Reusing Deep Learning Models: Challenges and Directions in Software Engineering

James C. Davis*, Purvish Jajal*, Wenxin Jiang*,
Taylor R. Schorlemmer*, Nicholas Synovic†, and George K. Thiruvathukal†
*Department of Electrical & Computer Engineering, Purdue University, West Lafayette, IN, USA
{davisjam,pjajal,jiang784,tschorle}@purdue.edu
†Department of Computer Science, Loyola University Chicago, Chicago, IL, USA
{nsynovic,gthiruvathukal}@luc.edu

***Abstract*—Deep neural networks (DNNs) achieve state-of-the-art performance in many areas, including computer vision, system configuration, and question-answering. However, DNNs are expensive to develop, both in intellectual effort (*e.g.,* devising new architectures) and computational costs (*e.g.,* training). Re-using DNNs is a promising direction to amortize costs within a company and across the computing industry. As with any new technology, however, there are many challenges in re-using DNNs. These challenges include both missing technical capabilities and missing engineering practices.**

This vision paper describes challenges in current approaches to DNN re-use. We summarize studies of re-use failures across the spectrum of re-use techniques, including conceptual (*e.g.,* re-using based on a research paper), adaptation (*e.g.,* re-using by building on an existing implementation), and deployment (*e.g.,* direct re-use on a new device). We outline possible advances that would improve each kind of re-use.

***Index Terms*—Machine learning, Deep learning, Pre-trained models, Re-use, Empirical software engineering, Position, Vision**

I. Introduction

The long "AI Winter" [1] is over. Machine learning, especially the deep learning paradigm, matches or exceeds human performance in diverse tasks [2], [3]. These tasks include signal processing (*e.g.,* sight [4], [5], hearing [6], smell [7]), general reasoning (*e.g.,* question-answering [8], synthesizing information), and artistic expression (*e.g.,* Dall-E [9], music-making [10]). Although we expect further scientific breakthroughs in performance, many techniques are mature and already being applied to domains such as marketing [11], medicine [12], and autonomous vehicles [13]. Machine learning has entered the domain of the software engineer.

One key software engineering technique is *reuse* [14]. To reduce engineering costs, software engineers recognize opportunities to reuse, and they also determine an appropriate reuse paradigm. Software reuse has been exhaustively studied in traditional software engineering, both conceptually (*e.g.,* [15], [16]) and empirically (*e.g.,* [17], [18], [19]). However, for deep learning software the study of reuse has just begun.

All authors are listed in alphabetical order as equal contributors to this manuscript. Please direct correspondence to davisjam@purdue.edu and gthiruvathukal@luc.edu.

Deep learning (DL) is an emerging field whose probabilistic nature and data-driven approach differs from traditional software [20]. The engineering community lacks long-term experience in appropriate engineering methods for deep learning [21]. Prior work has characterized the challenges of *developing* DL software [22], [23]. Yet relatively little is known about DL *reuse* and the specific challenges engineers face when trying to reuse these models.

In this vision paper, we examine challenges and future directions for the reuse of deep neural networks (DNNs). Figure 1 depicts the three paradigms of DNN reuse that we consider: conceptual reuse, adaptation reuse, and deployment reuse. *Conceptual reuse* is the replication of DNN techniques from sources like academic literature. *Adaptation reuse* is the modification of pre-trained DNNs to work in a particular use case. *Deployment reuse* is the conversion of pre-trained DNNs into forms which operate in different environments.

The paper proceeds as follows. In §II, we discuss the general landscape of work on DNNs. We define the different types of reuse. In §III, we go into detail about each type of reuse, focusing on the general nature of the challenge and prior work that clarifies it. In §IV, we focus on open problems worthy of community attention. The scope of this work is to introduce the current conceptual, adaptation, and deployment challenges facing DNN reuse, with a discussion of future directions to address these issues. We do not aim to address the (many!) other software engineering challenges related to DNNs, such as when to use them, for what purposes, how to integrate them into existing software systems, and how to debug them.

We hope the community's continued efforts to understand the software engineering implications of working with DNNs will lead to standardization of software engineering practices and the development of tools, both of which were greatly helpful to the advancement of traditional software development.

II. Background

A. Engineering Deep Neural Networks

1) DNN Concepts: Software that implements a DNN has three main components: the DNN model itself (a parameterized computational graph); a data pipeline (preprocessing to manipulate data into an appropriate format); and the training

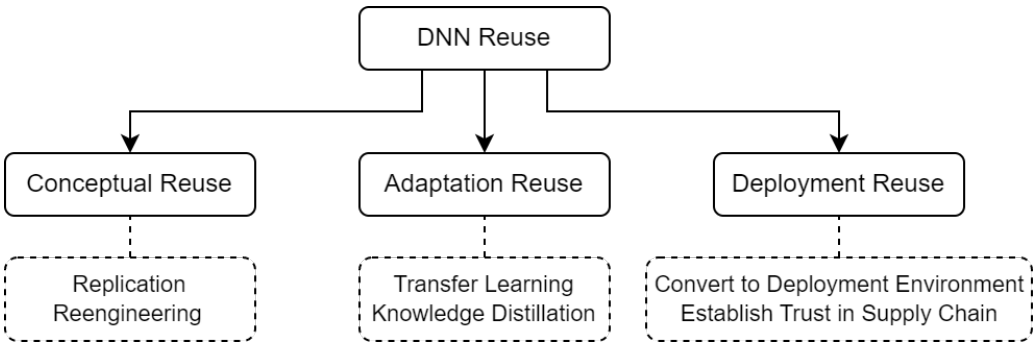

Fig. 1: Deep neural network reuse is the process of using existing DNN technology for another purpose. We focus on three distinct types: *conceptual reuse*, where existing theory is repurposed; *adaptation reuse*, where existing DNN models are modified; and *deployment reuse* where existing DNN models are converted for use in a new environment. Dashed boxes provide examples of each type.

mechanisms [24]. Due to the computational costs of training and inference, DNN software is also optimized. We discuss background for each.

a) The DNN Model: A DNN model is a parameterized computational graph [21], *i.e.,* the composition of weighted operations [25]. Layers can be grouped into blocks based on their purpose or frequency of occurrence. For example, Long Short-Term Memory (LSTM) units are commonly used blocks in recurrent neural networks, and they are particularly designed to handle time-series data [26]. Each layer and block can be designed and tested individually [27], [28], and then integrated into one structure prior to training. These model sub-elements are shown in Figure 2. The layer sizes must be noted to maintain the shape of the model and ensure that adjustments are made so that the output of one element is the same size as the input of the next.

The architecture of a DNN model will vary based on its purpose. As an example, in the computer vision domain, typical models have three main components: feature extractor (backbone), decoder [1], and head [29]. The backbone refers to the combination of layers responsible for feature extraction. The backbone is usually a large number of sequential layers and the majority of the model training process is to adjust the backbone parameters. The decoder, which usually has less layers than the backbone, handles post-processing and serves as a connection between the backbone and the head. The head, often the smallest element, is responsible for the final model task (ex: image classification). Depending on the model architecture, the components included within the backbone and decoder may vary.

b) The Data Pipeline: A data pipeline extracts raw data such as pictures from a storage source (*decoder*), transforms the data into the required input format (*parser*), and loads the transformed data into the GPU (*loader*). The Extract-Transform-Load (ETL) architecture [30] is a common pattern for data pipelines as it simplifies implementation. It also promotes reuse: different models can share datasets (Extract), vary in their data augmentations (Transform), and have similar final stages (Load). An example data pipeline is depicted in Figure 3.

The data pipeline may also be modified as part of training. Some changes are for robustness, while others are for performance [32]. For robustness, *data augmentations* can be added to ensure that a model can perform well not only on well formed input but also on more unusual input [33]. For example, in computer vision, images may be rotated, their colors may be changed, and "filters" may be added such as fog [34]. For performance, the data processing may be optimized to avoid bottlenecks during training. If a dataset fits in the available system memory then it can be cached. Else, the dataset must be processed in batches.

c) The Training Mechanisms: The final aspect of a DNN is its training — the identification of parameters for the DNN's operations ("weights") that yield acceptable functionality [21], [35]. When training, engineers must provide a loss function. A loss function defines the learning curve of the DNN and is used to adjust the network weights (*e.g.,* via inspecting the loss gradient over different parameter changes). Once the weights are finalized by minimizing the loss function, the loss function itself is no longer required.

d) Optimizations: Based on the device being used for training, the model training time and efficiency will vary drastically [36], [37]. Certain measures can be taken to increase the training efficiency on average. For example, the data pipeline operations have a strong effect on the training time. Thus, the operations used during the data pipeline stage must be efficient and based on built-in operations whenever possible (*e.g.,* vectorized mathematics rather than `for`-loops). Training hyper-parameters such as the number of epochs may be adjusted depending on the training device, (*e.g.,* whether it is a CPU, GPU or TPU).

However, optimizing the data pipeline may violate the modular design of the data pipeline. Breaking module boundaries is a common effect of performance optimization [38]. For exam-

[1]There exists some terminological confusion within the literature on "decoder". A *model decoder* refers to the post-processing component of the model. The *data pipeline decoder* transforms data into a format that is directly usable by downstream operations.

