ReLESS: A Framework for Assessing Safety in Deep Learning Systems

Nan Jia[1,*], Anita Raja[1,2] and Raffi Khatchadourian[1,2]

[1]*CUNY, the Graduate Center, 365 Fifth Avenue, New York, NY 10016, USA*
[2]*CUNY, Hunter College, 695 Park Avenue, New York, NY 10065, USA*

Abstract
Traditionally, software refactoring helps to improve a system's internal structure and enhance its non-functional features, such as reliability and run-time performance, while preserving external behavior including original program semantics. However, in the context of learning-enabled software systems (LESS), e.g., Machine Learning (ML) systems, it is unclear which portions of a software's semantics require preservation at the development phase. This is mainly because (a) the behavior of the LESS is not defined until run-time; and (b) the inherently iterative and non-deterministic nature of ML algorithms. Consequently, there is a knowledge gap in what refactoring truly means in the context of LESS as such systems have no guarantee of a predetermined *correct* answer. We thus conjecture that to construct robust and safe LESS, it is imperative to understand the flexibility of refactoring LESS compared to traditional software and to measure it. In this paper, we introduce a novel conceptual framework named *ReLESS* for evaluating refactorings for supervised learning by (i) exploring the transformation methodologies taken by state-of-the-art LESS refactorings that focus on singular metrics, (ii) reviewing informal notions of semantics preservation and the level at which they occur (source code vs. trained model), and (iii) empirically comparing and contrasting existing LESS refactorings in the context of image classification problems. This framework will set the foundation to not only formalize a standard definition of semantics preservation in LESS but also combine four metrics: *accuracy, run-time performance, robustness,* and *interpretability* as a multi-objective optimization function, instead of a single-objective function used in existing works, to assess LESS refactorings. In the future, our work could seek reliable LESS refactorings that generalize over diverse systems.

Keywords
learning-enabled software systems, machine learning systems, refactoring, trusted AI software architectures, AI safety

1. Introduction

Developers of Learning-Enabled Software Systems (LESS) face the challenge of constructing highly reliable large-scale systems, as evidenced in previous research [1, 2]. With the pervasive integration of dynamic Machine Learning (ML) models in these operational software systems, safety, efficiency, and adaptability with respect to evolving user requirements become paramount. Moreover, software systems inherently evolve throughout their life-cycle [3], which traditionally incurs substantial costs and risks, particularly in the context of large, complex systems [4]. Although LESS shares these traits with conventional software, its data-driven nature accentuates the propensity for evolution [5]. This divergence from traditional software poses unique challenges for testing and verification due to its data-driven and uncertain requirements. Notably, the efficacy of resulting ML models, including Large Language Models (LLMs), improves with more extensive data inputs, necessitating a delicate balance between user privacy protection and model refinement in large-scale systems. Consequently, there arises a pressing need for validation and testing methodologies tailored to the distinctive characteristics of AI-driven systems.

This evolving research agenda underscores a critical reassessment of priorities in AI system development. Furthermore, as AI technologies permeate various sectors of society, scalable systems must effectively consider and adapt to legal, policy, and employment implications. These technical attributes not only underpin the functional aspects of AI applications but also facilitate their alignment with essential ethical standards and societal expectations [6]. This imperative is further underscored by a recent U.S. government-issued Executive Order [7] and the EU AI Act [8], emphasizing the necessity for Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence. Moreover, to ensure the positive societal impact of AI systems, *accuracy*, *run-time performance*, *robustness*, and *interpretability* are crucial technical attributes that directly support broader ethical objectives.

Recent works [1, 9] have highlighted a variety of metrics for assessing the impacts of LESS transformation. These metrics include aspects such as ensuring safety and fairness, protecting privacy, fostering collaboration, considering legal and policy ramifications, and evaluating impacts on employment. Recent studies [10, 11, 12, 13, 14] have investigated whether original and transformed systems should behave consistently before and after transformation. These studies illustrate the potential trade-offs between *accuracy* and each respective metric. Although various metrics like fairness and privacy are considered, in this work, we focus on *accuracy*, *run-time performance*, *robustness*, and *interpretability* as a starting point with the intent to cover the majority of AI safety concerns in LESS [14, 15, 16]. We argue that comprehending and harnessing the flexibility of refactoring in LESS represents a pivotal stride toward enhancing the safety of AI systems.

A detailed exposition of these metrics, as discussed in the state-of-the-art literature, is provided in Section 2.

Traditionally, the criterion for refactoring [17, 18], is that the same input must produce the same output; any deviation is considered a behavior change of the program and a threat leading to system crash [19]. However, refactoring is underexplored in the context of LESS, including deep learning frameworks [1]. LESS, unlike traditional software systems, benefit from randomness but yet lack a guarantee of a predefined exact outcome due to their reliance on the quantity and quality of data, complicating predictions about the effects of refactoring.

This paper aims to bridge the knowledge gap between refactoring practices in traditional software [4, 20, 21, 22, 23,

The IJCAI-2024 AISafety Workshop
*Corresponding author.
njia@gradcenter.cuny.edu (N. Jia); anita.raja@hunter.cuny.edu (A. Raja); khatchad@hunter.cuny.edu (R. Khatchadourian)

24, 25, 26, 27], and LESS [12, 28, 29, 30] by introducing *ReLESS* (**Re**factoring of **L**earning-**E**nabled **S**oftware **S**ystems), an evaluation framework for standardizing and formalizing refactoring methodologies. In this work, we describe this framework in the context of supervised learning tasks, specifically image classification problems. **Our hypothesis posits that the criteria for successful refactoring—namely source-to-source transformation and semantic preservation—assume unique, yet complementary implications in the context of LESS as opposed to traditional software systems.**

Specifically, *ReLESS* will allow for the possibility that transformations might produce outputs that are slightly different from the original output as long as they lead to improvements in other performance metrics of the system. Determining how "different" the output can be from the original is a research question we seek to address. Moreover, our approach aims to discover and preserve safety-critical metrics during *ReLESS* while further mitigating the uncertainties introduced by their non-deterministic nature. While current approaches emphasize knowledge distillation (transferring knowledge from a large neural network to a smaller, resource-efficient one) and regularization (a technique for solving over-fitting), our vision for the future of *ReLESS* includes approaches that combine connectionist models (e.g., neural networks) and symbolic (e.g., decision tree) approaches as well as Bayesian and analogizer (e.g., K-nearest neighbor, support vector machine) approaches.

