

Approximation-based monitoring of ongoing model extraction attacks

- model similarity tracking to assess the
progress of an adversary

*Approximationsbaserad monitorering av
pågående modelextraktionsattacker
- modellikhetsövervakning för att uppskatta
motståndarens framsteg*

Christian Gustavsson

Supervisor : David Bergström
Examiner : Fredrik Heintz

External supervisor : Martin Karresand (FOI)

Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>.

Sammanfattning

Många organisationer vänder sig till löftet om artificiell intelligens och maskininlärning (ML) då dess användning vinner mark inom allt fler discipliner. Att utveckla högpresterande ML-modeller är dock ofta kostsamt. Designarbetet kan vara komplicerat. Att samla in stora träningsdataset är ofta dyrt och kan innehålla känslig eller proprietär information. Det finns många skäl till att maskininlärningsmodeller kan vara lockande mål för en motståndare som är ute efter att stjäla data, modellparametrar eller modellbeteende.

Det här arbetet utforskar modellextraktionsattacker och syftar till att utforma en approximationsbaserad monitorering som följer framstegen för en potentiell motståndare. När en attack är konstaterad kan åtgärder vidtas för att hantera hotet. Den föreslagna monitorn utnyttjar interaktionen med målmodellen. Den tränar kontinuerligt en monitor-modell som en fungerar som en approximation för vad angriparen skulle kunna uppnå med de data som samlats in från målmodellen.

Nyttan av den föreslagna övervakningsansatsen visas för två experimentella attackscenarier. Det ena utforskar användningen av parametriska och Bayesianska modeller för ett regressionsfall, medan det andra utforskar vanligt använda neurala nätverksarkitekturer för ett bildklassificeringsfall. Experimenten utvidgar aktuell forskning kring monitorer till att inkludera Ridge regression, Gauassian process regression och en uppsättning standardvarianter av convolutional neural networks: ResNet, VGG och DenseNet. Experimenten utforskar även likhet mellan ML-modeller och dataset med hjälp av mått från statistisk analys, linjär algebra, optimal transport samt rangapproximation.

Abstract

Many organizations turn to the promise of artificial intelligence and machine learning (ML) as its use gains traction in many disciplines. However, developing high-performing ML models is often expensive. The design work can be complicated. Collecting large training datasets is often costly and can contain sensitive or proprietary information. For many reasons, machine learning models make for an appetizing target to an adversary interested in stealing data, model properties, or model behavior.

This work explores model extraction attacks and aims at designing an approximation-based monitor for tracking the progress of a potential adversary. When triggered, action can be taken to address the threat. The proposed monitor utilizes the interaction with a targeted model, continuously training a monitor model as a proxy for what the attacker could achieve, given the data gathered from the target.

The usefulness of the proposed monitoring approach is shown for two experimental attack scenarios. One explores the use of parametric and Bayesian models for a regression case, while the other explores commonly used neural network architectures for image classification.

The experiments expand current monitoring research to include ridge regression, Gaussian process regression, and a set of standard variants of convolutional neural networks: ResNet, VGG, and DenseNet. It also explores model and dataset similarity using metrics from statistical analysis, linear algebra, optimal transport, and a rank score.

Acknowledgments

Writing one's master's thesis is probably never a straightforward journey. At least in my case, it hasn't been. Last summer, I set out to work on one thesis and ended up finishing another.

First of all, a thank you to my supervisor at FOI, Martin Karresand. You have provided me with much-appreciated guidance and feedback. And between you and me, we can still call this work by its working title: 'Computer Says No!'

I would like to thank my supervisor at LiU, David Bergström. You challenged me by introducing new ideas into my work, which made it more interesting. For instance, it would not contain Gaussian processes without you, and now I know about paper towns.

For inspiring this work, for encouragement on the way, and for the many changes along the way, I would also like to thank my examiner, Fredrik Heintz.

Lastly, a word of gratitude towards my husband, Mikael. During these last years, when I've been talking about what I do all day, I realize that you have sometimes gotten really, really bored. Yet, your support for me going back to school has been solid.

Contents

Acknowledgments	v
List of Figures	viii
List of Tables	ix
Acronyms	x
Definitions	xii
1 Introduction	1
1.1 Motivation	1
1.2 Aim	2
1.3 Research questions	2
1.4 Contributions	2
1.5 Delimitations	2
1.6 Limitations	3
1.7 Ethical considerations	3
1.8 Thesis outline	3
1.9 Tools	3
2 Machine Learning Fundamentals	4
2.1 Introduction to Machine Learning	4
2.2 Learning paradigms	4
2.3 Mathematical description	5
2.4 Validation and testing	5
2.5 Performance metrics for classification models	6
2.6 Performance metrics for regression models	7
2.7 Basic parametric models	8
2.8 Bayesian models	9
2.9 Neural networks	11
3 Model Extraction Attacks	19
3.1 Introduction to Extraction Attacks	19
3.2 Notation and terminology	20
3.3 A trivial example	20
3.4 Threat modeling	21
3.5 Attack approaches	23
3.6 Attack infrastructure	24
3.7 Defensive strategies and tactics	25
3.8 Model similarity metrics	26
3.9 Dataset similarity metrics	28

4	Related work	30
4.1	Monitor approaches	30
4.2	Model inference attacks	31
5	Method	32
5.1	Literature review	32
5.2	Proposal	32
5.3	Experimental evaluation	33
6	Design: An approximation based monitor	34
6.1	The proposal	34
6.2	Design choices	34
7	Experimental design	38
7.1	Scenario 1 setup: Mechanical system	38
7.2	Scenario 2 setup: Image classification	41
7.3	Experimental environment	46
8	Results	47
8.1	Scenario 1 results: Mechanical system	47
8.2	Scenario 2 results: Image classification	52
9	Discussion	56
9.1	Results	56
9.2	Method discussion	59
9.3	The work in a wider context	60
10	Conclusion	62
10.1	Conclusion	62
10.2	Future work	63
	Bibliography	65
A	Scenario 1: Experimental data	69
B	Scenario 2: Experimental data	78

List of Figures

2.1	Illustration of a confusion matrix	7
2.2	Illustration of Gaussian process regression.	11
2.3	A basic neural network model.	12
2.4	Tanh, ReLU and Sigmoid activation functions.	13
2.5	Process of training a neural network	14
2.6	Interpreting an image as input features to a CNN.	16
2.7	Illustration of a convolutional layer in a neural network.	16
2.8	Illustration of a pooling layer in a neural network.	16
2.9	Residual block used for residual learning	17
2.10	Schematics of a dense block used for densely connected networks	18
2.11	An example of a densely connected network	18
3.1	A trivial example using a linear regression model.	21
3.2	Illustration of the basic infrastructure involved for an attack.	24
6.1	Flow of information using the proposed method.	35
6.2	Illustration of desired behavior for a monitor threshold.	37
7.1	Scenario 1: The behavior of M_t , a mechanical system.	39
7.2	Scenario 1: Evaluating optimal parameters for M_t	40
7.3	Scenario 1: M_t fitted to training data.	40
7.4	Scenario 2: Examples from the CIFAR-10 dataset, 32x32 pixels, used for \mathcal{D}_t , \mathcal{D}_{query} and \mathcal{D}_m	44
8.1	Scenario 1: Aggregated prediction consistency	48
8.2	Scenario 1: Wasserstein-1 distance between \mathcal{D}_t and \mathcal{D}_m	49
8.3	Scenario 1: Prediction consistency for S1E1	50
8.4	Scenario 1: Parametric model similarity between M_t and S1E1.	51
8.5	Scenario 1: Prediction consistency for S1E2	51
8.6	Scenario 1: Illustration of probability density distribution for two random \mathcal{D}_m	52
8.7	Scenario 2: Aggregated prediction consistency	53
8.8	Scenario 2: Aggregated rank score results	54

List of Tables

3.1	Categorization used for threat modeling of MEA.	21
3.2	Categorization of defensive strategies and methods.	25
8.1	Scenario 1: Aggregated prediction consistency and parametric similarity	48
8.2	Scenario 1: Statistical metrics and Wasserstein-1 distance for datasets.	49
8.3	Scenario 2: Aggregated prediction consistency and rank similarity	54
8.4	Scenario 2: Distribution of images in used datasets	55
A.1	Scenario 1: Detailed experimental data for S1E1	69
A.2	Scenario 1: Detailed experimental data for S1E2	72
A.3	Scenario 1: Wasserstein-1 distances to M_t	75
B.1	Scenario 2: Detailed experimental data for S2E1-S2E2	78
B.2	Scenario 2: Detailed experimental data for S2E3-S2E4	79
B.3	Scenario 2: Detailed experimental data for S2E5-S2E6	79

Acronyms

AI artificial intelligence. 1, 2, 60

APT advanced persistent threat. 1, 23

CNN convolutional neural network. 15, 16, 22

FOI Swedish Defense Research Agency. 1

LLM large language model. 1

MEA model extraction attack. 1–3, 19–21, 24, 30–34, 60, 62, 63

ML machine learning. iv, 1–5, 8, 11, 12, 19, 20, 25, 30, 32, 36, 38, 41, 56, 59–62

MLaaS Machine-Learning-as-a-Service. 3, 19, 20, 23, 62

NN neural network. 3, 11–13, 19, 22, 26

OT optimal transportation. 29

Notation

General mathematics

Symbol	Meaning
b	a scalar
\mathbf{b}	a vector
\mathbf{B}	a matrix
\mathbf{I}	identity matrix
\mathbf{B}^T	the transpose of matrix \mathbf{B}
tr	trace operation [1]
∇	nabla; ∇f is the gradient of f .
$\ \cdot\ _1$	L1 norm
$\ \cdot\ _2$	L2 norm, i.e., Euclidean norm
$\mathbb{E}[\cdot]$	Expected value
$\text{cov}(\cdot)$	Covariance function

Machine learning

Symbol	Meaning
x	input feature scalar
\mathbf{x}	input feature vector
\hat{y}	model prediction scalar
$\hat{\mathbf{y}}$	model prediction vector
θ	parameters to be learned from training data
h	activation function
\mathbf{W}	weight matrix (neural networks)
\mathbf{b}	bias vector (neural networks)
γ	learning rate
κ	kernel function
ϵ	epsilon, error term
α	regularization term
M_t	target model, the model being attacked
\mathcal{D}_t	target dataset, $\mathcal{D}_t = \{\mathcal{D}_{train}, \mathcal{D}_{test}\}$
M_s	(antagonist) substitute model
\mathcal{D}_s	(antagonist) substitute dataset
M_m	monitor model
\mathcal{D}_m	monitor dataset ($\mathcal{D}_m = \mathcal{D}_s$)

Definitions

advanced persistent threat An adversary that possesses sophisticated levels of expertise and significant resources which allow it to create opportunities to achieve its objectives by using multiple attack vectors, including cyber, physical, and deception [2]. x, 1

adversary An individual, group, organization, or government that conducts or has the intent to conduct detrimental activities [3]. 1

artificial intelligence A branch of computer science devoted to developing data processing systems that perform functions normally associated with human intelligence, such as reasoning, learning, and self-improvement [3]. x, 1

kernel function A kernel function is a function of two arguments that computes an inner product between images of its arguments in some feature space [4]. 15

machine learning The use and development of computer systems that can learn and adapt without following explicit instructions by using algorithms and statistical models to analyze and draw inferences from patterns in data [3]. iv, x, 1, 3, 4

model extraction attack An attempt by an adversary to create a local copy of a targeted model, stealing exact model properties or model behavior [5, 6]. x, 1–3, 19, 32, 63



1 Introduction

Turning to the promise of artificial intelligence (AI), many organizations invest heavily in its development, often training machine learning (ML) models on proprietary datasets. Adopting these technologies opens the door to new possibilities, but it might also attract the unwanted attention of malicious actors. Attacking a ML model can provide damaging insights into sensitive data, the model's inner workings, and the capabilities of an organization. In some cases, stealing the entire model might be an effective shortcut for an adversary who wants to piggyback on the work of others.

This master thesis introduces the area of ML model theft, model extraction attacks, and explores a protective method that utilizes the black-box interactions between an attacker and a targeted model. The project was carried out at the Swedish Defense Research Agency (FOI).

This chapter introduces the problem studied in this thesis and project delimitations.

1.1 Motivation

Machine learning has proven useful in many disciplines, resulting in widespread use. Many everyday tools that are taken for granted, such as the smartphone in our pocket and the streaming service connected to our television sets, use AI and ML to make our lives easier and more entertaining. However, with the breakthrough of large language models (LLMs) and other generative models, more people have become aware of the potential in AI beyond auto-completing our text messages and what music to listen to next.

The development of ML models is costly, and training often depends on access to large sets of sensitive and labeled data. It could, therefore, be tempting for a competitor to take a shortcut by stealing results, reconstructing datasets, or even copying an entire model. A stolen model that is almost as good as the targeted model might be good enough for an adversary and provide important insights for future attacks.

Societally, the theft of models or the leakage of proprietary and sensitive data can seriously impact national interests. A state-backed attacker, or another advanced persistent threat (APT), might have more sinister motives than a curious competitor. It is also important to remember that attacks do not need to be significant to inflict long-term damage. A series of small, isolated, and barely noticeable incidents can also compound over time, eating away at citizens' trust in their national institutions [7].

Model and data theft can also have severe implications for individuals and organizations. Privacy issues, for example, leaks of sensitive personal information, are a growing concern. For an organization, disruptions of operations and claims of negligence can be a hard blow, resulting in loss of reputation and high economic costs. The average company cost of a data breach is estimated to be \$4.45 million per incident, according to a survey by IBM [8]. Model extraction attacks might have even more serious implications, risking the loss of both data and a trained model.

Naturally, there is great interest in exploring extraction attacks, prevention methods, and possible defenses. Protecting a model against aggression is not straightforward since it often includes trade-offs. Imposing heavy restrictions for accessing a model could, of course, be a powerful tool for protection. On the other hand, restricting access often defeats the purpose of the model, providing insights and assistance to the organization and its customers. A model can also quickly become obsolete if isolated and cut off from continued learning. Better preventive and defensive measures are needed to further the continued development of AI and ML models.

1.2 Aim

The thesis explores an approach for approximation-based monitoring of model extraction attack, utilizing interactions between an adversary and the targeted model. The aim is to approximate an ongoing attack by continuously evaluating the similarity between a targeted model and a monitor model.

1.3 Research questions

To realize the aim of this thesis, the research work seeks to answer these research questions:

RQ1 *How could black-box interactions with a targeted model be used for approximating an adversary's substitute model?*

RQ2 *What are relevant model and dataset similarity metrics?*

RQ3 *When an attempted model extraction attack is detected, what are possible defenses?*

1.4 Contributions

The main contribution of this work is that it expands current monitoring research to include ridge regression [4], Gaussian process regression [9], and a set of standard variants of convolutional neural networks [10]; ResNet [11], VGG [12], and DenseNet [13]. It explores model and dataset similarity metrics, experimenting with the usability of the RankMe score [14] for approximating model complexity and applying optimal transport to measure similarities between datasets.

This work also contributes to the notation for monitoring model extraction attacks (MEAs) and the discussion of threat modeling.