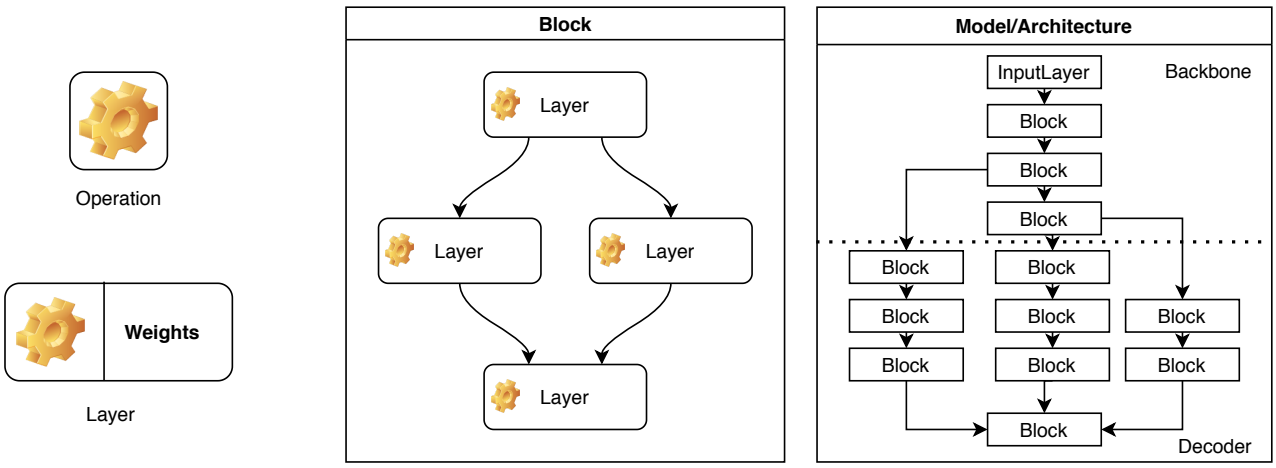

Fig. 2: Components of a deep neural network (DNN), represented at different levels of abstraction. A DNN is a composition of weighted operations. These are combined into a *layer*; a group of layers into a *block*; and a group of blocks into a sub-graph such as a "backbone" or a "head".

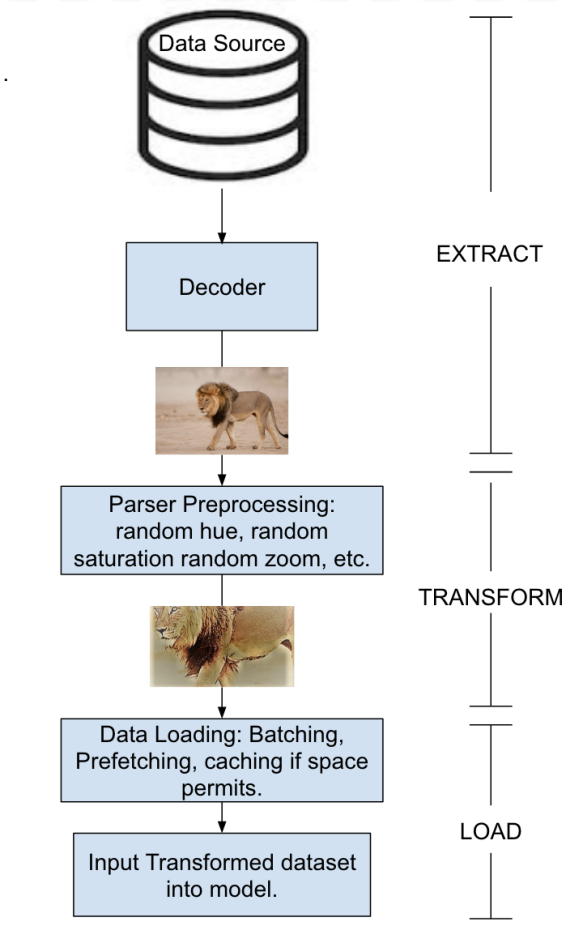

Fig. 3: Illustration of a data pipeline following the Extract-Transform-Load design pattern. The specific pipeline is for the You-Only-Look-Once (YOLO) model family [31].

ple, one common data augmentation applied while training a computer vision model is image blurring. Image blurring uses a convolution operation which is slow on a CPU but efficient on a GPU. Transitioning to a GPU only for one operation and then back to the CPU would be time-consuming, but this issue can be tackled by placing the blurring operation at the end of the data pipeline. This is because the blur operation would immediately be followed up by the DNN training process on the same device. Thus, the order of operations can influence the model's training time, and they may be re-arranged in an implementation for efficiency.

2) Distributing a DNN Model for re-use: DNN software is sometimes shared, either open-source or as binaries. The model may be shared alone, or with weights after training. A DNN model with both architecture and pre-trained weights is called a *Pre-Trained Model (PTM)* [39]. When a PTM is shared for re-use in an organized manner, it is called a *pre-trained model package* [40]. The package may include the PTM as well as the names of the authors, the bill of materials (SBOM), license information, versioning, test suite, and so on.

3) Engineering "Best Practices": Major technology companies have shared different kinds of studies on machine learning (ML) best practice (*e.g.,* Google [41], Microsoft [22], and SAP [23]). Google [41] and Microsoft [22] provide high level guidelines on the current state of the ML models and potential improvements. Breck *et al.* from Google present 28 quantified tests as well as monitoring needs and propose an easy-to-follow rubric to improve ML production readiness and reduce technical debt [41]. Rahman *et al.* present a case study on SAP [23]. They discuss the challenges in software engineering, ML, and industry-academia collaboration and specifically point out the demand for a consolidated set of guidelines and recommendations for ML applications. Unlike the guidance on high-level architectures and organizational processes shared by Google and Microsoft, we focus on lower-level engineering and programming patterns.

In addition to views of the industry, there are also works on software engineering practice for the engineering of DL software from academic perspectives [42], [43], [44], [45]. Zhang *et al.* present 13 valuable findings in the practice of the engineering life cycle of DL applications [43]. They indicate that we are in great demand for test cases which could be used to check the correctness of the system functions. Based on their findings, they give suggestions to both practitioners and researchers including using well-known frameworks, improving the robustness of DL applications, and proposing new bug locating tools and logs. Serban *et al.* conducted a survey and quantified practice adoption and demographic characteristics, then propose a positive correlation between practices and effects [44]. However, there has been limited focus on perspective of reusability in the context of DL models. In this vision paper, we present a comprehensive summary of existing reuse paradigms, and propose challenges as well as future directions in the field of software engineering

B. Reuse Paradigms

We identify three major reuse paradigms for deep learning models. These reuse paradigms are related to those proposed by Sommerville [46] and Krueger [15] for reuse in traditional software engineering, but take somewhat different forms in deep learning engineering. Specifically, the reuse of deep learning models necessitates unique considerations including the demand for significant computational resources, the need to accommodate specific hardware configurations, and the inherent dependencies on diverse datasets.

- **Conceptual Reuse:** *Replicate and reengineer the algorithms, model architectures, or other concepts described in academic literature and similar sources, integrating the replication into new projects*. An engineer might do this because of licensing issues or if they are required to use a particular deep learning framework (*e.g.,* TensorFlow) but are reusing ideas previously realized in another deep learning framework (*e.g.,* PyTorch) [47]. This paradigm is related to Sommerville's notion of "abstraction reuse", where an engineer reuses knowledge but not code directly. This paradigm is also related to reproducibility in the scientific sense, since an engineer independently confirms the reported results of a proposed technique [48], [49].
- **Adaptation Reuse:** *Leverage existing DNN models and adapt them to solve different learning tasks*. An engineer might do this using several techniques, such as transfer learning [50] or knowledge distillation [51]. This form of reuse is suitable if there is a publicly available implementation of an appropriate model (a pre-trained model). This paradigm is related to Sommerville's notion of "object/component reuse", since an engineer must identify existing models suited for a purpose and then customize them for a different task.
- **Deployment Reuse:** *Convert and deploy pre-trained DNN models in different computational environments and frameworks*. This form of reuse is suitable if there is a perfect fit for the engineer's needs, viz. a DNN trained on the engineer's desired task (*e.g.,* demonstrating proof of concept in a hackathon). This paradigm is related to Sommerville's notion of "system reuse", since an engineer is reusing an entire model (including its training) and deploying it in the appropriate context. Deployment often requires the conversion of a DNN from one representation to another, followed by compilation to optimize it for hardware.

These reuse paradigms are orthogonal. Multiple forms of reuse are possible in a single engineering project. For example, an engineering team might develop their own version of a decision-making component (conceptual reuse); re-use the implementation of a feature extractor DNN as a backbone (adaptation reuse); and after training, convert their model to a specialized hardware environment (deployment reuse).

III. Challenges

In this section, we describe the open software engineering challenges associated with deep learning model reuse. Each subsection addresses one of the three types of reuse: conceptual, adaptation, and deployment.

A. Challenges in Conceptual Reuse of DNNs

Conceptual reuse in deep learning primarily takes two forms. The first is reproducing results using the same code and dataset. The second form is replication and model reengineering, where the same algorithm is implemented (possibly with changes) in a new context, *e.g.,* a different deep learning framework or a variation on the original task. Each of these forms presents its own set of potential challenges.

