# A Cost-Aware Approach to Adversarial Robustness in Neural Networks

Charles Meyers[1*], Mohammad Reza Saleh Sedghpour[2,1], Tommy L´ofstedt[1], Erik Elmroth[1,2]

[1*]Department of Computing Science, Umeå University, Umeå, 901 87, Sweden.
[2]Elastisys AB, Sweden.

*Corresponding author(s). E-mail(s): cmeyers@cs.umu.se;
Contributing authors: msaleh@cs.umu.se; tommy@cs.umu.se;
elmroth@cs.umu.se;

**Abstract**

Considering the growing prominence of production-level AI and the threat of adversarial attacks that can evade a model at run-time, evaluating the robustness of models to these evasion attacks is of critical importance. Additionally, testing model changes likely means deploying the models to (*e.g.* a car or a medical imaging device), or a drone to see how it affects performance, making un-tested changes a public problem that reduces development speed, increases cost of development, and makes it difficult (if not impossible) to parse cause from effect. In this work, we used survival analysis as a cloud-native, time-efficient and precise method for predicting model performance in the presence of adversarial noise. For neural networks in particular, the relationships between the learning rate, batch size, training time, convergence time, and deployment cost are highly complex, so researchers generally rely on benchmark datasets to assess the ability of a model to generalize beyond the training data. However, in practice, this means that each model configuration needs to be evaluated against real-world deployment samples which can be prohibitively expensive or time-consuming to collect — especially when other parts of the software or hardware stack are developed in parallel. To address this, we propose using accelerated failure time models to measure the effect of hardware choice, batch size, number of epochs, and test-set accuracy by using adversarial attacks to induce failures on a reference model architecture before deploying the model to the real world. We evaluate several GPU types and use the Tree Parzen Estimator to maximize model robustness and minimize model run-time simultaneously. This provides a way to evaluate

the model and optimise it in a single step, while simultaneously allowing us to model the effect of model parameters on training time, prediction time, and accuracy. Using this technique, we demonstrate that newer, more-powerful hardware does decrease the training time, but with a monetary and power cost that far outpaces the marginal gains in accuracy.

**Keywords:** artificial intelligence, machine learning, adversarial AI, optimisation, compliance

# 1 Introduction

## 1.1 Motivation

Recently, machine learning (ML) using deep neural networks has become a popular way to classify large amounts of data — with applications ranging from medical imaging [1] to aviation [2] and from security [3–5] to self-driving cars [6]. Statistical learning theory [7, 8] provides us no guarantees about the generalization performance of deep neural networks due to the massive number of tunable parameters. To overcome this, neural networks need large amounts of data [9, 10] to train ever-larger model [9], which has yielded increasingly marginal gains on test-set accuracy [11]. It is also clear that reaching safety-critical standards using test-set accuracy would require an infeasibly large test set [12]. Therefore, we propose using Accelerated Failure Time (AFT) models to simulate edge-cases and verify models using a small number of samples. Modern neural networks are massive — they have have grown to be one of the largest consumers of data-center power, especially with the rise of generative AI [13]. For image systems, AlexNet [14] is now considered small with *only* 60 million parameters. ResNet152 has even more parameters at 116 million [15]. However, modern architectures have lead to an explosion in model size with the recent Mamba model boasting a massive 8 *billion* tunable parameters [16]. At these scales, we have no statistical guarantees about the performance of these models and test-set validation would require many, many billions of samples for each and every model change. Furthermore, even if we ignore the immense cost of training modern neural networks, the required number of test samples creates serious questions about the efficacy of the typical train/test split methodology for assessing model generalization [12] since regulatory standards around safety-critical software applications [17–20] clearly define the maximum failure rate to be in the range $[10^{-12}, 10^{-15}]$ depending on the number of lives at risk. Furthermore, ensuring the robustness of ML models against adversarial noise has become a critical concern since inducing a failure at run-time has consistently shown to be trivial [21–26]. Collecting, labelling, and testing a new set of data for every software change — as required by law in most of the world [17–20] — would be prohibitively expensive for these large models. Therefore, a new evaluation methodology is required. We propose to use survival analysis as a methodological framework to model the failure conditions of a machine learning (ML) model. Instead of having a test set large enough to cover all of the failure cases, we generate adversarial samples crafted specifically to make the model fail and then use survival analysis to predict these failures in general. In this work, we

used accelerated failure-time methods to predict model performance as a function of various model parameters. The methodology outlined in Section 3 allows the model-builder to minimize the training cost, optimise for adversarial robustness, estimate the effects of covariates on model performance during routine training procedures. Then, one can perform a cost analysis (Section 4). We then demonstrate the efficacy of this methodology in Section 6 by examining the role of hardware on model performance, the particulars of which are outlined in Section 5.

### 1.2 Contributions

To tackle the problems of minimizing deployment cost while maximizing the model performance on both the test set and in the presence of adversarial noise we present a scalable and effective methodology (see Figure 1) and software framework (see Figure 2) for the training (see Section 3) and evaluation (see Section 4) of ML models that:

- Optimises for benign and adversarial accuracy simultaneously,
- Demonstrates a scalable and effective method to train a model while simultaneously estimating the effect of various hyper-parameters, and
- Measures the power and monetary cost of deploying a model across different hardware architectures to model the trade-offs between deployment hardware and robustness.
- Demonstrates that, even when we separately measure the effect of the slower clock speed of "inference only hardware" compared to hardware tailored for training, that models trained using the "inference only" devices are more robust for image classification tasks than models built on hardware designed for training.

Section 2 defines the terminology used throughout the paper. Section 3 outlines the training and evaluation methodology presented in this paper. Section 4 discusses the resulting cost analysis framework, arising from the methodology in the previous section. Section 5 outlines the software components and specific experiments that were conducted, while Section 6 contains the results and discussions of those experiments. Section 7 and Section 8 reveal the caveats and the conclusions respectively.

## 2 Background

In this paper, we evaluate a comprehensive methodology for evaluating model robustness during training time and model the asymptotic effect of the various parameters in the hyper-parameter search space. For the sake of the reader, we provide a section for definitions and requisite background information below.

### 2.1 Cloud Architectures

ML pipelines play a crucial role in the development and deployment of robust and accurate models. However, managing complex pipelines across diverse central processing unit (CPU) and graphics processing unit (GPU) architectures, ensuring robustness against adversarial attacks, and understanding the relationship between computational

cost, model loss, and prediction accuracy remain ongoing challenges. One popular solution is using a network of interconnected services [27–30] where a *service* is the smallest component of a "cloud-native" software stack. In the context of ML, that might be some software component meant for training, inference, pre-processing, sampling, or any other arbitrarily small part of the data pipeline. A service *mesh* typically consists of a set of interconnected components or proxies deployed alongside the services within the system. These components facilitate various capabilities and functionalities essential for managing the communication between services. Kubernetes has become one of the largest open source projects on the code-sharing website Github [31] and provides a framework for managing, monitoring, and networking a self-scaling set of tools across arbitrary software and hardware architectures using a service mesh. For ML applications, these services are often divided into "training" and "inference" configurations that often have distinct hardware and software configurations [32]. In this work, we leveraged Kubernetes [33] to manage a multi-stage ML pipeline (see Figure 1) and measure the power and cost of training and evaluating the pipeline (see Figure 2) across a variety of different hardware architectures (see Table 1).

## 2.2 ML Pipelines

ML pipelines are often long-running and complex software tool-chains with many tunable hyper-parameters. Managing, tracking, and controlling for various parameters is non trivial, but many management tools are available [33–35]. In general, a dataset is split into *training* and *test* sets. The test set is then used to determine the best configuration of a given model architecture on a given hardware architecture with the expectation that it will generalize both on the withheld test set and on new data generated and submitted by users. Since different hardware devices have differing amounts of video random access memory (VRAM), the resulting models will have differing optimal configurations if both robustness and training cost are considered. To verify the training process, the test set is validated against the *inference* configuration of a model which may run on different hardware than the *training* configuration to reduce cost, latency, or power consumption. Likewise, Nvidia offers hardware that is marketed as either for training (*e.g.*, v100 and p100) or only for inference (*e.g.*, l4).

## 2.3 Classifiers

We consider ML classifiers, $K(x;\theta)$ with model parameters, $\theta$. The true labels are denoted by $y$, and the model predictions by $\hat{y} = K(x;\theta)$ where $x$ is a mini-batch of size $N$ of data samples. The loss function, the measure of discrepancy between the true label and the predicted label, is denoted by $L(y,\hat{y})$. Because of the complexity of modern ML models, researchers rely on numerical optimisation, and one popular choice of optimisation algorithm is *stochastic gradient descent*, which updates the model parameters by taking a step in the negative gradient direction, where the gradient is computed using a subset (a mini-batch) of the training samples instead of all training samples. This procedure is repeated for some number of *epochs* (iterations through

the entire training set). Every iteration takes a step,

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_{\theta^{(i)}} L(y, K(x, \theta^{(i)})) \tag{1}$$

where the gradient is approximated using $b$ samples per mini-batch and $\eta$ is the *learning rate* (or step size) that is tuned to the particular model and data.

