saturation threshold of 95%. Assuming now that we have coded n interviews, we examine whether the codes extracted from these n interviews could cover all codes extracted from n+2 interviews. If this is the case, we consider that saturation has been achieved. Otherwise, we code a new interview and check the saturation again. Given this process, the minimum number of interviews that will be conducted is eight. In total, we conducted 22 interviews, although we found that saturation was achieved after coding the 21st interview. We did continue to code the 22nd interview in order to not waste the interview effort, although no new codes emerged.

D. Data Availability

To protect the privacy of our participants, we are not able to share the raw interview transcripts. However, the consent letter, recruitment letter, interview questions, and the final codebook exported from Altas.ti can be found in our replication package [45], [46].

IV. Results

A. Codebook

The final codebook consists of 102 codes, which are categorized into six types of challenges (RQ1), five types of solutions (RQ2), and five types of support needed (RQ3), as illustrated in Fig. 1. The numbers in rectangles represent the number of codes for each category, and the numbers in circles represent the number of participants who provide specific insights.

B. RQ1: Challenges

Twenty-eight codes were extracted and categorized into six types of challenges, including regression faults, well-performing offline models failing online, difficulties in collecting reliable test data, lack of mature approaches and metrics, comprehending black box decision-making, and performance preservation.

Regression faults. Facing regression faults is the most common challenge faced by developers, mentioned by 16 developers. One common type of regression fault is related to the metrics. This often occurs during system optimization, when developers focus on improving certain metrics, other metrics might unexpectedly degrade, as mentioned by P2:

"During an optimization of text summarization function of our model, we collected new data for finetuning and successfully improved the accuracy of text summarization from more than 70% to more than 80%. However, subsequent testing revealed that the correct matching rate between text summaries and original paragraphs significantly decreased from the original level to nearly 40%."

The second type of regression fault concerns specific test cases. After an update, the DL model might not be able to correctly make predictions for some cases that it could handle without issues previously. This is particularly problematic in sensitive fields such as security, politics, or human healthcare:

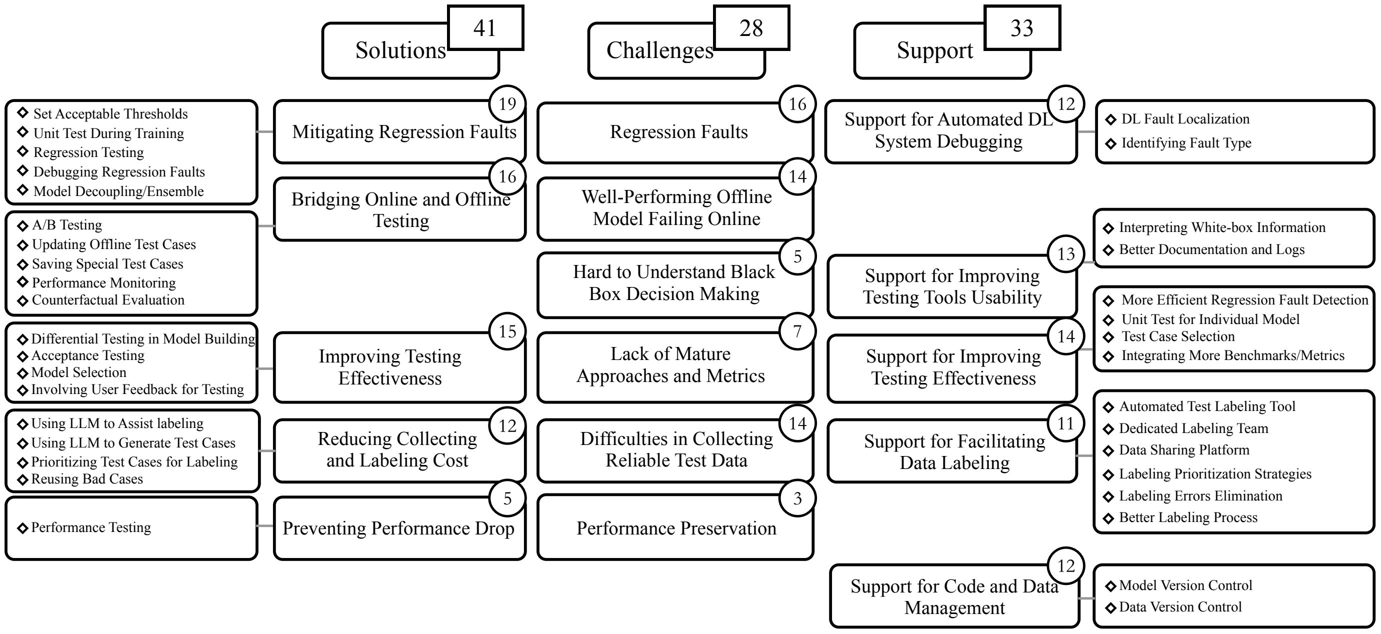
Solutions
41
Challenges
28
Support
33
Set Acceptable Thresholds
Unit Test During Training
Regression Testing
Debugging Regression Faults
Model Decoupling/Ensemble
Mitigating Regression Faults
19
A/B Testing
Updating Offline Test Cases
Saving Special Test Cases
Performance Monitoring
Counterfactual Evaluation
Bridging Online and Offline Testing
16
Differential Testing in Model Building
Acceptance Testing
Model Selection
Involving User Feedback for Testing
Improving Testing Effectiveness
15
Using LLM to Assist labeling
Using LLM to Generate Test Cases
Prioritizing Test Cases for Labeling
Reusing Bad Cases
Reducing Collecting and Labeling Cost
12
Performance Testing
Preventing Performance Drop
5
Regression Faults
16
Well-Performing Offline Model Failing Online
14
Hard to Understand Black Box Decision Making
5
Lack of Mature Approaches and Metrics
7
Difficulties in Collecting Reliable Test Data
14
Performance Preservation
3
Support for Automated DL System Debugging
12
DL Fault Localization
Identifying Fault Type
Support for Improving Testing Tools Usability
13
Interpreting White-box Information
Better Documentation and Logs
Support for Improving Testing Effectiveness
14
More Efficient Regression Fault Detection
Unit Test for Individual Model
Test Case Selection
Integrating More Benchmarks/Metrics
Support for Facilitating Data Labeling
11
Automated Test Labeling Tool
Dedicated Labeling Team
Data Sharing Platform
Labeling Prioritization Strategies
Labeling Errors Elimination
Better Labeling Process
Support for Code and Data Management
12
Model Version Control
Data Version Control