1) Reproducibility of Results: Conceptual reuse of DNN necessitates an understanding of state-of-the-art DNN architectures and algorithms. As part of this process, both engineers [52], [53] and researchers [54], [55] regard the reproduction of reported DNN results as essential for enhancing comprehension and trust.

Reproducibility is considered a key quality of machine learning software, with a particular emphasis on the thorough exploration of experimental variables and the requirement for comprehensive documentation to achieve reproducibility [56]. Despite this recognition, achieving DNN reproducibility remains a challenging task and continues to be a focal point within the research community [48], [56], [57], [58], [59].

Many state-of-the-art models are publicly available, but they often exist in the research prototype stage. This stage is typically characterized by an absence of rigorous testing, inadequate documentation, and a lack of considerations for portability. These factors contribute to the ongoing "reproducibility crisis" in the field of deep learning [41], [53].

2) Model Replication and Reengineering: Replicating and reengineering DNNs is tricky, even when referring to the original code of the research prototypes [24]. Deep learning frameworks contain many sources of variability that can limit replicability [60], [61].

Jiang *et al.* described three challenges of DNN replication and reengineering: model operationalization, portability of DL operations, and performance debugging [62]. First, the *model*

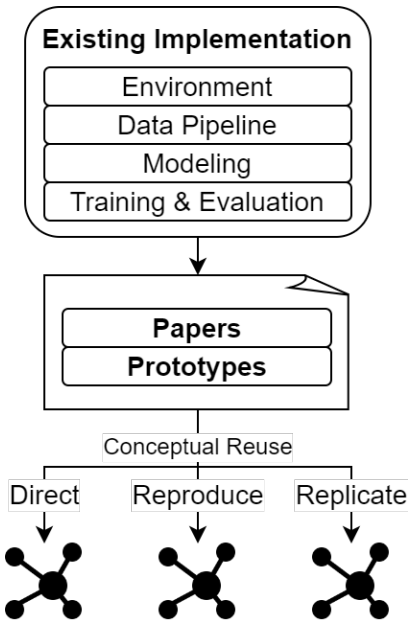

Fig. 4: Overview of the conceptual reuse of deep learning models. An engineer learns about deep learning ideas from existing implementations, *e.g.,* research papers and prototypes. They use this to guide their own development of a deep learning model, *e.g.,* implementing it directly, reproducing the model, or replicating the model.

operationalization, or the difficulty in choosing and validating the correct model for the task, can be confusing for reusers. It can be hard to distinguish between multiple implementations, to identify the implemented conceptual algorithm(s), and to evaluate their trustworthiness. Second, the *portability of deep learning operations* is inconsistent across frameworks. Some deep learning operations only support certain hardware which makes it harder to transfer the implementation to frameworks and environments. Consequently, *performance debugging* becomes necessary. This step, however, poses its own set of challenges for engineers because of the stochastic nature of deep learning systems [23].

B. Challenges in Adaptation Reuse of DNNs

Software engineers use multiple techniques to modify existing DNNs for use in their specific applications. Even though adapting a model is more efficient than training a model from scratch, engineers still face challenges with adaptation. Software engineers face both *technical* (with respect to an adaptation technique) and *decision-making* (with respect to the engineering process) challenges when adapting a DNN to a particular use case. This section explores these challenges.

a) Technical Adaptation Challenges: **Techniques** Existing deep learning models can be adapted by engineers to new tasks through various techniques, such as transfer learning, dataset labeling, and knowledge distillation [50], [39], [63], [64]. The practice of reusing DL models as pre-trained models (PTMs) reduces the need for users to train their own models,

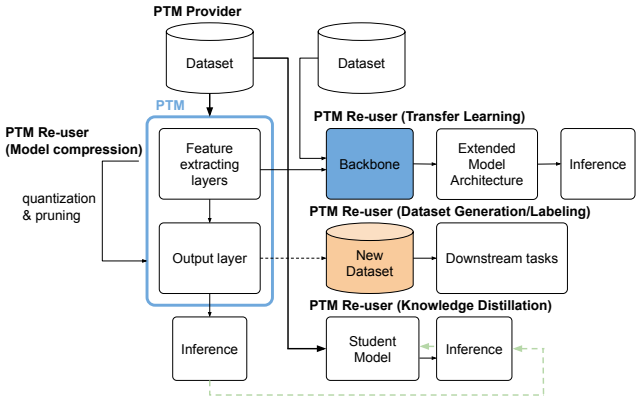

Fig. 5: A pre-trained deep neural network model (PTM) can be adapted for different tasks via transfer learning, quantization and pruning, and knowledge distillation. This figure is reused from [66] with their permission.

thereby facilitating rapid model development for an expanded range of tasks [39], [65].

Figure 5 shows multiple methods for PTM reuse. A PTM provider will train a deep learning model on a dataset, resulting in a DNN checkpoint that includes specific architecture and weight parameters. A PTM user can then employ one of the aforementioned methods to adapt this model to a new context or application. The resulting model may be more compact, easier to manage, and more resource-efficient [39], [64].

Accuracy and Latency: Engineers adapt DNNs to different hardware constraints and environments. For example, the adaptation of DNNs to embedded devices (also known as IoT or Edge devices) often demands significant engineering efforts, encompassing techniques such as model compression and hardware acceleration [67]. Engineers report that they are unaware of "push-button" techniques to adapt DNNs across hardware environments [68].

Fairness and Robustness In addition to accuracy and latency, engineers reusing Deep Neural Networks (DNNs) may also consider properties such as fairness and robustness [69], [70], [71]. A multitude of strategies has been developed to boost the fairness of DNNs, encompassing local interpretability techniques [72], [73] and model-agnostic methods [74]. Regarding robustness, recent research has shown that decomposition techniques can contribute to the resilience of DL systems [75], [76]. By enabling model modularization, these techniques provide engineers with the flexibility to substitute faulty or inefficient components during the adaptation reuse process [77]. This approach underscores the potential benefits of modular design in bolstering the robustness and maintainability of DL systems in diverse deployment scenarios. Nevertheless, enhancing fairness and reducing algorithmic bias remain considerable challenges that require further exploration and advancement [69].

b) Decision-Making Challenges: Adapting DNNs (*e.g.,* as PTMs) is also challenging because of the complicated decision-making process and its costly evaluation loops. Soft-

Aspect	Software Supply Chains	DNN Supply Chains
Actors	Software Engineers Testers Maintainers	Software Engineers Testers Maintainers
Artifacts	Source Code Software Dependencies Infrastructure	Source Code, Datasets Frameworks, Pre-trained Models Infrastructure
Operations	Development Building, Testing Deployment	Data Collection, Development Model Training, Testing Model Deployment

Table I: Comparison of Software Supply Chains and DNN Supply Chains [66] by actors, artifacts, and operations described by Okafor *et al.* [79].

ware engineers must assess if they can reuse a DNN for their task, select a model, adapt the model, evaluate the model, and then deploy the model [40]. Several challenges exist for this decision-making process — they include: model selection, discrepancies, and security and privacy risks [40].

First, selecting an appropriate DNN for reuse is a difficult process for software engineers. While model registries, such as Hugging Face, are popular in the deep learning community, they often lack infrastructure that provides attributes helpful to the reuse process. These attributes would include DNN information like model provenance, reproducibility, and portability on top of traditional software information like popularity, quality, and maintenance [40]. On the other hand, missing attributes lead to an increased engineering effort and an expensive evaluation process during model selection.

Next, potential performance discrepancies may arise from poor documentation or non-robust models. For example, Montes *et al.* [78] demonstrate that popular models can have significant accuracy and latency discrepancies in their implementations in different model hubs. Jiang *et al.* [40] also note the lack of documentation for DNNs on Hugging Face. These issues make it hard for engineers to adapt models to their tasks. Engineers must spend additional effort discovering different or undocumented behaviors for the DNNs they choose to reuse.

Lastly, attacks against DNNs pose security and privacy risks to products that rely on them. Because of the unique aspects of machine learning, new types of attacks have emerged specifically for DNNs. These attacks can target several aspects of the DNN engineering processing including datasets, model parameters, and even model behaviors. Many of these attacks take advantage of poor documentation, inadequate security features, and performance discrepancies common among open-source DNNs [78]. Traditional software supply chain threats also apply. Software supply chains contain actors, artifacts, and operations which interact to create some final software product [79]. When DNNs are reused, they create their own supply chain consisting of models, datasets, maintainers, and training processes [66] — see Table I. Actors play a similar role in both supply chain types, but DNN artifacts and operations differ because of the nature of machine learning.