## 2.4 Learning Rate Selection

Choosing a good learning rate is critical for model performance — greatly effecting both the accuracy and the training cost. A small learning rate will more accurately approximate the class boundaries [36], but will converge slower, all other things being equal. Of course "small" is arbitrarily defined, but the scale of the optimal learning rate will vary with *e.g.*, the mini-batch size, number of epochs, dimensionality of the input data, *etc.* [37]. GPUs with larger amounts of VRAM are able to hold more data in memory at a time, increasing the effective mini-batch size, and reducing the number of model-tuning steps per epoch. A "good" learning rate will allow a model to converge quickly [37, 38] and the ideal mini-batch size will be determined by the memory bandwidth of the GPU and the size of the model and the size and dimensionality of the data. However, since cloud infrastructure is virtualized and shared with other users, the available bandwidth will not necessarily match the peak available bandwidth specified by the manufacturer [39], so researchers must evaluate this *in situ*. Furthermore, since hardware is typically billed by the hour on public clouds, optimising the training and inference times allows a model-builder to minimize the cost of deployment. To optimize for training time as well as robustness (both $\lambda_{ben}$ and $\lambda_{adv}$), the TPE optimization was used (see: Section 3.3). However, before progressing any further, it is first necessary to define robustness.

## 2.5 Adversarial Attacks

In the context of ML, an adversarial attack refers to deliberate and malicious attempts to manipulate or exploit ML models. Adversarial attacks are designed to deceive or change the model's behavior by introducing carefully crafted input data that can cause the model to make incorrect predictions or otherwise produce undesired outputs. That is, a successful attack is one in which the model outputs on the original, unperturbed data, $\hat{y}$, are not the same as the model outputs on perturbed data, $\hat{y_a}$. That is *adversarial success* or *accelerated failure* is one in which

$$\hat{y} \neq \hat{y}_a = K(x + \varepsilon; \theta), \tag{2}$$

where $\varepsilon$ is a noise distance bounded by $0 < \varepsilon \leq \varepsilon^*$ Additionally, one can measure the accuracy of the model when tasked with these adversarial samples, giving us a metric called *adversarial accuracy* which is Eq. 4 calculated on the perturbed samples. The goal of an adversarial attack is often to exploit vulnerabilities in the model's decision-making process or to probe its weaknesses. These attacks can occur during various stages of the ML pipeline, including during training [40, 41], inference [42, 43], or

deployment [21, 22, 24, 42, 44–46]. One of many possible attacks is designed to induce failures as quickly as possible, and is conveniently called the *fast gradient method* [47]. It works by applying noise to a set of samples, $x$, to generate adversarial examples, $x_a$, such that,

$$x_a = x + \eta \cdot \text{sign}(\nabla_x L(y, K(x, \theta))). \tag{3}$$

In essence, this is process seeks to increase the loss by changing the input data, in contrast to model training which seeks to minimize the loss by changing the model parameters.

### 2.6 Adversarial Analysis

In the case of safety- or security-critical domains, considering the worst-case scenario is routine [39]. Whether in the context of automotive safety [6], crytographic systems [48, 49], or healthcare malpractice [1], a component, algorithm, or system is considered broken if the *failure rate* exceeds a certain amount, depending the risk to human life [17]. An order of magnitude more automotive accidents, security breaches, or deaths due to negligence would be unacceptable and, as such, these standards are non-negotiable. However, this would mean testing many millions of samples for every model change that has the potential to injure a human, with orders of magnitude more stringent requirement in the case of potentially fatal systems. This is just not computationally feasible. Instead, researchers can use adversarial failure analysis [12, 22, 50] to improve the precision of our estimates while only using a small set of test-data.

## 3 Survival Analysis for Robustness Verification during Training

We propose a methodology for model training and verification using accelerated failure-time (AFT) models, drawn from the field of *survival analysis*, with a focus on cost-efficient evaluation methods. AFT models are statistical models used to analyze multivariate effects on the observed failure rate to predict the time-to-failure across a wide variety of circumstances [51, 52]. In medical science, these models are used to make claims like "smokers are twice as likely to die from lung cancer" or used to set the operating limits of manufactured components. This methodology can be used to map the relationship between various model tuning parameters and their effect on model performance. The next section outlines the methodology for modelling the effect of model hyper-parameters during routine training procedures. We precisely outline the methodology for using survival analysis during the training of ML methods.

### 3.1 Accuracy

Accuracy measures the vulnerability or susceptibility of the model to failures. A larger accurate indicates a higher rate of true classifications, signifying a weaker model in terms of *robustness* against (noise-induced) failures. Throughout, we use the terminology *benign* accuracy to refer to the performance on the test set using unperturbed data and *adversarial* to refer to the performance in the presence of additive noise that

is intended to confuse the model. The subscripts *ben* and *adv* are used respectively. The accuracy, $\lambda$, is defined as

$$\lambda := \text{Accuracy} := 1 - \frac{\text{False Classifications}}{\text{Total Classifciations}}, \tag{4}$$

which is generally assumed to indicate the rate of successes in real-world data sampled from the same distribution as the training data [53]. However, the normal test/train split methodology consistently overestimates the model's performance in the presence of adversarial noise [23]. In addition, it ignores the run-time cost of a given architectural decision, caring only about accuracy on benchmark data [9, 10]. Additionally, it has been shown that it is trivial to generate adversarial counter examples that reveal the test/train split methodology to be optimistic at best [12, 21, 22, 24, 40, 42, 46, 54].

## 3.2 Failure Rate

The failure rate refers to the percentage or proportion of examples that cause the targeted ML model to misclassify or produce incorrect outputs [12]. To encompass the cost of a particular model or attack, the proposed methodology considers failures to be a function over some time interval (*e.g.*, training time, inference time, attack generation time, *etc.*) and some covariates, $\theta$, so that the failure rate is the average time until a failure in a time interval around time, $t$, such that:

$$h_\theta(t) := \frac{p(\text{False Classification} \mid \theta)}{\Delta t} t,$$

where $p(\text{False Classification} \mid \theta)$ is the probability of a false classification given a particular set of hyper-parameters, $\theta$, $\Delta t$ is a time interval, and $t$ is a point in time. Note that $1 - \lambda$ is an estimate of this value when $\Delta t = t$ and converges to this value as the number of samples, $N \to \infty$. By modelling accuracy as a function of time, one can then compare the probability of failure to the cost, which is also measured in time, allowing one to make deployment decisions based on the risk analysis standards outlined in IEC61508 [17]. This then allows us to make claims like "increasing training time by X% will yield survival time gains of Y%" [55].

## 3.3 Optimisation

ML models are typically trained by examining the effect of the entire hyper-parameter space on the resulting accuracy. However, the number of hyper-parameter combinations are often infinite or at least exponential in the number of hyper-parameters, making it infeasible to exhaustively evaluate the entire hyper-parameter search space. Additionally, the goals of test-set accuracy and adversarial accuracy are often at odds [22] — with several researchers noting an inverse relationship between model accuracy and model robustness [12, 22, 54]. Therefore, a proper search would keep this dual-objective in mind. In addition, we attempt to minimize the training time for each piece of hardware since optimal batch size and, therefore, learning rate will be determined by the VRAM, the size bit depth of the data as well as the size and

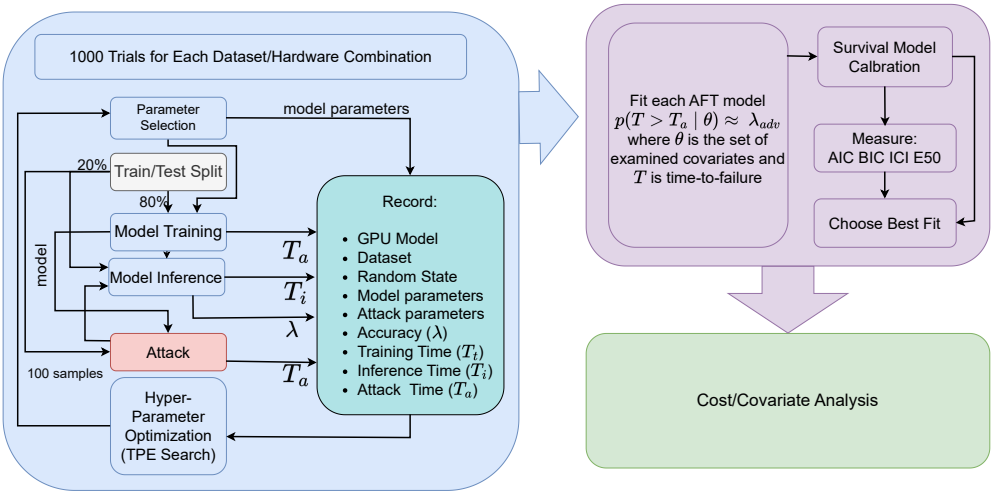


**Fig. 1**: For each dataset and hardware combination, a random state parameter was chosen at random to decide the test and train sets. Next, model parameters and attack parameters are chosen at random. After 128 random trials, the TPE algorithm attempts to maximize benign and adversarial accuracy while minimizing training time by tuning the model parameters. The random seed for the data split and the attack parameters are sampled independently from this optimization, which is why they are colored differently. The model tuning (blue-box) is discussed in Section 3.3. After the trials are completed, several AFT models are fit (see Section 3.4) and compared (see Section 3.5) using the process depicted in the purple box. Finally we conduct the cost analysis outlined in Section 4 (green box).

bit depth of the model. To maximise adversarial and benign accuracy simultaneously while minimizing training time, we propose the use of the Tree-structured Parzen Estimator (TPE) because it has been shown to converge over tens or hundreds of trials rather the the 1000s of trials typical of other multi-objective optimisation algorithms like CMAES or NSGA-II [56–58]. In addition, we seek to minimize the training time in order to maximize the number of hyper-parameters that can be examined on a fixed budget. Further discussion of the cost analysis can be found in Section 4.