Fig. 1: The Findings Summarized from the Interview

"After the update, we have to ensure that all test cases involving national laws are correctly predicted, as any omission may lead to legal disputes." -P8

Well-performing offline model failing online. The DL model that passed offline testing may not perform well in online applications. Specifically, P11 points out:

"The (online-offline) gap may come from data collection methods, differences in data update frequency, or mismatches in timestamps. For recommendation systems, especially during specific periods such as seasonal changes, this online-offline difference is particularly significant."

P11 further explained this issue with the example of COVID-19. The recommendation system well tested offline did not perform as expected due to dramatic changes in user behaviors and social interaction patterns during the pandemic.

Difficulties in collecting reliable test data. As DL systems continue to evolve, many enterprises opt to establish annotation teams to continuously label new test cases, safeguarding the model performance. However, the process of collecting and labeling these test cases is costly, as commented by P3:

"We once recruited an annotation team to label the summarization of articles, with each article costing upwards of 200-300 RMB (approximately equivalent to 27 to 41 USD). Though expensive, the labeling quality is not always satisfying."

Another issue in obtaining reliable test data is the subjectivity of labeling test cases:

"As for the AI-generated images, whether they look good or not is really up to each person's taste. Different users will label them different scores because it's so subjective." - P7

Even if all these issues are resolved, there is a still major obstacle to hinder developers in their efforts to collect a large amount data conveniently and rapidly, which is privacy concerns. For example, for collecting test cases, P21 shares:

"In the medical domain, for tasks such as predicting patient survival time, obtaining test cases is really hard. Part of the problem is that patient information is sensitive, so we have to be careful with privacy. In Japan, they've got really strict rules about protecting patients' privacy, which makes it really hard for hospitals to share data with each other. And when we need to collect more data from other hospitals for testing, well, let's just say it's really hard."

Sometimes, when there are data available, it is pre-processed, making it difficult to label, as mentioned by P17:

"(In QR code recognition), the images are related to privacy. We can't directly see the images but only get the embedding, which makes it nearly impossible to label."

Lack of mature approaches and metrics. Some developers complain that although academia is constantly proposing DL testing approaches and metrics, these methods may not be applicable in real industrial scenarios:

"For (mechanical parts) defect detection tasks, the scenario is complex, with numerous elements present in the images to be recognized. In addition, defects can appear on objects of different materials and shapes, making it difficult for existing testing methods to cover possible scenarios and accurately assess performance in these scenarios. While using LLM to generate images for testing may be useful, it risks introducing new biases into the testing results. - P14

Moreover, developers often do not know what approaches they can use to test DL systems, as mentioned by P7:

"In traditional software, we have regression testing. For DL systems, we don't know any mature and widely-used approaches to effectively test them after model evolution."

The lack of intuitive testing criteria also poses issues, as suggested by P8:

"I know testing criteria like neuron coverage, but we never use it in our testing process. We cannot understand the direct correlation between a neuron's activation state and specific words or semantics in our application scenarios. We don't know what it means to cover a neuron."

Comprehending black box decision making. DL systems typically comprise a large number of parameters interconnected through complicated computations, making it challenging to directly analyze their internal decision-making processes. Therefore, developers and testers often resort to end-to-end black-box testing approaches, as mentioned by P13:

"Traditional software usually has clear logic and is easier to interpret. When we get the test report, it's easier to trace the exact lines of code causing the bugs. But a DL model is like a black box; it's really hard to understand what's happening inside, and there are so many things that can lead to buggy performance, like the training process, the model design, and the quality of training data. That makes it really tough to figure out how to debug just by looking at the test results."

The issue intensifies when the model comparison is needed:

"It's difficult to compare and explain the differences between the decision-making logic of different models, let alone measuring the changes during model evolution." - P10

Performance preservation. Sometimes, conducting feature evolution (e.g., adding or deleting features) or updating model structures (e.g., increasing the number of layers) can significantly impact the performance, such as inference time and memory usage. However, developers often pay more attention to the performance related to functional requirements (e.g., prediction accuracy), neglecting the performance of non-functional requirements (e.g., time needed for prediction):

"They (hospital medical staff) need our system to quickly provide (bone age) prediction results in their daily work, especially when using devices with limited computation capabilities. If the model's performance decreases after updating the structure, medical staff may have to wait too long and complain to us." - P2

C. RQ2: Solutions

The solutions, consisting of 19 pieces of advice summarized from 41 codes, aim to address five types of challenges.

Mitigating regression faults. Regression faults frequently occur in industrial applications. To prevent such faults, developers have proposed various approaches, which can be categorized into three stages: Pre, In, and Post Evolution.