This paper is structured as follows: in Section 2 we first provide a comprehensive analysis of state-of-the-art refactoring methodologies in LESS and discuss how these works trade-off *accuracy* with respect to specific metrics such as *run-time performance*, *robustness* [11, 13, 14, 16], or *interpretability*. In Section 3, we contrast existing practices and scrutinize informal notions of semantics preservation across different levels (source code vs. trained model). We then motivate a novel thread of inquiry for the *ReLESS* evaluation framework and its multi-objective optimization function that combines the aforementioned multiple metrics to guarantee the AI system's safety. Section 4 presents preliminary experiments utilizing *ReLESS* to gauge LESS safety and associated parameters. Finally, in Section 5 we discuss the main insights gleaned from this work and our future work.

2. Related Work

In recent years, various research has been conducted on LESS refactoring, with significant observations in balancing single metric against *accuracy* of models. Several studies have focused on image classification or object detection, addressing this tension and presenting innovative verification. However, these approaches often face limitations in lack of generalization and narrow scope of metrics, which we aim to address in our work.

2.1. Refactoring Types in Software Development

Refactoring [17], a well-known technique for the evolution and maintenance of traditional software, alters a system's internal structure without changing its behavior [18] to improve non-functional characteristics such as run-time performance, security, and modularity, and to pay down technical debt [31, 32, 33, 34, 35]. It can be considered as a series of typically automatic procedures for modifying code, such as variable name changes to enhance comprehension [19], without an explicit focus on automated refactoring, as these modifications frequently occur automatically within a system-based environment. Formally, a refactoring is a program transformation potentially spanning multiple, non-adjacent program statements or expressions that is: (i) source-to-source and (ii) semantics-preserving, i.e., the behavior of the program is the same before and after the refactoring.

Even though refactoring is a well-established practice in traditional software development, it is not as well clear in LESS. Existing refactoring attempts in LESS are implicitly performed via controlling randomness [11], decomposing trained models [12], or defining new requirements [13]. The lack of refactoring tools and techniques, and an evaluation framework for LESS is a significant challenge for developers and researchers [2]. Our research aims to develop a multi-objective evaluation framework for LESS. We study it in the context of a specific class of supervised learning problems, namely image classification tasks.

2.2. Image Classification Problems and Evaluation

While the continuous evaluation of ML models [11, 12, 13, 36] has highlighted modularity, reliability, robustness, and interpretability, these assessments done independently fall short of ensuring the safety of AI systems as a whole. Consider for instance the role of *accuracy*, which is the widely accepted metric [37] for gauging the success of models in the image classification task. Benchmark models for this problem class originating from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) have continually improved on *accuracy*. To date, the record for the highest *accuracy* on the ImageNet benchmark is an impressive 92.4%, set by OmniVec(ViT)[1]. However, while high *accuracy* is indispensable, it is not the sole criterion for the adequacy of a model, especially within contexts of safety-critical applications. In such applications, other non-functional metrics demand equal consideration to ensure the comprehensive robustness and reliability of LESS.

Ensuring AI systems maintain safety and fairness [38] across various conditions and inputs is crucial for applications like autonomous driving and medical diagnosis [13, 16, 39]. Reliability is equally important, as dependable systems yield consistent results, fostering trust among stakeholders and accountability among developers. While state-of-the-art models often match or exceed human performance in image classification tasks, understanding errors and their solutions remains challenging [40]. Evaluating model performance is vital, especially in safety-critical scenarios, yet the opaque nature of the learning component hinders transparency and interpretability.

Our proposed framework *ReLESS* combines *accuracy, run-time performance, robustness*, and *interpretability*, using a multi-objective optimization function. By experimenting with metrics drawn from existing literature and through preliminary evaluations of them, we validate target systems' performance both before and after refactoring. Our findings illuminate the trade-offs researchers make between *accuracy* and other performance metrics. Importantly, our evaluation

[1]https://paperswithcode.com/paper/omnivec-learning-robust-representations-with

process considers not just a single metric versus *accuracy* but integrates multiple metrics to understand various system maintenance challenges. This approach helps mitigate the "black-box" nature of AI learning components, providing clearer insights into system behavior and performance.

2.3. Baselines for Comparison

Chen et al. [11] analyzed refactoring for image classification tasks at the algorithmic level with various models using dynamic analysis, record-and-replay, and profile-and-patch. The focus of their approach is to control randomness and hardware non-determinism to guarantee that the $Output$ and performance metrics are the same as the original system in seven models (Lenet1/4/5, ResNet-38/56, WRN-28-10, and ModelX). Models are then reproduced efficiently and accurately across different hardware.

Pan and Rajan [12] hypothesized that decomposing learning models into reusable components can affect refactoring outputs and statistical performance in the MNIST [41] dataset. They run four DNN models across sixteen experiments with varying hidden layers and datasets, demonstrating that removing irrelevant edges in the network can lead to similar *accuracy* and preserve the most semantics. They found that 9 out of 16 cases were functionally equivalent to the original models, based on the Jaccard Index, with intra-dataset performance from decomposed models slightly outperforming models built from scratch (e.g., MNIST(+0.30%)).

Adopting the methodology from Hu et al. [13], we succeeded in obtaining the original and filtered images from ImageNet [42]. Image filters such as brightness, contrast, defocus/blur, frost, gaussian noise, and jpeg compression, are crucial for testing the robustness of the refactored systems [43] because they involve pictures that human can recognize correctly and easily before and after filtering, thus setting a baseline for model performance in similar conditions.