## 3.4 AFT Models

AFT models are widely used in industrial, medical, or risk-mitigation contexts [51, 52] to model the effect of covariates on a model's expected time-to-failure (also called survival time). For industrial components, this often means testing the component under a variety of extreme circumstances to induce failures prematurely. For medical applications, this is used to model the effect of demographic characteristics or the efficacy of certain treatments on a given disease. For ML, this means inducing failures during training to measure generalization performance.

The point of this is to model the *survival time*, $S_\theta(t)$, as a function of the time, $t$, and some set of model parameters, $\theta$ such that,

$$S_\theta(t) = p(T > t \mid \theta) = \exp\left(-\int_0^t h_\theta(u)\, du\right)$$

where $p(T > t)$ is the probability that a model has not failed by time $t$ ("survives" beyond time $t$). The expected survival time is

$$\mathbb{E}_{S_\theta}[T] = \int_0^{t^*} S_\theta(u)\, du, \tag{5}$$

where $t^*$ is the latest time observed in the survival data. However, modelling $S_\theta(t)$ requires a choice in modelling function for $S_\theta$. The Log-Logistic, Log-Normal, and Weibull functions are widely used alternatives [52, 55]. For each trial, one can measure the attack generation time to define the time interval and the accuracy to estimate the number of failures and successes in that time interval.

### 3.4.1 Survival Time

By using adversarial samples (see Equation 2), failures can be induced. The likelihood of that failure is dependent on the amount of adversarial noise, $\varepsilon$, since adversarial noise is known to *induce* failures. That is, the rate of adversarial success (Equation 2), will increase as $\varepsilon$ increases. In terms of AFT models, this can be expressed as the expected lifetime of an adversarial sample when treating $\varepsilon$ as a covariate. In the language of accelerated failure time models, this can be expressed in terms of the accelerated failure time assumption [52]

$$S_\theta(t) = S_0\left(\frac{t}{\phi_\theta(x)}\right), \tag{6}$$

where $\phi_\theta$ is the acceleration factor, described by the joint effect of the covariates, such that

$$\phi_\theta(x) = \exp\left(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n\right),$$

and where $x = (x_0, \ldots, x_n)$ is a vector of covariates and $\theta = (\theta_0, \ldots, \theta_n)$ describe the fitted parameters, and $\varepsilon$ is used as one of the covariates. That is, when the adversarial noise level changes, it will also alter the expected survival time of a sample. This assumption means we can evaluate the generalization performance using a small number of adversarial samples with our precision coming from careful time measurements rather than massive test sets [52, 55]. However, in order to find the best AFT model, one must compare the tested models using the techniques outlined below.

## 3.5 Choosing the best AFT Model

To choose a best-fit from the possible AFT functions, one should prepare the collected metrics and scores and then compare them using *e.g.*, Akaike Information Criterion

(AIC), Bayesian Information Criteria (BIC), or Concordance, as per the best-practices for this methodology [51, 52]. For AIC and BIC, that means choosing the smallest value. Concordance, however, is a number between 0 and 1 that quantifies the degree to which the survival time is explained by the model, where a 1 reflects a perfect explanation [52] and 0.5 reflects random chance. By evaluating $\mathbb{E}_{S_\theta}[T]$ under extreme perturbations, one can test the model and minimize the number of evaluated samples [51, 52] rather than relying on the $> 10^{12}$ samples as required by IEC61508 [17]. While the aforementioned metrics measure goodness-of-fit, one should conduct a *survival probability calibration*, where one fits the AFT model to the data and compares it to a cubic-spline that is meant to capture the un-modelled relationship between failures and time [59]. These plots can be used to visually inspect the goodness-of-fit of various models, as in Figure 7, and this method is called *survival probability calibration* (used in Figure 1). The integrated calibration index (ICI) as well as the error at the 50th percentile E50 [59] can then be calculated. These are the mean absolute difference between observed and predicted probabilities and the median absolute difference between observed and predicted probabilities, respectively [59]. Additionally, the data were split into a training set that was used to fit the model (80%) and an unseen test set (20%) and the concordance, ICI, and E50 were measured for both.

# 4 Cost Analysis

In addition to the survival analysis, we can use an estimate of survival time to conduct a cost analysis as dictated by IEC61508 [17] which allows us to quantify the marginal risk. To quantify this marginal risk, one must measure the benign and adversarial accuracies (see Equations 2 and 4), the model training time, $t_t$, the model inference time or latency, $t_i$, the attack generation time, $t_a$, the cost per hour for a particular hardware, $C$, as well as the power consumption, $P$, of each tested model and attack.

## 4.1 Accuracy

In order to estimate the number of failures in a given time period, we measured both the benign accuracy and the adversarial accuracy, which reflect the normal test-set accuracy and the test-set accuracy in the presence of additive noise in the direction that maximizes loss. Under the AFT framework above, we can use the adversarial accuracy as a measure of the survival time across a specified time period as outlined in Figure 1.

## 4.2 Training Time

The training time, $T_t$, is the time it takes to evaluate $n$ samples when $t_t$ is the training time per sample. It is defined as

$$T_t := t_t \cdot n \cdot m,$$

where $m$ is the number of epochs.

### 4.3 Latency

Latency is the time it takes to respond to a query. We assume that latency per sample is

$$T_i := t_i \cdot n,$$

which will be driven by the memory bandwidth (measured in bits/second) of a given CPU or GPU and the size [60] and complexity [15] of a given neural network architecture.

### 4.4 Attack Generation Time

A successful attack is one that induces failure in a model. That is, the expected survival time, $\mathbb{E}_{S_\theta}[T]$, can be thought of as the average time it takes for an attacker to induce a change in the model output. Assuming a uniform distribution, $T_a$, for $n$ samples, $i$ iterations, and attack time per sample, $t_a$, as

$$T_a := t_a \cdot n. \tag{7}$$

In reality, the attack time will not be drawn from a uniform distribution and prior research [55] has shown that by treating model and attack parameters as covariates, one can model the survival time accurately by using and AFT model, such that:

$$t'_a \approx \mathbb{E}_{S_\theta}[T] = \int_0^{t^*} S_\theta(t)dt. \tag{8}$$

### 4.5 Cost

Furthermore, we approach the cost of deployment at two scales. Firstly, we consider the cloud-rental scale, where a small-business might test and deploy a model using *e.g.*, the Google Cloud Platform (GCP) compute costs as a measure of total cost. However, at a certain scale or with certain applications, it is more appropriate to talk about cost in terms of power (*e.g.*, to deploy a self-driving car with a useful operating range). Finally, we define metrics that provide an efficient way to minimize the latency, cost of deployment, and maximize the generalized performance of a model. We define the training cost as

$$C_t = C_h \cdot T_t,$$

the cost of model inference time as,

$$C_i = C_h \cdot T_i,$$

and the cost of an attack as,

$$C_a = C_h \cdot T_a.$$

where $C_h$ is the cost per unit time for the hardware, $T_t$ is the training time, $T_i$ is the inference time, and $T_a$ is the attack generation time, as defined above.

### 4.6 TRASH Score

With an estimate of the expected survival time in hand, quantify the cost-normalized failure rate, or the ratio of training time to attack time. Assuming that the cost scales linearly – as discussed above – the model builder's cost is proportional to the training time ($C_t \propto t_t$), and the attacker's cost is proportional to the attack time, which is approximately the expected survival time ($C_a \propto t'_a \approx \mathbb{E}_{S\theta}[T]$). Using the definition of $\varepsilon$ from Equation 2 and definition of $t'_a$ from Equation 8, we can express the cost of failure in adversarial terms as follows:

$$\text{TRASH} = \frac{t_t}{\mathbb{E}_\theta[T]} = \frac{t_t}{t'_a}. \tag{9}$$

#### 4.6.1 Power

The power consumption for a particular piece of hardware, $P_h$, measured in Watts (Joules per second), can be thought of similarly such that the total power consumption of model training is

$$P_t = P_h \cdot T_t,$$

the power consumption during model inference is

$$P_i = P_h \cdot T_i,$$

and the power consumption during attack generation is

$$P_a = P_h \cdot T_a,$$

where $P_h$ is the cost per unit time for the hardware, $T_t$ is the training time, $T_i$ is the inference time, and $T_a$ is the attack generation time, as defined above.