We divide attacks against DNNs into four types:

1) **Train-Time Attacks:** These attacks manipulate the model's training procedure (often through a malicious dataset) to affect its behavior [80]. A prime example is BadNets [81] where a model is trained on a malicious dataset to illicit improper classifications when later used.
2) **Idle-Time Attacks:** These attacks maliciously alter existing DNNs to introduce new behavior. Examples of this type include EvilModel [82], [83] and StegoNet [84]. These examples implant malware directly into the parameters of models, hiding a malicious payload in a trained DNN without significant drops in performance. Another example includes LoneNeuron [85] which injects a malicious function into an existing DNN.
3) **Inference-Time Attacks:** These attacks exploit unexpected input-output pairs or recover confidential dataset or parameter information [80], [86]. For example, Papernot *et al.* [87] describe a method to generate adversarial inputs for DNNs; Shokri *et al.* [88] describe a method for membership inference attacks to reveal sensitive training data; and MAZE [89] is a model stealing attack which recreates the target model without access to its dataset.
4) **Traditional Supply Chain Attacks:** These attacks compromise upstream software components with the intent of exploiting downstream DNN vulnerabilities [79], [66]. Ladisa *et al.* [90] and Ohm *et al.* [91] enumerate several classifications of attacks on traditional open-source supply chains. These attacks are also applicable to DNN supply chains. For example, typo-squatting (whereby an attacker uses similar package names) could confuse DNN users into using the wrong PTM.

C. Challenges in Deployment Reuse of DNNs

Once a deep learning model has been developed, possibly through conceptual or adaptation reuse, it must then be deployed. This deployment is not simple, so we classify this as a form of reuse rather than just a final step. Deployment faces its own series of challenges because the development environment of DNNs (for training and testing) may differ substantially from deployment environments. For this reason, engineers typically face two main types of deployment reuse challenges: interoperability between frameworks and hardware, and trust establishment in DL supply chains.

a) Deep Learning Interoperability: The advent of DNNs has brought computing platforms that specialize in accelerating their execution [92], consequently, there has been a greater demand for tools enabling the deployment of DNNs onto diverse hardware. *Deep learning compilers* [92] such as TVM [93], OpenVINO [94], and Glow [95] aim to bridge this gap by taking DL models and converting them to optimized binary code targeting the multitude of hardware available. Prior work has focused on developing [93], [95] and testing DL compilers [96], [97], as well as empirical studies of DL compiler failures [98].

The introduction of DL compilers largely addresses the challenges of deploying to diverse hardware but it introduces challenges in the use of compilers. Many compilers do not

support all frameworks thus the interoperability of frameworks and compilers poses a challenge in the deployment of DNNs. To address this common representations such as ONNX act as intermediaries between DL frameworks and compilers [99]. Figure 6 depicts how common representations can act as intermediaries between frameworks and compilers. Model converters are used to convert between framework representations and ONNX such as *tf2onnx* [100].

Prior work on interoperability has largely focused on common representations or model conversion. Common representations such as ONNX [99] and NNEF [101] have been introduced to further interoperability. Model converters like MMDnn [102] allow for faithful conversions between frameworks, or allow conversion to and from common representations. Though this type of software is largely understudied in a DL context. Studies on converters have largely focused on DNN properties after conversion [103]. ONNX conformance test generation has been proposed to ensure ONNX implementations match the ONNX specification [104].

Recently, model converter failures have been studied in the context of ONNX [105]. It was found that model converters exhibit two common failure symptoms: crashes and wrong outputs. Crashes are largely due to the converter being unable to convert operators of the DNN to ONNX. This can be due to the converter not yet implementing this conversion, or ONNX not supporting the operator. Wrong outputs are when a successfully converted model is not semantically equivalent to the original model. Such failures suggest that engineers should weigh the risks of model conversion against potential benefits.

b) Establishing Trust in DL Supply Chains: Establishing trust in traditional software products is a difficult task [106]. This is no different for DNNs. In §III-B we discuss the wide array of challenges engineers face when attempting to adapt existing DNNs to solve different tasks. Specifically, we mention many of the attacks which threaten how engineers can reuse models and that the reuse of DNNs creates a supply chain structure. These attacks make the decision-making process difficult when adapting DNNs, but they also make it difficult for engineers to establish trust in the DNNs they are about to deploy. In other words, once an organization decides to release a DNN into the wild, they have a hard time making sure users can trust it.

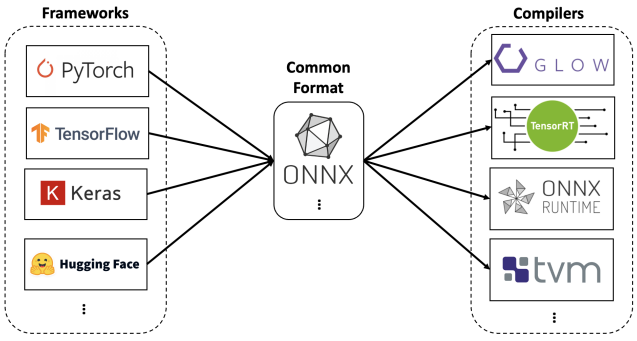

Fig. 6: A common format like ONNX (Open Neural Network eXchange) is used as an intermediary to adapt DNN written with general-purpose DL frameworks so that it works on hardware-specific DL compilers.

Novel characteristics of the DNN supply chain introduce new methods for attackers to degrade trust [66], [107]. Furthermore, users are often either unwilling or unable to check for these attacks [85]. This means users must blindly trust the DNNs they download from open-source registries. Although traditional security features such as software signing may help to verify file integrity, many attacks slip by these simple measures. As an example, Figure 7 illustrates a common dependency structure for DNNs that makes it hard for users to validate DNN models and datasets at deployment time (see caption). Cryptographic methods [108], [109] exist to verify that particular files have not changed through deployment, but the relationships between files — and in particular, DNN dependency relationships — are not easily verified in this way. Traditional software dependencies can be verified by checking dependencies through methods like Software Bills of Materials (SBOMs) and reproducible builds [110], [111]. Similar techniques are more difficult for DNNs because of non-determinism and training costs [112].

IV. Directions

A. Directions in Conceptual Reuse of DNNs

Comparing the conceptual reuse of DNNs to that of traditional software [113], [114], DNN reuse is mainly based on research products while traditional software reuse is focused on the outputs from engineering teams. Consequently, the goal and focus of DNN reuse are different [62]. Considering the differences of conceptual DNN reuse, we propose several research directions, including promoting reusable artifacts and developing engineering tools.

1) Evaluating Artifacts for Reuse: Although conceptual reuse is primarily focused on methods presented in research papers and technical reports, there is a growing effort to include artifact evaluation to support the claims of conference and journal papers. Typical artifact evaluation when it comes to machien learning includes a minimum-viable prototype, training and evaluation datasets, and results. While the focus of artifact evaluation in machine learning is primarily on *reproducibility* of a paper's claims, evaluating the *software engineering* of these accepted artifacts would seemingly be a key ingredient of ensuring that a research artifact has some hope of being reproduced by others at the conceptual level (and beyond). As more and more publication venues incentivize artifacts, an empirical software engineering study to evaluate the software engineering process across these venues, specifically in support of reuse, is a topic worthy of further study. Separately, we would also encourage artifact evaluation to include a checklist of minimal expectations, which is already a practice in software-focused journals such as the Journal of Open Source Software.

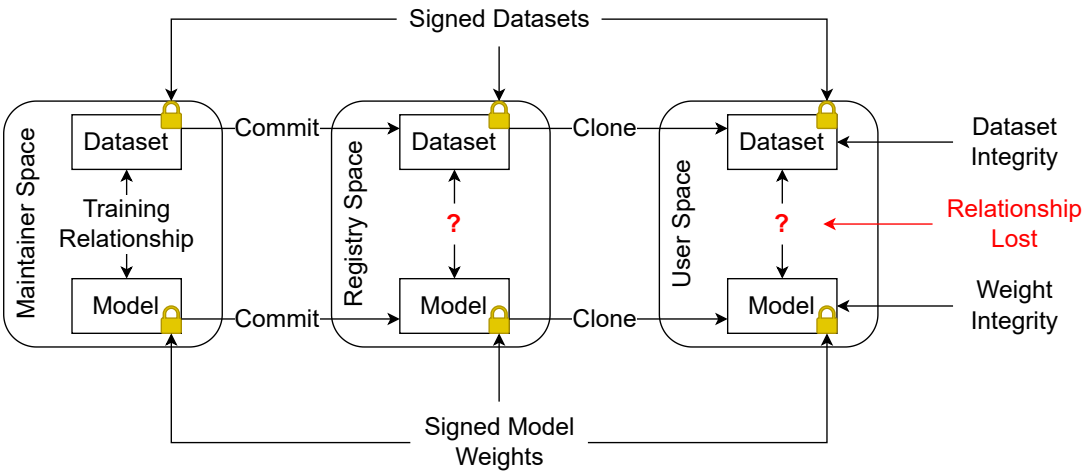

Fig. 7: Maintainers can sign datasets or models before commiting those files to a registry. Attaching signatures to these files enables validation of files between trust domains. This enables users to verify dataset and model integrity. These signatures only validate the source of a model or dataset — any relationship between the two cannot be validated. Users must trust that maintainers have trained a DNN as described in documentation. There is no known way to validate this with high confidence.

2) Testing Tools: Testing tools and frameworks are among the high-impact practices in software engineering and greatly improve the reuse potential of software libraries in general. Testing can also help to improve the DNN reuse process. Researchers have developed some tools for automated deep learning testing [115]. Despite these advancements, there is a noticeable lack of adoption of specialized testing techniques for conceptual DNN reuse.