In the pre-evolution stage, some developers propose prioritizing functions based on their importance according to the requirements and setting acceptable thresholds for performance degradation for unimportant ones:

"In autonomous driving, we think that common scenarios related to vehicle recognition and traffic are more critical, while a certain degree of decline in the recognition accuracy of secondary elements, such as trees and streetlights, is acceptable." - P22

During the evolution process, developers employ methods such as continual learning to assist training and closely monitor the current model's performance during training to decide whether it is time to stop:

"We conduct tests during the training of a new model. We monitor details like the convergence rate every 100 epochs and compare it with the performance of the previous (version of the) model at the same epoch to check if the performance of the monitored metrics is degrading. Then, we decide whether to continue with the training." - P8

In the post-evolution stages, developers have recommended conducting thorough regression testing on DL systems:

"After each model update, we evaluate its effectiveness on existing test sets. If critical functions deteriorate beyond our acceptance threshold, we will roll back to the previous version of the model." - P18

Given the high cost of training, some developers use debugging to eliminate regression faults:

"We really try hard to find the root causes of those regression faults, but it's tough sometimes. If we can't directly find them, we may analyze which functions' performance is not good enough and add relevant data to alleviate the issue, even though it might lead to overfitting." - P8

Exploring other compensating approaches is another way to alleviate regression faults, and one such example is decoupling and integrating old and new models:

"After an update, our text summarization model struggled to balance the performance for short and long texts. We finally decoupled the model into two independent ones. We used different models and prompts for short and long user inputs, respectively, to effectively resolve the issue." - P3

Bridging online and offline testing. During our interviews, developers have proposed several strategies for bridging online and offline testing, which encompass three key approaches: 1) A/B Testing; 2) Updating Offline Test Cases; and 3) Supervision.

Regarding how to conduct A/B Testing, P15 outlines their comprehensive process:

"For our recommendation models, after passing offline testing, we deploy them online with a small fraction of user traffic (around 0.25% or less). Once we confirm that the model's overall performance metrics are satisfactory online, we gradually increase the traffic allocation to the new version of the model, typically in increments of 0.25%, 0.5%, 1%, 5%, 10%, 20%, 50%, and ultimately 100%. Throughout this process, if at any stage the model is unable to make correct recommendations, we immediately roll back to the original model. After the new model has fully replaced the old one and consistently demonstrates stable performance over time, we designate it as the default model. This entire process is referred to as solidification."

In addition to the above testing approaches, it is crucial to assess the similarity in distribution between offline and online test cases and update offline test cases when needed. This prevents discrepancies between offline and online performance:

"We employ a test case lifecycle strategy. We record the timestamp of each test case and set a retention threshold. When the model is updated, test cases whose saving time exceeds this retention threshold are removed. By leveraging this threshold, we ensure a balance between using new test cases and retaining certain older test cases, thereby maintaining the comprehensiveness and representativeness of our testing. For database-related tasks, our typical retention threshold ranges from 3 to 6 months." - P6

P15 adds insights on which tests to update:

"For recommendation tasks, we generally prefer the latest test cases. However, we must also consider special days such as Singles' Day and Black Friday (shopping carnivals), when there's a surge in purchases. During those times, we reuse test cases from the exact same period last year for testing."

Lastly, a robust online performance monitoring mechanism is essential to prevent data and concept drift:

"We continuously monitor the model's performance on data, tracking metrics like the average and variance of certain values, and conducting counterfactual evaluations for early warning. This helps us to decide whether to update the model or test cases." - P11

Improving testing effectiveness. Developers are seeking to further enhance testing efficacy to ensure the quality of DL systems throughout their continuous evolution. P15 illustrated how they applied differential testing in model building:

"We initiate testing during the model-building process. For example, even for the same model structure, we use different libraries and languages, or we just put it on different devices, such as CPUs and GPUs, to construct a model with an identical structure and then employ differential testing to check whether the training results of these two models are similar."

P19 shared their experience in terms of acceptance testing:

"We conduct Alpha, Beta, and Gamma testing. Firstly, Alpha testing is the self-validation stage to ensure the model's basic functions and performance meet expectations. Beta testing focuses on a deeper validation of the model's core functionalities to ensure they meet requirements. Lastly, during Gamma testing, in addition to continuing to test core functionalities, we introduce monkey testing to simulate users' random and unusual queries, comprehensively assessing the model's responsiveness and stability."

P14 shared some tips on model selection:

"We provide testers with several models that we believe perform well. They may need to select test cases to differentiate their performance and select the best one for deployment."

Given that high scores on some testing metrics do not mean good performance, especially in NLP domains, multiple developers incorporate feedback from users as the gold standard:

"For summarization tasks, if errors occur in the first few sentences, users will lose trust in the subsequent detailed summaries. No metric can surpass user feedback, and we need to regularly conduct user surveys." - P3

Reducing test case collecting and labeling cost. Testing of DL systems often requires a large amount of test data, but the cost of collecting and labeling is high. To reduce the cost of labeling, P14 shared their experience in using LLMs to assist labeling:

"We originally relied on manual labeling, but now we are trying to use LLM to assist in labeling, as well as double-checking the labeling results."

The benefit of LLMs is not limited to assisting labeling, developers can use LLMs to generate test cases:

"If there is a real lack of test cases, LLM can be used to generate some test cases to initiate the training and testing of this project. After collecting real data provided by users, the test cases and model can be updated." - P5

P6, instead, focused on prioritizing test cases for labeling:

"For our task, correctly identifying slow queries with optimization potential is more meaningful. So, test cases are easier to wrongly predict as the slow queries are more important to us. We consider using prioritization strategies and fingerprint matching techniques to find these test cases for labeling."

Given the limited labeling resources, reusing bad cases could be an effective way to save the effort:

"We have established a bad case library to store and manage representative fault-triggering cases encountered previously. These cases are classified by tags to help us quickly find cases related to specific faults to improve the model performance accordingly. For example, in our tasks (animal imaging generation), we will collect all errors related to ears and label them with ear-related tags. After each model update, we will reuse these bad cases to test the new model to see if its performance has been optimized or deteriorated." - P7

Preventing performance drop. Developers believe that during the model evolution, it is necessary to pay close attention to non-functional requirements, such as inference time and memory usage. To this end, developers stress the importance of performance testing after evolution:

"After updating features and model structures, we conduct performance testing. We send requests to the model to check that the response time of both the coarse-grained and fine-grained recommendation models does not exceed the preset millisecond threshold. In addition, we also conduct stress testing to simulate user traffic peak scenarios to ensure that the QPS of the model does not decrease by more than 5%, in order to maintain service stability." - P15

P15 further indicated that for feature evolution, they prefer to replace old features with new ones to maintain the relative stability of the total number of features. This can prevent performance degradation caused by changes in model structure.