## 5 Experiments

This section outlines specific implementation details for the evaluation of the survival analysis methodology outlined in Section 3 and the cost analysis methodology outlined in Sec 4.

### 5.1 Cloud Platform and Hardware

To conduct the experiments and have access to different types of hardware, we utilized GCP. Six virtual machines running container optimised operating system provided by GCP constituted the test-bed. Using Google Kubernetes Engine 1.27.3 and Containerd 1.7.0, a cluster consisting of six worker nodes was created. Three worker nodes were responsible for running the monitoring platforms– Prometheus 2.47.2 and Grafana 10.2.0. These nodes were of the "e2-medium" instance type provided by GCP. In total, three GPU architectures were used–the Nvidia P100, V100, and L4. For P100 and V100 GPUs, the "n1-standard-2" type was used for the nodes and for L4 GPUs the "g2-standard-4" was used.

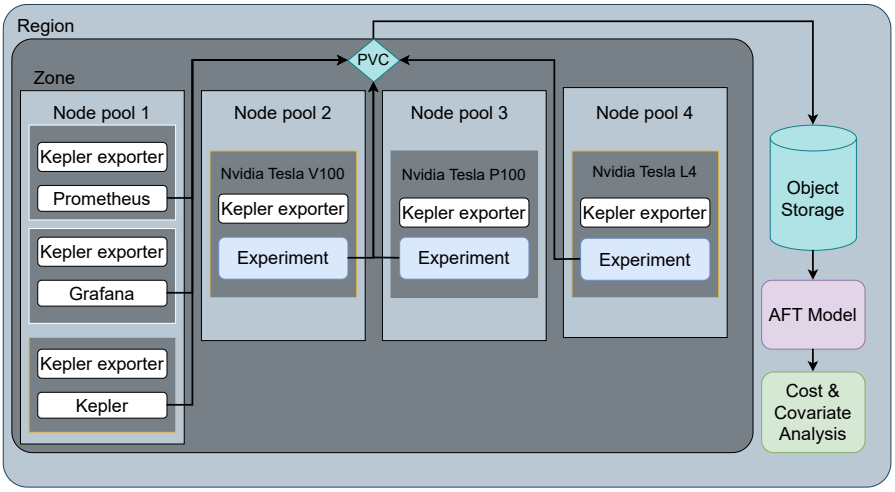


**Fig. 2**: For our experiments we used four node pools from Google Cloud Platform, each has a particular responsibility. The first node pool includes 3 different nodes responsible for hosting monitoring services such as Prometheus and Grafana. The other node pools each had one node with a specific GPU. The KEPLER exporter is then deployed on each node as DaemonSet to monitor the resource usage. All the storage requirements during the experimentation such as storage for experiments and monitoring data were then stored on storage provided by persistent volume claim (PVC). A PVC is a request for storage by user in Kubernetes which is then connected to the object storage. The blue experiment, blue-green object storage, green analysis component, and purple AFT component correspond to the same colors in Figure 1.

To assess the energy consumption of the experiments, KEPLER deployed on a was used to measure the power consumption of each experiment [61]. This approach enables gathering energy consumption data on granular levels as it runs in Kubernetes cluster and capable of collecting energy consumption of Kubernetes components. In essence, KEPLER uses extended Berkeley Packet Filter to probe energy-related system stats and exports them as Prometheus metrics. This filter can be described as a lightweight and sandboxed virtual machine (VM) in kernel space. The filter programs are invoked by the kernel when certain events occur. Examples of such events include system calls or network activity. These processes enable deep analysis and full control over different events with low overhead [62]. A diagram of the cloud architecture can be found in Figure 2. Finally, for a cost-effective model evaluation technique, all of the experiments were restricted to a single one-thousand United States dollar budget (a research grant from Google). Approximately 10% of this was used for development and 90% was used for the evaluations. Over the course of the evaluations, 6% of the budget went to the storage, 6% to the monitoring, and the remaining 88% went the GPUs.

## 5.2 AFT Models

For each hardware configuration and dataset, the TPE algorithm was used to select the hyper-parameters [56, 63], and it was iterated for 1000 trials. This was chosen to be substantially larger than the 200 used successfully by Watanabe *et al.* [58] while still allowing for reasonable run times [64]. We used three optimisation criteria: benign accuracy, training time, and adversarial accuracy, seeking to maximize both benign and adversarial accuracy while minimizing training time (and therefore deployment cost). We selected a set of parameters as per the TPE algorithm and trained on 80% of the samples for each of the MNIST, CIFAR10, and CIFAR100 datasets. Of the remaining samples, 100 were withheld to be attacked and used to evaluate the adversarial accuracy. Figure 1 illustrates this methodology. For each dataset, we tested this on ten random splits of the data to create 10 unique test/train pairs. For each trial, we recorded attack generation time, model training time, model inference time, benign accuracy and adversarial accuracy, and the size of the training set, test set, and attack set. Using these values, we fit an AFT model to the number of failures (indicated by accuracy and sample size) and the attack generation time after adding dummy variables for the dataset and hardware device. Additionally, we approximate the AFT model using the methodology depicted in Figure 1.

## 5.3 Datasets

The AFT models were evaluated using the MNIST [65], CIFAR10 [66], and CIFAR100 [66] datasets, chosen primarily for their standardized use in adversarial analysis [22, 23, 67, 68] and decades of experimental results. Before training, we centered and scaled the data so that the attack distance would be analogous for all tested datasets. Furthermore, to reduce the complexities of system overhead, distributed or federated training, and the effect of shared cloud environments, we restricted ourselves to datasets that were small enough to reside entirely within GPU memory with the model, since the disk read speed in cloud environments is incredibly variable.

## 5.4 Models

The evaluations were restricted to a single model. Primarily, this was done to meet the budgetary constraints since evaluating more models would mean evaluating fewer pieces of hardware. As discussed in Section 2.4, the relationship between hardware specifications, hyper-parameters, and performance is highly complex and hard to predict. So, we sampled learning rates $\in [10^{-6}, 1]$, batch sizes $\in [1, 10^5]$, and epochs $\in [1, 50]$ for MNIST and CIFAR10 on the P100 and V100. For CIFAR100, the range of the tested epochs was increased to be $\in [1, 100]$. The Feature Squeezing defence [69] was used to evaluate the efficacy of different bit-depths on the L4 hardware, as provided by IBM's adversarial robustness toolbox [26] with bit depths $\in [4, 8, 16, 32, 64]$ which casts the inputs into `pytorch` compatible arrays. Model parameters were chosen using the `optuna` optimisation framework, the configuration was handled by `hydra`, and `dvc` was used to ensure reproducibility and aid in collaborative development.

## 5.5 Attacks

To examine the effect of model parameters at run-time, evasion attacks, which attack the model at the prediction stage, were examined. Prior research [12, 55] has shown that the Fast Gradient method (see Eq. 3) is consistently the most effective at inducing a large number of failures in a small amount of time. To evaluate the effect of adversarial noise on the samples, the noise levels were varied $0 < \varepsilon \leq 0$. This was done using the `adversarial-robustness-toolbox` package maintained by a team at IBM [26].

## 5.6 GPU Configurations

Several hardware configurations were tested, that had various hourly costs, peak power demands, and theoretical memory bandwidths. The V100 was chosen as a baseline, since it is routinely used in the literature [70, 71]. The P100 architecture comes from the same line of server-grade GPUs, but from an older generation. The L4, however, is advertised as a machine built for inference — not training — relying on a smaller number of bits per tensor core. Consequently, the number of operations per second depends on the bit depth of the data and model weights, with peak numbers outlined in Table 1 for 8-bit inputs. The rental cost of the hardware, measured in United States Dollars per hour, indicates the operating cost of a given model. To calculate this, the price per hour from each cloud service pricing page [72, 73] were used and the cost of training ($C_t$) and the cost of inference ($C_i$) calculated from the cost of hardware ($C_h$), the training time ($T_t$), and the inference time ($T_i$).

| | V100 | P100 | L4 |
|---|---|---|---|
| Cost (USD/hour) | 2.55 | 1.60 | 0.81 |
| Power (Watts) | 250 | 250 | 72 |
| Memory Bandwidth (GB/s) | 900 | 732 | 300 |

**Table 1**: Hardware specifications for the tested GPUs. The specifications were retrieved from Nvidia's website at the following links: V100 Datasheet, P100 Datasheet, and L4 Datasheet. Prices were retrieved from Google Cloud Platform for the `europe-west4` region on 3 December 2023.