We urge the research community to consider the application of comprehensive testing tools for conceptual DNN reuse tasks. Since conceptual reuse includes comprehending documentation such as research papers, checklists and tools for extracting knowledge would be helpful. Validation tools might leverage pre-existing model implementations. For instance, emerging fuzzing technologies could use adversarial inputs to verify the accuracy of early-stage training, using the original model as a benchmark. Refining unit testing methods could also reduce overall costs. We encourage exploration of strategies for effective unit and differential testing in DNN software.

B. Directions in Adaptation Reuse of DNNs

Adapting DNNs presents challenges, including complicated decision-making processes, costly evaluation loops, and potential security and privacy risks. We describe several research directions that support adaptation reuse: large-scale model audit, infrastructure optimization, recommendation systems, and attack detection.

1) Model Audit: Prior work found that the trustworthiness of DNNs are concerning due to the lack of DNN transparency. Future work can measure attributes of DNNs that are not currently accessible to engineers. For example, DL-specific attributes can be extracted from provided documentation, source code, and metadata [40]. Potential risks could be identified by measuring the discrepancies among different DNN models. These measurements could largely improve the transparency of DNNs and therefore facilitate better adaptation reuse.

2) Infrastructure Optimization: Another challenge of adaptation reuse is the model selection. Engineers struggle to compare different DNNs and identify a good way to adapt to their downstream task. To facilitate the adaptation, researchers can identify different approaches to support the model selection process. For instance, providing enhanced documentation, similar to the badges used by GitHub, could offer greater transparency about the capabilities and limitations of a model. This could facilitate a more informed model selection process.

Standardization tools can also play a pivotal role. For example, the use of large language models (LLMs) could facilitate the extraction of metadata for model cards, enabling a systematic comparison of models' performance and requirements. Additionally, a standardized model interface or API could be developed. This would provide a uniform way to interact with and evaluate different models, thereby simplifying the model selection process.

3) PTM Recommendation Systems: Existing literature highlights the challenges associated with technical adaptation and the resulting decision-making process to select a suitable starting point (§III-B). Numerous studies have sought solutions to the problem of PTM selection. For instance, You et al. proposed a transferability metric called *LogME* that is used to assess and rank a list of models [116], [117]. Certain works have also employed deep learning techniques to enhance the AutoML process [118], [119]. Despite these advancements, open-source PTMs remain under-utilized [40], suggesting the need for a robust PTM recommendation system to aid engineers in adaptation reuse. We encourage researchers to consider various factors during the adaptation reuse process, including diverse fine-tuning approaches, necessary engineer-

ing efforts, and the trustworthiness of the model.

4) Attack Detection Tool: Our work indicates that specific attack detection tools are currently missing in DL model registries. For example, Hugging Face only uses ClamAV which can only detect traditional malware but not applicable for DL-specific attacks [40]. The potential risks existing in the model registries make the adaptation reuse harder for engineers. Developing detection tools for DL-specific attacks can therefore improve the trustworthiness of DNNs and help engineers easier adapt existing DNNs. Detection tools for copyright infringement such as DeepJudge [120] might be adapted for this purpose.

C. Directions in Deployment Reuse of DNNs

In §III-C we identified interoperability and trust as key challenges for DNN deployment. Here, we identify future research directions that may help engineers deploy DNNs more effectively.

Interoperability plays a key role in the deployment of DNNs, namely the use of model converters (ONNX) to connect frameworks to compilers (as shown in Figure 6). ONNX model converter failures have been studied [105]. Converters often are not compatible with all ONNX operators, moreover, converted models often are not semantically equivalent to their original models.

Based on prior work, we propose the following directions related to enhancing interoperability, converter testing, and increasing DNN supply chain security.

1) *Model Converter Testing:* Model converters often produce models that are semantically inequivalent to the original models. Consequently, testing to identify the source of semantic inequivalence in converters in important.
2) *Supporting Model Converter Engineering:* ONNX model converters suffer from incompatibilities with the evolution of intermediate representation. This results in converters needing updates as ONNX updates to ensure the specification is faithfully followed. It follows that these efforts must be supported. Specifically, engineering efforts can be better focused by understanding operator popularity, and domain-specific languages can be used to automatically generate converter code. Additionally, these efforts can be avoided with the development of a small stable operator set that can represent all possible operators that can be used in a deep learning model [121] — similar to RISC [122] and JVM [123].
3) *Supply Chain Security for DNNs:* The software engineering community has been working on systems such as TUF [124] and Sigstore [108] to increase the usability and effectiveness of signatures for package managers. The community has also began to develop standards such as Supply-chain Levels for Software Artifacts (SLSA) [125] and Software Supply Chain Best Practices from the Cloud Native Computing Foundation (CNCF) [126] to help engineers implement appropriate supply chain security measures. Similar efforts should be taken to create systems and standards for the DNN community, *e.g.,* determining which concepts still apply to DNN environments and which concepts need to be changed. For example, Figure 7 shows how traditional signing methods may be inadequate in DNN supply chains. Future work could consider how to preserve the relationship between a particular DNN and the dataset(s) it was trained on. This might be accomplished by watermarking [127] or some form of limited reproduction, whereby users validate model-dataset relationships via retraining.

D. Assessing the Software Engineering Process of DNNs

As with any other software engineering effort, conceptualizing, adapting, and deploying DNNs is a measurable process. A high-quality software artifact is typically associated with an effective software engineering process [129]. By measuring the software engineering process of DNNs, engineers can gain insights into the quality of the DNN. These process measurements can assist engineering teams in evaluating DNNs as dependencies within their projects.

There are no tools that quantify what is and is not an effective software engineering process tailored to DNNs. Existing understanding [130] and tooling [131] can be applied to the DNN engineering process. However, as the reuse, conceptualizing, adapting, and deploying process of DNNs is different than traditional software projects, it is unclear as to what is and is not a good engineering process for DNNs.

Figure 8 illustrates the challenges of measuring the DNN engineering (and reuse) process, suggesting the difficulties in mining to elicit empirical data about effective processes. Unlike traditional software engineering packages, DNN packages cannot be analyzed in isolation;[2] the training dataset, config, and documentation all provide crucial information as to both what the model is, as well as the performance is. Due to the additional information, the data extraction that is required to mine a DNN repository is more complex.

To assist in the mining efforts of DNNs, the PTMTorrent dataset [132] provides a dataset of ∼60,000 PTMs from 5 model registries. This dataset contains the full `git` repository for each DNN package, including the model, documentation, and configuration information. The follow-up PeaTMOSS dataset adds connections to use in open-source projects from GitHub [133].

V. Discussion

A. DNN Reuse as an Accelerator

Reuse is a key engineering technique because it offers such substantial cost savings [134]. Whether a software engineer reuses via the conceptual paradigm, the adaptation paradigm, or the deployment paradigm (or all three!), they benefit.

Our purpose in framing deep learning engineering in terms of reuse is to emphasize the potential benefits of deep learning reuse to software engineers. However, we acknowledge that at present the benefit may not always be realized. For example,

[2]Of course, traditional software should also not be analyzed in isolation, but more useful data may be derived from an isolated view in that case.

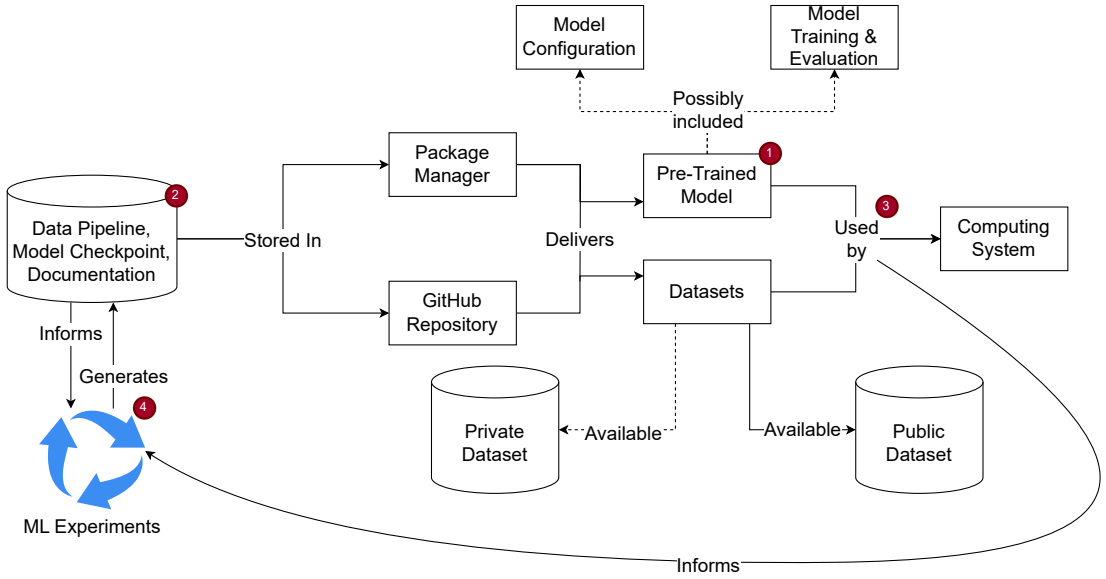

Fig. 8: This figure illustrates the complexity of mining and using DNN packages. We note four challenges in mining DNN packages. (1) A DNN package may solely contain the pre-trained model with no additional information, limiting what can be learned from it. (2) ML experiments generate new data pipelines, model checkpoints, and documentation as they are iterated upon and improved on. It is challenging to describe and learn from the interrelationships of the experiments. (3) ML experiments are affected by pre-trained models trained on similar tasks or datasets, or by the dataset itself. There is a cyclical dependency that affects mining these artifacts. (4) ML experiments may not be made publicly available. This "secret" [128] process is crucial to deep learning engineering but difficult to observe.

in recent interviews we found that skilled software engineers sometimes take months to identify a good model to adapt and fine-tune to reach acceptable performance (adaptation paradigm) [40]. We are not aware of any direct comparisons in engineering cost between from-scratch development and reuse development. However, we expect that advances in DNN reuse techniques can substantially lower the current costs. We challenge the research community to show the way forward.