1.5 Delimitations

For the extent of this work, the assumption is made that external access to the targeted model is restricted to an API and that the model owner can differentiate between queries made by separate users. It is also assumed that the attacker has black-box knowledge, knowing nothing about the targeted model. On the other hand, the monitor model has white-box knowledge, knowing everything about the targeted model.

Other forms of model inference attacks against ML models, such as model inversion and membership inference, are not a part of this work.

1.6 Limitations

For practical reasons, the experimental work has been conducted on models and datasets of limited size, often explicitly designed for the project. Commercial services might work differently and compound training data from many different sources [15], [16]. Generalizing the experimental results for large-scale Machine-Learning-as-a-Services (MLaaS), trained on datasets of numerical values, text, and images, might not be feasible.

1.7 Ethical considerations

In works that aim at exploring vulnerabilities in systems, there is a necessity to tread carefully. It would be unlawful to attack a random network, a system, or a ML model unless necessary permissions were granted beforehand. Findings could also be used by some antagonist when attacking a system that has been explored for research purposes. Sharing results with system owners before publication, allowing them to mitigate identified issues, is of importance.

For this work, only custom made ML models based on openly available data are used.

1.8 Thesis outline

The thesis starts by laying a two-fold theoretical foundation. The fundamentals of machine learning are somewhat of a prerequisite. Hence, it begins with introducing ML in Chapter 2, together with the models used for the experimental work and relevant performance metrics. In Chapter 3, the focus turns to the main topic, model extraction attacks. Notation, threat modeling, and strategies for attacking and defending models are presented. Metrics for model and dataset similarity are also described. Related work is presented in Chapter 4.

The method used for this work is introduced in Chapter 5. The proposed design for approximation-based monitoring is presented in Chapter 6. In Chapter 7, some experimental scenarios are designed to evaluate the proposed approach, while the results of the experiments are presented in Chapter 8. The results are discussed in Chapter 9, along with a discussion on the method and the work in a wider context.

Finally, in Chapter 10, conclusions of the work are drawn, and the research questions addressed. This chapter also includes a section on future work.

1.9 Tools

Experimental setups have been run Ubuntu 22.04.3 LTS equipped with an Intel Core i7 processor, 64 GB memory, and NVIDIA GeForce RTX 3060 GPU. Scenarios designed in Python 3.10.12. Scikit-learn has been used for work on basic parametric and non-parametric models in scenario 1. For Scenario 2, doing work on neural network, Pytorch 2.2.0.dev (20231018) has been used with Cuda version 11.8.



2

Machine Learning Fundamentals

Machine learning (ML) is a term that describes the ability of a system to learn from data, generalize the results, and apply them to new, previously unseen data. A mathematical model, learned from observing training data, is at the heart of all ML. Many models and architectures have unique strengths, making them suitable for different tasks [4]. This chapter introduces the concept of ML before presenting models, architectures, and performance metrics used for this work.

2.1 Introduction to Machine Learning

Using ML has proven to be a powerful tool since large amounts of data can be absorbed into a model, capturing complex relationships that would be impossible to do manually. On the other hand, modeling is highly dependent on access to high-quality training data, which can be time-consuming and often dependent on manual labor.

Generally, training of ML models is divided into three basic paradigms: *Supervised*, *unsupervised*, and *reinforcement* learning. The output type can also be used to describe a model. A *regression* model means the output is numerical, while *classification* means the output is categorical. The output could also be the rank of items in a set, making it a *ranking* model.

2.2 Learning paradigms

A *supervised learning* algorithm builds its mathematical model by observing labeled data, i.e., it trains by observing the input and the desired output from the training data. In an iterative process, the model builds an input-output relationship and generalizes the results. The model learns a function to predict the output for new inputs, i.e., supervised learning can be said to be predictive. The model's quality depends on the quality of the training datasets [4].

The basis for *unsupervised learning* is unlabeled training data. In this case, the learning algorithm aims to find structure in the input data. One example is clustering, where similar input values are grouped based on commonalities. When observing a new input sample, the model identifies common traits, or lack thereof, with the previously grouped samples [4]. Unsupervised learning can be said to be descriptive.

There is also some middle ground. For example, *semi-supervised learning* describes using datasets where some labels are missing. It falls in between unsupervised and supervised learning.

Reinforcement learning is a training method that rewards desired behaviors and punishes undesired ones. At the center of the learning algorithm is an agent exploring an unknown environment to achieve its goal, such as maximizing its expected cumulative reward [4]. Reinforcement learning can be said to be active.

2.3 Mathematical description

Before describing the specific ML models used in this work, it will be helpful to provide a basic mathematical description of machine learning [4], [17]. This is not a general description for all ML models, but it is useful as a stepping stone towards basic parametric models and neural networks. For instance, a Bayesian approach is dissimilar to what is described here. For this discussion, a supervised training model is assumed. The model can be described as a function f . The function maps the input vector \mathbf{x} to some output prediction vector $\hat{\mathbf{y}}$. It is denoted as

$$\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta}) \quad (2.1)$$

where $\boldsymbol{\theta}$ is the model parameter vector. The complexity of the model is dependent on $\boldsymbol{\theta}$. For an already trained model asked to perform a prediction, $\hat{\mathbf{y}}$ is the requested result.

However, when training a model, the prediction is used to fine-tune the parameters in $\boldsymbol{\theta}$ to make as good future predictions as possible. In learning paradigms where a ground truth, \mathbf{y} , is available in the training dataset, a *loss function* l can be modeled by

$$l(\hat{\mathbf{y}}, \mathbf{y}) = l(f(\mathbf{x}; \boldsymbol{\theta}); \mathbf{y}) \equiv L(\mathbf{x}; \boldsymbol{\theta}) \quad (2.2)$$

where L is introduced for a shorter notation. The loss function quantifies the difference between the prediction and the ground truth. One common loss function is the MSE, described in Equation (2.9), but other functions can be used. Creating a bespoke function tailored to a specific task is also possible. Therefore, when describing an implemented model, it is necessary to specify which loss function is used.

The gradient $\nabla L(\mathbf{x}; \boldsymbol{\theta})$ can be calculated with respect to the parameter set $\boldsymbol{\theta}$. This description assumes that L satisfies the requirements for a gradient to be calculated, but that is not true for all cases. To decrease the loss, a small step of size γ is then taken in the opposite direction of the gradient,

$$\boldsymbol{\theta}^* = \boldsymbol{\theta} - \gamma \nabla L(\mathbf{x}; \boldsymbol{\theta}) \quad (2.3)$$

resulting in an updated parameter set $\boldsymbol{\theta}^*$. The step size γ is a hyperparameter chosen for the model. The updated set of parameters results in a smaller loss, such that $L(\mathbf{x}; \boldsymbol{\theta}^*) < L(\mathbf{x}; \boldsymbol{\theta})$. Step-by-step, the model improves its ability to give accurate predictions as it is trained on samples from the training data set.

2.4 Validation and testing

When developing a ML model, it is important to evaluate how well it performs, i.e., its ability to give accurate predictions. For this to be meaningful, the model must be tested on data it has not seen before. Otherwise, it is hard to know whether the model generalizes well for new data or has just memorized the answers provided during training.

Hold-out is a method that splits the dataset into two, one for training and one for testing. Their name describes their intended use. A standard split is using 80% of the data for training,

keeping 20% for testing. This method is suitable for work on large datasets; otherwise, the number of samples in the respective sets might be too small for any work of value [17].

However, there could also be a use case for validating a model. For instance, this could be used to fine-tune a model's settings, its *hyperparameters*. In this case, we also need a validation dataset. In practice, this can be managed using *cross-validation*. It is a method where the training dataset is randomly split into k groups. It is also called *k-fold cross-validation*. One group is used for validation, while the rest is used for training. This train-validation process repeats k times until all groups have been used for validation. This method allows the model to train on multiple train-validation splits, providing a clearer indication of how it will perform on new, previously unseen data [17].

In general, the dataset should be shuffled before splitting. There are exceptions to this guideline; for example, if the data captures a series of events, the splitting might need to consider temporal order.

2.5 Performance metrics for classification models

A set of standard metrics is often used for evaluating the performance of classification models, also known as logistic models. The metrics used for this work are presented based on a description by Lindholm et al. [4], using these common acronyms throughout:

- TP, True positive: The model predicted positive, and it is correct.
- TN, True negative: The model predicted negative, and it is correct.
- FP, False positive: The model predicted positive, and it is incorrect.
- FN, False negative: The model predicted negative, and it is incorrect.

These acronyms will be used for the listed performance metrics.

Accuracy

The accuracy metric

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.4)$$

measures how many percent of the predictions were correct. It is a good metric to use when the focus of a model is correctly predicting both true positives and true negatives. The result is sometimes presented in a confusion matrix, illustrated by Figure 2.1. For this example, the calculated accuracy would be 45 percent.

Precision

The precision metric

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2.5)$$

evaluates how well a model predicts positive labels. Precision is often used when the cost of a false positive is considered high, and the cost of a false negative is considered low. For the example in Figure 2.1, the precision would be 35 percent.

		Predicted label	
		Positive	Negative
True label	Positive	7	9
	Negative	13	11

Figure 2.1: Example confusion matrix, where correct predictions (TP, TN) are on the green diagonal. The calculated accuracy for this example would be 45 percent.

Recall

The recall metric

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2.6)$$

calculates the percentage of actual positives a model correctly identified. This is often used when the cost of a false negative is considered high. For the example in Figure 2.1, the recall would be close to 44 percent.

F1 Score

The F1 score combines precision and recall

$$\text{F1 Score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (2.7)$$

and is a useful metric when false positives and negatives are essential. For the example in Figure 2.1, the F1 score would be close to 39 percent.

2.6 Performance metrics for regression models

For regression models, metrics that calculate numerical differences between predictions and the ground truth are needed [18].

Squared error

The squared error metric describes the difference between two values, such as the difference between a ground truth sample, y , and a prediction, \hat{y} :

$$\text{squared loss} = (\hat{y} - y)^2 \quad (2.8)$$

Mean Square Error

Abstracting the idea of squared loss, Mean Squared Error (MSE) metric is introduced, being the average loss over a set of data samples. This is described by

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (2.9)$$

where n is the number of samples in the set.

Root Mean Squared Error

A take on the MSE is the Root Mean Squared Error (RMSE):

$$\text{RMSE} = \frac{1}{n} \sqrt{\sum_{i=1}^n (\hat{y}_i - y_i)^2} \quad (2.10)$$

2.7 Basic parametric models

Since the foundation of ML has been presented, the next step is to introduce the models. A natural starting point is basic parametric models. The description of being basic should not be misinterpreted as simplistic. These models' strengths are that they can both complete somewhat complex tasks while being transparent in their design.

A parametric model's meaning is that after training, the model only depends on a fixed set of parameters, θ , learned from training data. The training data can be discarded after the training is complete. This differs from non-parametric models, which also utilize the training data during predictions, making them slower than their parametric counterparts but less affected by outlier data samples [4].

2.7.1 Linear regression

The linear regression model is widely used for many tasks, often capturing linear problems straightforwardly. This model is a stepping stone for introducing more complex concepts and models.

The object of linear regression is to learn a model, f , that maps an input vector, \mathbf{x} , to an output prediction, \hat{y} , such that $\hat{y} = f(\mathbf{x}) + \epsilon$. The term ϵ is an error term that we consider as a random variable, or noise, independent of \mathbf{x} and with a mean value of zero. It accounts for errors in the data not captured by the model [4].

Expanding on this, a linear regression model can be described by

$$\hat{y} = \theta^T \mathbf{x} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_p x_p + \epsilon \quad (2.11)$$

where $\theta = \{\theta_0, \dots, \theta_p\}$ are the model parameters and $\mathbf{x} = \{x_1, \dots, x_p\}$ are the input variables. p is the number of parameters, and θ_0 is the offset term. T indicates *transpose* [1]. The choices of the parameters θ constitute the model.

Training the model means finding the parameters θ that perform best on observed training data [4]. This is done under a supervised learning paradigm. Assume that there is a training dataset $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^p$. The aim is to find θ , such that the difference between the ground truth from the training data, \mathbf{y} , and the model's predictions, $\hat{\mathbf{y}}$, is small. In practice; this means minimizing the cost

$$\hat{\theta} = \arg \min_{\theta} \overbrace{\frac{1}{n} \sum_{i=1}^p L(\hat{y}(\mathbf{x}; \theta), y)}^{\text{cost function}} \quad (2.12)$$

loss function

where L is the loss function and n is the number of samples. In many cases, L is the squared error loss,

$$L(\hat{y}(\mathbf{x}; \theta), y) = (\hat{y}(\mathbf{x}; \theta) - y)^2, \quad (2.13)$$

but other functions could also be used.

2.7.2 Polynomial regression

Many problems are non-linear and, therefore, require a non-linear model to describe their behavior. Linear regression, described in Section 2.7.1, uses a first-degree polynomial. For a polynomial regression [4], polynomials up to the p th degree are used and can be described by

$$\hat{y} = \boldsymbol{\theta}^T \mathbf{x} = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_p x^p + \epsilon \quad (2.14)$$

where $\boldsymbol{\theta}$ are the model parameters and \mathbf{x} are the input variables. p is the number of parameters, and θ_0 is the offset term. T indicates transpose [1].

It might be tempting to make p large to get a better model. However, a large p might result in *overfitting*, i.e., the model memorizes individual samples rather than learning relevant patterns from data [4]. In practice, a useful approach is to use validation data, see Section 2.4, on models with different p to identify an appropriate degree polynomial.

The principle for training the model is the same as for linear regression, aiming to find $\boldsymbol{\theta}$, such that the difference between the ground truth from the training data, \mathbf{y} , and the model's predictions, $\hat{\mathbf{y}}$, is small. Since the complexity of the model has increased, the cost function is now best described in matrix multiplication form. To find the best parameters [4], minimize

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \frac{1}{n} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 \quad (2.15)$$

where n is the number of samples, and \mathbf{X} , $n \times p$, is the input matrix.

Another method for mitigating overfitting is to use regularization. Regularization means an extra penalty term is added to the cost function to keep $\hat{\boldsymbol{\theta}}$ small. A common choice for regularization is using the L2 norm, $\|\boldsymbol{\theta}\|_2^2$, which transforms the polynomial regression model into a *Ridge regression* model, described in Section 2.7.3.

2.7.3 Ridge regression

Polynomial regression, described in Section 2.7.2, can capture non-linear behavior but is prone to overfitting. This can be mitigated by regularization, adding a penalty term to the cost function from Equation (2.15). When using the L2 norm for this penalty, polynomial regression is instead called Ridge regression.

The weight of the penalty is set by a regularization parameter, $\alpha \geq 0$, chosen to balance between overfitting and making the model too simplistic. Training the model means finding $\boldsymbol{\theta}$, such that the difference between the ground truth from the training data, \mathbf{y} , and the model's predictions, $\hat{\mathbf{y}}$, is small. To find the best parameters [4], minimize

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \frac{1}{n} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 + \underbrace{\alpha \|\boldsymbol{\theta}\|_2^2}_{\text{penalty term}} \quad (2.16)$$

where n is the number of samples, and \mathbf{X} , $n \times p$, is the input matrix.