## 5.7 Survival Analysis

In addition to the optimisation criteria of benign/adversarial accuracy and training time, prediction times, attack times, power consumption, batch size, and number of epochs were also collected to be used as covariates in the AFT model, $S_\theta(t)$. To fit the AFT model and to plot the effect of the covariates, the `lifelines` Python package was used [74]. The Weibull, Log Logistic, and Log Normal AFT models (see Section 3) were also compared.

# 6 Results and Discussion

This section presents and discusses the results for all of the experiments across all datasets and hardware. First, the accuracy, training time, inference, and monetary cost are discussed in Sections 6.1–6.2, followed by the fitted AFT model in Section 6.3.

## 6.1 Accuracy

Figure 3 shows the benign (left) and adversarial (right) accuracies for all datasets and hardware. It demonstrates little to no change in accuracy or adversarial accuracy, regardless of hardware. The benign accuracy decreases with difficulty (CIFAR10 vs MNIST) or with the number of classes (CIFAR10 vs CIFAR100). For all three datasets, the adversarial accuracy becomes the reciprocal of the number of classes (*i.e.*, the accuracy we would expect with random data), demonstrating the efficacy of the attack outlined in Section 3.3.

## 6.2 Time, Power, Cost

Figure 4 reveals the training time, inference time, and attack generation time across all datasets and models. Slower hardware (see Memory Bandwidth in Table 1) is only a few milliseconds slower, so this is unlikely to be noticeable to an end-user during inference. This should be no surprise since the less-capable hardware is meant to be used exclusively for inference. Additionally, the attack time (right sub-figure) increases with both the training (left) and prediction times (center) across hardware and datasets — an expected result since this is driven by the number of samples and the inference time (see Equation 7). We can also see that average training time per sample is takes slightly more than than attack generation time per sample (see Figure 4 left and right plots).

The power consumption during training, inference, and attack generation for all hardware and datasets is illustrated in Figure 5. It tracks monetary cost (Figure 6) closely, probably because that's the predominant cost for the datacenter itself [75], so it is unsurprising that cloud billing is correlated with the power requirement. Furthermore, we see that the largest dataset (CIFAR100) and smallest GPU (L4) require the least amount of power (Figure 5) for prediction, while differences between hardware and datasets are insignificant for the training and attack metrics.

The monetary cost for each dataset and each piece of hardware is shown in Figure 6. For all three datasets across all three pieces of hardware, the cost of training on a single sample often exceeds the cost of attacking a single sample. In the best case scenario, they are comparable, but attacks consistently succeed with only 100 samples (Figure 3) while model training requires orders of magnitude more.

## 6.3 AFT Models

Table 2 shows the performance metrics for all three AFT models, as outlined in Section 3.4. A total 75% of the available data were used to build the AFT models and 25% were withheld for evaluation. The concordance is both strong ($> 0.5$) and similar for all three AFT models and both the train and test sets. We observed no more than

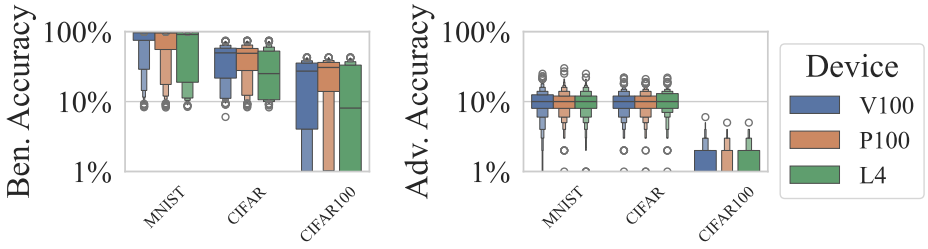


**Fig. 3**: Benign and adversarial accuracy across all hardware and datasets for all 1000 trials using plots that depict the distribution of the y-axis values using the width of the plot. Each color is a different device and the datasets are displayed along the x-axis. Outliers are denoted with a white dot.

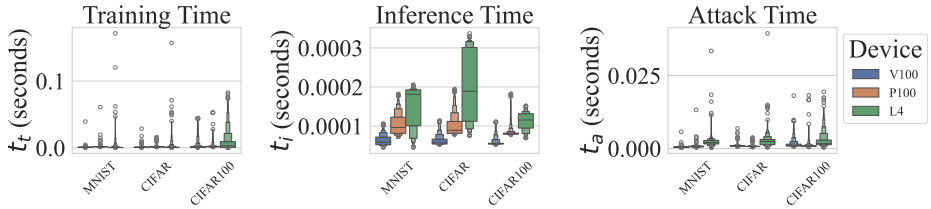


**Fig. 4**: This depicts the training, inference, and attack times for all hardware and datasets for all 1000 trials using plots that depict the distribution of the second axis values using the width of the plot. The time per sample was assumed to be uniform across the batch of samples for each training, inference, or attack measurement. Each color is a different device and the datasets are displayed along the first axis. Outliers are denoted with a white dot. For these plots, the second axes have been scaled by the number of samples for the sake of comparison.

1% mean absolute error in the probabilities ICI and no error in the median probability E50 across all three AFT models. Figure 7 depicts the observed and predicted probabilities for all three AFT models. All three models have strong predictive power (see Table: 2), with an average test error of around 1% ICI and a similar error for the median sample E50. We also see that these metrics are functionally identical across all AFT models as well as between the test and train set of each model.

Figure 8 shows the log-scale coefficients for the AFT model. It shows that epochs, batch size, and training time have no effect on the survival time. Additionally, the covariate value of 0 for random state indicates that random permutations of the training and test data have no effect on the survival time, which is expected. However, we can clearly see that inference time is as strong an indicator as the attack noise distance, revealing that model speed is nearly as important as the the noise value of the attack

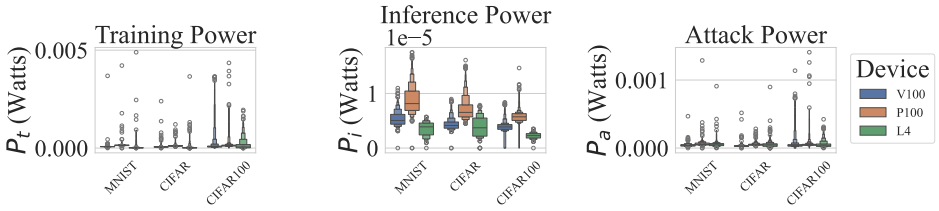


**Fig. 5**: This depicts the training, inference, and attack times for all hardware and datasets for all 1000 trials using plots that depict the distribution of the second axis values using the width of the plot. The power per sample was assumed to be uniform across the batch of samples for each training, inference, or attack measurement. Each color is a different device and the datasets are displayed along the first axis. Outliers are denoted with a white dot. For these plots, the second axes have been scaled by the number of samples for the sake of comparison.

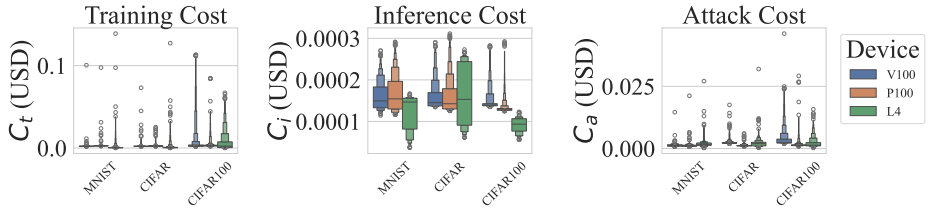


**Fig. 6**: This depicts the training, inference, and attack times for all hardware and datasets for all 1000 trials using plots that depict the distribution of the second axis values using the width of the plot. The cost per sample was assumed to be uniform across the batch of samples for each training, inference, or attack measurement. Each color is a different device and the datasets are displayed along the first axis. Outliers are denoted with a white dot. For these plots, the second axes have been scaled by the number of samples for the sake of comparison.

($\varepsilon$). Furthermore, we see that benign accuracy is negatively correlated with the survival time, confirming previous assertions that robustness ($S_\theta(t)$) is inversely related to benign accuracy (which is the standard indicator of generalization performance) [22].

Using the $S_\theta(t)$ described in Section 3, we can model the effect of various hardware devices, which is depicted in Figure 8. It clearly shows that hardware choice has a relatively small effect ($\pm 20\%$) on robustness, despite the much larger disparity in cost outlined in Table 1.