B. DNN Reuse in the Age of Foundational Models

The trend in deep learning has been to develop larger and larger models, *e.g.,* supporting multiple modes of input and capable of solving a wide range of problems. The resulting models are generalists. One might wonder, when a single model like GPT-4 can solve math problems and interpret jokes [135], why researchers and engineers should be concerned with model reuse as opposed to simply model application. While large-language models at the time of writing are appearing to become less open as part of an industry "arms race" of sorts, investigating reuse is important regardless of whether building proprietary and/or open source deep learning solutions (similar to the corresponding situation in general software development). We point out two weaknesses of the current trend towards a single powerful model.

First, this approach centralizes power in the hands of those who control the model [136]. Foundational models such as ChatGPT and GPT-4 are trained by one organization with unclear data sources, and they impose a client-server model where one organization mediates all responses. Centralization and unmonitored data processing bode poorly for user privacy [137] and the concomitant problems of a monopoly [138].

Second, this approach is highly energy-consumptive [139]. Resource-intensive models can achieve state-of-the-art performance, but sometimes this is not necessary [140]. In resource-constrained environments, such as those involved in real-time decision-making, fast and adequate may suffice. In this context, the reuse paradigms allow engineers to explore alternative approaches that trade off resource utilization and performance.

VI. Conclusion

This paper described the challenges of re-using deep neural networks (DNNs). DNNs have demonstrated exceptional performance in various domains, but their development and computational costs remain significant. DNN re-use can reduce development costs. However, re-use presents its own set of challenges, encompassing technical capabilities and engineering practices. We hope this work improves understanding DNN re-use. We discussed three types of reuse — conceptual, adaptation, and deployment. By identifying the gaps in current re-use practices, we contribute to the understanding of DNN reuse and offer insights for future research in this area.

Acknowledgments

We acknowledge financial support from Google, Cisco, and NSF awards 2229703, 2107230, 2107020, and 2104319.

References

[1] P. McCorduck, *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. A K Peters/CRC Press, 1991.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[3] W. G. Hatcher and W. Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, vol. 6, pp. 24 411–24 432, 2018.

[4] A. Goel, C. Tung, Y.-H. Lu, and G. K. Thiruvathukal, "A Survey of Methods for Low-Power Deep Learning and Computer Vision," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, Jun. 2020, pp. 1–6.

[5] A. Goel, C. Tung, X. Hu, G. K. Thiruvathukal, J. C. Davis, and Y.-H. Lu, "Efficient computer vision on edge devices with pipeline-parallel hierarchical neural networks," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 532–537.

[6] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE access*, vol. 7, pp. 19 143–19 165, 2019.

[7] E. L. Hines, E. Llobet, and J. Gardner, "Electronic noses: a review of signal processing techniques," *IEE Proceedings-Circuits, Devices and Systems*, vol. 146, no. 6, pp. 297–310, 1999.

[8] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, X. Jiang, K. Cobbe, T. Eloundou, G. Krueger, K. Button, M. Knight, B. Chess, and J. Schulman, "WebGPT: Browser-assisted question-answering with human feedback," Jun. 2022, arXiv:2112.09332 [cs]. [Online]. Available: http://arxiv.org/abs/2112.09332

[9] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-Shot Text-to-Image Generation," Feb. 2021, arXiv:2102.12092 [cs]. [Online]. Available: http://arxiv.org/abs/2102.12092

[10] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A Generative Model for Music," Apr. 2020, arXiv:2005.00341 [cs, eess, stat]. [Online]. Available: http://arxiv.org/abs/2005.00341

[11] M. Tkáč and R. Verner, "Artificial neural networks in business: Two decades of research," *Applied Soft Computing*, vol. 38, pp. 788–804, 2016.

[12] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan, "Medical image analysis using convolutional neural networks: a review," *Journal of medical systems*, vol. 42, pp. 1–13, 2018.

[13] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 712–733, 2020.

[14] M. Hendrickson, "Thinking about glue," *O'Reilly Radar*, April 2021. [Online]. Available: https://www.oreilly.com/radar/thinking-about-glue/

[15] C. W. Krueger, "Software reuse," *ACM Computing Surveys (CSUR)*, vol. 24, no. 2, pp. 131–183, 1992, publisher: ACM New York, NY, USA.

[16] W. B. Frakes and K. Kang, "Software reuse research: Status and future," *IEEE transactions on Software Engineering*, vol. 31, no. 7, pp. 529–536, 2005.

[17] C. Sadowski, K. T. Stolee, and S. Elbaum, "How developers search for code: a case study," in *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, 2015, pp. 191–201.

[18] R. Abdalkareem, O. Nourry, S. Wehaibi, S. Mujahid, and E. Shihab, "Why do developers use trivial packages? an empirical case study on npm," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 2017, pp. 385–395.

[19] L. G. Michael IV, J. Donohue, J. C. Davis, D. Lee, and F. Servant, "Regexes are Hard: Decision-making, Difficulties, and Risks in Programming Regular Expressions," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019.

[20] A. Karpathy, "Software 2.0," 2017. [Online]. Available: https://medium.com/@karpathy/software-2-0-a64152b37c35

[21] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[22] S. Amershi, A. Begel, C. Bird, R. DeLine, and H. Gall, "Software Engineering for Machine Learning: A Case Study," in *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019.

[23] S. Rahman, E. River, F. Khomh, Y. G. Guhneuc, and B. Lehnert, "Machine learning software engineering in practice: An industrial case study," *arXiv*, pp. 1–21, 2019.

[24] V. Banna, A. Chinnakotla, Z. Yan, A. Vegesana, N. Vivek, K. Krishnappa, W. Jiang, Y.-H. Lu, G. K. Thiruvathukal, and J. C. Davis, "An experience report on machine learning reproducibility: Guidance for practitioners and TensorFlow model garden contributors," *arXiv preprint arXiv:2107.00821*, 2021.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] H. B. Braiek and F. Khomh, "On testing machine learning programs," *Journal of Systems and Software (JSS)*, vol. 164, p. 110542, 2020.

[28] S. Li, J. Guo, J.-G. Lou, M. Fan, T. Liu‡, and D. Zhang, "Testing Machine Learning Systems in Industry: An Empirical Study," in *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2022, pp. 263–272.

[29] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer, 2020, pp. 213–229.

[30] Wikipedia, "Wikipedia, extract, transform, load," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Extract,_transform,_load

[31] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[32] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, 2019.

[33] H. Yokoyama, S. Onoue, and S. Kikuchi, "Towards Building Robust DNN Applications: An Industrial Case Study of Evolutionary Data Augmentation," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Sep. 2020, pp. 1184–1188.

[34] D. Yin, R. Gontijo Lopes, J. Shlens, E. D. Cubuk, and J. Gilmer, "A fourier perspective on model robustness in computer vision," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[35] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[36] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[37] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[38] M. Shen, J. C. Davis, and A. Machiry, "Towards automated identification of layering violations in embedded applications (wip)," in *2023 ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. ACM, 2023.

[39] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J.-R. Wen, J. Yuan, W. X. Zhao, and J. Zhu, "Pre-trained models: Past, present and future," *AI Open*, vol. 2, pp. 225–250, 2021.

[40] W. Jiang, N. Synovic, M. Hyatt, T. R. Schorlemmer, R. Sethi, Y.-H. Lu, G. K. Thiruvathukal, and J. C. Davis, "An empirical study of pre-trained model reuse in the hugging face deep learning model registry," in *IEEE/ACM 45th International Conference on Software Engineering (ICSE'23)*, 2023.

[41] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML test score: A rubric for ML production readiness and technical debt reduction," *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, vol. 2018-Janua, pp. 1123–1132, 2017.

[42] T. Zhang, C. Gao, L. Ma, M. Lyu, and M. Kim, "An Empirical Study of Common Challenges in Developing Deep Learning Applications," *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, vol. 2019-Octob, pp. 104–115, 2019.

[43] X. Zhang, Y. Yang, Y. Feng, and Z. Chen, "Software engineering practice in the development of deep learning applications," *arXiv*, 2019.