3. Methodology

Given the context of refactoring in ML systems as discussed in the previous section, we present *ReLESS*, a conceptual framework created especially to tackle two important research goals. First, to investigate the definition and operation of semantic preservation during system transformation procedures inside the LESS. Second, to explore approaches for evaluating the safety of LESS during the system's transition, building on our formalization of semantic preservation from the previous goal.

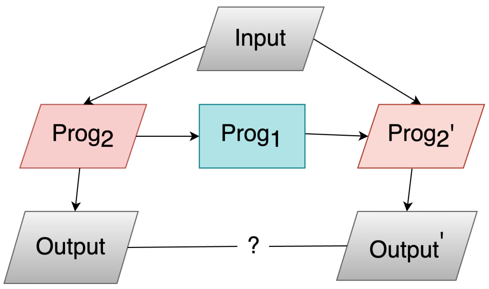

Figure 1: Traditional refactoring. $Prog_1$ is the refactoring. $Prog_2$ and $Prog_2'$ are programs to be refactored and the refactored program, respectively. $Output$ and $Output'$ are the outputs of $Prog_2$ and $Prog_2'$, respectively, given $Input$. Assume $Prog_2 \neq Prog_2'$.

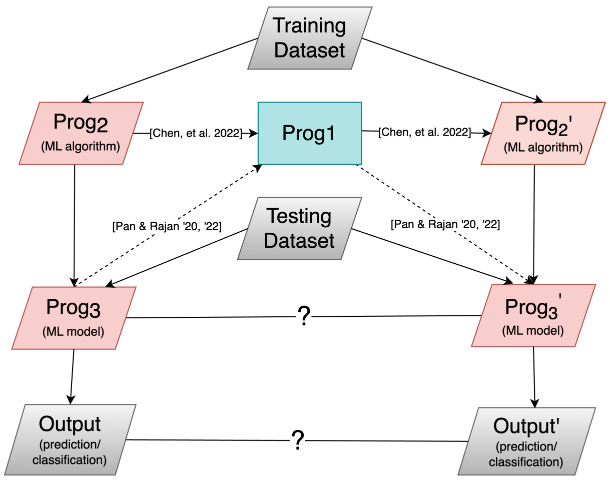

Figure 2: Refactoring LESS. $Prog_1$ is the refactoring. $Prog_2$ and $Prog_2'$ is the ML algorithm to be refactored and the refactored ML algorithm, respectively. Assume $Prog_2 \neq Prog_2'$. $Prog_3$ and $Prog_3'$ are the outputs (trained models) of $Prog_2$ and $Prog_2'$, respectively, given a training set. $Output$ and $Output'$ are the outputs (predictions/classifications) of $Prog_3$ and $Prog_3'$, respectively, given a testing set.

Consider Fig. 1 which depicts the situation representing the refactoring of traditional software systems. Here, $Prog_1$ represents an (automated) refactoring that takes as input a program $Prog_2$ to be refactored and produces a refactored program $Prog_2'$. Note that $Prog_2$ and $Prog_2'$ are *source* code, i.e., textual descriptions. We assume that $Prog_1$ is a non-trivial refactoring, i.e., that $Prog_2 \neq Prog_2'$. As refactoring typically deals with real-world languages with non-trivial semantics, the semantic equivalence of $Prog_2$ and $Prog_2'$ is normally assessed empirically by executing $Prog_2$'s test suites and comparing the results. Thus, to evaluate the refactoring $Prog_1$, $Input$ is fed to both $Prog_2$ and $Prog_2'$ for all test suite inputs. The $Output$ is then compared—ideally, all tests have the same results before and after the refactoring. If so, then $Output = Output'$, and $Prog_1$ is considered validated. Otherwise, $Output \neq Output'$, meaning there is a bug [18, 19] in the system. Since traditional software is typically deterministic and its logic is not driven by dynamic data models, the process works in a relatively straightforward fashion. In fact, the larger the test suite, the greater the confidence that the refactoring works.[2] On the other hand, given the non-deterministic intricacies inherent in LESS, the traditional refactoring as described in Fig. 1 is insufficient. Consequently, we construct an auxiliary diagram Fig. 2 that facilitates a more direct and nuanced evaluation of the transformations.

Now consider Fig. 2 representing *ReLESS* with citations of related work in the supervised learning context. [3] Here, $Prog_1$ represents an (automated) refactoring that takes as input an ML algorithm $Prog_2$ to be refactored and produces a refactored ML algorithm $Prog_2'$. Note that $Prog_2$ and $Prog_2'$ are ML algorithm *source* code, i.e., textual descriptions. We

[2]Traditional software may be concurrent, potentially experiencing race conditions, or may rely on its (changing) environment. In such cases, "flaky" tests may arise, which would challenge refactoring validation. In this case, the test suites can be executed several times to identify stable tests.

[3]While our current investigation focuses on supervised learning, we plan to extend the framework to other types of learning (unsupervised, reinforcement) as part of of our future work.

again assume that $Prog_1$ is a non-trivial refactoring, i.e., that $Prog_2 \neq Prog_2'$. To evaluate the refactoring $Prog_1$, two steps are taken: (a) a training dataset is fed to both $Prog_2$ and $Prog_2'$, which then produces the compiled, aka trained ML models $Prog_3$ and $Prog_3'$, respectively; (b) an evaluation (testing) dataset is fed to both $Prog_3$ and $Prog_3'$. One or more such datasets (both training and evaluation) may be used. The *Output*—in this case, predictions or classifications—is then compared. If $Prog_1$ results in no *accuracy* loss, then $Output = Output'$. Otherwise, $Output \neq Output'$, meaning $Prog_1$ causes some *accuracy* loss when refactoring $Prog_2$. Note that unlike in the traditional refactoring evaluation case, whether there is a bug in $Prog_1$ in this situation is not straightforward to determine and is not a topic of focus in this paper. Because LESS can be non-deterministic and has logic that is driven by dynamic data models, whether $Prog_1$ is considered valid may depend on multiple factors. For instance, if the *accuracy* loss is within a certain threshold, then $Prog_1$ may be considered valid. If the *accuracy* loss is above the threshold, then $Prog_1$ may be considered invalid.