## 6.4 Why Cost Matters

In security analysis, it is routine to think in optimistic terms for both the attacker and defender. In cryptography, these ideal attack- and defence-scenarios are used to test

**Table 2**: Comparison of AFT Models across all hardware and datasets. AIC and BIC are measures of goodness-of-fit, with a smaller value being preferred. Concordance (Conc) is a value between 0 and 1 that reflects how what proportion of events (failures) can be explained by the model. ICI measures the average error between a cubic-spline and the model and E50 measures the median error between the spline and the model. These measured on both the train and test sets of the collected data, with the latter being denoted "Test".

| | AIC | BIC | Conc | Test Conc | ICI | Test ICI | E50 | Test E50 |
|---|---|---|---|---|---|---|---|---|
| Weibull | $-2.11 \cdot 10^4$ | $-2.11 \cdot 10^4$ | 0.84 | 0.83 | 0.00 | 0.01 | 0.00 | 0.00 |
| Log Logistic | $-2.18 \cdot 10^4$ | $-2.18 \cdot 10^4$ | 0.84 | 0.83 | 0.01 | 0.01 | 0.00 | 0.00 |
| Log Normal | $-2.25 \cdot 10^4$ | $-2.25 \cdot 10^4$ | 0.84 | 0.83 | 0.01 | 0.00 | 0.00 | 0.00 |

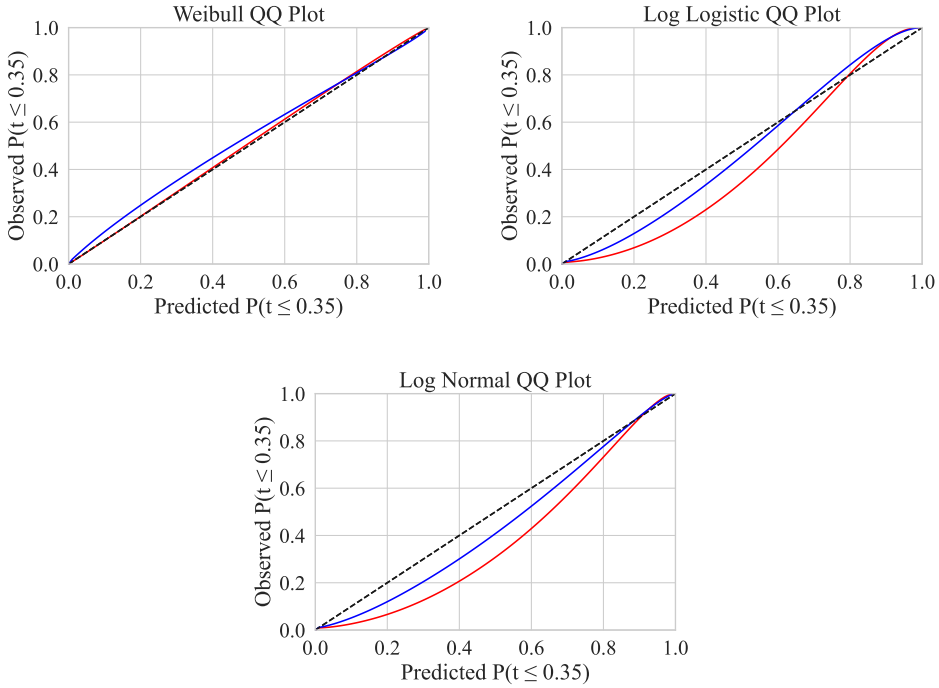


**Fig. 7**: The red and blue lines depict the relationship between the modelled number of failures (x-axis) and the measured number of failures (y-axis) for the training set (blue) and test set (red). The dotted line indicates a cubic spline fit to $S_\theta(t = t_a) \approx \lambda_{adv}$. A model that fits this cubic spline exactly would be a diagonal line at $y = x$. The process of comparing the fitted AFT model to a default cubic spline is called *survival probability calibration* and is discussed in more detail in Section 3.5.

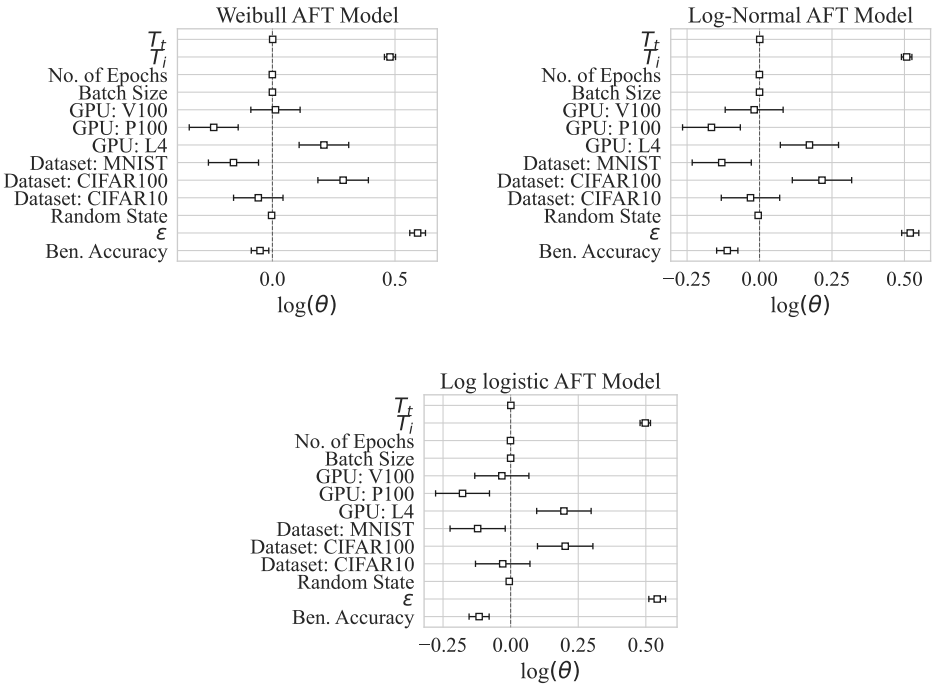


**Fig. 8**: Coefficients of the Covariates for the Weibull, Log-Normal, and Log-Logistic AFT models. Here, "Random Stat" is used as a control variable that should be (and is) close to 0. A positive value indicates that covariate increases the survival time and negative value indicates that covariate decreases the survival time. The symbols $T_i$ and $T_t$ refer training and inference time for all samples, while *Ben. Accuracy* refers to the benign accuracy (accuracy on the un-altered samples), and $\varepsilon$ is the noise distance.

the computational feasibility of subverting a particular cryptographic system [48, 49] (*e.g.*, whether or not a given cryptographic method should be considered "broken"). Using AFT, one can model the Pareto front — the set of points that are superior to all others in the search space — for all objectives [63]. By using a whitebox attack to generate the failures, we ensure that our defender operates under the worst case scenario while the attacker operates under their best case scenario to yield a worst-case failure rate. As such, we center our analysis on the whitebox Fast Gradient Method (FGM) [47] (see Eq. 3) which is both effective and fast [12]. If the cost to a model builder is much larger than the cost to an attacker, then it is clear that the model is "broken" in this cryptographic sense and can be discarded as ineffective [55]. Figure 9 depicts the TRASH score defined in Equation 9, that quantifies the per-sample ratio of training to attack time. It is clear that even with the reduced GPU bandwidth (Table 1), that the L4 model is not only superior in terms of cost (Table 1), but also in terms of robustness (Table 1), presumably since the extra bit-depth provided by the

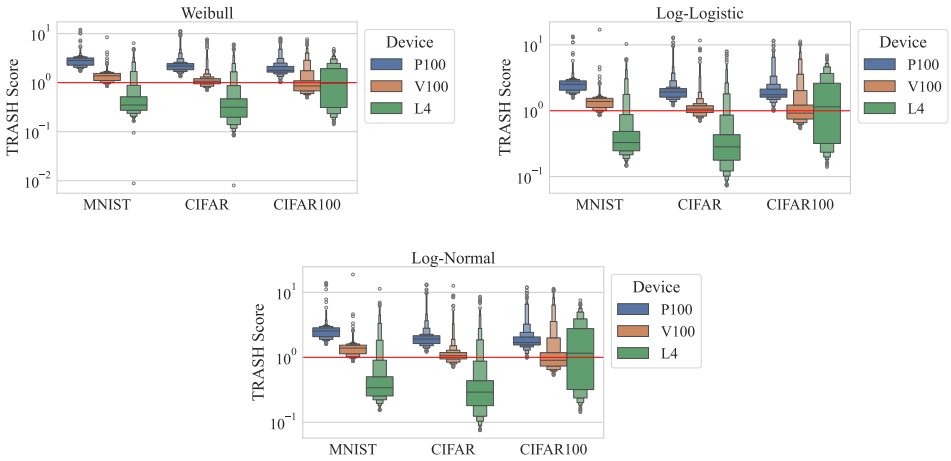


**Fig. 9**: The *training rate and survival heuristic* score (TRASH score) that depicts the per-sample ratio of training time to attack time for each of the tested AFT models. If this value is greater than one (the red line), then the model can be discarded as ineffective. The x-axis shows each dataset, the y-axis shows the TRASH score and each GPU model is given its own color. Outlier scores are depicted with a white dot.

P100 and V100 ends up being useless noise anyway when operating on 8-bit images like MNIST and CIFAR.