D. RQ3: Support

In terms of support developers needed to improve their testing activities, we extracted 16 actionable items from 33 codes, which were classified into five different categories.

Support for automated DL system debugging. The most frequently mentioned support needed is automated debugging tools. Specifically, 12 of the 22 participants expressed that their current test results are insufficient for effectively identifying and fixing regression faults in DL systems. They urgently need tools that can accurately locate the faults:

"Our debugging process relies heavily on experience, and test reports are not very helpful. For instance, after an update, we noticed that the model was incorrectly outputting German verbs in lowercase when they should be capitalized. We suspected that the root cause might be that the developers accidentally lowered all letters of verbs in new data, causing a conflict with the original training data. While this was a straightforward case to debug, sometimes when the model outputs strange words, we can't even guess which stage introduced the bug. The sources of errors can be numerous, including the data, hyperparameters, and the training process. [. . .] Compared with bugs in the code, faults from training data are hard to debug. There are too many samples to check. We desperately need a tool." - P8

P7 instead focused on the identification of fault types:

"For our tasks, like animal image generation, we need a tool to precisely identify the type of fault, such as whether it's an issue with the ears, color, or the number of legs in the generated images. Identifying the specific defect for each test case would significantly enhance our ability to find the root cause and debug the model."

While there are already some debugging tools for DL systems documented in the literature, the industry has been hesitant to adopt them in practice. Many developers express concerns about their effectiveness in real-world industrial scenarios. P7 raised her concerns that using these tools might introduce new problems, such as overfitting the current training set, which could be counterproductive. That is, developing tools is one thing, and how to gain trust from developers in using these tools is another story.

Support for improving the usability of DL testing tools. Some developers believe that the existing DL testing tools suffer from low usability, which is mainly reflected in the following two aspects:

First, these tools have a steep learning curve. Many testing approaches rely on white-box information to improve testing effectiveness. Unlike traditional software, whose white-box information is mainly source code, the white-box information of DL systems is usually tensors with complicated semantics. Therefore, enhancing the visualization and interpretability of this information can improve the ease of use of the tools, as suggested by P21 who worked in a cross-disciplinary field of medical care and AI:

"It is very difficult for me and my colleagues to understand the specific meaning of each tensor, and the learning curve for using these tools is really steep. I would appreciate it if they (tool developers) could show me what these tensors mean."

Second, the lack of comprehensive documentation and logs undermines the usability of these tools. Many developers of testing tools and DL systems do not pay attention to providing documentation and logs, which increases the learning cost for end users and testers:

"The quality of existing tool documentation varies greatly, and it is difficult to configure tools based on these unclear documents and apply them to our models." - P4

Support for improving testing effectiveness. The developers also emphasize that they need more effective approaches to test DL systems during the evolution process. Some developers believe that there is a need for more efficient testing methods to detect regression bugs.

"If, after model evolution, we can effectively determine whether critical functionalities have introduced regression faults, it would enhance our testing effectiveness, although I'm not sure if it's easy to achieve that." - P14

While this is indeed a challenging problem, new approaches like LLM-based test case generation techniques might help to reveal fault like regression faults in a shorter time period. In reality, DL systems can be quite complex, and some tasks might require not just one but multiple models to complete. Relying solely on end-to-end black-box testing is ineffective. In this case, unit tests for individual models would come in handy:

"Currently, many tasks rely on a series of models working in concert. As such, we may not only need end-to-end testing but also unit tests for each model to guarantee the testing effectiveness." - P10

Another issue is that the amount of test cases is huge, and there is not enough time to run all the tests. Therefore, they would expect approaches for test case selection:

"In our ad recommendation task, we have a large number of test cases, reaching billions or even trillions. Running tests for our DL system might be quicker than for traditional software, but with all these test cases, it still adds up to a lot of time. Therefore, we need to either select a subset of test cases to run or find some way to estimate model performance with fewer predictions." - P15

Moreover, existing metrics, testing approaches, and benchmarks are scattered across multiple platforms. Despite efforts by some platforms to integrate these resources, they struggle to keep them updated in a timely manner. P4 envisions a more integrated LLM testing platform:

"All metrics and testing methods are proposed by different institutions and papers. We expect an LLM testing platform or tool that can integrate multiple benchmarks and metrics."

Support for facilitating data labeling. According to our interviews, a significant amount of time was dedicated to collecting and labeling test cases. P13 emphasizes the need for automated labeling tool:

"We need more automated or assisted labeling tools to support domain experts, especially when dealing with niche data domains. This would enhance labeling efficiency and ease the burden on experts."

Other developers have different opinions on how to handle to time-consuming labeling process. For example, P3 envisions a dedicated labeling team to help with this task:

"Enterprises should try to establish a labeling team to continuously provide data during the evolution process, if possible because high-quality data is very important for DL training and testing."

However, P20 focuses more on a data sharing platform:

'All departments involved in model training should share their high-quality data on a public platform within the enterprise to build a general dataset so that all other departments to select the data they need for testing from it."

Furthermore, P17 stresses the necessity of better strategies for prioritizing the test case labeling:

"If our goal is to detect more bugs in DL systems, we may need test prioritization and selection strategies. By labeling the test cases which are more likely to trigger faulty behaviors, such as those triggering regression faults, being closely related to critical functions, or capable of inducing specific defect types, we can more efficiently allocate resources and improve testing efficiency."

In practice, not all labeled data possess high quality. Five developers expressed their concerns about the quality of test cases, believing that it would directly affect the accurate evaluation of model effectiveness. P17 expects a tool to eliminate labeling errors:

"Even with manual annotation of data, it is easy to introduce labeling errors. Therefore, if there were tools to assist us in eliminating such errors, including incorrect labeling and bias in test data, it would be extremely beneficial."