A supplementary contribution of our proposed framework is that it has an additional layer where both the transformation and output comparison could occur. For instance, there is a dashed line in Fig. 2 from $Prog_3$ to $Prog_1$ and $Prog_1$ to $Prog_3'$, indicating that the program transformation can also take place on the *trained* ML models. In the traditional setting (Fig. 1), because the transformation is not source-to-source, it would not be considered a refactoring in the traditional sense but instead viewed as compiler optimization. However, in the LESS context, ML algorithms are typically written in interpreted languages (e.g., Python), where a compiler is not involved. It is because the model training (compilation) process can potentially be lengthy (days or even weeks) depending on the dataset size, transforming the ML algorithm to produce a new ML model as part of the refactoring process can be time-consuming [44]. Instead, it may be advantageous in this context to perform the refactoring at the testing level to avoid retraining. Such a "refactoring" is done on LESS by Pan and Rajan [12, 45]. Although the transformation is on the trained ML model, their goal of enhanced modularity is a classical refactoring outcome.

3.1. Determination of Semantic Equivalence

Our objective is to ultimately build a tool, where users provide original code (old system), that determines which refactorings (new systems) would satisfy semantic equivalence. We identify different levels at which this could occur: semantic equivalence at: (a) the ML algorithm level (case 1), and (b) the ML model level (case 2). We will demonstrate how existing works perform semantic equivalence from a single-lens point of view. Drawing on these effects, however, our approach will create a multi-objective evaluation (instead of a single-objective function used by the current state-of-the-art).

Case 1: Semantic Equivalence at the ML Algorithm Level

$Prog_2 = Prog_2'$: This equivalence implies that $Prog_3$ and $Prog_3'$ are also semantically equivalent as shown in Fig. 2. In this case, $Prog_2$ and $Prog_2'$ are semantically equivalent since $Output = Output'$. But, the average training time (in hours) of the model for this refactoring in Chen et al. [11] increases from 0.017 to 0.023 for Lenet1 and from 7.08 to 14.979 in the case of ModelX. Their approach has higher storage overhead for $Prog_3'$ (due to random seed recording). Such an approach does not facilitate model generalization to unseen data by making the training process explore various possibilities. This will constrain the *robustness* of an ML model. Deterministic methods are also more susceptible to overfitting, as models can memorize the training data too closely, limiting their performance on new data. Lastly, ensuring complete determinism can be computationally expensive and challenging, especially in complex, multi-threaded, or distributed computing environments. This work highlights the tension between semantic preservation and model optimization.

Case 2: Semantic Equivalence at the ML Model Level

$Prog_3 = Prog_3'$: This means that the trained ML models *are* the same. It follows again that $Output = Output'$, and $Prog_2$ and $Prog_2'$ are semantically equivalent in the traditional sense. As we are considering non-trivial refactorings as discussed earlier, we assume $Prog_2 \neq Prog_2'$, meaning that the refactoring $Prog_1$ has made some non-trivial transformation. An example of such a transformation would be to enhance the *run-time performance* of the training; the trained model would be the same but the training process would be faster. For instance, Castro Vélez et al. [46] show that the $Prog_3'$ run-time is ~9.22 seconds faster than $Prog_3$ by applying a hybrid training technique in imperative Deep Learning (DL) programs. In TensorFlow 2 [47], for example, the `tf.function` decorator can be applied to certain (model) Python functions found in imperative code to speed up the training process. Developers and scientists, then, can write natural, debuggable DL code in an imperative style while retaining the run-time performance typically found in legacy DL frameworks that support deferred-execution style programming models. Applying `tf.function` to (otherwise eagerly-executed) imperative DL code can be—if done correctly—a semantics-preserving refactoring [46].

$Prog_3 \neq Prog_3'$: This means that the trained models are *not* the same. It follows that it is possible that $Output \neq Output'$, meaning that $Prog_2$ and $Prog_2'$ may not be semantically equivalent in the *traditional* sense. There are several situations that may occur here, e.g., (i) different hyperparameters are used. (ii) hybridization is misused, resulting in semantically–in-equivalent code [46], (iii) $Prog_3'$ may be an optimized DL model, e.g., having fewer edges, being more modular, and avoiding over-fit. In Fig. 2, $Prog_3'$ represents a modular and refactored system from $Prog_3$ via $Prog_1$, where semantics is preserved through separation of concerns, such as using supervised classification labels for maintenance and reduced model training time [12]. This indicated that $Prog_3'$ does better than $Prog_3$ with respect to *ReLESS* optimization while preserving the potential to explore generalizability and scalability.

Our analysis not only sheds light on the current state-of-the-art but also establishes a linkage between program transformation techniques and their operational viability in scenarios where safety is of paramount concern. We then use these observations to formally define semantic preservation using a multi-objective optimization function rather than a single-objective one in LESS.

3.2. Semantic Preservation: Formal Definition and Verification Metrics

We first define the semantic preservation of LESS based on varying ranges of the output. The Venn diagram Fig. 3 shows the outputs from the original code and proposed *ReLESS*. The upper circle in blue is the output from the original code, e.g., the probability of correct labels for a classification or prediction task. The lower circle in yellow is the output from *ReLESS*. This diagram examines where the two outputs are equivalent (overlapping area) and where they are different. Suppose δ is the acceptable range of overlap, i.e., how much developers/engineers/scientists are willing to trade *accuracy* with other factors viz. *robustness*, *run-time performance*, *interpretability* etc. Ideally, the overlapped area should be as large as possible, but this is not always the case and is application-dependent. For instance, if the system is time-critical, then the response time is emphasized in the optimization even though there are marginal *accuracy* losses. If the system is safety-critical, then the *accuracy* should be preserved as much as possible. That said, we posit that to achieve semantic preservation in *ReLESS*, it is inadequate to consider *accuracy* as the sole optimization metric.