## 6.5 Advantages of this Methodology

The primary advantage of this methodology over the traditional test/train split measurements is that the precision of the latter is determined by the number of samples; whereas, the precision of the survival time estimate is driven by the resolution of our timing measurements. For test-set accuracy, meeting the weakest safety-critical IEC standard (one failure in a million) would require many millions of samples to confidently conclude that a model is safe. However, the presented AFT model has an average error rate of 1% while requiring a very small number of samples (10 sets of 100) (see Table 2). Because of the small number of samples needed for building AFT models (when compared to testing against massive in-distribution test sets), AFT models could, for example, act as a unit test in ML applications. This is in contrast to a full-system integration tests to evaluate changes to a single model, signal processing technique, data storage format, or API access mechanism [76, 77]. It could also be used to highlight error-prone classes or other subsets of data to reduce error or create synthetic samples as is common in medical research [52]. Furthermore, by isolating changes and testing them as quickly as possible, it is easier to parse cause and effect when compared to full-system integration tests that could include many changes from many different development teams and require live and potentially dangerous systems (like cars, drones, or industrial equipment) to effectively test. To further increase development velocity, these models can quantify the marginal risk associated with each

change, as dictated by the IEC 61508 standard [17], allowing the model builder to make a quantitative assessment about the efficacy of a model change without testing the effect on real users. Additionally, since AFT models have strong predictive power for untested hyper-parameter combinations, this method has a potential to reduce the search space by quickly eliminating candidates that are unlikely to increase the survival time.

# 7 Considerations

We have taken much care to conduct all timing measurements as carefully as possible. Primarily, to minimize timing jitter and account for GPU parallelization, we assumed that the time-to-failure during the benign and adversarial accuracy measurements was uniform across the samples in each trial. While it is very possible (if not altogether guaranteed) that some classes or samples are easier to attack than others, we assume that this averages out over the 100 samples given to each attack. Further work examining, for example, the effect of imbalanced datasets on the the distribution of survival times across classes is outside the scope of this work, though the methodology would remain identical. Similarly, one could examine survival times for samples near the class boundary and compare them with prototypical samples near the center of the class cluster. However, given the strong results and uniformity across AFT functions in Table 2, the uniform time assumption appears to be insignificant, though accounting for the failure rate per sample or class is likely to account for some of the remaining unexplained variance. Additionally, we chose a model and set of datasets small enough to fit entirely in GPU memory to minimize the confounding factors around parallelized, distributed, and/or federated learning as well as the complexities around storage hardware, file systems, and various access mechanisms. Larger models acting on larger datasets are likely to perform better on hardware with a higher GPU bandwidth, but the 48GB of VRAM provided by the L40 should be more than sufficient for vision tasks based on 8-bit images standard in the literature (*e.g.* MNIST, CIFAR10, CIFAR100). Furthermore, the attack batch size and the training batch size were set to be the same for every trial for each dataset and piece of hardware so that the attack timing would mimic the usage of regular users. If anything, the smaller sample size of the attacks means that proportionally more parallelization overhead is needed per sample, leading to an underestimate of the attack efficacy compared to the benign measurements. Furthermore, while FGM is fast, it is not guaranteed to find failures as quickly as possible, meaning that the survival time estimate is likely to be overestimated. In general, distributed/federated models were out of scope for this work, but architectural choices regarding the configuration of such methods could be evaluated using the cost and survival analysis techniques outlined above. The hyper-parameter search was restricted to only a single evasion attack (see Eq. 3) in the interest of time and budget, though this analysis would generalize to other evasion, extraction, inversion, or poisoning attacks [40, 43, 44, 50].

# 8 Conclusions

In this work, we applied survival analysis to cloud-native ML, developed a method to quickly and efficiently test model parameter choices and evaluated them in the presence of adversarial noise, allowing one to the model performance as a function of tested model parameters. Using this technique, it is clear that reduced run-times with newer and/or larger hardware, that adversarial robustness is not substantially effected and that the specific choice of attack parameters is far more important than model training time.

The experiments conducted confirm that this methodology is both sound and cost-effective. While 88% of the cloud budget went to GPU rentals, the data show how cutting that cost by 75% and using less power-intensive hardware designed for 8-bit image processing would be *more* effective than using the 32-bit datacenter GPUs (V100 and P100) for typical image classification tasks. Using  proved to be a cost-effective as real-time monitoring only used a fraction of the budget (6%). Figure 3 confirms the efficacy of the TPE algorithm discussed in Section 3.3, allowing an infrastructure engineer to ignore the intricacies of learning rate optimization while tuning the model to a particular piece of hardware. Additionally, Table 2 and Figure 7 show that AFT models are an effective way to model the survival time across hardware variants. By comparing the training, inference, and attack times, costs, and power consumption across different hardware devices (Figures 4- 6), it is clear that attacks generally cost less than training, even when we ignore the fact that attacks are consistently effective using a small number of samples while training relies on many thousands (see Figures 1 & 3). This analysis is further confirmed by examining the coefficients of the AFT models in Figure 8 wherein it is evident that neither training time ($T_t$) nor the number of epochs have noticeable effect on the survival time, but that model latency ($T_i$) does.

Regardless of which AFT model is chosen, the model suggests that the P100 decreases the survival time compared to the V100 and the L4 increases the survival time (both compared to the V100, which has no significant effect). Furthermore, the coefficients again confirm an inverse relationship between test set (benign) accuracy and the adversarial robustness (*i.e.*, survival time). Finally, even when we account for the reduced GPU bandwidth (see Table 1) and increased training time (see Figure 4), it is clear the the L4 model is superior in terms of robustness and cost-effectiveness (see Figure 9), despite it being advertised as "inference only" by the manufacturer. Furthermore, the data demonstrate the un-advertised training capability of the Nvidia L4 GPU which yields a 20% increase in survival time while costing 75% less than the V100 that's been typical in the literature over the last several years.

## 8.1 List of Abbreviations

# References

[1] Erickson, B.J., Korfiatis, P., Akkus, Z., Kline, T.L.: Machine learning for medical imaging. Radiographics **37**(2), 505–515 (2017)

[2] Maheshwari, A., Davendralingam, N., DeLaurentis, D.A.: A comparative study of machine learning techniques for aviation applications. In: 2018 Aviation Technology, Integration, and Operations Conference, p. 3980 (2018)

[3] Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K.: Dos and don'ts of machine learning in computer security. In: 31st USENIX Security Symposium (USENIX Security 22), pp. 3971–3988 (2022)

[4] Mery, D., Svec, E., Arias, M., Riffo, V., Saavedra, J.M., Banerjee, S.: Modern computer vision techniques for x-ray testing in baggage inspection. IEEE Transactions on Systems, Man, and Cybernetics: Systems **47**(4), 682–692 (2016)

[5] Travaini, G.V., Pacchioni, F., Bellumore, S., Bosia, M., De Micco, F.: Machine learning and criminal justice: A systematic review of advanced methodology for recidivism risk prediction. International journal of environmental research and public health **19**(17), 10594 (2022)

[6] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)

[7] Vapnik, V.N.: An overview of statistical learning theory. IEEE transactions on neural networks **10**(5), 988–999 (1999)

[8] Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Cambridge university press, Cambridge, UK (2014)

[9] Desislavov, R., Martínez-Plumed, F., Hernández-Orallo, J.: Compute and energy consumption trends in deep learning inference. arXiv:2109.05472 (2021)

[10] Bailly, A., Blanc, C., Francis, É., Guillotin, T., Jamal, F., Wakim, B., Roy, P.: Effects of dataset size and interactions on the prediction performance of logistic regression and deep learning models. Computer Methods and Programs in Biomedicine **213**, 106504 (2022)

[11] Sun, C., Shrivastava, A., Singh, S., Gupta, A.: Revisiting unreasonable effectiveness of data in deep learning era. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 843–852 (2017)

[12] Meyers, C., Löfstedt, T., Elmroth, E.: Safety-critical computer vision: An empirical survey of adversarial evasion attacks and defenses on computer vision systems. Artificial Intelligence Review (2023)

[13] O'Brien, M., Fingerhut, H., Press, T.A.: A.I. tools fueled a 34% spike in Microsoft's water consumption, and one city with its data centers is concerned about the effect on residential supply. Fortune (2023). https://fortune.com/2023/

09/09/ai-chatgpt-usage-fuels-spike-in-microsoft-water-consumption/

[14] Alom, M.Z., Taha, T.M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M.S., Van Esesn, B.C., Awwal, A.A.S., Asari, V.K.: The history began from alexnet: A comprehensive survey on deep learning approaches. arXiv preprint arXiv:1803.01164 (2018)

[15] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

[16] Waleffe, R., Byeon, W., Riach, D., Norick, B., Korthikanti, V., Dao, T., Gu, A., Hatamizadeh, A., Singh, S., Narayanan, D., et al.: An empirical study of mamba-based language models. arXiv preprint arXiv:2406.07887 (2024)

[17] Commission, I.E.: IEC 61508 Safety and Functional Safety. International Electrotechnical Commission (2010)

[18] International Standards Organization: ISO 26262-1:2011, Road vehicles — Functional safety. https://www.iso.org/standard/43464.html (2018)

[19] Axelrod, C.W.: Applying lessons from safety-critical systems to security-critical software. In: 2011 IEEE Long Island Systems, Applications and Technology Conference, pp. 1–6 (2011). IEEE