2.8 Bayesian models

Taking a Bayesian approach, also referred to as a probabilistic approach. The aim is a model that predicts a distribution, rather than a single value. The model parameters are considered *random variables*, meaning their value depends on random events [4]. For instance, extending linear regression with a Bayesian approach finds the distribution over updated parameters whenever new data samples are observed.

A *Gaussian process* furthers the Bayesian approach, being a *stochastic process* [18], a collection of random variables indexed in some order. It is a non-parametric, kernel-based method

for finding distribution over all the possible functions, $f(\mathbf{x})$, consistent with the observed data. This could be used for regression and classification tasks.

A complete derivation, founded in Bayesian methods, is omitted from this work. However, some intuition is provided along with the definition and the mathematics of Gaussian processes in a *function-space* [9] setting.

2.8.1 Gaussian process regression

In their work, Rasmussen and Williams [9] defines a Gaussian process as a collection of random variables, any finite number of which have a joint Gaussian distribution. As with all Bayesian methods, work starts with a prior belief about a distribution and updates this as new data samples are observed, resulting in a posterior distribution over functions. Since it is non-parametric, there is no training phase, such as for parametric models, only inference. The training points are used during prediction, making the model slow on large datasets.

For input vectors, \mathbf{x} and \mathbf{x}' , Rasmussen and Williams [9] writes a Gaussian process as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) , \quad (2.17)$$

meaning that f is distributed as a Gaussian process and completely described by its mean, $m(\cdot)$, and covariance function, $k(\cdot, \cdot)$, which is the kernel function for this case.

A Gaussian process regression model aims not to predict a single output but to sample the possible functions from $f(X_*) = f_*$, where the matrix X_* is one or a set of data sample vectors. In Figure 2.2, this is illustrated by two versions of the same example based on a sine function. In each case, three function samples are taken from the posterior, f_* , for X_* trained on two and five data samples, $\{X, \mathbf{y}\}$. As more data samples are observed and used for training, the predicted function improves, and the confidence in that prediction increases.

However, sampling from all the possible functions would be useless without constraints. There would be an infinite amount, and most would be irrelevant. Instead, the distribution is calculated conditioned by training data samples X , with known outputs y . Following the notation by Rasmussen and Williams [9], the conditional distribution to be sampled is written as

$$\begin{aligned} \mathbf{f}_* | X, \mathbf{y}, X_* &\sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)) , \text{ where} \\ \bar{\mathbf{f}}_* &= \mathbb{E}[\mathbf{f}_* | X, \mathbf{y}, X_*] = K(X_*, X_*) (K(X, X) + \sigma_n^2 I)^{-1} \mathbf{y} , \text{ and} \\ \text{cov}(\mathbf{f}_*) &= K(X_*, X_*) - K(X_*, X) (K(X, X) + \sigma_n^2 I)^{-1} K(X, X_*) . \end{aligned}$$

K is the covariance matrix, \mathbb{E} denotes the statistical operation of calculating the expected value [18], and I denotes the identity matrix [1].

The random variables that define the Gaussian process are indexed in some order. The function value is a random variable for each such index. These values are also assumed to correlate. For \mathbf{x} and \mathbf{x}' , the function values, $f(\mathbf{x})$ and $f(\mathbf{x}')$, should be highly correlated if the index difference between the two input vectors is small.

The covariance matrix, K , consists of values calculated with a kernel function. A common use for this is the *squared exponential* (SE), specifying the covariance between pairs of random variables as

$$\text{cov}(f(\mathbf{x}), f(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}')_{SE} = \sigma^2 \exp\left(-\frac{(\mathbf{x} - \mathbf{x}')^2}{2l^2}\right) , \quad (2.18)$$

where \exp is the exponential function [18], l is a length scale, and the output variance, σ^2 , is a scale factor.

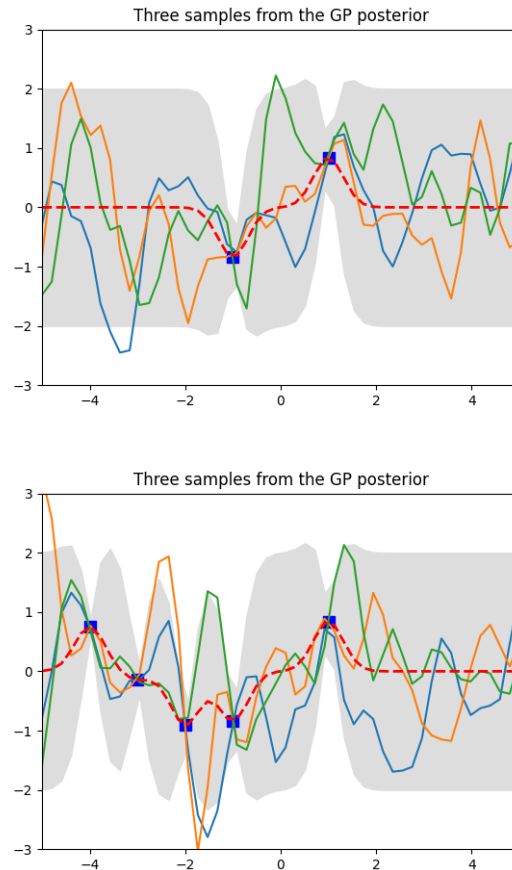


Figure 2.2: A Gaussian process example with three samples taken from the posterior, f_* . The underlying function is a sine function, for which X_* contains two (left) and five (right) observed samples. The red dashed line is the mean of the three samples. The gray field visualizes the confidence interval. As the number of observations increases, the possible functions to sample from decrease, making the model better at predicting.

2.9 Neural networks

Neural networks (NNs), also called *artificial neural networks*, is a branch of ML that loosely mimics the concept of a biological brain. They are useful for large, non-linear problems or complex data, such as images. The drawback is that they are computationally costly, often making them a poor choice for simple tasks. It can also be hard to fully comprehend the inner workings of such complex architectures.

A NN is comprised of layers with *neurons*, also referred to as *nodes*. The basic structure consists of an input *layer*, one or more hidden layers, and an output layer. Illustrated in Figure 2.3 is a network with three nodes in the input layer, two hidden layers with five nodes each, and an output layer with two nodes. This means it takes an input vector of three *features*, and the network will output a vector of two values, such as the probability of the input sample belonging to one of two classes.

Connections of nodes between different layers represent a flow of information from one node to another. All such *edges* are associated with a *weight*. Each layer could also have a constant, non-zero *bias* vector. It is added to the product of node features and edge weights, offsetting the result in some direction.

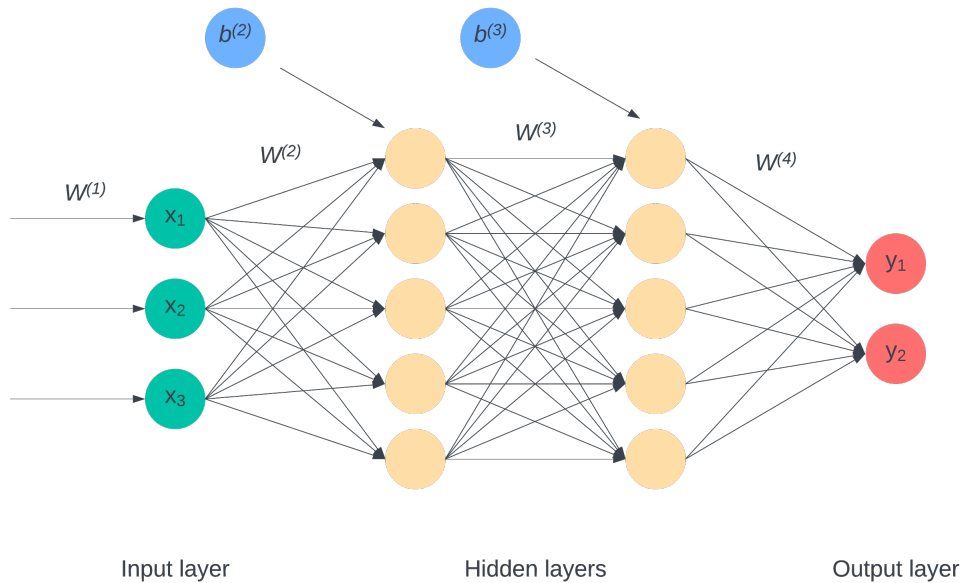


Figure 2.3: A neural network with three nodes in the input layer, two hidden layers, and two nodes in the output layer. A weight is associated with each connection, and a bias is associated with each node.

The network in Figure 2.3 is an illustrative example to derive some generally valid expressions. Following the notation used by Lindholm et al. [4], the layers are denoted $l = 1, \dots, L$, where $L = 4$ in the example, the input feature vector as \mathbf{x} and the output vector $\hat{\mathbf{y}}$. The set of edge weights leading into a node is denoted $\mathbf{W}^{(l)}$, and the bias vectors are denoted $\mathbf{b}^{(l)}$. The activation function for each layer, later described in Section 2.9.3, is denoted h . No activation function or bias vector is typically associated with the input layer.

For our example, it is assumed that all weights $\mathbf{W}^{(1)}$ are equal to 1, meaning that there are no weights applied for the input vector. It is also assumed that $\mathbf{b}^{(1)} = \mathbf{b}^{(4)} = 0$. These assumptions are true for our example, but not for all cases. The layers of our network, \mathbf{q} , can now be described as

$$\begin{aligned} \mathbf{q}^{(1)} &= \mathbf{x} \\ \mathbf{q}^{(2)} &= h(\mathbf{W}^{(2)}\mathbf{q}^{(1)} + \mathbf{b}^{(2)}) \\ \mathbf{q}^{(3)} &= h(\mathbf{W}^{(3)}\mathbf{q}^{(2)} + \mathbf{b}^{(3)}) \\ \hat{\mathbf{y}} &= \mathbf{q}^{(4)} = h(\mathbf{W}^{(4)}\mathbf{q}^{(3)}) \end{aligned} \quad (2.19)$$

Based on this, a general description of a neural network of L layers can be presented as

$$\begin{aligned} \mathbf{q}^{(1)} &= h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}), \text{ for } l = 1 \\ \mathbf{q}^{(l)} &= h(\mathbf{W}^{(l)}\mathbf{q}^{(l-1)} + \mathbf{b}^{(l)}), \text{ for } 1 < l < L \\ \hat{\mathbf{y}} &= \mathbf{q}^{(L)} = h(\mathbf{W}^{(L)}\mathbf{q}^{(L-1)} + \mathbf{b}^{(L)}), \text{ for } l = L \end{aligned} \quad (2.20)$$

2.9.1 Hyperparameters

The choice of hyperparameters can significantly impact how a NN operates and, therefore, its results. A hyperparameter is a parameter with a constant value set before training. One such constant is the learning rate, also called step size, γ , used in the mathematical description of ML in Section 2.3. Other hyperparameters are the number of hidden layers in the model

or the batch size used for training. The constants can be chosen using optimizers, such as Hyperopt¹ or Optuna², or based on experience.

2.9.2 Basic architectures

The NN architecture can be adapted for the intended use. Some terms are commonly used and should briefly be mentioned [4, 17].

If data passes through a layer once, that layer is referred to as a *feed-forward layer*. If the entire network is one-directional, it is a *Feed-forward Network*. A network with bi-directional flows is called a *Recurrent Neural Network*. If all nodes in one layer are fully connected to all nodes in the next layer, these layers are said to be fully connected, or an *affine layer*. If there are more than two hidden layers in a network, it is a *Deep Neural Network* used for *deep learning*.

The NN example in Figure 2.3 is a feed-forward and a fully connected network.

2.9.3 Activation functions

An activation function, denoted h , takes the node input, z , and calculates the node output. It decides whether a node will be activated or not. What function to use for this purpose is a design choice, where the Rectified Linear Unit (ReLU) function is the most commonly used for general cases:

$$h(z) = \max(0, z) \quad (2.21)$$

ReLU ensures that the node output is always ≥ 0 [4]. The Sigmoid function, also called the *Logistic function*,

$$h(z) = \frac{1}{1 + e^{-z}} \quad (2.22)$$

is another common choice. It is a continuous and monotonic function, also ensuring output values in the interval $[0, 1]$, often used when predicting a probabilistic output [4]. The tangens hyperbolicus function

$$h(z) = \tanh(z) \quad (2.23)$$

shares characteristics with the sigmoid function but instead maps the output towards the interval $[-1, 1]$ [19]. In Figure 2.4, these three functions are illustrated for values of z close to zero. Numerous other functions can be used, including building a bespoke function tailored to the task.

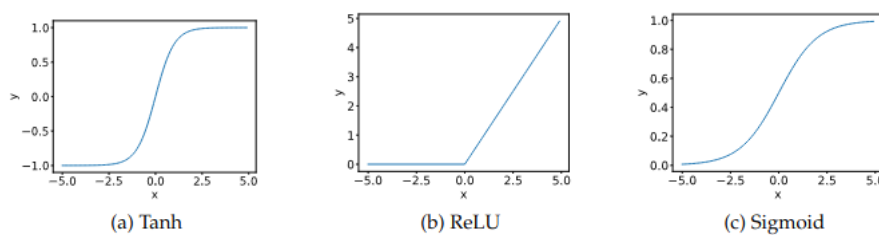


Figure 2.4: Tanh, ReLU and Sigmoid activation functions.

To normalize the output vector, making it possible to interpret it as a probability distribution, it is common to use the softmax activation function

¹<http://hyperopt.github.io/hyperopt/>

²<https://optuna.org/>

$$h(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}} \quad (2.24)$$

that applies the exponential function for the input element z_i , divided by the sum of the exponentials of all values in the input vector \mathbf{z} . The softmax function extends the Sigmoid function, sharing its characteristics shown in Figure 2.4 (c).

2.9.4 Training neural networks

The mathematical fundamentals for supervised learning of machine learning models have already been presented in Section 2.3. However, it is helpful to expand on this topic for neural networks.

A neural network is parametric. Training a neural network is to find the optimal parameters for each layer, which can be described using the same mathematical notation as in Equation (2.12). The parameter matrix is denoted θ , and consists of the weights, $\mathbf{W}^{(l)}$, and biases, $\mathbf{b}^{(l)}$, where $l = 1, \dots, L$ and L is the number of layers.

$$\theta = \left[\mathbf{W}^{(1)T} \mathbf{b}^{(1)T} \dots \mathbf{W}^{(L)T} \mathbf{b}^{(L)T} \right] \quad (2.25)$$

The supervised learning process for a neural network can be described in three stages: the *forward pass*, the calculation of the loss function, and the *backward propagation*. These three steps are schematically illustrated in Figure 2.5 and then expanded on step-by-step.