Prior works [13, 38] has formalized balancing between *accuracy* and reliability/robustness and fairness. OBrien et al. [48] define non-functional LESS metrics as run-time performance (speed), security, privacy, and memory (storage). Building upon these foundational studies, we extend the evaluation framework for semantic preservation to explicitly encompass safety as an overarching theme. *Run-time performance*, as highlighted by OBrien et al. [48], serves not only as a measure of efficiency but also influences system safety by ensuring timely responses in critical scenarios. *Robustness*, as documented by Hu et al. [13], is directly linked to safety, reflecting the system's capacity to withstand errors and adversities. Finally, *interpretability*, introduced by [36], enhances safety by providing clarity on decision-making processes, thereby allowing for greater accountability and easier identification of potential safety breaches. These three metrics collectively forge a more resilient and safety-conscious framework for assessing semantic preservation in LESS.

This tailored approach allows for a more integrated and holistic assessment of LESS, aligning closely with contemporary LESS development and deployment needs. All three

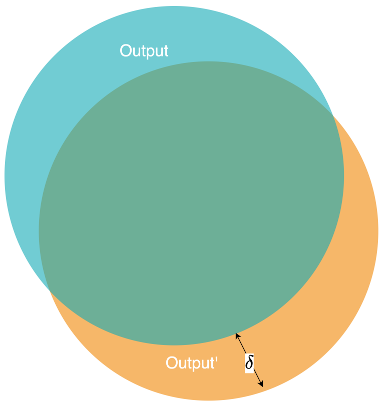

Figure 3: $|Output'| \setminus |Output|$. $Output$ and $Output'$ are supervised classification tasks' labels from the original and refactored models, respectively. $|\delta| = |Output' \setminus Output|$.

transformations do not change LESS's external behavior (semantics) [49]. The formal notation is built to combine *accuracy* with those non-functional metrics with customized importance factors to guide which degree of flexibility the engineers, scientists, and researchers want the model they work on to emphasize. We propose a multi-objective optimization function, akin to Nguyen et al. [38]'s approach, to determine the difference (loss function) between a LESS and its corresponding *ReLESS*. We argue that if the loss is below a certain threshold with constraints (as discussed in Fig. 3), then semantic preservation is maintained.

As one of the state-of-the-art formal methods, optimization via loss functions is central to the training of ML/DL models [38]. It is recognized for its adaptability to a wide range of applications. Different trade-offs exist when refactoring in ML/DL systems, so a multi-objective optimization function is constructed. Besides, optimization can standardize each metric term that needs to be balanced with *accuracy* in loss function to make the whole system understandable to the target audience. The range of optimization applied is from classical ML models (random forest, gradient boost) to DNN models with supported libraries, e.g., auto-sklearn [50] and AutoKeras [51].

3.2.1. Accuracy, Run-time Performance, Robustness, and Interpretability

To comprehensively evaluate the performance of *ReLESS*, we will consider *accuracy* with three key loss functions: *run-time performance*, *robustness*, and *interpretability*.

a. **ACC**uracy is the number of correct outputs over the total number of instances.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

where TP and TN are the number of positive instances and negative instances correctly classified, and FP and FN are the number of instances incorrectly classified.

b. **R**un-**T**ime **P**erformance **I**mprovement (*RTPI*) is determined by comparing the observed run-time of the original (old) code and new (transformed) code.

$$RTPI = \frac{RTP_{old} - RTP_{new}}{RTP_{old}} \tag{2}$$

c. **ROB**ustness **I**mprovement is indicated as *ROBI*.

$$ROBI = \frac{1}{D} * \Sigma_{x,y \in D}(1 - \frac{Loss_{new}(y,f) - Loss_{old}(y,f)}{Loss_{old}(y,f)}) \tag{3}$$

where D is the input dataset, x is the training dataset, and y is the set of corresponding labels for a supervised learning task, such as image classification. Similar to *RTPI*, the observed difference is captured in *loss* function in old and new models. *ROBI* is observed by the difference in the loss function between the old and new models. Our definition for robustness is based on [10, 52], where a robustness system after refactoring is verified by its loss function after adversarial training, and for classical ML after refactoring can be also verified by its loss function after data augmentation, feature engineering, and ensemble learning.

d. **INT**erpretability **I**mprovement is indicated as *INTI*.

$$INTI = \frac{1}{|D_{subset}|} * \Sigma_{x,y \in D_{subset}} Loss(y, f_{explainable(x)}) \tag{4}$$

Molnar [53] define interpretability in machine learning using interpretable models and a simplified loss function. The loss function serves as a quantitative measure to compare the interpretability of different models while maintaining *accuracy*. This approach blends the conceptual understanding of model behavior (through interpretable models) with a practical, measurable way (using the loss function) to assess and compare the clarity and comprehensibility of different models. We compute this metric using the definition of interpretability where a subset as input of $\mathcal{D}$. We are able to compare the difference between new and old models and the corresponding interpretability score [36]. The implication is that the simplified loss function on the explainable refactored system correlates with higher interpretability, which is plausible, but the exact method of determining explainability is essential here.

To sum up, we define a multi-objective optimization loss function that facilitates balancing the importance of various objectives depending on the application domain. Each metric is formulated as a ratio or a normalized value, which is typical in performance evaluation to provide a standardized measure of improvement or degradation. In the equation below, each metrics term is the loss measurements calculated from Eqs. (2) to (4) respectively for *ROBI*, *RTPI*, and *INTI* metrics respectively and *accuracy* is *ACC*; $\mathcal{M}_\theta$ is the model with its parameters, $\mathcal{D}$ is the dataset. Each term's weight coefficient ω_i is assumed to be user-defined and indicates the importance of each of the metrics during model evaluation.

$$\begin{aligned} min\mathbb{L}(\mathcal{M}_\theta, \mathcal{D}) = \omega_1 \times \mathrm{ACC} \\ + \omega_2 \times \mathrm{RTPI} \\ + \omega_3 \times \mathrm{ROBI} \\ + \omega_4 \times \mathrm{INTI} \end{aligned} \tag{5}$$

where $\omega_1, \omega_2, \omega_3$, and ω_4 are weights that reflect the importance of each term in the loss function.