P8 further elaborates on a potential problem in the current annotation process and longs for a better labeling process:

"The current annotation process often relies on tools to provide suggested labels first, and then the annotators check and provide final annotations. However, some of these tools have design issues that can trick annotators into making specific types of errors. Therefore, annotators should be cautious when using these tools and should not rely solely on their suggestions. At the same time, we urgently need to establish a better process for evaluating the correctness of annotations, and develop tools that can effectively detect errors in annotated data, to ensure the quality of test data."

Support for code and data management. 10 developers mentioned that they need tools for code, data, and test management for DL systems. Some developers advocate that the development of DL systems should get inspiration from traditional software development, such as introducing continuous integration and deployment (CI/CD) to document each version of the model, helping effectively prevent regression faults:

"In DL systems, updates are not limited to code but also include data sets and test sets. So, every time we make a change, we should conduct regression testing through the CI/CD process. This way, we can be sure that our new version of the DL model is regression-free." - P20

Similarly, some other developers have voiced the need for better data version control. They emphasize that one of the key differences between DL systems and traditional software systems is that they not only involve the management of code versions but also involve the management of data versions. Therefore, strict management and backup strategies need to be implemented for both code and data to ensure the ability to reproduce models:

"We not only need to track the history of code changes to ensure that we can roll back to previous versions or merge changes from different developers, but we also need to pay special attention to the recording of data versions, including data cleaning methods, data labels, and test cases, to maintain the accuracy, integrity, and consistency of data." - P2

V. Discussion

A. Findings Comparison with Exiting Work

In Section II, we introduced related DL Testing work. They primarily focus on academic issues (e.g., generating tests to maximize coverage), paying less attention to the practical needs of industry. Moreover, they often overlook the critical context of testing evolving DL systems. For instance, Marjin et al. [41] mentions testing challenges, i.e., large input space, oracle absence, and high white-box testing efforts, through literature reviewing. In contrast, our study, from a practitioner perspective, reveals that developers primarily face regression faults, online-offline differences, and the cost of continuously collecting reliable tests. These findings differ significantly from those reported in existing academic work. Specifically, Amershi et al. [17] primarily concludes with one solution about automatically creating test sets for DL systems. However, our solutions offer a more detailed perspective on testing, encompassing five categories, including strategies for mitigating regression faults. Compared with existing works, we are the first to give a comprehensive overview of DL Testing covering challenges, solutions, and support.

B. Differences Between Testing DL and non-DL Software

DL systems evolve differently from non-DL software, involving more changes at feature/data/workflow levels rather than at the code level. They are data-driven and hard to interpret. Specifically: Regarding **challenges**, DL systems require more test data. The difficulty in collecting reliable test data differs from that of non-DL software. Since DL systems are hard to interpret, exploring effective ways to understand black-box decision-making to facilitate testing is challenging. Regarding **solutions**, developers acknowledge the difficulty in avoiding all regression faults. Consequently, they set acceptable thresholds based on the importance of requirements to protect important functionalities. Additionally, they use LLM to assist labeling to reduce labeling costs, which is uncommon in non-DL software. Regarding **support**, DL faults and their root causes differ from those in non-DL software. Therefore, developers need more specific testing and debugging approaches for DL systems. Additionally, to collect enough tests, developers require better teams, tools, and working processes for labeling, whereas non-DL software often does not have such concerns.

C. Implications for Researchers

The DL testing and debugging approaches are not generally used in industry. The reluctance of developers to adopt DL testing approaches from academia primarily stems from their perception of these methods as immature.

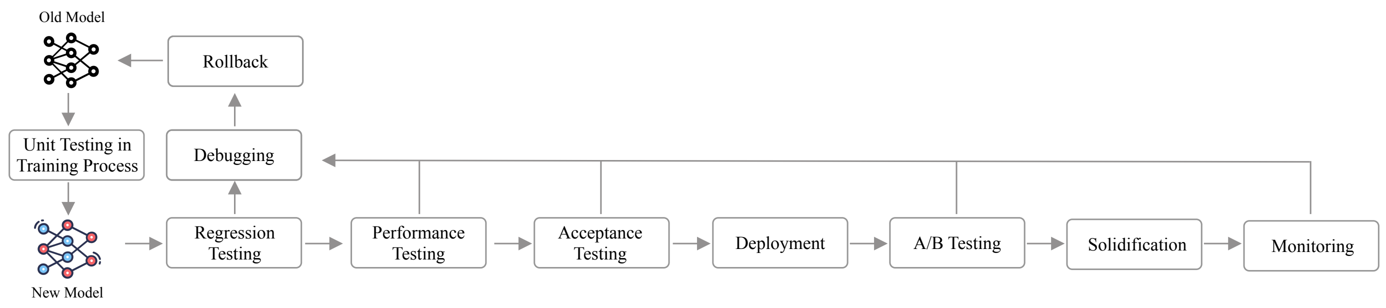

Fig. 2: The Recommended Testing Framework of Practice

There are a few concerns: First, there are hurdles in utilizing the tools themselves due to various reasons. For example, these tools lack comprehensive documentation (P4) and proper logging (P11). Additionally, the tools are not kept up-to-date or seamlessly integrated into existing platforms (P4), further hindering their adoption. Second, developers are uncertain about the effectiveness of these academic approaches in real-world scenarios. This stems from the fact that researchers often do not have the resources to conduct evaluations in industry, leading to a gap in demonstrating their applicability and impact in real-world scenarios. Consequently, we urge researchers to consider conducting comprehensive comparisons with industry tools in more realistic settings if possible. Separately, another area of concern revolves around debugging tools, mentioned by 63% of participants. These developers refrain from using available tools for two primary reasons: first, they are unsure of their effectiveness (P7), and second, they fear that these tools may cause unpredictable side effects, e.g., overfitting offline test sets or generated data and hurting other properties, limiting their usefulness in fixing practical faults (P12).