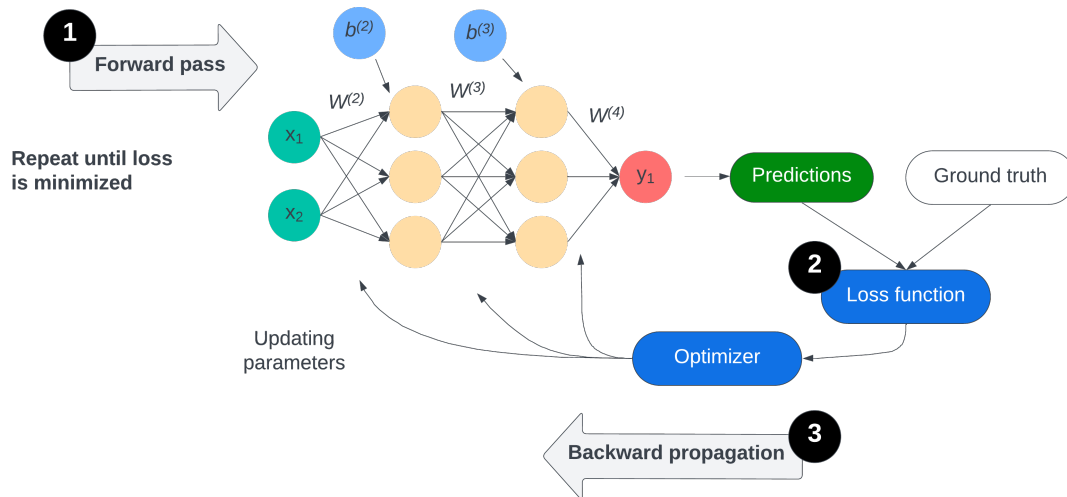


Figure 2.5: Schematics of training a neural network, with (1) forward pass, (2) calculation of loss function, and (3) backward propagation.

Training a neural network repeats these three steps until reaching some desired level of performance or some preset criterion for stopping.

Forward pass

The forward pass feeds input features into and through the model, as shown in Figure 2.5. This process has previously been described in Section 2.9. The purpose of this is to retrieve predictions for calculating the loss.

Calculating loss function

Loss is calculated by using the prediction outputs of the network and comparing them to ground truth from the training dataset, as shown in Figure 2.5. For regression tasks, squared

error loss is often used for a loss function, described in Section 2.5. For classification tasks, cross-entropy is often used [4]. The loss function, L , using cross-entropy is described as:

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\ln g_y(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})) = -z_y + \ln \sum_{j=1}^M e^{z_j}, \quad (2.26)$$

where $\mathbf{z} = [z_1, \dots, z_M]^T$ is the output of the last layer, referred to as logits, before the softmax function, and z_j is the j th output, $j = 1, \dots, M$. The softmax function maps \mathbf{z} to the class probability $\mathbf{g} = [g_1, \dots, g_M]^T$:

$$\mathbf{g}(\mathbf{z}) = \text{softmax}(\mathbf{z}) = \frac{1}{\sum_{j=1}^M e^{z_j}} [e^{z_1}, e^{z_2}, \dots, e^{z_M}]^T \quad (2.27)$$

The next step is to update the network parameters using the loss score calculated using the loss function. Testing and validation, previously introduced in Section 2.4, is also commonly used after training.

Backward propagation

During the backward propagation, the third step of Figure 2.5, the network parameters get updated based on the new knowledge from calculating the loss using an optimizer function. For this work, stochastic gradient descent (SGD) is used for the optimizer function, introduced for network training by Amari [20].

In short, a gradient decent optimizer computes the gradient of the loss with respect to the parameters. The network parameters are then changed in the direction opposite of the steepest gradient, multiplied with a hyperparameter learning rate, γ , set before training. However, while ordinary gradient descent uses all training data to compute the gradient, stochastic gradient descent uses randomly chosen subsets of training data for each iteration.

SGD has the advantage of being efficient for large datasets and is generally considered to provide fast convergence. It also has a regularization effect, preventing overfitting. However, the stochastic nature also contributes a risk for noisy, oscillating updates during training.

2.9.5 Convolutional neural networks

The Convolutional neural network (CNN) architecture was initially constructed for problems where the model input is grid-like, introduced in work by [10] on recognition of handwritten zip codes. A typical use case is image processing, where two pixels close to each other typically have more in common than two far apart. They are also useful for other inputs, such as a time series of radio waves [17]. Samples neighboring each other typically have more in common than two samples placed far apart.

This section introduces the fundamentals of convolutional networks while variants adding to this architecture are introduced in Sections 2.9.6 to 2.9.8.

Consider an example of a grayscale image, 6×6 pixels, as illustrated in Figure 2.6. Each pixel can be represented by a value ranging from 0 (black) to 1 (white). Values in between represent different shades of grey. Putting these values in a 6×6 matrix provides a numerical image representation.

The 36 numerical values in the matrix of Figure 2.6 could be made into a long feature vector and fed into the following hidden layer. However, by doing so, the structural data of the image would be lost. We want to process the data in a way that preserves the structure. This is done by introducing the notion of *convolutional* and *pooling* layers.

In a convolutional layer, a kernel function, κ , is slid over the input features matrix. It operates on every element while at the same time extracting spatial information from the neighboring region, as illustrated in Figure 2.7. The result is then fed forward in the network.

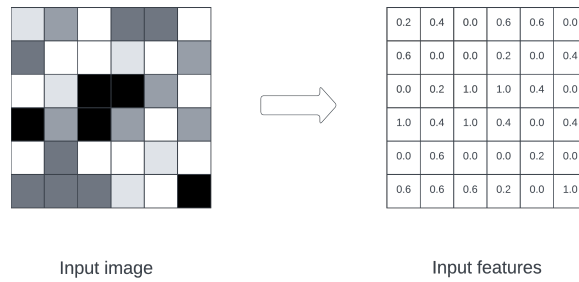


Figure 2.6: Example of an input image being converted into a numerical matrix representation, making up the input features for the network.

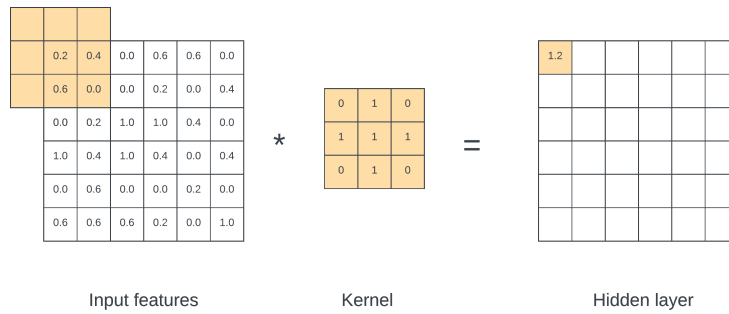


Figure 2.7: Illustration of a convolutional layer, where a kernel slides across the input features, populating the nodes of the following hidden layer. The kernel is arbitrarily chosen for the case of illustration.

A pooling layer summarizes information from a previous layer, often used after or as a part of a convolutional layer. Here, a filter function also slides over a region of elements, pooling the included elements. *Stride* is the term used for how many steps the function slides each time. *Max pooling* layers are the most common, where the maximum value in the region is fed forward, as illustrated in Figure 2.8. *Min pooling* and *average pooling* are other common filters used.

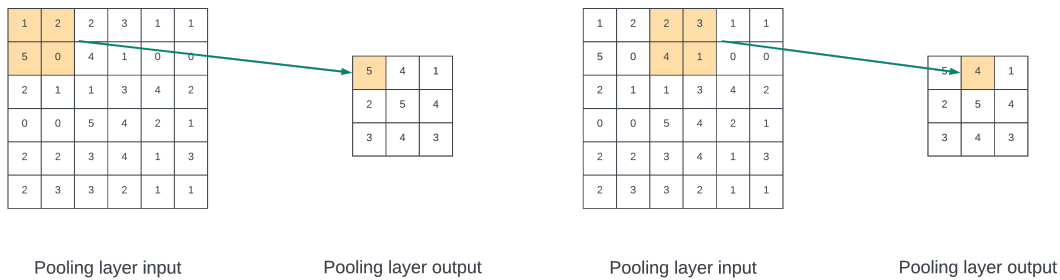


Figure 2.8: Illustration of a pooling layer with a filter size of 2 × 2 and stride 2.

When implementing a CNN, a series of convolutional layers of different configurations would typically be used in the first part of a network. It would then look more like the standard, often affine layers for the resulting predictions.

Note that the theory presented can be abstracted into higher dimensions, such as a color image input.

2.9.6 Very Deep Convolutional Networks

In their work on investigating how accuracy for CNNs is affected by network depth, Simonyan and Zisserman [12] introduced Very Deep Convolution Networks (VGG). In their work, they show that representation depth, using a deep network with more hidden layers, can be beneficial for accuracy in image classification. The standard VGG models range from 11 to 19 layers.

2.9.7 Deep Residual Neural Networks

The work on the VGG architecture [12] shows the benefits of using more layers. However, adding more layers might not always result in higher accuracy. Using models with many layers, deep learning is prone to vanishing or exploding *gradients* during backward propagation. In their work, He et al. [11] show that *residual learning* can improve the accuracy of deep convolutional networks for image classification.

Residual learning allows input to a *residual block* (of layers), also called residual learning functions, to be added to its output, as illustrated in Figure 2.9. This means that for an input x to a residual block F , the output is $F(x) + x$. This is known as adding a *residual connection*. Besides being processed by the layers, the input also trickles past them.

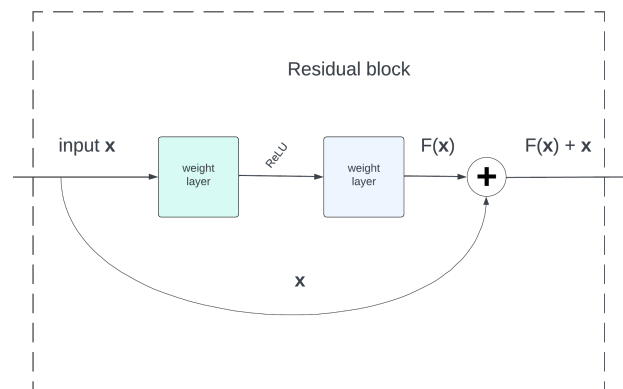


Figure 2.9: A residual block used for a residual learning architecture such as ResNet. Based on work by He et al. [11].

These residual blocks can be stacked together for a residual network, forming a very deep neural network. The standard ResNet architectures range in depth from 18 to 152 layers.

2.9.8 Densely Connected Neural Networks

In their work, Huang et al. [13] also tries to address the issues of increasingly deep neural networks. *Dense blocks* are introduced based on maximizing information flow between layers. Each layer is connected in a feed-forward fashion within each dense block, as illustrated in Figure 2.10. Each layer gets inputs from all preceding layers and passes on its own output features to all subsequent layers.

The DenseNet architecture shares similarities with the residual learning approach. However, where ResNets sum features before being passed on to the next layer, illustrated in Figure 2.9, DenseNets combine features by concatenating them. This means that DenseNets explicitly differentiates between information added to the network and information preserved by the connections.

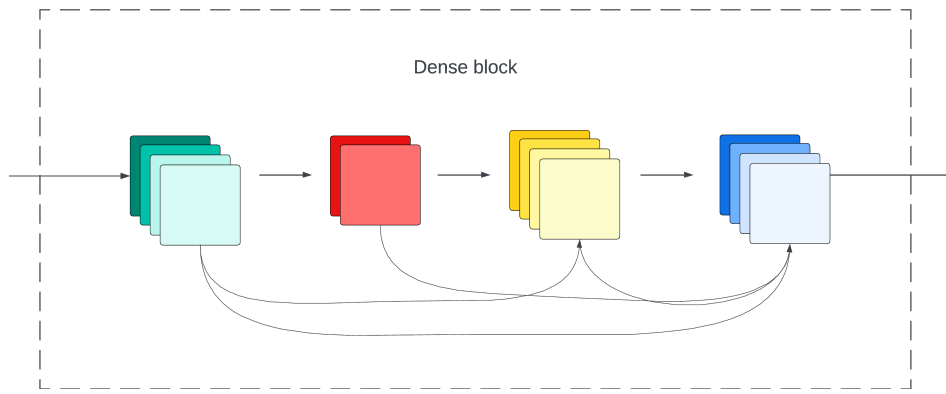


Figure 2.10: Schematic example of a four-layer dense block, a building block of densely connected networks. Based on work by Huang et al. [13].

A deep network can be constructed by combining several dense blocks, as illustrated in a simple example model in Figure 2.11. Intermittent layers, so-called *transition* layers, change the feature size by convolution and pooling layers.

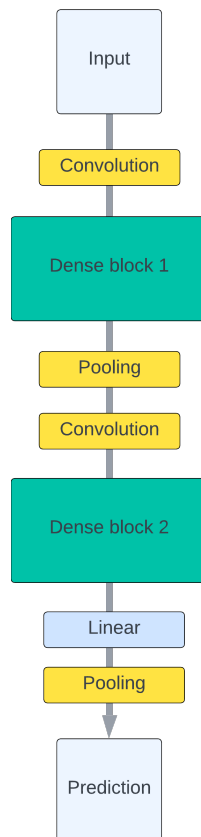


Figure 2.11: A deep DenseNet consisting of two dense blocks. Intermittent layers are called transition layers, changing feature size using convolution and pooling. Based on work by Huang et al. [13]

In their work, Huang et al. [13] showed that the densely connected network approach can achieve state-of-the-art accuracy. The standard DenseNet architectures range in depth from 121 to 201 layers.



3 Model Extraction Attacks

Developing high-performing ML models is expensive. The design of a model is often complex work based on experience and testing. Collecting large training datasets constitutes another costly challenge and might expose an organization's sensitive and proprietary data. Finally, the model training can in itself be computationally costly [21, 22]. For many reasons, ML models make for an appetizing target to an adversary.

A model extraction attack (MEA) aims to steal the exact model properties or its behavior, achieved without gaining access to the original training data. Gaining this knowledge about a model can serve several purposes. One is circumventing restrictions placed on the use of a model by making a substitute model that copies the behavior of the targeted model. Another is cutting time and resources spent on developing some other model by stealing already proven properties. Knowledge of a model's inner workings can also facilitate future attacks, such as getting past a ML based surveillance system, triggering model actions, targeting training data, or revealing an organization's capabilities.

This chapter introduces the topic of extraction attacks, laying a foundation of relevant notation before describing threat modeling and approaches for attacks. Metrics for model and dataset similarities are also included, providing tools for evaluating results for this work. Finally, related work is presented, putting the work of this master thesis in its context.

3.1 Introduction to Extraction Attacks

The idea of ML can not be said to be one thing. It comes in many forms, trained for different purposes, from basic concepts that could be calculated with pen and paper to neural networks with complex architectures. This means that different models are susceptible to different forms of attack. There is no one-attack-fits-all method, and a higher model complexity tends to make models more vulnerable to attacks by malicious actors [23].

There are several recent surveys into the area of MEA. Oliynyk et al. [5] contributes a comprehensive systematization of knowledge. Their work lays a foundation for the notation, terminology, and threat modeling. Alongside a survey by Genç et al. [6] it also provides an overview of suitable attack methods for different model architectures. It is noted that there is a lack of studies for extracting regression models [6].

The specific case of stealing models through commercial Machine-Learning-as-a-Services (MLaaS) is explored in several research papers. Tramèr et al. [15] present a generic *equation-*

solving attack for classification models and a novel path-finding algorithm for attacking *decision trees*. Targeting two major cloud service providers shows how prediction APIs can be efficiently utilized for an attack, violating training data privacy and aiding in model extraction. Similarly, Liu et al. [16] show successful attacks against two major MLaaSs. Their paper focuses on cutting the cost of attack, a perspective that is otherwise lacking in most research and this work. Their approach uses pre-trained models and task-relevant data for querying the target model.