The multi-objective optimization function in this formalism enables the determination of whether a *ReLESS* is a semantically preserving transformation to its LESS. Moreover, when fusing these measurements, it is also essential to include the measure of *accuracy* because regardless of the importance of the speed of operation, robustness, and interpretability, producing correct outputs is the cornerstone of model evaluation. In other words, *accuracy* is always a first-class objective. Only by considering the critical role of *accuracy* can we ensure that a model is trustworthy [54].

Expanding on the conceptual structure presented in the preceding part, we describe a preliminary experimental configuration intended to closely assess the *accuracy, run-time performance, robustness*, and *interpretability* of LESS.

4. Experimental Setup

This section describes the experimental setup employed for a preliminary evaluation of the proposed *ReLESS* framework for a simple case study. We describe the datasets used for experiments, followed by an explanation of the experimental design and the metrics adopted to assess the efficacy of the refactorings.

4.1. Datasets and Models

As indicated in Section 2, we study *ReLESS* in the context of two image classification datasets: the ImageNet dataset and the MNIST dataset. The ImageNet dataset [42], comprised of 1.2 million images across 1000 categories, is utilized for the evaluation to assess *reliability* and *robustness*, with a specific subset of 50,000 images filtered from [43]. The MNIST dataset [41], containing 60,000 training images and 10,000 test images of handwritten digits, serves as the basis for initial evaluations. Those datasets enable preliminary assessments of the refactorings' effectiveness before proceeding to more complex scenarios. Our experimental models include fully connected neural networks with 1 to 4 layers for the MNIST dataset, and pre-trained complex architectures such as AlexNet [55], ResNet50 [56], VGG16 [57], and GoogleNet [58] for the ImageNet dataset.

4.2. Experiment and Results

In our experimental setup, we applied the methodologies outlined by Pan and Rajan [12] and Hu et al. [13], along with techniques detailed in Section 3, across both datasets to scrutinize the refactored systems with respect to *accuracy, run-time performance, robustness*, and *intepretebility*. The results of experiments are summarized in Table 1.

From Table 1, we observe that the refactored models exhibit a marginal decrease in *accuracy* on the MNIST dataset, with a difference of 0.001. This decrease is attributed to the expanded modular complexity, which results in a run-time increase of 414.7 seconds. The modularity of the refactored model is significantly higher than the original model, with a difference of 8. The robustness of the refactored model is also higher, with a difference of 2.0476. The interpretability of the refactored model is higher, with an *accuracy* difference of 0.0769. Increases in both metrics indicate that the refactored system exhibits improved robustness and interoperability after decomposing. However, although the robustness has improved, the *accuracy* for refactored systems using the ImageNet dataset has decreased, falling below that of a coin flip. Therefore, modularity appears not only to be harmless but also beneficial to system safety, as it maintains *accuracy* and improves robustness. But, for the optimization of the aforementioned complex systems, more efforts are required to prevent *accuracy* loss, particularly in safety-critical tasks. More details can be found in https://github.com/NanJ90/ReLess-testing-tool

To summarize, we present an initial assessment for *ReLESS* evaluation framework and describe the datasets used, the experimental design, and the metrics for evaluating refactorings. The comparative analysis of original and refactored models reveals that different datasets and models can exhibit significant variations across different performance metrics. For instance, while the performance of the ImageNet model remained relatively consistent after speedup, the modularized MNIST model took 168 times longer than the original. This underscores the critical importance of evaluating effects across multiple datasets and models to gain comprehensive insights into performance implications w.r.t *accuracy*.

5. Conclusions and Future Work

Our contribution in this work includes a review of literature focused on refactoring in LESS, particularly with an emphasis on safety considerations. This review critically analyzes the spectrum of assessments presented across various studies, each contributing to a facet of the AI safety standard. We further explore and elucidate the interrela-

Table 1
Performance comparison of the original and refactored models on the MNIST and ImageNet datasets.

Metric	MNIST			ImageNet		
	Original	Refactored	Difference	Original	Refactored	Difference
Accuracy	0.9491	0.9490	0.0001	0.7948	<0.5	>0.2948
Run-time (seconds)	4.2	419.3	414.7	163.7	174	10.3000
Robustness	0.1744	2.2220	2.0476	0.9453	10.0754	9.1301
Interpretability (accuracy)	0.7882	0.8651	0.0769	<0.5	<0.5	0

tionships between these safety metrics and the *accuracy* of AI systems, highlighting the implications for model development and deployment. Our preliminary results set a potential foundation to help drive the long-term evolution, and robustness of LESS that are traditionally enjoyed by conventional systems during development and deployment, and then improve the safety of LESS. The scientists and engineers who develop AI systems will be able to rely on the refactored systems and trust them to make decisions that are safe, secure, and trustworthy. This work includes understanding how the thresholds in Fig. 3 will be determined for various applications and how the user can determine the weights for the various metrics. We have described an initial validation of our framework; however, further experimentation that includes more metrics such as fairness and privacy, extending the validation to a variety of problem domains and case studies is essential to comprehensively assess its effectiveness and generalizability. This would also enable practitioners to prioritize specific components when evaluating LESS and could even lead to design-to-criteria LESS.

Acknowledgments

We thank Ayan Kohli for the initial investigation into semantic similarity and the anonymous reviewers for their helpful comments. This work is supported by the National Science Foundation (NSF) under Agreement No.CCF-2200343.

References

[1] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, S. Wagner, Software Engineering for AI-Based Systems: A Survey, ACM Transactions on Software Engineering and Methodology 31 (2022) 1–59. URL: http://arxiv.org/abs/2105.01984. doi:10.1145/3487043. arXiv:2105.01984.

[2] E. Breck, S. Cai, E. Nielsen, M. Salib, D. Sculley, The ML test score: A rubric for ML production readiness and technical debt reduction, in: 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 1123–1132. doi:10.1109/BigData.2017.8258038.

[3] ISO/IEC 14764, Software Engineering – Software Life Cycle Processes – Maintenance, International Organizations for Standardization, Geneva, Switzerland, 2006.