[44] A. Serban, K. Van Der Blom, H. Hoos, and J. Visser, "Adoption and effects of software engineering best practices in machine learning," *International Symposium on Empirical Software Engineering and Measurement*, 2020.

[45] H. Washizaki, H. Uchida, F. Khomh, and Y. G. Guéhéneuc, "Studying Software Engineering Patterns for Designing Machine Learning Systems," *Proceedings - 2019 10th International Workshop on Empirical Software Engineering in Practice, IWESEP 2019*, pp. 49–54, 2019.

[46] I. Sommerville, *Software engineering*, 2016.

[47] J. Kim and J. Li, "Introducing the model garden for tensorflow 2," 2020. [Online]. Available: https://blog.tensorflow.org/2020/03/introducing-model-garden-for-tensorflow-2.html

[48] M. Hutson, "Artificial intelligence faces reproducibility crisis," *American Association for the Advancement of Science*, vol. 359, no. 6377, pp. 725–726, 2018.

[49] S. S. Alahmari, D. B. Goldgof, P. R. Mouton, and L. O. Hall, "Challenges for the Repeatability of Deep Learning Models," *IEEE Access*, 2020.

[50] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," 2020. [Online]. Available: https://arxiv.org/abs/1911.02685

[51] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv:1503.02531 [cs, stat]*, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[52] J. Villa and Y. Zimmerman, "Reproducibility in ML: why it matters and how to achieve it," 2018. [Online]. Available: https://determined.ai/blog/reproducibility-in-ml

[53] J. Pineau, "How the AI community can get serious about reproducibility," 2022. [Online]. Available: https://ai.facebook.com/blog/how-the-ai-community-can-get-serious-about-reproducibility/

[54] "Papers with Code - ML Reproducibility Challenge 2021 Edition," 2020. [Online]. Available: https://paperswithcode.com/rc2021

[55] Z. Ding, A. Reddy, and A. Joshi, "Reproducibility," 2021. [Online]. Available: https://blog.ml.cmu.edu/2020/08/31/5-reproducibility/

[56] J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché Buc, E. Fox, and H. Larochelle, "Improving reproducibility in machine learning research (A report from the neurips 2019 reproducibility program)," *arXiv*, 2020.

[57] O. E. Gundersen and S. Kjensmo, "State of the art: Reproducibility in artificial intelligence," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 1644–1651, 2018.

[58] O. E. Gundersen, Y. Gil, and D. W. Aha, "On reproducible AI: Towards reproducible research, open science, and digital scholarship in AI publications," *AI Magazine*, 2018.

[59] B. Chen, M. Wen, Y. Shi, D. Lin, G. K. Rajbahadur, Z. Ming, and Jiang, "Towards Training Reproducible Deep Learning Models," in *International Conference on Software Engineering (ICSE)*, May 2022, pp. 2202–2214.

[60] H. V. Pham, S. Qian, J. Wang, T. Lutellier, J. Rosenthal, L. Tan, Y. Yu, and N. Nagappan, "Problems and Opportunities in Training Deep Learning Software Systems: An Analysis of Variance," in *International Conference on Automated Software Engineering (ASE)*, 2020.

[61] H. V. Pham, M. Kim, L. Tan, Y. Yu, and N. Nagappan, "DEVIATE: A Deep Learning Variance Testing Framework," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov. 2021, pp. 1286–1290, iSSN: 2643-1572.

[62] W. Jiang, V. Banna, N. Vivek, A. Goel, N. Synovic, G. K. Thiruvathukal, and J. C. Davis, "Challenges and practices of deep learning model reengineering: A case study on computer vision," *arXiv preprint arXiv:2303.07476*, 2023.

[63] M. Yuan, L. Zhang, X.-Y. Li, and H. Xiong, "Comprehensive and efficient data labeling via adaptive model scheduling," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1858–1861.

[64] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, pp. 1789–1819, 2021.

[65] P. Marcelino, "Transfer learning from pre-trained models," Apr. 2022. [Online]. Available: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

[66] W. Jiang, N. Synovic, R. Sethi, A. Indarapu, M. Hyatt, T. R. Schorlemmer, G. K. Thiruvathukal, and J. C. Davis, "An empirical study of artifacts and security risks in the pre-trained model supply chain," in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2022, pp. 105–114.

[67] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.

[68] N. K. Gopalakrishna, D. Anandayuvaraj, A. Detti, F. L. Bland, S. Rahaman, and J. C. Davis, ""If security is required": Engineering and Security Practices for Machine Learning-based IoT Devices," in *International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT)*, 2022.

[69] D. Pessach and E. Shmueli, "A review on fairness in machine learning," *ACM Computing Surveys (CSUR)*, vol. 55, no. 3, pp. 1–44, 2022.

[70] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–35, 2021.

[71] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Ieee Symposium on Security and Privacy*. Ieee, 2017, pp. 39–57.

[72] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 3319–3328.

[73] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[74] J. Gesi, X. Shen, Y. Geng, Q. Chen, and I. Ahmed, "Leveraging Feature Bias for Scalable Misprediction Explanation of Machine Learning Models," in *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, 2023.

[75] R. Pan and H. Rajan, "On decomposing a deep neural network into modules," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2020, pp. 889–900.

[76] ——, "Decomposing convolutional neural networks into reusable and replaceable modules," in *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 524–535.

[77] S. M. Imtiaz, F. Batole, A. Singh, R. Pan, B. D. Cruz, and H. Rajan, "Decomposing a recurrent neural network into modules for enabling reusability and replacement," in *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, 2023.

[78] D. Montes, P. Peerapatanapokin, J. Schultz, C. Guo, W. Jiang, and J. C. Davis, "Discrepancies among pre-trained deep neural networks: a new threat to model zoo reliability," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE-IVR track).*, 2022.

[79] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, "Sok: Analysis of software supply chain security by establishing secure design properties," in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, ser. SCORED'22. New York, NY, USA: Association for Computing Machinery, 2022, p. 15–24. [Online]. Available: https://doi.org/10.1145/3560835.3564556

[80] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Computer Science Review*, vol. 34, p. 100199, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013718303289

[81] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain," 2019. [Online]. Available: http://arxiv.org/abs/1708.06733

[82] Z. Wang, C. Liu, and X. Cui, "EvilModel: Hiding Malware Inside of Neural Network Models," in *IEEE Symposium on Computers and Communications (ISCC)*, 2021.

[83] Z. Wang, C. Liu, X. Cui, J. Yin, and X. Wang, "EvilModel 2.0: Bringing Neural Network Models into Malware Attacks," *Computers & Security*, 2022.

[84] T. Liu, Z. Liu, Q. Liu, W. Wen, W. Xu, and M. Li, "StegoNet: Turn Deep Neural Network into a Stegomalware," in *Annual Computer Security Applications Conference*. Austin USA: ACM, Dec. 2020, pp. 928–938. [Online]. Available: https://dl.acm.org/doi/10.1145/3427228.3427268

[85] Z. Liu, F. Li, Z. Li, and B. Luo, "LoneNeuron: a Highly-Effective Feature-Domain Neural Trojan Using Invisible and Polymorphic Watermarks," in *ACM SIGSAC Conference on Computer and Communications Security*. Los Angeles: ACM, 2022.

[86] Y. Miao, C. Chen, L. Pan, Q.-L. Han, J. Zhang, and Y. Xiang, "Machine learning–based cyber attacks targeting on controlled information: A survey," *ACM Comput. Surv.*, vol. 54, no. 7, jul 2021. [Online]. Available: https://doi.org/10.1145/3465171

[87] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016, pp. 372–387.

[88] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2017, pp. 3–18. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2017.41

[89] S. Kariyappa, A. Prakash, and M. K. Qureshi, "MAZE: Data-Free Model Stealing Attack Using Zeroth-Order Gradient Estimation," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Nashville, TN, USA: IEEE, Jun. 2021, pp. 13 809–13 818. [Online]. Available: https://ieeexplore.ieee.org/document/9577631/

[90] P. Ladisa, H. Plate, M. Martinez, and O. Barais, "Taxonomy of Attacks on Open-Source Software Supply Chains," 2022. [Online]. Available: http://arxiv.org/abs/2204.04008

[91] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, C. Maurice, L. Bilge, G. Stringhini, and N. Neves, Eds. Springer, 2020, pp. 23–43.

[92] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian, "The deep learning compiler: A comprehensive survey," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, p. 708–727, Mar 2021, arXiv:2002.03794 [cs].

[93] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: An automated end-to-end optimizing compiler for deep learning," no. arXiv:1802.04799, Oct 2018, arXiv:1802.04799 [cs]. [Online]. Available: http://arxiv.org/abs/1802.04799

[94] OpenVINO, "Openvino documentation," https://docs.openvino.ai/latest/home.html, 2021.

[95] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein, J. Montgomery, B. Maher, S. Nadathur, J. Olesen, J. Park, A. Rakhov, M. Smelyanskiy, and M. Wang, "Glow: Graph lowering compiler techniques for neural networks," no. arXiv:1805.00907, Apr 2019, arXiv:1805.00907 [cs]. [Online]. Available: http://arxiv.org/abs/1805.00907

[96] J. Liu, J. Lin, F. Ruffy, C. Tan, J. Li, A. Panda, and L. Zhang, "Nnsmith: Generating diverse and valid test cases for deep learning compilers," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. Vancouver BC Canada: ACM, Jan 2023, p. 530–543. [Online]. Available: https://dl.acm.org/doi/10.1145/3575693.3575707

[97] D. Xiao, Z. Liu, Y. Yuan, Q. Pang, and S. Wang, "Metamorphic testing of deep learning compilers," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 1, p. 1–28, Feb 2022.