3.2 Notation and terminology

This research area has no well-established use of notation and terminology, even if some basic elements are reoccurring [6, 5]. Some common terminologies are used for this work, but it is also extended to suit work on monitoring MEA and to provide the reader with a more intuitive notation.

An adversary aims to steal the *target model*, denoted M_t . It is trained on the *private dataset*, which for this work will be called the *target dataset*, denoted \mathcal{D}_t . In turn, this dataset contains the train and test dataset, such that $\mathcal{D}_t = \{\mathcal{D}_{train}, \mathcal{D}_{test}\}$.

The adversary uses the target model as an *oracle*, collecting pairs (x, y) for the *substitute dataset*, denoted \mathcal{D}_s . x is a data sample, a query, that the attacker sends to the oracle. The output y is the prediction made by the target model, i.e., $y = M_t(x)$. A copy, or an approximation, of the model obtained by an adversary is the *substitute model*, denoted M_s .

To perform an attack, the adversary has some infrastructure in place. It is used for the systematization of querying the target model and using the resulting input-output pair to train M_s . The queries could consist of random numbers scanning some interval or a dataset of images prepared in advance. For more advanced attacks, queries could be generated by a machine learning model, an *attack model*, that adapts to observations. If the attack model is trained on some dataset, this dataset is denoted a *surrogate dataset*.

Since this work adds a monitor to the setup, a notation for that model and dataset group is needed. The monitoring dataset that also collects the input-output pairs (x, y) is called the *monitor dataset*, denoted \mathcal{D}_m . The monitor model, approximating M_s , is denoted M_m .

The term *query* describes a question put to the targeted model. However, 'trained on queries' refers to a model's training on the input-output pair(s). The input-output pairs are the results of querying M_t .

If y is the only output an adversary can obtain from M_t , the target model is described as a *black-box*. A user has black-box access to the target model. If the architecture and parameters of the model are known, then it is described as a *white-box* model. States in-between are called *grey-box* models.

3.3 A trivial example

Stealing a ML model is not necessarily a trivial task, yet turning to a trivial example might prove helpful in describing its underlying methodology.

Imagine that data has been gathered for a simple, two-dimensional, linear problem. Provided x , the result is y . Even if you might not think of it as ML, turning the data into a linear regression model, $y = f(x)$, is learning and generalizing behavior from data, even if it is done in a spreadsheet program.

The linear regression model is a straight-line equation, $y = mx + c$. From mathematics, it is known that only two measurements are needed to solve for the unknown coefficients, m and c . Turning to the example in Figure 3.1, it is easily observable that the model is fully described by $y = f(x) = 2x + 1$. A successful MEA is performed, and the model is stolen.

This introductory example is simple. Putting two queries into the targeted model gives up all its secrets. Spending time stealing this simple linear regression model is probably not

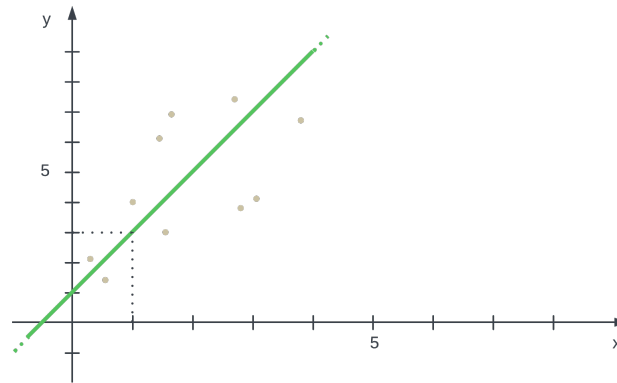


Figure 3.1: A trivial example using a linear regression model.

worthwhile, but it provides an accessible introduction to how a query-based extraction attack operates. More complicated models can be copied or approximated by probing a model with enough queries and learning from its output.

3.4 Threat modeling

From the perspective of a model owner, preventive and defensive measures might be needed in order to protect against theft. What strategies and tactics are used might depend on a potential adversary's objectives, incentives, and capabilities. The basis for this should be threat modeling, providing important insights into the mechanics of a potential attack.

Ascertaining the objectives and capabilities of an adversary is important when evaluating whether an attack has achieved its goals. Naturally, it is also fundamental to evaluating whether a defense has been successful. For work on exploring MEA, the premise of an attack is relevant for drawing conclusions [5].

For the topic of threat modeling and categorization of attack objectives, this work adheres to descriptions formulated by Oliynyk et al. [5] (pages 7-11), which, for convenience, is summarized below. However, regarding the issue of adversarial capabilities, the description has been extended somewhat to suit this work.

To facilitate an overview, the categorization is summarized in Table 3.1 before being expanded on in detail.

Table 3.1: Categorization used for threat modeling of MEA, later expanded on in detail.

Attacker incentives	Attacker objectives	Attacker capabilities
<ul style="list-style-type: none"> • Exploit a copy of the targeted model • Opening the model for further attacks 	<ul style="list-style-type: none"> • Stealing exact model properties • Stealing model behavior 	<ul style="list-style-type: none"> • Knowledge • Actions • Resources • Skill level

3.4.1 Attacker incentives

This work distinguishes between two main incentives for attempting a model extraction, described by Oliynyk et al. [5]:

- *Exploit a copy of the target model*: For a target model, restrictions might be imposed on its use. For an adversary, obtaining some copy allows unlimited use, avoiding daily caps or fees. Stealing parts of the model structure might also be a strong motivation, providing shortcuts or helping an adversary to improve its models, not necessarily within the same field.
- *Opening a target model for further attacks*: An attack is often greatly simplified if the attacker has some knowledge of the targeted model. An attack that reveals the structure of a target model must be considered as a possible intermediate step, prying a black-box model open for future grey-box or white-box attacks.

3.4.2 Attack objectives

For this work, attack objectives are categorized into two main categories based on work by Oliynyk et al. [5]:

- An attack with the objective of **stealing exact model properties** aims to reveal a model’s design. This knowledge can greatly benefit an adversary since malicious access to parameters, hyperparameters, and architecture means an entire model can be replicated. Even if only parts of model properties are stolen, they can provide a convenient shortcut. Design choices are complicated, and stealing the parameters or architecture from a proven model can save time, increase efficiency, and, therefore, be of great value. Insights into a model design can also be a stepping stone for future attacks aimed at a model or an organization.

The three potential targets of this type of attack are:

- *Training hyperparameters*: The aim is to reveal the hyperparameters used for training a target model.
 - *Architecture*: Attacks focused on charting the architecture of a target model, mainly aimed at NNs. The objective can be to ascertain the number of layers, layer types, or kernels used in specific types of networks, such as a CNN.
 - *Learned parameters*: An attack that often requires the architecture of a target model to be known and aims at stealing the weights of a model. If successful, the adversary has made a copy of identical behavior as the target model.
- An adversary might not be interested in the structure of the target model. Instead, the aim might be to **steal model behavior** — something that works just as well, especially where the details are of no concern.

The two different descriptions of what it means to steal a model behavior are:

- *Same level of effectiveness*: Given a target model, M_t , the attacker aims at making an approximate copy, a substitute model, M_s , that performs similarly on original data. The adversary can use M_s to solve tasks without the restrictions on using M_t . If the focus of the attack is reaching the same level of effectiveness, M_s doesn’t need to share the architecture of the targeted model.
- *Prediction consistency*: The attack aims to make an approximation of a target model, M_t , such that a substitute model, M_s , outputs predictions consistent with M_t . The substitute model should do the same if the targeted model misclassifies an input. The adversary gains knowledge of predictions available to the targeted organization.

3.4.3 Attacker capabilities

The possible capabilities of an adversary are relevant in modeling threats. A point will be made for extending this categorization, but for this work, attack objectives are initially categorized into three main categories based on work by Oliynyk et al. [5]:

- *Knowledge*: What knowledge does the adversary have about the target model? White-box, grey-box, or black-box knowledge not only makes a difference in the possibility of a successful attack, but it can also affect an adversary's objectives.
- *Actions*: The assumption for this work is that an adversary uses *query-based* attacks. If an adversary has hardware or software access to the computing resource where the model is deployed, it opens up the model for *side-channel attacks*.
- *Resources*: What resources does an adversary have that could be put towards the model? If a query-based attack is considered, then the number of queries that can be put into the model, the *query budget* of the adversary, is important.

In considering an adversary's capabilities, it is also reasonable to discuss its skill level. The overall technical skills and resources available to an adversary are important for correctly assessing a threat. Knowledge is not only about knowledge of the system but also technical knowledge in a greater sense. An APT, like an industrial competitor or a state-backed attacker, would have far greater access to knowledge and resources than a curious individual.

- *Skill level*: What are the potential adversary's overall level of technical skills? What overall resources could a potential adversary put towards extracting the target model?

3.5 Attack approaches

Extraction attacks can be categorized by the method used for the attack. This work will explore two of these categories [6, 5]: *Equation-solving attacks* and *Retraining attacks*. These approaches are query-based attacks, querying a model and using the model output to extract the desired information. For both approaches, research lacks work on regression models.

3.5.1 Equation-solving attacks

An equation-solving attack commonly aims to acquire the exact model properties. They were first proposed by Lowd and Meek [24] as a method for investigating linear classifiers. The algorithms proved not to be query-efficient. However, later work by Tramèr et al. [15] showed that attacks using this approach could effectively target two MLaaS. Their approach to the problem was to input samples \mathbf{x} , collecting outputs $\hat{\mathbf{y}}$. Together, these values constitute a system of equations, $\hat{\mathbf{y}} = M_t(\mathbf{x})$, where the solutions reveal the parameters of the model. Since the values are only used for setting up the mathematical equations, the input-output values do not need a real meaning.

Even though the equation-solving approach seems straightforward and can perform good results, it has clear limitations in actual deployment. Its architecture has to be known to set up a relevant system of equations to break a model. This requires a white-box knowledge of the model, either by insider knowledge or by a prelude attack [6].

Reith et al. [25] conducted the only study using this approach for regression models aimed at a support vector regression machine in a white-box setup.

3.5.2 Retraining attacks

In retraining attacks, the target model's inputs and outputs are used as training data for the adversary substitute model [6]. This approach was introduced by Tramèr et al. [15], targeting

models that output class labels. As in the case of equation-solving attacks, Section 3.5.1, Reith et al. [25] have conducted the only study on regression models. This work is limited to attacks on discriminative models, where the output is numerical values or a class label. However, research has also explored retraining attacks aimed at generative models, even if they are not explored here.

A retraining attack considers the target model, M_t , as an oracle. By querying the oracle with input \mathbf{x} , an output, $\hat{\mathbf{y}} = M_t(\mathbf{x})$, is obtained. The set of input-output pairs, (x_i, \hat{y}_i) , where $i = 1, \dots, n$, constitutes the substitute dataset, \mathcal{D}_s , used to train a substitute model, M_s . Since M_s trains on the relationship described by the input-output pair rather than the original training data for the target model, there is no need for M_s to have the same design as the target model [5].

3.6 Attack infrastructure

An adversary must have an infrastructure to attempt an MEA. Assuming a query-based attack, the adversary needs access to the target model API, a method for putting relevant queries to it, a method for collecting the API output, and a pipeline for training a substitute model. The design of the individual components might depend on the objective and approach. Stealing a linear regression model, as presented in the trivial example from Figure 3.1, requires less work than engaging a complex neural network architecture.

Considering the illustration in Figure 3.2, some components are already familiar. In the substitute dataset, \mathcal{D}_s , an adversary stores the input-output pairs collected from API querying. The substitute model, M_s , is generally the end goal of an attacker, making a copy of the targeted model.

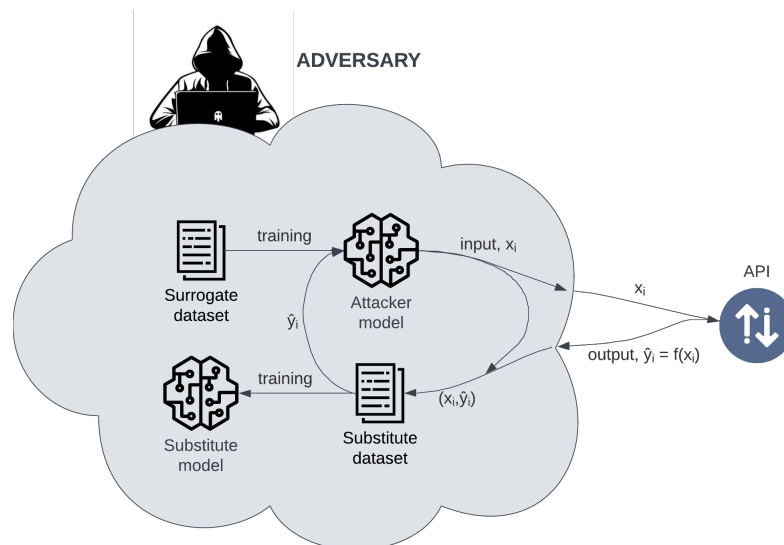


Figure 3.2: Illustration of the basic infrastructure involved for an attack.

The two components not previously discussed are the ones performing the actual attack. The term *attacker model* refers to a systematic method for putting queries to the API. This could be as simple as picking random values from a defined numerical space. It could also be a neural network, for example, a generative model, tasked with inputting queries into M_t . If the attacker model is trained on a dataset, this is called the *surrogate dataset*. The adversary could also access the input-output-pairs already collected in \mathcal{D}_s for the attack.

3.7 Defensive strategies and tactics

Protecting information against an adversary is seldom an easy task. An attacker only needs to find one way in, while a defender might need to find all such loopholes to protect against attacks. One strategy for protecting an ML model would be restricting access, letting almost no one use it. This might be a solution in some extreme cases; however, in most cases, it would defeat the purpose of developing the model. There needs to be a trade-off between protection and accessibility, rooted in a threat modeling for the model at hand. Sometimes, these strategies could also be combined to meet multiple threats [5].

In research, there is confusion about the use of the terms strategy and tactics. For this work, the presented concepts are categorized according to Table 3.2. One concept can have applications both as a strategy and as a tactic.

Table 3.2: Categorization of defensive strategies and methods, later expanded on in detail.

Defensive strategies	Defensive tactics
<ul style="list-style-type: none"> • Monitoring • Differential privacy • Model obfuscation • Model distillation 	<ul style="list-style-type: none"> • (user) Cut-off • Watermarking • Differential privacy • Output perturbation

The basic assumption for the listed defense strategies and tactics is that user access to a model is restricted to an API and that the model owner can differentiate between calls made by separate users. This is itself considered a defense mechanism.

3.7.1 Monitoring

A monitor-based defense infers information from the user’s interactions with the API or by studying the model’s behavior. When some threshold is surpassed, actions can be taken. Several methods are available for implementation depending on the monitoring purpose. Some approaches are focused on statistical measures to find malicious intent in the user’s interactions [26, 27, 28]. In contrast, others focus more on the behavior of the target model [29, 30, 31]. This topic is explored extensively in Related work, Chapter 4.