[4] D. Dig, J. Marrero, M. D. Ernst, Refactoring sequential java code for concurrency via concurrent libraries, in: International Conference on Software Engineering, IEEE, 2009, pp. 397–407. doi:10.1109/icse.2009.5070539.

[5] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, D. Dennison, Hidden technical debt in Machine Learning systems, in: Neural Information Processing Systems, volume 2 of *NIPS '15*, MIT Press, 2015, pp. 2503–2511.

[6] J. Dolby, A. Shinnar, A. Allain, J. Reinen, Ariadne: Analysis for machine learning programs, in: International Workshop on Machine Learning and Programming Languages, MAPL 2018, ACM SIGPLAN, ACM, New York, NY, USA, 2018, pp. 1–10. doi:10.1145/3211346.3211349.

[7] Executive order on the safe, secure, and trustworthy development and use of artificial intelligence, 2023. URL: https://www.whitehouse.gov/briefing-room/presidential-actions/2023/10/30/executive-order-on-the-safe-secure-and-trustworthy-development-and-use-of-artificial-intelligence/.

[8] T. Madiega, Artificial intelligence act, European Parliament: European Parliamentary Research Service (2021). URL: https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/698792/EPRS_BRI(2021)698792_EN.pdf.

[9] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach., 4 ed., Pearson, 2020. doi:https://doi.org/10.1007/978-3-030-82681-9.

[10] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, A. Madry, Robustness may be at odds with accuracy, 2019. arXiv:1805.12152.

[11] B. Chen, M. Wen, Y. Shi, D. Lin, G. K. Rajbahadur, Z. M. J. Jiang, Towards training reproducible deep learning models, in: International Conference on Software Engineering, ICSE '22, Association for Comput-

ing Machinery, New York, NY, USA, 2022, pp. 2202–2214. doi:10.1145/3510003.3510163.

[12] R. Pan, 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, ACM, 2020, pp. 889–900. URL: https://dl.acm.org/doi/10.1145/3368089.3409668. doi:10.1145/3368089.3409668.

[13] B. C. Hu, L. Marsso, K. Czarnecki, R. Salay, H. Shen, M. Chechik, If a Human Can See It, So Should Your System: Reliability Requirements for Machine Vision Components, in: Proceedings of the 44th International Conference on Software Engineering, 2022, pp. 1145–1156. URL: http://arxiv.org/abs/2202.03930. doi:10.1145/3510003.3510109. arXiv:2202.03930.

[14] J. Huang, V. Rathod, C. Sun, M. Zhu, A. K. Balan, A. Fathi, I. S. Fischer, Z. Wojna, Y. Song, S. Guadarrama, K. P. Murphy, Speed/accuracy trade-offs for modern convolutional object detectors, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 3296–3297. URL: https://api.semanticscholar.org/CorpusID:206595627.

[15] S. A. Seshia, A. Desai, T. Dreossi, D. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, X. Yue, Formal Specification for Deep Neural Networks, Technical Report UCB/EECS-2018-25, EECS Department, University of California, Berkeley, 2018. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-25.html.

[16] Y. Zhuo, Z. Song, Z. Ge, Security versus accuracy: Trade-off data modeling to safe fault classification systems, IEEE Transactions on Neural Networks and Learning Systems (2023) 1–12. doi:10.1109/TNNLS.2023.3251999.

[17] W. F. Opdyke, Refactoring object-oriented frameworks, Ph.D. thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1992.

[18] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, Boston, MA, USA, 1999.

[19] W. G. Griswold, W. F. Opdyke, The birth of refactoring: A retrospective on the nature of high-impact software engineering research 32 (2015) 30–38. doi:10.1109/MS.2015.107, conference Name: IEEE Software.

[20] M. Kim, T. Zimmermann, N. Nagappan, A field study of refactoring challenges and benefits, in: Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM, Cary, North Carolina, 2012, p. 1. doi:10.1145/2393596.2393655.

[21] E. A. AlOmar, M. W. Mkaouer, C. Newman, A. Ouni, On preserving the behavior in software refactoring: A systematic mapping study, Information and Software Technology 140 (2021) 106675. doi:10.1016/j.infsof.2021.106675.

[22] R. Khatchadourian, Y. Tang, M. Bagherzadeh, S. Ahmed, Safe automated refactoring for intelligent parallelization of Java 8 streams, in: International Conference on Software Engineering, ICSE '19, ACM/IEEE, IEEE, Piscataway, NJ, USA, 2019, pp. 619–630. doi:10.1109/icse.2019.00072.

[23] F. Tip, R. M. Fuhrer, A. Kieżun, M. D. Ernst, I. Balaban, B. De Sutter, Refactoring using type constraints, ACM Transactions on Programming Languages and Systems 33 (2011) 9:1–9:47. doi:10.1145/1961204.1961205.

[24] R. Khatchadourian, J. Sawin, A. Rountev, Automated refactoring of legacy Java software to enumerated types, in: International Conference on Software Maintenance, ICSM '07, IEEE, Paris, France, 2007, pp. 224–233. doi:10.1109/ICSM.2007.4362635.

[25] R. Khatchadourian, H. Masuhara, Automated refactoring of legacy Java software to default methods, in: International Conference on Software Engineering, ICSE '17, ACM/IEEE, IEEE Press, Piscataway, NJ, USA, 2017, pp. 82–93. doi:10.1109/ICSE.2017.16.

[26] R. Khatchadourian, H. Masuhara, Proactive empirical assessment of new language feature adoption via automated refactoring: The case of Java 8 default methods, in: International Conference on the Art, Science, and Engineering of Programming, volume 2 of *Programming '18*, AOSA, Nice, France, 2018, pp. 6:1–6:30. doi:10.22152/programming-journal.org/2018/2/6.

[27] T. Mens, T. Tourwe, A survey of software refactoring, IEEE Transactions on Software Engineering 30 (2004) 126–139. doi:10.1109/TSE.2004.1265817.