Opportunities in regression testing for DL systems. Regression faults are often overlooked in the development of DL systems. As the field evolves at an ever-increasing pace, it is particularly important to reduce the occurrence of regression faults during DL system evolution. Currently, regression faults can be roughly divided into two categories: erroneous predictions on some individual test cases and performance degradation on certain functions. Regression faults on individual test cases are difficult to avoid, and in key areas such as healthcare, politics, law, and security, it is particularly critical to avoid regression faults on individual test cases, as these errors may lead to serious consequences (P21 and P8). In industrial practice, performance degradation is more common, and developers need to ensure that core functions are not affected after evolution. For example, in the field of autonomous driving, the accuracy of vehicle recognition is more important than that of tree recognition (P22), and the accuracy of common scenarios is more important than rare scenarios (P1). How to achieve a balance between different functions remains a major challenge in the industry.

With the development of DL systems, from simple small models to complex LLM, the evolution speed and types have shown a significant growing trend. While data evolution (P3), model structure evolution (P2), and feature evolution (P21) are common evolution types traditionally, more complex evolution scenarios have emerged. For example, there are innovative workflows in the field of LLM such as Retrieval-Augmented Generation (RAG) [55] and Low-Rank Adaptation of Large Language Models (LoRA) [56] (P16). Introducing new agents (e.g., agents for optimizing prompts for follow-up prediction processes) in the workflow to further improve model performance is becoming increasingly popular (P18). Due to different evolution methods, the causes of regression faults become more diverse, e.g., the faults caused by model structure evolution and data evolution might be entirely different. It is essential to understand these root causes to improve the effectiveness of testing, fault localization, and model fixing.

In short, such frequent and complex evolution has brought new opportunities for researchers. Future research could focus on: 1) how to ensure that important functions and individual test cases are not harmed during evolution; 2) how to effectively disentangle different components of DL systems and quickly diagnose the issues behind regression faults, and 3) how to identify evolution-relevant test cases among a large number of test cases for improving testing efficiency.

Opportunities in augmenting interpretation with DL testing. The black-box nature of DL often forces developers to rely heavily on end-to-end black-box testing due to the difficulty in comprehending the semantics of complex tensors. Despite the existence of interpretation methods (e.g., Grad-CAM [57]) and tools (e.g., CNN Explainer [58]), there are limited efforts to use them to facilitate testing. One key reason is the challenge of effectively using these tools to improve test effectiveness, as the impact of interpretability metrics on test adequacy and error detection capabilities has not been well-studied. Using interpretability metrics can help practitioners better understand the changes in the decision-making process of DL systems and derive more targeted and effective testing strategies. Moreover, we need to lower the bar for using such tools as much as possible. Providing users with exhaustive documentation, error logs, and practical usage examples can significantly enhance the user-friendliness of these tools and increase their accessibility to a broader range of developers.

D. Key Takeaways for Industry Practitioners

The testing framework for evolving DL systems. To enhance testing effectiveness, we present a practical testing framework (Fig. 2) integrating the insights from the interviews.

First, during the training phase, developers should consider conducting unit testing, including comparing the model with its previous version regarding the performance on specific epochs, to ensure the training effectiveness (P8, P15). Upon successful training, regression testing should be conducted to identify any potential regression faults, followed by performance testing (P2), acceptance testing (P19), and essential offline testing to effectively guarantee the model's quality. After deployment, developers should embark on online testing, specifically A/B testing (P15), to assess the model's real-world performance. If the model passes the A/B test, it can be safely updated. However, ongoing monitoring of online performance (P11) and user feedback (P3) remains crucial. If the DL model fails any testing phase, the failures should be reported to the developers for debugging (P8), and the system should roll back to its most recent stable version (P18) to maintain operational integrity. This framework is in its early stages, requiring further studies to enrich and verify it, ultimately guiding future recommendations for practitioners. Our study lays a crucial foundation for such endeavors.

Different kinds of regression testing frameworks. In traditional software, regression testing tends to reuse test cases. However, in DL, test cases should be updated due to differences between online deployment and offline training. Generally, the frequency of test case updates varies depending on the difficulty of collecting and labeling new test cases. For example, in the medical field, it is difficult to collect test cases, so there is little updating of test cases (P21); in recommendation systems, due to the huge amount of available data, the developers basically always use the latest test cases and even need sampling to complete the entire testing (P15). For most common tasks, it is necessary to gradually accumulate new test cases (P16) or replace existing test cases with new ones (P6). Industrial developers can choose different regression testing case update strategies, including deciding how to discard outdated test cases and continue to label test cases in regression scenarios, based on business requirements.

VI. Threats to Validity

Threats to construct validity concern the relationship between theory and observation. The years of experience could impact how developers view the challenges and the strategies they would propose. We have invited participants with a wide range of experience. However, we acknowledge that potential bias might be introduced given the very different backgrounds of the participants in the interviews. Another threat is related to the design of interview questions, namely if the questions could guide developers to provide meaningful insights. To ensure that the responses could answer our research questions, we conducted a pilot study with three participants and refined the interview questions based on their responses.

Threats to internal validity concerns the selection bias in this study and the subjectiveness of the authors during the interview and coding processes. For the prior issue, due to privacy concerns, not all participants accepted our interview invitations. Therefore, we might miss out on some insights directly impacting the interests of the companies. This is inevitable in such studies. For the latter issue, there is a possibility that the authors might misinterpret the responses of participants. To mitigate this issue, two people independently coded the transcripts and discussed the conflicts until a consensus was reached. To avoid imposing our opinions on developers during the interviews, we thoroughly explained the goal of our study and asked for clarifications when we were not entirely sure what the participants meant to better understand the perspectives of the participants.