[98] Q. Shen, H. Ma, J. Chen, Y. Tian, S.-C. Cheung, and X. Chen, "A comprehensive study of deep learning compiler bugs," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Athens Greece: ACM, Aug 2021, p. 968–980. [Online]. Available: https://dl.acm.org/doi/10.1145/3468264.3468591

[99] "ONNX | Home," https://onnx.ai/, 2019.

[100] "tf2onnx," https://github.com/onnx/tensorflow-onnx, Feb 2023.

[101] Oct 2016. [Online]. Available: https://www.khronos.org//

[102] Y. Liu, C. Chen, R. Zhang, T. Qin, X. Ji, H. Lin, and M. Yang, "Enhancing the interoperability between deep learning frameworks by model conversion," in *Proceedings of the 28th ACM Joint Meeting on ESEC/FSE*. Virtual Event USA: ACM, Nov 2020, p. 1320–1330. [Online]. Available: https://dl.acm.org/doi/10.1145/3368089.3417051

[103] M. Openja, A. Nikanjam, A. H. Yahmed, F. Khomh, Z. Ming, and Jiang, "An Empirical Study of Challenges in Converting Deep Learning Models," *IEEE Transactions on Software Engineering*, Jun. 2022.

[104] X. Cai, P. Zhou, S. Ding, G. Chen, and W. Zhang, "Sionnx: Automatic unit test generator for onnx conformance," no. arXiv:1906.05676, Jun 2019, arXiv:1906.05676 [cs]. [Online]. Available: http://arxiv.org/abs/1906.05676

[105] P. Jajal, W. Jiang, A. Tewari, J. Woo, Y.-H. Lu, G. K. Thiruvathukal, and J. C. Davis, "Analysis of failures and risks in deep learning model converters: A case study in the onnx ecosystem," no. arXiv:2303.17708, Mar 2023, arXiv:2303.17708 [cs]. [Online]. Available: http://arxiv.org/abs/2303.17708

[106] European Union Agency for Cybersecurity, "ENISA Threat Landscape 2021," ENISA, Report/Study, Oct. 2021. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2021

[107] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8685687/

[108] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2353–2367. [Online]. Available: https://doi.org/10.1145/3548606.3560596

[109] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1393–1410. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias

[110] "Software Bill of Materials," https://www.cisa.gov/sbom.

[111] C. Lamb and S. Zacchiroli, "Reproducible Builds: Increasing the Integrity of Software Supply Chains," *IEEE Software*, vol. 39, no. 2, pp. 62–70, Mar. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9403390/

[112] B. Chen, M. Wen, Y. Shi, D. Lin, G. K. Rajbahadur, and Z. M. J. Jiang, "Towards training reproducible deep learning models," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, Jul. 2022, pp. 2202–2214. [Online]. Available: https://doi.org/10.1145/3510003.3510163

[113] J. Singh, K. Singh, and J. Singh, "Reengineering framework for open source software using decision tree approach," *International Journal of electrical and computer engineering (IJECE)*, vol. 9, no. 3, pp. 2041–2048, 2019.

[114] K. Bhavsar, V. Shah, and S. Gopalan, "Machine learning: a software process reengineering in software development organization," *International Journal of Engineering and Advanced Technology*, vol. 9, no. 2, pp. 4492–4500, 2020.

[115] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated Whitebox Testing of Deep Learning Systems," in *Symposium on Operating Systems Principles (SOSP)*, Oct. 2017.

[116] K. You, Y. Liu, J. Wang, and M. Long, "LogME: Practical Assessment of Pre-trained Models for Transfer Learning," in *International Conference on Machine Learning (ICML)*. PMLR, Jul. 2021, pp. 12 133–12 143. [Online]. Available: https://proceedings.mlr.press/v139/you21b.html

[117] K. You, Y. Liu, J. Wang, M. I. Jordan, and M. Long, "Ranking and Tuning Pre-trained Models: A New Paradigm of Exploiting Model Hubs," *The Journal of Machine Learning Research (JMLR)*, vol. 23, no. 1, pp. 9400–9446, Oct. 2021. [Online]. Available: http://arxiv.org/abs/2110.10545

[118] E. Öztürk, F. Ferreira, H. Jomaa, L. Schmidt-Thieme, J. Grabocka, and F. Hutter, "Zero-shot AutoML with Pretrained Models," in *Proceedings of the 39th International Conference on Machine Learning*. PMLR, Jun. 2022, pp. 17 138–17 155. [Online]. Available: https://proceedings.mlr.press/v162/ozturk22a.html

[119] B. Lu, J. Yang, L. Y. Chen, and S. Ren, "Automating Deep Neural Network Model Selection for Edge Inference," in *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*, Dec. 2019, pp. 184–193.

[120] J. Chen, J. Wang, T. Peng, Y. Sun, P. Cheng, S. Ji, X. Ma, B. Li, and D. Song, "Copy, Right? A Testing Framework for Copyright Protection of Deep Learning Models," in *2022 IEEE Symposium on Security and Privacy (SP)*, May 2022, pp. 824–841, iSSN: 2375-1207.

[121] Y. Liu, C. Chen, R. Zhang, M. Research, H. Lin, and M. Yang, "Enhancing the Interoperability between Deep Learning Frameworks by Model Conversion," *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pp. 1320–1330, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Interoperability.

[122] D. A. Patterson and J. L. Hennessy, *Computer organization and design ARM edition: the hardware software interface*. Morgan Kaufmann, 2016.

[123] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java virtual machine specification*. Pearson Education, 2014.

[124] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 61–72. [Online]. Available: https://doi.org/10.1145/1866307.1866315

[125] The SLSA Project, "Supply-chain Levels for Software Artifacts." [Online]. Available: http://slsa.dev/

[126] S. T. A. Group, "Software Supply Chain Best Practices," Cloud Native Computing Foundation, Tech. Rep., May 2021. [Online]. Available: https://project.linuxfoundation.org/hubfs/CNCF_SSCP_v1.pdf

[127] Y. Li, H. Wang, and M. Barni, "A survey of deep neural network watermarking techniques," *Neurocomputing*, vol. 461, pp. 171–193, 2021.

[128] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *International Conference on Software Engineering (ICSE)*, 2009.

[129] I. Sommerville, "Software engineering 10th Edition," *ISBN-10*, vol. 137035152, p. 18, 2015.

[130] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach, Third Edition*, 3rd ed. Boca Raton: CRC Press, Oct. 2014.

[131] N. M. Synovic, M. Hyatt, R. Sethi, S. Thota, Shilpika, A. J. Miller, W. Jiang, E. S. Amobi, A. Pinderski, K. Läufer, N. J. Hayward, N. Klingensmith, J. C. Davis, and G. K. Thiruvathukal, "Snapshot Metrics Are Not Enough: Analyzing Software Repositories with Longitudinal Metrics," in *37th IEEE/ACM International Conference on Automated Software Engineering — Tools track*. Rochester MI USA: ACM, Oct. 2022, pp. 1–4. [Online]. Available: https://dl.acm.org/doi/10.1145/3551349.3559517

[132] W. Jiang, N. Synovic, P. Jajal, T. R. Schorlemmer, A. Tewari, B. Pareek, G. K. Thiruvathukal, and J. C. Davis, "PTMTorrent: A Dataset for Mining Open-source Pre-trained Model Packages," in *Proceedings of the 20th Annual Conference on Mining Software Repositories — Data and Tool Showcase Track (MSR-Data'23)*, Mar. 2023.

[133] W. Jiang, J. Yasmin, J. Jones, N. Synovic, J. Kuo, N. Bielanski, Y. Tian, G. K. Thiruvathukal, and J. C. Davis, "Peatmoss: A dataset and initial analysis of pre-trained models in open-source software," *arXiv preprint arXiv:2402.00699*, 2024.

[134] C. Jones and O. Bonsignour, *The economics of software quality*. Addison-Wesley Professional, 2011.

[135] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Noria, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of artificial general intelligence: Early experiments with gpt-4," *arXiv preprint arXiv:2303.12712*, 2023.

[136] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "OPT: Open Pre-trained Transformer Language Models," *arXiv*, 2022. [Online]. Available: http://arxiv.org/abs/2205.01068

[137] Z. Sun, X. Du, F. Song, M. Ni, and L. Li, "Coprotector: Protect open-source code against unauthorized training usage with data poisoning," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 652–660.

[138] A. M. Spence, "Monopoly, quality, and regulation," *The Bell Journal of Economics*, pp. 417–429, 1975.

[139] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," *arXiv preprint arXiv:1906.02243*, 2019.

[140] J. You, J.-W. Chung, and M. Chowdhury, "Zeus: Understanding and optimizing GPU energy consumption of DNN training," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 119–139.