[28] C. Eleftheriadis, N. Kekatos, P. Katsaros, S. Tripakis, On neural network equivalence checking using SMT solvers, in: S. Bogomolov, D. Parker (Eds.), Formal Modeling and Analysis of Timed Systems, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, pp. 237–257. doi:10.1007/978-3-031-15839-1_14.

[29] M. Dilhara, A. Ketkar, D. Dig, Understanding software-2.0: A study of machine learning library usage and evolution, ACM Transactions on Software Engineering and Methodology 30 (2021). doi:10.1145/3453478.

[30] M. Dilhara, A. Ketkar, N. Sannidhi, D. Dig, Discovering repetitive code changes in python ml systems, in: International Conference on Software Engineering, ICSE '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 736–748. doi:10.1145/3510003.3510225.

[31] G. Bavota, B. Russo, A large-scale empirical study on self-admitted technical debt, in: International Conference on Mining Software Repositories, MSR '16, ACM, New York, NY, USA, 2016, pp. 315–326. doi:10.1145/2901739.2901742.

[32] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, N. Zazworka, Managing technical debt in software-reliant systems, in: FSE/SDP Workshop on Future of Software Engineering Research, FoSER '10, ACM, New York, NY, USA, 2010, pp. 47–52. doi:10.1145/1882362.1882373.

[33] B. Christians, Self-admitted technical debt–an investigation from farm to table to refactoring, 2020.

[34] E. Tom, A. Aurum, R. Vidgen, An exploration of technical debt, Journal of Systems and Software 86 (2013) 1498–1516. doi:10.1016/j.jss.2012.12.052.

[35] Y. Tang, R. Khatchadourian, M. Bagherzadeh, R. Singh, A. Stewart, A. Raja, An empirical study of refactorings and technical debt in Machine Learning systems, in: International Conference on Software Engineering, ICSE '21, IEEE/ACM, IEEE, Madrid, Spain, 2021, pp. 238–250. doi:10.1109/ICSE43902.2021.00033.

[36] Y. Li, H. Wu, H. Zhao, Semantic-preserving adversarial code comprehension, 2023. URL: http://arxiv.org/abs/2209.05130. doi:10.48550/arXiv.2209.05130.

arXiv:2209.05130 [cs].

[37] Z.-L. Zhang, Y.-Y. Chen, J. Li, X.-G. Luo, A distance-based weighting framework for boosting the performance of dynamic ensemble selection 56 (2019) 1300–1316. URL: https://www.sciencedirect.com/science/article/pii/S030645731830712X. doi:10.1016/j.ipm.2019.03.009.

[38] G. Nguyen, S. Biswas, H. Rajan, Fix fairness, don't ruin accuracy: Performance aware fairness repair using automl, arXiv preprint arXiv:2306.09297 (2023).

[39] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, C.-J. Hsieh, Towards stable and efficient training of verifiably robust neural networks, 2019. arXiv:1906.06316.

[40] S. Srivastava, G. Sharma, Omnivec: Learning robust representations with cross modal sharing, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2024, pp. 1236–1248.

[41] Y. LeCun, C. Cortes, MNIST handwritten digit database (2010). URL: http://yann.lecun.com/exdb/mnist/.

[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, International Journal of Computer Vision (IJCV) 115 (2015) 211–252. doi:10.1007/s11263-015-0816-y.

[43] D. Hendrycks, T. Dietterich, Benchmarking neural network robustness to common corruptions and perturbations, 2019. arXiv:1903.12261.

[44] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J. rong Wen, J. Yuan, W. X. Zhao, J. Zhu, Pre-trained models: Past, present and future, ArXiv abs/2106.07139 (2021). URL: https://api.semanticscholar.org/CorpusID:235421816.

[45] R. Pan, H. Rajan, Decomposing convolutional neural networks into reusable and replaceable modules, in: International Conference on Software Engineering, number arXiv:2110.07720 in ICSE '22, arXiv, New York, NY, USA, 2022, pp. 524–535. doi:10.1145/3510003.3510051. arXiv:2110.07720.

[46] T. Castro Vélez, R. Khatchadourian, M. Bagherzadeh, A. Raja, Challenges in migrating imperative deep learning programs to graph execution: An empirical study, in: International Conference on Mining Software Repositories, MSR '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 469–481. doi:10.1145/3524842.3528455.

[47] Google LLC, Better performance with tf.function, 2021. URL: https://tensorflow.org/guide/function.

[48] D. OBrien, S. Biswas, S. M. Imtiaz, R. Abdalkareem, E. Shihab, H. Rajan, 23 Shades of Self-Admitted Technical Debt: An Empirical Study on Machine Learning Software (2022) 13.

[49] T. Mens, S. Demeyer, B. Du Bois, H. Stenten, P. Van Gorp, Refactoring: Current research and future trends, Electronic Notes in Theoretical Computer Science 82 (2003) 483–499. URL: https://www.sciencedirect.com/science/article/pii/S1571066105826246. doi:https://doi.org/10.1016/S1571-0661(05)82624-6, lDTA'2003 - Language descriptions, Tools and Applications.

[50] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: Advances in Neural Information Processing Systems 28 (2015), 2015, pp. 2962–2970.

[51] H. Jin, F. Chollet, Q. Song, X. Hu, Autokeras: An automl library for deep learning, Journal of Machine Learning Research 24 (2023) 1–6. URL: http://jmlr.org/papers/v24/20-1355.html.

[52] T. Pang, M. Lin, X. Yang, J. Zhu, S. Yan, Robustness and accuracy could be reconcilable by (proper) definition, in: International Conference on Machine Learning, 2022. URL: https://api.semanticscholar.org/CorpusID:247011694.

[53] C. Molnar, Interpretable machine learning, Lulu. com, 2020.

[54] S. Ao, S. Rueger, A. Siddharthan, Empirical optimal risk to quantify model trustworthiness for failure detection, arXiv preprint arXiv:2308.03179 (2023).

[55] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, 2014. arXiv:1404.5997.

[56] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[57] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2015. arXiv:1409.1556.

[58] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.