Threats to external validity concern the generalizability of our findings. We recruited 22 DL developers, which might not represent the whole DL software community. However, these developers are from companies of different sizes and sectors in various countries, and they work on different tasks in different roles. This to a certain extent mitigates the threats to generalizability. We acknowledge that our participants are predominantly from medium to large companies, with a smaller representation from small companies. This is partly due to difficulties in reaching out experts who are both experienced in DL system evolution and in DL system testing. It is worth noting that not all companies have specific testers for DL systems, especially small companies. However, we believe that the numerous challenges presented in this paper are also frequently encountered by practitioners in small companies. Solutions presented, along with real-world examples from other practitioners, may serve as a learning resource and inspire innovative solutions among them.

VII. Conclusion

Testing evolving DL systems has been challenging in practice. To provide insights on how developers can effectively test their DL systems along with system evolution, we conducted a semi-structured interview with 22 DL developers to understand the challenges they face, propose practical solutions to these challenges, and reveal future research and practical opportunities for better testing support.

Our results, accompanied by real industry examples, can serve as a reference for developers when testing evolving DL systems. Our future work will focus on converting the insights gained from interviews to concrete approaches and tools that could benefit developers in real-world industrial scenarios.

Acknowledgment

We thank all the interview participants and are grateful for the suggestions from the reviewers. We specifically thank Zhenfei Huang for his assistance in managing the participants, transcribing the recordings, and checking the codebook. We also thank Yushan Xiao, Qiyu Li, Shunda Pei, Zuowei Cui, and Sisi Li for helping recruit interview participants. This work would not have been possible without their invaluable support.

This work is partially supported by the National Natural Science Foundation of China (Grant Nos. 62472310, 62322208, 62232001, 12411530122).

References

[1] X. Zhu, H. Wang, H. You, W. Zhang, Y. Zhang, S. Liu, J. Chen, Z. Wang, and K. Li, "Survey on testing of intelligent systems in autonomous vehicles," *Journal of Software*, vol. 32, no. 7, pp. 2056–2077, 2021.

[2] R. Tozuka, N. Kadoya, S. Tomori, Y. Kimura, T. Kajikawa, Y. Sugai, Y. Xiao, and K. Jingu, "Improvement of deep learning prediction model in patient-specific qa for vmat with mlc leaf position map and patient's dose distribution," *Journal of Applied Clinical Medical Physics*, p. e14055, 2023.

[3] X. Chen, X. Hu, Y. Huang, H. Jiang, W. Ji, Y. Jiang, Y. Jiang, B. Liu, H. Liu, X. Li *et al.*, "Deep learning-based software engineering: Progress, challenges, and opportunities," *arXiv preprint arXiv:2410.13110*, 2024.

[4] Y. Kang, Z. Wang, H. Zhang, J. Chen, and H. You, "Apirecx: Cross-library API recommendation via pre-trained language model," in *EMNLP (1)*. Association for Computational Linguistics, 2021, pp. 3425–3436.

[5] Z. Huang, J. Chen, J. Jiang, Y. Liang, H. You, and F. Li, "Mapping apis in dynamic-typed programs by leveraging transfer learning," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 4, pp. 102:1–102:29, 2024.

[6] C. Gartenberg, "Apple's face id with a mask works so well, it might end password purgatory," Accessed: 2024. [Online]. Available: https://www.theverge.com/2022/2/2/22912677/apple-face-id-mask-update-ios-15-4-beta-hands-on-impressions

[7] T. Gerken, "Dpd error caused chatbot to swear at customer," Accessed: 2024. [Online]. Available: https://www.bbc.com/news/technology-68025677

[8] News, "Gpt-4 is getting worse and worse every single update," Accessed: 2024. [Online]. Available: https://community.openai.com/t/gpt-4-is-getting-worse-and-worse-every-single-update/

[9] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *ASE*. ACM, 2018, pp. 132–142.

[10] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *ISSTA*. ACM, 2019, pp. 146–157.

[11] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *ICSE*. IEEE, 2021, pp. 397–409.

[12] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepmutation: Mutation testing of deep learning systems," in *ISSRE*. IEEE Computer Society, 2018, pp. 100–111.

[13] M. Ojdanic, E. O. Soremekun, R. Degiovanni, M. Papadakis, and Y. L. Traon, "Mutation testing in evolving systems: Studying the relevance of mutants to code evolution," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 1, pp. 14:1–14:39, 2023.

[14] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN Comput. Sci.*, vol. 2, no. 6, p. 420, 2021.

[15] C. Hill, R. K. E. Bellamy, T. Erickson, and M. M. Burnett, "Trials and tribulations of developers of intelligent systems: A field study," in *VL/HCC*. IEEE Computer Society, 2016, pp. 162–170.

[16] X. Zhang, Y. Yang, Y. Feng, and Z. Chen, "Software engineering practice in the development of deep learning applications," *CoRR*, vol. abs/1910.03156, 2019.

[17] S. Amershi, A. Begel, C. Bird, R. DeLine, H. C. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: a case study," in *ICSE (SEIP)*. IEEE / ACM, 2019, pp. 291–300.

[18] T. Zhang, C. Gao, L. Ma, M. R. Lyu, and M. Kim, "An empirical study of common challenges in developing deep learning applications," in *ISSRE*. IEEE, 2019, pp. 104–115.

[19] E. Eijkelenboom, "Trends in software evolution," Ph.D. dissertation, Citeseer, 2005.

[20] H. You, Z. Wang, J. Chen, S. Liu, and S. Li, "Regression fuzzing for deep learning systems," in *ICSE*. IEEE, 2023, pp. 82–94.

[21] S. J. Chen, Z. Qin, Z. Wilson, B. Calaci, M. Rose, R. Evans, S. Abraham, D. Metzler, S. Tata, and M. Colagrosso, "Improving recommendation quality in google drive," in *KDD*. ACM, 2020, pp. 2900–2908.