[20] Acharyulu, P.S., Seetharamaiah, P.: A framework for safety automation of safety-critical systems operations. Safety Science **77**, 133–142 (2015)

[21] Brown, T.B., Mané, D., Roy, A., Abadi, M., Gilmer, J.: Adversarial patch. arXiv:1712.09665 (2017)

[22] Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57 (2017)

[23] Croce, F., Hein, M.: Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In: International Conference on Machine Learning, pp. 2206–2216 (2020). PMLR

[24] Chen, J., Jordan, M.I., Wainwright, M.J.: HopSkipJumpAttack: A query-efficient decision-based attack. In: IEEE Symposium on Security and Privacy (SP), pp. 1277–1294 (2020). IEEE

[25] Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., Mukhopadhyay, D.: Adversarial attacks and defences: A survey. arXiv:1810.00069 (2018)

[26] Nicolae, M.I., Sinn, M., Tran, M.N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I., Edwards, B.: Adversarial robustness toolbox v1.2.0. CoRR **1807.01069** (2018)

[27] Panchal, D., Verma, P., Baran, I., Musgrove, D., Lu, D.: Reusable mlops: Reusable deployment, reusable infrastructure and hot-swappable machine learning models and services. arXiv preprint arXiv:2403.00787 (2024)

[28] Hasselbring, W., Steinacker, G.: Microservice architectures for scalability, agility and reliability in e-commerce. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), pp. 243–246 (2017). IEEE

[29] Zhou, R., Pang, J., Zhang, Q., Wu, C., Jiao, L., Zhong, Y., Li, Z.: Online scheduling algorithm for heterogeneous distributed machine learning jobs. IEEE Transactions on Cloud Computing (2022)

[30] Singh, N., Hamid, Y., Juneja, S., Srivastava, G., Dhiman, G., Gadekallu, T.R., Shah, M.A.: Load balancing and service discovery using docker swarm for microservice based big data applications. Journal of Cloud Computing **12**(1), 4 (2023)

[31] Github: Octoverse Projects. Github.com (2019). https://octoverse.github.com/2018/projects.html

[32] Wang, Y.E., Wei, G.-Y., Brooks, D.: Benchmarking tpu, gpu, and cpu platforms for deep learning. arXiv preprint arXiv:1907.10701 (2019)

[33] Kubernetes: Kubernetes–an open source system for managing containerized applications. Github (2019). https://github.com/kubernetes/kubernetes

[34] dvc.org: DVC- Data Version Control. Github (2023). https://github.com/iterative/dvc.org

[35] Yadan, O.: Hydra - A framework for elegantly configuring complex applications. Github (2019). https://github.com/facebookresearch/hydra

[36] Cao, Y., Gu, Q.: Generalization bounds of stochastic gradient descent for wide and deep neural networks. Advances in neural information processing systems **32** (2019)

[37] Granziol, D., Zohren, S., Roberts, S.: Learning rates as a function of batch size: A random matrix theory approach to neural network training. Journal of Machine Learning Research **23**(173), 1–65 (2022)

[38] Smith, L.N., Topin, N.: Super-convergence: Very fast training of neural networks using large learning rates. In: Artificial Intelligence and Machine Learning for Multi-domain Operations Applications, vol. 11006, pp. 369–386 (2019). SPIE

[39] Sajid, M., Raza, Z.: Cloud computing: Issues & challenges. In: International Conference on Cloud, Big Data and Trust, vol. 20, pp. 13–15 (2013)

[40] Biggio, B., Nelson, B., Laskov, P.: Poisoning Attacks against Support Vector

Machines. arXiv:1206.6389 [cs, stat] (2013)

[41] Saha, A., Subramanya, A., Pirsiavash, H.: Hidden trigger backdoor attacks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 11957–11965 (2020)

[42] Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., Mukhopadhyay, D.: Adversarial attacks and defences: A survey. arXiv:1810.00069 [cs, stat] (2018)

[43] Orekondy, T., Schiele, B., Fritz, M.: Knockoff nets: Stealing functionality of black-box models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4954–4963 (2019)

[44] Choquette-Choo, C.A., Tramer, F., Carlini, N., Papernot, N.: Label-only membership inference attacks. In: International Conference on Machine Learning, pp. 1964–1974 (2021). PMLR

[45] Li, Z., Zhang, Y.: Membership leakage in label-only exposures. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 880–895 (2021)

[46] Kotyan, S., Vargas, D.V.: Adversarial robustness assessment. PloS one **17**(4), 0265723 (2022)

[47] Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv:1412.6572 (2014)

[48] Leurent, G., Peyrin, T.: SHA-1 is a shambles: First Chosen-Prefix collision on SHA-1 and application to the PGP web of trust. In: 29th USENIX Security Symposium (USENIX Security 20), pp. 1839–1856 (2020)

[49] Kamal, P.: A study on the security of password hashing based on gpu based, password cracking using high-performance cloud computing. Master's thesis, St. Cloud State. St. Cloud, Minnesota, USA. (2017)

[50] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13, pp. 387–402 (2013). Springer

[51] Bradburn, M.J., Clark, T.G., Love, S.B., Altman, D.G.: Survival analysis part II: multivariate data analysis–an introduction to concepts and methods. British journal of cancer **89**(3), 431–436 (2003)

[52] Kleinbaum, D.G., Klein, M.: Survival Analysis a Self-learning Text. Springer, New York, NY, USA (2012)

[53] Tan, J., Yang, J., Wu, S., Chen, G., Zhao, J.: A critical look at the current train/test split in machine learning. arXiv preprint arXiv:2106.04525 (2021)

[54] Dohmatob, E.: Generalized No Free Lunch Theorem for Adversarial Robustness. In: Proceedings of the 36th International Conference on Machine Learning. PMLR, vol. 97 (2019)

[55] Meyers, C., Reza, M., Löfstedt, T., Elmroth, E.: A systematic approach to robustness modelling. In: Accepted for International Conference on Machine Learning and Cybernetics (2023)

[56] Ozaki, Y., Tanigaki, Y., Watanabe, S., Onishi, M.: Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 533–541 (2020)

[57] Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2623–2631 (2019)

[58] Watanabe, S.: Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. arXiv preprint arXiv:2304.11127 (2023)

[59] Austin, P.C., Harrell Jr, F.E., Klaveren, D.: Graphical calibration curves and the integrated calibration index (ICI) for survival models. Statistics in Medicine **39**(21), 2714–2742 (2020)

[60] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[61] Amaral, M., Chen, H., Chiba, T., Nakazawa, R., Choochotkaew, S., Lee, E.K., Eilam, T.: Kepler: A framework to calculate the energy consumption of containerized applications. In: 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), pp. 69–71 (2023)

[62] Sedghpour, M.R.S., Townend, P.: Service mesh and ebpf-powered microservices: A survey and future directions. In: 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE), pp. 176–184 (2022). https://doi.org/10.1109/SOSE55356.2022.00027

[63] Zitzler, E., Knowles, J., Thiele, L.: Quality assessment of pareto set approximations. Multiobjective optimization: Interactive and evolutionary approaches, 373–404 (2008)

[64] Legriel, J., Le Guernic, C., Cotton, S., Maler, O.: Approximating the pareto front

of multi-criteria optimization problems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 69–83 (2010). Springer

[65] Deng, L.: The MNIST database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)

[66] Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Technical report, Toronto, ON, Canada (2009)

[67] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv:1706.06083 (2017)

[68] Moosavi-Dezfooli, S.-M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2574–2582 (2016)

[69] Xu, W., Evans, D., Qi, Y.: Feature squeezing: Detecting adversarial examples in deep neural networks. arXiv:1704.01155 (2017)

[70] Svedin, M., Chien, S.W., Chikafa, G., Jansson, N., Podobas, A.: Benchmarking the nvidia gpu lineage: From early k80 to modern a100 with asynchronous memory transfers. In: Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, pp. 1–6 (2021)

[71] Xu, R., Han, F., Ta, Q.: Deep learning at scale on nvidia v100 accelerators. In: 2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), pp. 23–32 (2018). IEEE

[72] Google: GPU pricing. Compute Engine: Virtual Machines (VMS). google cloud. Google. https://cloud.google.com/compute/gpus-pricing

[73] Google: GPU pricing. Compute Engine: Virtual Machines (VMS). google cloud. Google. https://cloud.google.com/compute/vm-instance-pricing#accelerator-optimised

[74] Davidson-Pilon, C.: lifelines: survival analysis in python. Journal of Open Source Software **4**(40), 1317 (2019) https://doi.org/10.21105/joss.01317

[75] Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: A survey. IEEE Communications surveys & tutorials **18**(1), 732–794 (2015)

[76] Schmoor, C., Sauerbrei, W., Schumacher, M.: Sample size considerations for the evaluation of prognostic factors in survival analysis. Statistics in medicine **19**(4), 441–452 (2000)

[77] Lachin, J.M.: Introduction to sample size determination and power analysis for clinical trials. Controlled clinical trials **2**(2), 93–113 (1981)