3.7.2 Watermarking

The use of watermarks has a long history of protecting intellectual property [32]. In the context of ML it could be used for marking training data or the model output. This method does not explicitly prevent model extraction attacks. However, knowing that the material contains watermarks might be a deterrent since it facilitates the claim of ownership. Another approach for watermarking datasets can be to add fictitious data, such as some map makers historically adding paper towns to see if others copied their work¹.

In their work, Jia et al. [32] identifies limitations in simple methods for watermarking material, proposing a method for what they call *entangled watermarks*. This approach is then implemented and tested for image and audio datasets. For text-based data, the work by Kirchenbauer et al. [33] proposes an approach for watermarks in Large Language Models. Using some unique model identifier or model watermarking could also be a possibility [5].

¹<https://www.thesimplethings.com/blog/paper-towns>

3.7.3 Output perturbations

For a retraining attack, an adversary depends on getting as precise output as possible from the target model. A small perturbation in the output might have little consequence for a benign user, but training a substitute model on large sets of input-output pairs with small imperfections could make the model drift. This could be in the passive form of rounding or truncating values. In their work, Orekondy et al. [34] propose an active approach for controlled perturbations against attacks on deep neural networks while maintaining utility for the benign user.

3.7.4 Differential privacy

Differential privacy is a mathematical framework introduced by Dwork [35], aimed at ensuring privacy for individuals in datasets. Later on, this has evolved into methods that can protect entire models. In their work, Zheng et al. [36] propose a boundary-sensitive approach to differential privacy for models. A cached response is given as output if a query has already been processed to minimize the risk of an adversary learning from access to multiple perturbed predictions for the same query. If the query has not been seen before, the prediction is calculated and checked against the sensitive boundaries in the model. If the query is deemed sensitive, a boundary response algorithm is invoked.

3.7.5 Model obfuscation

Even if two NN have the same functionality, their weights might differ. Since initial training generally starts with random weights, the result might be networks with similar accuracy but different weights - even when using the same architecture and training on the same data. An adversary could take advantage of this, stealing model properties, making minor weight changes (scaling, noise, etc.), and claiming ownership.

In their work, Szentannai et al. [37] proposes a method for making this theft more difficult by obfuscating the architecture but not the functionality. Their approach transforms the target model into one extremely sensitive to small weight changes but with equivalent predictions. This is done by extending a hidden layer with nodes without changing the model output. In addition, the suggested approach adds what they refer to as deceptive nodes to decrease the risk that an adversary sees through the protection.

3.7.6 Model distillation

Model distillation is primarily a method for reducing a large-scale model into one more manageable for deployment. However, since the distilled model has never seen the original training set, a theory is that it is less susceptible to inference attacks. The distilled model also masks the architecture of the original target model. In their work, Wang et al. [38] suggest a model for adversarial robustness distillation without original data, distilling large models into smaller data-free models with improved performance against various adversarial attacks.

3.8 Model similarity metrics

Since the extraction attacks aim to steal a model, metrics for comparing model similarity are essential in evaluating the success of such work. However, what metric to use might vary on whether the adversary's objective is to steal the exact parameters of a model or to steal its behavior. In an evaluation, a combination of metrics might provide valuable insights.

3.8.1 Model behavior similarity

A method for comparing the behavior of two models is to run the same set of test cases through them, comparing how well they perform the task. The metrics introduced below add to metrics already described in Section 2.4 and Section 2.5.

Fidelity

A model's fidelity is calculated in the same way as accuracy, described in Section 2.5, but where the predictions of the targeted model are considered ground truth. This metric shows the prediction consistency of the substitute model. Since it is not dependent on labeled training data, as it uses the target model output for ground truth, it can be calculated on any data or distribution without losing its relevance [5].

Transferability

The metric shows how many adversarial examples are generated for the substitute model, M_s , and are also adversarial for the target model, M_t . Let x be a real data sample, $y = M_t(x) = M_s(x)$ and x^* be an adversarial example for M_s , so that $M_s(x) \neq M_s(x^*)$. Having $M_t(x) \neq M_t(x^*)$ then means that there is transferability between the substitute model and the target model [5].

3.8.2 Model parameter similarity

For parametric models, calculating the similarities of their respective parameter sets is a way of measuring how similar the models are.

Euclidean distance

The Euclidean distance [18] is one of the most common ways of calculating a distance between two points in space or between two vectors. The distance, d , between two parameter sets, $\theta^{(1)}$ and $\theta^{(2)}$ of size N is calculated by

$$d(\theta^{(1)}, \theta^{(2)}) = \sqrt{\sum_{i=1}^N (\theta_i^{(1)} - \theta_i^{(2)})^2}, \quad (3.1)$$

where $d = 0$, if the two vectors are identical.

Even if implied above, in the case of the Euclidean distance, the two vectors don't need to be of the same length, N , for this calculation to be meaningful. The distance between a non-zero point in space and the origin (null) is a distance.

Data normalization is often applied to the vectors when performing calculations since the relative difference between two vectors might be more interesting than the actual difference. When calculating the Euclidean distance, it is natural to use the Euclidean norm, also called the L2 norm, denoted $\|\cdot\|_2$.

In Equation (3.1), by using

$$\theta_{L2}^{(1)} = \frac{\theta^{(1)}}{\sqrt{\sum_{i=1}^N (\theta_i^{(1)})^2}}, \quad \theta_{L2}^{(2)} = \frac{\theta^{(2)}}{\sqrt{\sum_{i=1}^N (\theta_i^{(2)})^2}}, \quad (3.2)$$

instead of $\theta^{(1)}$ and $\theta^{(2)}$, the calculated distance would be normalized by the L2 norm.

Cosine similarity

The angle between them can also measure the similarity between two vectors. The definition of the Euclidean dot product [18] provides the derivation for this metric. The cosine similarity, S_C , between two parameter sets, $\theta^{(1)}$ and $\theta^{(2)}$ of size n , is defined by

$$S_C(\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}) \equiv \cos(\phi) = \frac{\boldsymbol{\theta}^{(1)} \cdot \boldsymbol{\theta}^{(2)}}{\|\boldsymbol{\theta}^{(1)}\|_2 \|\boldsymbol{\theta}^{(2)}\|_2} = \frac{\sum_{i=1}^n \theta_i^{(1)} \theta_i^{(2)}}{\sqrt{\sum_{i=1}^n (\theta_i^{(1)})^2} \cdot \sqrt{\sum_{i=1}^n (\theta_i^{(2)})^2}} \quad (3.3)$$

where $\|\cdot\|_2$ is the L2 norm. $S_C = 1$ for identical, while $S_C = 0$ for orthogonal (dissimilar) vectors.

For the dot product to be defined, the parameter vectors must be the same length, and all included elements must be non-zero. Therefore, the use case for the cosine similarity metric is more limited than the Euclidean distance.

3.8.3 Rank similarity

In their work on assessing the downstream performance of pre-trained self-supervised representations by their rank, Garrido et al. [14] introduces RankMe, a method for approximating the rank of a model in a computationally-friendly way. RankMe is added to improve the performance of *self-supervised* learning in their use case. This approach could also be used for comparing two models coarsely based on their rank.

RankMe builds on previous work by Roy and Vetterli [39], where the smooth rank measure in Equation (3.4) originally was introduced. It is, in turn, an extension of the equation for calculating *entropy* [4].

Denoting σ_k as the k th singular value of the $N \times M$ matrix \mathbf{Z} , the rank of \mathbf{Z} can be approximated

$$\text{RankMe}(\mathbf{Z}) = R(\mathbf{Z}) \equiv \exp\left(-\sum_{k=1}^{\min(N,M)} p_k \log p_k\right), \quad (3.4)$$

$$\text{where } p_k = \frac{\sigma_k(\mathbf{Z})}{\|\sigma(\mathbf{Z})\|_1} + \epsilon, \text{ and}$$

ϵ is a small constant, typically in the magnitude of 10^{-7} , N is the number of rows of \mathbf{Z} , and M is the number of columns of \mathbf{Z} .

The singular values, σ_k , of \mathbf{Z} are the elements of the diagonal matrix $\boldsymbol{\Sigma}$, calculated by performing a Singular Value Decomposition such that $\mathbf{Z} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ [1].

For large matrices, sampling also can provide a good approximation of its rank [14].

3.9 Dataset similarity metrics

During an extraction attack, an adversary would collect data from the target model and store it in a substitute dataset. A substitute model can then be trained on this data. Hence, the success of an attack can also be measured by comparing similarities between a target dataset and a substitute dataset.

3.9.1 Basic metrics

For dataset analysis, some basic statistical metrics will be useful. The (arithmetic) mean for a set of values, \mathbf{x} , is defined as:

$$\text{mean}(\mathbf{x}) = \mu(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N x_i, \quad (3.5)$$

where $i = 1, \dots, N$. The standard deviation for the same set of values \mathbf{x} are defined as:

$$\sigma(\mathbf{x}) = \sqrt{\sum_{i=1}^N (x_i - \mu(\mathbf{x}))^2}. \quad (3.6)$$

3.9.2 Optimal Transport

Optimal transportation (OT) problems have a long history, and their origin is attributed to the work of French mathematician Gaspard Monge (1746–1818). Monge solved an engineering problem, moving stones from one place to another [40]. In 1942, Leonid Kantorovich (1912–1986) introduced the notion of applying OT to probability distributions rather than piles of stone [41]. This has been shown to have many modern-day applications within statistics, computer science, and applied mathematics [42, 43].

The Wasserstein distance, or the Kantorovich-Rubinstein metric, measures dissimilarity between two probability distributions. It describes the minimum cost of transforming one distribution into the other. Formally, the Wasserstein distance between two probability distributions A and B on a metric space X is defined as

$$Wd(A, B) = \left(\inf_{\gamma \in \Gamma(A, B)} \int_{X \times X} c(x, y) d\gamma(x, y) \right)^{1/p}$$

where $Wd(A, B)$ is the Wasserstein distance, γ represents the set of all joint distributions on $X \times X$ with marginals A and B , and $c(x, y)$ is the cost of moving a unit of mass from x to y . The optimization problem seeks to find the joint distribution γ that minimizes the total cost of moving mass from A to B . When $p = 1$, it is called the Wasserstein-1 distance, the Earth movers distance.

3.9.3 Fréchet Inception Distance

The Fréchet Inception Distance (FID), introduced by Heusel et al. [44], is a commonly used metric for generative image models. However, for the example of images, the comparison is not made pixel for pixel. Instead, it compares the mean and standard deviation for a feature space defined by a function f . The function f is chosen depending on the task. A common, and the default choice in Pytorch, for f is an Inception v3 model trained on the ImageNet [44].

Given two datasets, \mathcal{D}_S and $\mathcal{D}_{S'}$, f maps the data to two new datasets $f(\mathcal{D}_S)$ and $f(\mathcal{D}_{S'})$. Gaussian distributions, $\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$ and $\mathcal{N}(\boldsymbol{\mu}', \mathbf{C}')$ are fitted to the new datasets. The metric is then calculated as

$$\text{FID} = \|\boldsymbol{\mu} - \boldsymbol{\mu}'\|_2^2 + \text{tr}(\mathbf{C} + \mathbf{C}' - 2(\mathbf{C}\mathbf{C}')^{1/2}) \quad (3.7)$$

where $\|\cdot\|_2$ is the L2 norm and tr is an operation that sums the diagonal elements of a matrix [1].



4 Related work

As research into MEAs has evolved, work has also gone into methods for countering these attacks. This chapter describes related work for monitor-based approaches and adjacent work in the broader area of model inference attacks.

4.1 Monitor approaches

Kesarwani et al. [26] proposes a design for an extraction monitor that could warn about model extraction, both from individual users and groups of users. Focusing on decision trees, they propose metrics to track an adversary's learning rate. The first of the two metrics focuses on the input-output pairs of individual users, incrementally learning a local decision tree. An entropy-based metric then uses a validation set to measure adversarial information gain leading up to a threshold. The second metric uses the historical queries of all users, assessing the coverage of the possible combined knowledge of the users. The experimental work shows that the metrics can provide approximate information about the extracted knowledge. The approach has similarities with this work in that it infers an approximation from the input-output pair and utilizes entropy calculations. However, while their work is limited to decision trees, the work of this thesis expands into other ML models.

Pal et al. [27] focuses on monitoring the distribution of queries put to a model. Closely related to an anomaly detection approach, a modified variational encoder is trained to separate three types of attacker samples from benign. Experimental work shows that the accuracy of the extracted models, the substitute models, is reduced. Their work aims at a similar problem as this work; however, they address it in the input phase of querying.

Zhang et al. [29] propose a detector for model extraction attacks, where the adversary iteratively expands the set of samples in the substitute dataset near a decision boundary in the target model. This is done by using a similarity encoder trained by adversarial training. The detector closes the accounts of users when making queries that indicate an extraction attack is in progress. Experimental work shows an increased cost for the adversary in extracting models while maintaining a working system for benign users. Their work is also aimed at analyzing model inputs while addressing the same issue as this work.

Liu et al. [28] uses a method for amplifying the query distribution discrepancy and making distinguishing malicious from benign users easier. While queries are normally analyzed one at a time, they propose a semantic feature-based detection method harnessing context in-

formation not usually used for monitor-based approaches. Their work also aims to increase efficiency and decrease monitoring latency, proposing a heterogeneous framework for defending against attacks. Experimental work shows increased accuracy in detecting attacks compared to the baseline while reducing response latency by up to 54 percent.

Dziedzic et al. [30] impedes attacks by exploring an approach for increasing the cost of an adversary. Before accessing model predictions, users must complete a proof-of-work puzzle where the difficulty level is tied to an estimate of information leakage for that user. This is claimed to greatly increase the computational effort of an attacker, making the target model less appetizing while only introducing a small overhead for a benign user. Their approach takes a more active role in protecting a model from theft, while this work takes a more passive stance.

Sadeghzadeh et al. [31] introduces the notion of a hardness degree of samples founded on learning difficulty. The expectation is that predictions of labels of easy samples should converge during early training epochs while the same process for hard samples converges during later epochs. In the proposed hardness-oriented detection approach, the hardness degree for each query is calculated and added to the user's hardness histogram. The distance between the user's histogram and a reference histogram is calculated when the user reaches a set number of queries. The user is flagged as a possible adversary if the distance surpasses a threshold. Their approach turns the attention more to the behavior of models than the model inputs. This, as an implementation of a threshold idea, is a shared similarity with this work.

4.2 Model inference attacks

Taking one step back, it should be noted that the topic of MEAs is part of a broader field called Model Inference attacks. The threat of other neighboring inference attacks could also be important to assess to keep a model safe.

Liu et al. [23] explores the whole field of inference attacks, presenting a holistic risk assessment model. They find that the complexity of the training datasets plays a vital role in aiding the performance of an inference attack. However, the opposite is shown for attacks aimed at model extraction and membership inference.

Dibbo [45] provides a literature review of relevant research for model inversion attacks, aiming to infer or reconstruct the target dataset. The work describes attack methods and possible defenses. It also briefly discusses preventive measures. The review concludes that generalized attack techniques are yet to be investigated and that developing agnostic defenses is an important future task.