[22] Z. Li, M. Zhang, J. Xu, Y. Yao, C. Cao, T. Chen, X. Ma, and J. Lu, "Lightweight approaches to DNN regression error reduction: An uncertainty alignment perspective," in *ICSE*. IEEE, 2023, pp. 1187–1199.

[23] J. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *IEEE International Conference on Computer Vision, ICCV 2017*. IEEE Computer Society, 2017, pp. 5068–5076.

[24] M. Dilhara, D. Dig, and A. Ketkar, "PYEVOLVE: automating frequent code changes in python ML systems," in *ICSE*. IEEE, 2023, pp. 995–1007.

[25] J. Jiang, J. Yang, Y. Zhang, Z. Wang, H. You, and J. Chen, "A post-training framework for improving the performance of deep learning models via model transformation," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 3, pp. 61:1–61:41, 2024.

[26] H. Zhang and W. K. Chan, "Apricot: A weight-adaptation approach to fixing deep learning models," in *ASE*. IEEE, 2019, pp. 376–387.

[27] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Trans. Software Eng.*, vol. 48, no. 2, pp. 1–36, 2022.

[28] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *ICSE*. ACM, 2020, pp. 1110–1121.

[29] M. Wardat, W. Le, and H. Rajan, "Deeplocalize: Fault localization for deep neural networks," in *ICSE*. IEEE, 2021, pp. 251–262.

[30] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *SOSP*. ACM, 2017, pp. 1–18.

[31] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepgauge: multi-granularity testing criteria for deep learning systems," in *ASE*. ACM, 2018, pp. 120–131.

[32] M. Yan, J. Chen, X. Zhang, L. Tan, G. Wang, and Z. Wang, "Exposing numerical bugs in deep learning via gradient back-propagation," in *ESEC/SIGSOFT FSE*. ACM, 2021, pp. 627–638.

[33] X. Zhang, N. Sun, C. Fang, J. Liu, J. Liu, D. Chai, J. Wang, and Z. Chen, "Predoo: precision testing of deep learning operators," in *ISSTA*. ACM, 2021, pp. 400–412.

[34] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Deepconcolic: testing and debugging deep neural networks," in *ICSE (Companion Volume)*. IEEE / ACM, 2019, pp. 111–114.

[35] J. Wang, J. Chen, Y. Sun, X. Ma, D. Wang, J. Sun, and P. Cheng, "Robot: Robustness-oriented testing for deep learning systems," in *ICSE*. IEEE, 2021, pp. 300–311.

[36] N. Humbatova, G. Jahangirova, and P. Tonella, "Deepcrime: mutation testing of deep learning systems based on real faults," in *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021*. ACM, 2021, pp. 67–78.

[37] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, "Deepct: Tomographic combinatorial testing for deep learning systems," in *SANER*. IEEE, 2019, pp. 614–618.

[38] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: automated testing of deep-neural-network-driven autonomous cars," in *ICSE*. ACM, 2018, pp. 303–314.

[39] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empir. Softw. Eng.*, vol. 25, no. 6, pp. 5193–5254, 2020.

[40] F. Tambon, G. Laberge, L. An, A. Nikanjam, P. S. N. Mindom, Y. Pequignot, F. Khomh, G. Antoniol, E. Merlo, and F. Laviolette, "How to certify machine learning based safety-critical systems? A systematic literature review," *Autom. Softw. Eng.*, vol. 29, no. 2, p. 38, 2022.

[41] D. Marijan, A. Gotlieb, and M. K. Ahuja, "Challenges of testing machine learning based systems," in *AITest*. IEEE, 2019, pp. 101–102.

[42] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *IEEE METRICS*. IEEE Computer Society, 2005, p. 23.

[43] S. van Breukelen, A. Barcomb, S. Baltes, and A. Serebrenik, ""still around": Experiences and survival strategies of veteran women software developers," in *ICSE*. IEEE, 2023, pp. 1148–1160.

[44] G. T. Henry, *Practical sampling*. Sage, 1990, vol. 21.

[45] H. You, "Github homepage," Accessed: 2024. [Online]. Available: https://github.com/youhanmo/DLTestInterview

[46] ——, "Zenodo homepage," Accessed: 2024. [Online]. Available: https://doi.org/10.5281/zenodo.14197934

[47] O. Elazhary, C. M. Werner, Z. S. Li, D. Lowlind, N. A. Ernst, and M. D. Storey, "Uncovering the benefits and challenges of continuous integration practices," *IEEE Trans. Software Eng.*, vol. 48, no. 7, pp. 2570–2583, 2022.

[48] M. Greiler, M. D. Storey, and A. Noda, “An actionable framework for understanding and improving developer experience,” *IEEE Trans. Software Eng.*, vol. 49, no. 4, pp. 1411–1425, 2023.

[49] J. Saldaña, “The coding manual for qualitative researchers,” 2021.

[50] /, “Atlas.ti,” Accessed: 2024. [Online]. Available: https://atlasti.com/

[51] D. R. Thomas, “A general inductive approach for qualitative data analysis,” 2003.

[52] M. Williams and T. Moser, “The art of coding and thematic exploration in qualitative research,” *International management review*, vol. 15, no. 1, pp. 45–55, 2019.

[53] C. B. Seaman, “Qualitative methods in empirical studies of software engineering,” *IEEE Trans. Software Eng.*, vol. 25, no. 4, pp. 557–572, 1999.

[54] G. Guest, A. Bunce, and L. Johnson, “How many interviews are enough? an experiment with data saturation and variability,” *Field methods*, vol. 18, no. 1, pp. 59–82, 2006.

[55] P. S. H. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in *NeurIPS*, 2020.

[56] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” in *ICLR*. OpenReview.net, 2022.

[57] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” in *ICCV*. IEEE Computer Society, 2017, pp. 618–626.

[58] Z. J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman, M. Kahng, and D. H. P. Chau, “CNN explainer: Learning convolutional neural networks with interactive visualization,” *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 1396–1406, 2021.