Shokri et al. [46] provides an overview of membership inference attacks, where the attacker seeks to infer whether a specific target sample, such as a person's medical records, has been included in the model training dataset. Their key advance for aiding an attack is a shadow training method that trains an attack model to distinguish the target model output on members versus non-members of the training set. They show how this is successful, targeting two major cloud service providers.



5 Method

Having been introduced to the problem of model extraction attacks (MEAs) and the threat they pose, there is an interest in preventing and defending against them and evaluating their success. In this chapter, the method for the work of this thesis is described.

5.1 Literature review

A literature review has been conducted on machine learning and the specific area of model extraction attacks. The starting point for this work has been two recent surveys of knowledge into extraction attacks by Oliynyk et al. [5] and Genç et al. [6]. These were chosen using convenience selection and based on a Google search for 'model extraction attack.'. Following the snowball principle, references for relevant topics within these surveys were backtracked to add depth and broaden the theory introduced in this work. Articles used for this work were filtered, firstly based on their abstracts and secondly on their entire content.

Initial research used Google for metrics on model and dataset similarity and monitoring approaches, using phrases 'machine learning model similarity metrics' and 'dataset similarity metrics.'. After identifying relevant topics, detailed searches were made using both Google and databases available through Linköping University.

Based on theory on MEA, a foundation of knowledge about ML and performance metrics was introduced. Two main sources on this topic were books. One, by Lindholm et al. [4], affiliated with Linköping University by one of its authors. The other, Goodfellow et al. [17], is often referenced in other works on machine learning. Research material was also used to add depth, especially in describing the model architectures used for the experimental work. For a search on these terms and concepts, a primary search was made using Google, followed by a database search for the referenced articles through Linköping University.

5.2 Proposal

Based on the material presented in theory, a design for an approximation-based monitor was proposed. It uses the interaction between an adversary and a targeted model to approximate how far along the attack has proceeded. Based on the results of the experimental work in scenario 1, the proposal was amended.

5.3 Experimental evaluation

Experimental work has been conducted for a set of MEA scenarios to evaluate the proposed approximation-based monitor. For one scenario, the targeted model was a basic parametric regression model. The targeted model was a neural network classification model in the other scenario. Attacks against the models have been simulated, tracking the progress of a monitor model. The design of each scenario follows the same steps:

- An experimental scenario is defined, describing the setup and purpose of the experiment. It also touches on what is to be evaluated during the experiment.
- The targeted model is described, along with the (private) target dataset used for training the model. The origin and the characteristics of the dataset are described.
- A simulated attack is described, motivating how the targeted model is queried and what data is being used for this purpose.
- The design of the model and dataset for the monitor is described. The choice of model(s) is motivated.
- For the monitors, metrics implemented are motivated and described.

Afterward, the results of the simulated attack are presented, discussed, and used for conclusions.

Based on the results from scenario 1, more architectures are evaluated for a monitor model in scenario 2. An adversary's actions are unknown. Evaluating more possible models as a basis for a monitor, using the one best at stealing the targeted model, is a systematic way of handling uncertainty. From a protection point-of-view, the worst-case monitor model increases the chance of a well-performing protection.



6 Design: An approximation based monitor

This chapter proposes an approximation-based monitor approach for tracking an ongoing MEA in the context of a query-based attack. In Chapter 7, the monitor approach is implemented for two experimental scenarios to evaluate its usefulness.

6.1 The proposal

Assume a target model, M_t , is trained on a (private) target dataset, \mathcal{D}_t , of sensitive information, as illustrated in Figure 6.1. From the owner's point of view, there is an interest in protecting the model and the underlying data against theft. Conversely, there is also an interest in making M_t available for users. It might provide an important in-house service in an organization or be a service sold to external clients.

Continuing to build on Figure 6.1, also assume that an adversary wants to steal the target model and make a copy, called a substitute model M_s . Access to M_t is restricted to an API, making it a black-box attack. By querying the API, input-output pairs of data are collected, which could be used for training M_s and stored for future use. The theory of attacks is described in Chapter 3. For this work, however, as M_t is a black box for the adversary, everything outside the API is unknown to the owner of M_t .

The proposed approach uses the query interactions between a potential adversary and the API to assess the threat. The added monitoring components are marked in green in the Figure 6.1. The input-output pairs are fed to the monitor dataset, \mathcal{D}_m , and used for training a monitor model, M_m . It is then used as a proxy, approximating how close M_s is to copying M_t . When similarities reach a set threshold, actions could be taken to protect M_t .

6.2 Design choices

The suggested approach mimics actions that an adversary could take. The monitor, M_m , tries to make a copy of M_t , working as a proxy for the unknown substitute model, M_s . By applying some metric to compare M_t and M_m , the idea is to get an approximation of how an attack is progressing based on the assumption:

$$\text{metric}(M_m) \sim \text{metric}(M_s) \tag{6.1}$$

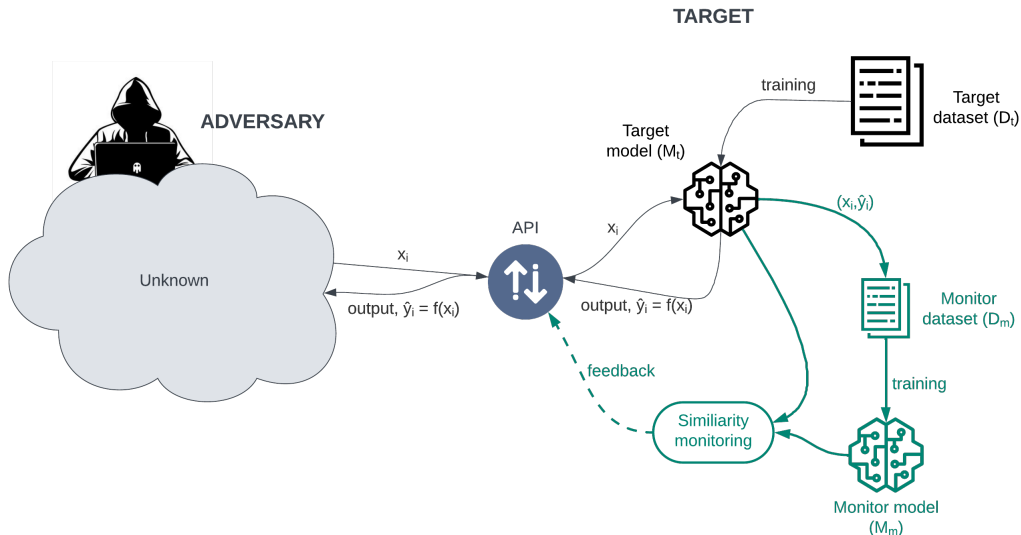


Figure 6.1: Flow of information between an adversary and a targeted model, with the proposed monitoring approach included.

However, this concept is not plug-and-play. Just as the adversary must make design choices for an attack depending on the target model, its purpose, and the API input-output constraints, a defender must also make choices for the design of M_m and what metric to use for tracking similarity. These choices can be heavily domain-specific. Step by step, the components of this approach will be discussed in detail:

- Design choices of the monitor dataset.
- Choice of model architecture and hyperparameters.
- Choices for making a similarity, or threat, monitor.
- Protective and defensive actions.

Since M_m has white-box knowledge of M_t it should be possible to perform equal or better than any substitute model based on black-box knowledge.

6.2.1 Monitor dataset

When a user puts a query, x_i , to the API, M_t answers the call by making a prediction, $\hat{y}_i = M_t(x_i)$, and returning it to the user through the API. A copy of the input-output pairs (x_i, \hat{y}_i) , $i = 1, \dots, n$ are stored in D_m for future use. Saving this data is not always necessary. A parametric monitor model could discard the training data when the training is complete. However, saving the input-output pairs could serve other purposes, such as analyzing the data and gaining insights into the purpose of an attack.

The design of D_m is not necessarily dependent on the target dataset, D_t . The input-output pair might differ greatly from the training data used for M_t . Yet, the pair contains a mapping that condenses the knowledge stored in the target dataset. When choosing the monitor dataset's design, white-box knowledge has an advantage, making it possible to avoid mistakes made by incorrect assumptions.

For example, if the input-output-pair is two floating point values, the monitor dataset should completely capture the relationship mapped in that pair. Truncating or rounding values could make M_m drift and provide a poor approximation of M_t .

Two factors that should influence the dataset design:

- The dataset design must capture the input-output pair completely.
- Since the aim is to train a monitor model, the design choices of the dataset should, if possible, facilitate the use of the data.

6.2.2 Monitor model

The aim of M_m is to approximate M_s , tracking the progress of an attack. The input-output pairs, (x_i, \hat{y}_i) , $i = 1, \dots, n$, are used for continuously training M_m .

While an adversary is limited by black-box knowledge about the target model, M_m can be designed based on white-box knowledge. This might not completely remove the need to make assumptions. A threat modeling could be performed to influence design choices.

Analogous to the discussion on choices for the \mathcal{D}_m design, the choice of model and its properties are not necessarily dependent on M_t . The input-output pair might be very dissimilar to the data used for training M_t . Therefore, M_m design choices should be made based on what would perform best for the task, trained on the actual input-output pairs. To address this systematically, one should test many possible choices for M_m , choosing the best model to steal M_t .

While M_m is trained on input-output pairs, its similarity to the M_t will also be tracked continuously. Design choices could be made to ease this work and increase monitoring efficiency.

Three factors that should influence the model design:

- The model must capture the mapping provided by the input-output pairs.
- Testing several possible models, choosing the best model to steal M_t .
- Computational cost in training and calculating model similarity to the target model.

For example, if M_t is based on a polynomial regression model, the output might be two floating point values. An M_m could be designed as a polynomial or neural network model. Choosing a polynomial model would, in general, be computationally cheaper and could ease a similarity comparison with the target model.

6.2.3 Similarity assessment

The key to the proposed monitoring approach is a similarity assessment, using some metric to track the similarity between M_t and $M_m \sim M_s$. Tracking the progress of M_m functions as a proxy for the unknown M_s . Some actions could be taken to protect and defend the target model when reaching a set threshold, as illustrated in Figure 6.2.

Choosing a suitable metric poses a series of challenges. First, it might be data-specific, adapting to models based on images, text, or numerical values. Second, it might be model-specific, adapting to a wide range of ML models. Third, calculation efficiency might be a priority if monitoring is to be continuous. In a live scenario, there might not be any room for extensive tests each time a query has been put into the model.

When exploring model similarity metrics, in Section 3.8, three main approaches were introduced: behavior similarity, parameter similarity, and rank similarity:

- Metrics on behavior similarity, such as accuracy, described in Section 2.5, or transferability, described in Section 3.8.1, are good for whether the monitor and target model behave similarly on test data. On large test datasets, calculating accuracy could be inefficient. However, a subset of the test dataset could be used if needed.

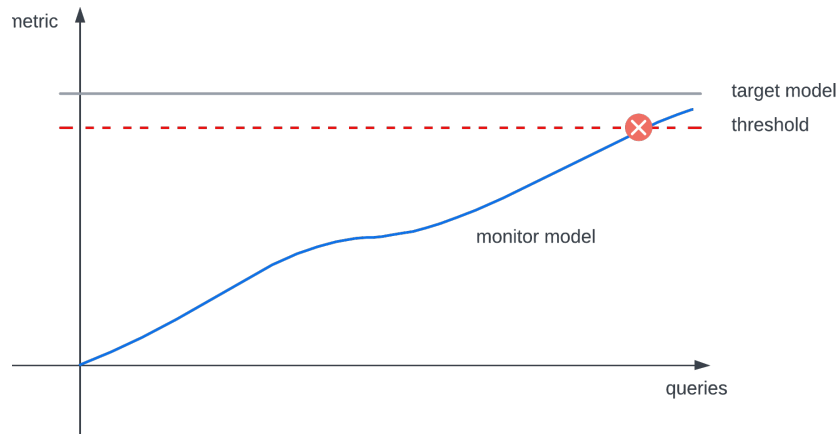


Figure 6.2: Conceptual illustration of the desired behavior of the similarity assessment, taking action when the monitor model reaches a threshold.

- As metrics on parameter similarity, the Euclidean distance and cosine similarity, described in Section 3.8.2, are efficient methods for comparing models with similar architecture. These could be used in some implementations for similarity assessment but are not versatile for all cases. For instance, it would not be implementable if the models have a very different parameter set or are non-parametric.
- Rank similarity, described by Equation (3.4), is the most versatile metric when comparing model similarity. It is potentially more computationally costly than calculating parameter similarity but should be much cheaper than calculating behavior similarity.

All three approaches will be explored in experimental work to compare the similarity between the target and the monitor models.

For this work, setting up a threshold for each scenario is omitted in the experimental work. When a monitor is implemented and tested, the choice of threshold is specific to the M_t it aims to protect.

6.2.4 Defensive actions

When M_m has passed a set threshold, under the assumption that $M_m \sim M_s$, actions could be taken to protect M_t from further attacks. Defensive measures, such as cut-off and output perturbation, are explored in research and described in Section 3.7.



7 Experimental design

To explore the approach for an approximation-based monitor, proposed in Chapter 6, a set of experimental scenarios will be played out. This chapter describes the purpose and design of each scenario. The results are presented in Chapter 8.

7.1 Scenario 1 setup: Mechanical system

A mechanical system is assumed as the basis for an ML model. The system itself is not important, but it has the behavior of a sine function. The target model is denoted M_t and is available through an API. To protect M_t against query-based theft of model behavior, a monitor as proposed in Chapter 6 is to be implemented.

To an adversary, stealing the model behavior could make sense. If each query to the API is associated with a cost, making a substitute model could save money. Having a copy could also give important insights into the mechanical system and reduce dependency on another entity.

A potential adversary has black-box knowledge about M_t . However, the opposite is also true. There is no way of knowing what is used for a substitute model and, consequently, no obvious architecture choice for a monitor model, M_m . More than one architecture will be tested, and the worst-case scenario, the one most successful at copying the behavior of M_t , will be chosen as the basis for M_m .

7.1.1 Purpose and evaluation

This experiment will measure that the approximation-based monitor approach suggested in Chapter 6 can be useful for a simple regression scenario. It also explores how different monitor model architectures affect the usefulness of monitors.

To mimic an attack, the M_t will be queried using random numbers $[0, 2\pi]$. For practical reasons, the number of possible queries is limited to 100. The resulting input-output pairs are stored in a monitor dataset, denoted \mathcal{D}_m , and used for training M_m . For this scenario, two different monitor models, M_m , will be used for this experiment.

To evaluate the behavior similarity between M_t and M_m , mean square error (MSE), described by Equation (2.9), is used as the main evaluation metric. It is calculated for prediction

consistency, using the target test dataset for ground truth. Applicable model and dataset similarity metrics, presented in Sections 3.8 and 3.9, are also evaluated.

The gathered results will be averaged over 20,000 experimental runs to mitigate the randomness involved in this scenario. A visual inspection will be conducted for the resulting datasets from two random experiments, showing that the outcome of one experiment might differ substantially from the average over 20,000 experiments.

7.1.2 Wasserstein distance setup

Work made by [47] will be implemented to calculate the Wasserstein distance-1, also called Earth movers distance, for both monitors. This metric was introduced in Section 3.9.2. For this implementation, the cost matrix, c , is the Euclidean distance between all points in \mathcal{D}_t and \mathcal{D}_s respectively.

7.1.3 Target dataset

The mechanical system has been measured to create a dataset for training a simulation model. Some systematic measurement error is added to the collected values, as illustrated in Figure 7.1. The target dataset, denoted \mathcal{D}_t , contains 50 samples from the interval $[0, 2\pi]$. An 80/20 split is used for target training and testing datasets, denoted \mathcal{D}_{train} and \mathcal{D}_{test} .

Since the system has the behavior of a sine function, it is assumed to repeat itself in cycles of 2π . The added noise results in a small discrepancy between the mechanical system and \mathcal{D}_t but is attributed to systematic measurement drift and not the system itself.

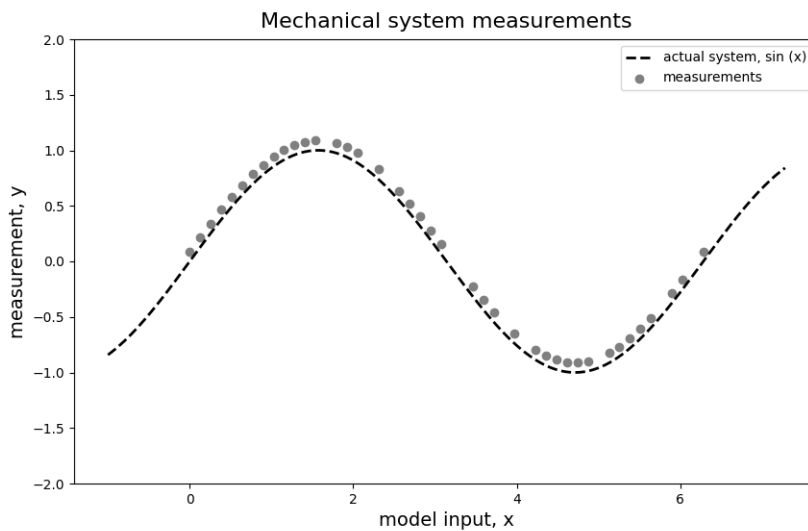


Figure 7.1: The behavior of the mechanical system in scenario 1, a sine function. Measurements affected by a systematic error are made in the interval $[0, 2\pi]$.

7.1.4 Target model

For this scenario, the target model, M_t , is designed specifically for this experiment. The system to be modeled has a non-linear behavior. Hence, a model that can handle non-linear problems is needed. A Ridge regression model is chosen, described in Section 2.7.3. It can capture the behavior of the mechanical system while being computationally effective, and it mitigates potential issues with overfitting.

When using Ridge regression, two design choices have to be made. The first is the choice of degree to use for the underlying polynomial regression, from Equation (2.14). The second is the choice of the regularization parameter, α , for the penalty term. The optimal parameters for the model are chosen empirically, searching for the minimum loss using MSE, described in Equation (2.9), for a range of degrees and α s.

The result of this test is illustrated in Figure 7.2, where the minimum loss on \mathcal{D}_{test} is approximately $3.61 \cdot 10^{-8}$, occurring at $\alpha = 0.001$ and degree = 11. This will be the hyperparameter choice for the target model in this scenario.

The model shows an expected offset in the y-direction compared to the actual mechanical system, as shown in Figure 7.3. However, it implements a good fit relative to the training data samples.

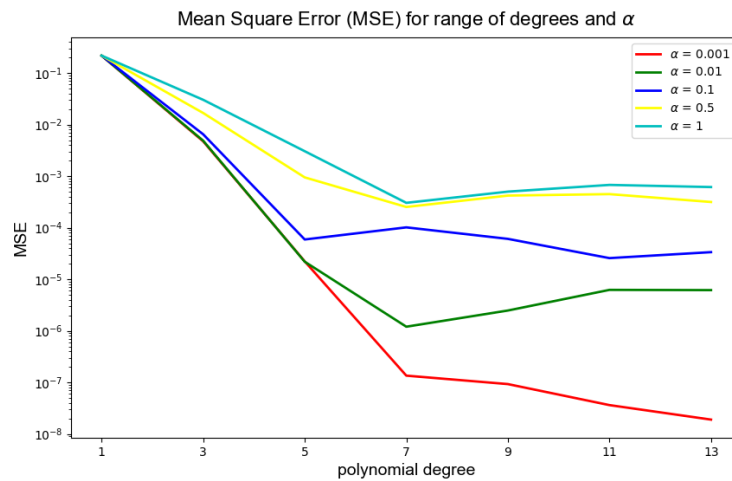


Figure 7.2: Evaluating optimal choices for regularization parameter, α , and polynomial degree for a Ridge regression model. Minimal loss when $\alpha = 0.001$ and degree = 11.

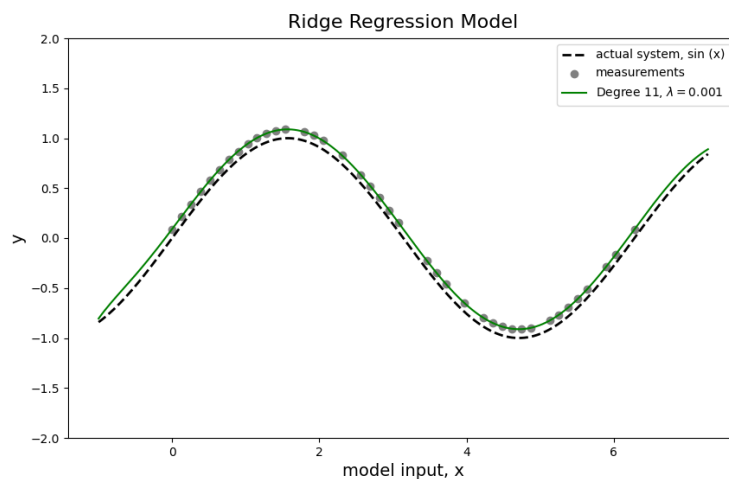


Figure 7.3: The Ridge regression model fitted to the training data samples. The relevant interval is $[0, 2\pi]$.

7.1.5 Monitor: Ridge regression (S1E1)

For the monitor, a Ridge regression using the same hyperparameters as the target model is chosen. This takes advantage of the white-box knowledge of the target model properties, mimicking M_t as closely as possible. Ridge regression performs well on training data and should perform well on the input-output pairs. A concern could be that the model is a slow learner, with parameters shifting greatly for each new data sample before converging toward the target model.

The monitor dataset, \mathcal{D}_m , stores two floating point values for each query. The dataset is empty from the start for this monitor.

7.1.6 Monitor: Gaussian process regression (S1E2)

A Gaussian process regression model (GPR), described in Section 2.8.1, is chosen for the monitor. This approach disregards the white-box knowledge and instead tries to optimize a situation where M_m is trained on one input-output pair at a time. Gaussian processes perform well in uncertainty, and a monitor based on its Bayesian approach should be a quick learner.

Since the black-box knowledge goes both ways, making detailed adaptations of the model properties should add, rather than reduce, risk. Hence, the standard setup in Scikit Learn is chosen for the implementation. It implements a combination of a constant and a radial basis kernel (RBF), such as

$$\text{kernel} = 1.0 \cdot \exp\left(-\frac{(\mathbf{x} - \mathbf{x}')^2}{2 \cdot 1^2}\right) \quad (7.1)$$

with bounds $10^{-5}, 10^5$. RBF has previously also been described in Equation (2.18).

The standard implementation of GPR also uses L-BFGS-B, which runs once. Alpha value, for kernel matrix during fitting, is set to 10^{-10} .

The monitor dataset, \mathcal{D}_m , stores two floating point values for each query. The dataset is empty from the start for this monitor.

7.2 Scenario 2 setup: Image classification

A system for image classification is the basis for a ML model. The system is trained to differentiate between ten categories of objects: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The model could combined with a camera supplying still images to the model, automating detection. To protect M_t against query-based theft of model behavior, an approximation-based monitor, as proposed in Chapter 6, is to be implemented.

Targeting an already-trained image recognition model is tempting for an adversary. Training a well-functioning image classifier requires time and access to labeled image samples. Instead of going through the strenuous process of building its model from scratch, part of this work could be outsourced by attacking the target model.

7.2.1 Scenario 1 takeaways

One of the purposes of the experimental work in scenario 1 was to let the results influence the design of scenario 2. Hence, the key takeaway should be noted early on.

Scenario 1 shows that the proposed method for an approximation-based monitor that trains on an adversary's interactions can be successful for a simple non-linear regression problem. However, since the adversary's design of a substitute model and dataset are unknown, the monitor is designed in a black-box setting. To design a monitor that is as successful as possible, the choice of monitor architecture and properties should be based on the worst-case option from the owner's perspective. Hence, for scenario 2, many architectures will be evaluated as possible M_m choices.

To enable the evaluation of many possible M_m , the compromise is that less attention will be put on the details of model design for this scenario. Standard implementations from the Torchvision model library and the same transform, hyperparameters, optimizer, and validation criterion will be used throughout.

7.2.2 Purpose and evaluation

Based on the experiences of scenario 1, Section 7.1, this experiment aims to show that the approximation-based monitor approach, suggested in Chapter 6, can be used in a more complex setup. It also explores the monitoring effectiveness for standard implementations of neural networks from the Torchvision library.

To mimic an attack, the M_t will be queried using a set of 20,000 images. The resulting input-output pairs are stored in a monitor dataset, denoted \mathcal{D}_m , and used for training M_m . For this scenario, six different M_m will be used for this experiment.

Accuracy, described in Section 2.5, is used as the main evaluation metric to evaluate the behavior similarity between M_t and M_m . It calculates prediction consistency using the target test dataset for ground truth. Applicable model and dataset similarity metrics, presented in Sections 3.8 - 3.9, are also presented.

The presented results will be obtained from one experimental run of each monitor.

7.2.3 Rank similarity setup

For this scenario, a version of the RankMe is implemented, described in Section 3.8.3, approximating the rank complexity based on the output layer. These calculations are performed on an output layer matrix. It is a 20 x 10 matrix constructed by letting a fixed set of 20 images pass through the model, collecting the 10-value vector from the output layer. Using 20 images is a choice that makes sure the output matrix is over-determined.

Any set of images that the model could process could be used. For this work, the set of images is a subset of the CINIC-10 dataset, organized in the categories listed above by Darlow et al. [48]. Two images from each category, only originating from the ImageNet dataset, by Deng et al. [49], are randomly chosen.

The algorithm for calculating the rank score is implemented as follows:

```
import torch
import numpy as np
import torchvision.transforms as transforms

def get_outputmatrix(model):
    # standard cast into tensor, normalization in [-1, 1]
    transform = transforms.Compose([transforms.ToTensor(),
                                    transforms.Normalize((0.5, 0.5, 0.5),
                                                         (0.5, 0.5, 0.5))])

    # File paths to 20 image files used for the calculation
    files = [PATH1, ..., PATH20]

    outputmatrix = np.zeros((len(files), 10))

    for i, file in enumerate(files):
        image = Image.open(file)
        x = transform(image)
        x.unsqueeze_(0)
        _, vector = predict_one(model, x)
```

```

        outputmatrix[i,:] = vector

    return outputmatrix

def get_rank(outputmatrix):
    epsilon = 0.0000001 # Based on article

    # Perform SVD decomposition
    _, Sigma, _ = np.linalg.svd(outputmatrix,
                                full_matrices=False)

    l1_norm_sigma = np.linalg.norm(Sigma, ord=1)
    p = np.zeros(10)
    sum = 0

    for k in range(Sigma.size):
        p[k] = (Sigma[k] / l1_norm_sigma) + epsilon
        sum += p[k] * np.log(p[k])

    rankscore = np.exp(-sum)

    return rankscore

```

7.2.4 Target dataset

A subset of the CIFAR-10 dataset, by Krizhevsky [50], is used for the target dataset, denoted \mathcal{D}_t . The complete dataset contains 60,000 color images in 32 x 32 pixels. They are categorized into ten categories: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each category contains 6,000 images. Some random example images are presented in Figure 7.4

For the test dataset, denoted \mathcal{D}_{test} , 1,000 images from each category are randomly selected and held back from training. From the remaining images in each category, 3,000 are randomly selected for the target training dataset, denoted \mathcal{D}_{train} . In total, \mathcal{D}_{train} contains 30,000 images. \mathcal{D}_{train} and \mathcal{D}_{test} are distributed as presented in Table 8.4.

The remaining 20,000 unused images, evenly distributed over the ten categories, are used for querying M_t API.

7.2.5 Target model

Considering the image classification task, a convolutional neural network is a natural choice for M_t . For this scenario, the focus shifts to using standard and already-proven architectures instead of custom-built models.

The choice of model is the ResNet-18 model from the model sub-package of Torchvision. It is a ResNet model, previously described in Section 2.9.7.

The choices of data transform, hyperparameters, optimizer, and validation criterion do not result from extensive model testing. These are somewhat arbitrary choices based on experience, early experimental log files, examples made by others, and a compilation by Singh Ganger [51]. The model starts with random weights and is not pre-trained.

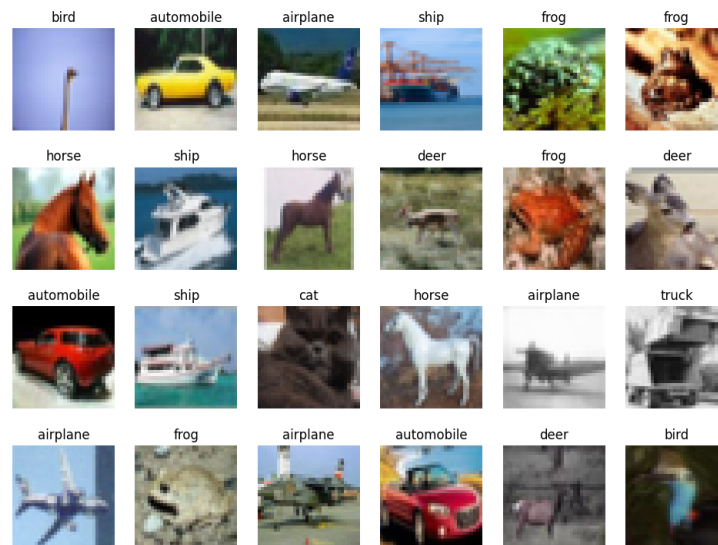


Figure 7.4: Random examples of images from categories in the CIFAR-10 dataset.

The target model is implemented as follows¹:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torchvision.models import resnet18

# The ResNet-18 model, using torchvision.models.resnet18
class Resnet18(nn.Module):
    def __init__(self, num_classes=10):
        super(ResNet18, self).__init__()
        self.model = resnet18(pretrained=False,
                               num_classes=num_classes)

    def forward(self, x):
        return self.model(x)

# Data transformation used
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

n_epochs = 50
```

¹<https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html>

```

batch_size = 64

target_model = ResNet18()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(target_model.parameters(),
                       lr=0.01, momentum=0.9, weight_decay=1e-4)

```

The model is trained and validated on the target training dataset. Using the set hyperparameters, the model performs the best in epoch 48 with a prediction accuracy, described in Section 2.5, of 77.23 % on \mathcal{D}_{test} .

The rank score for M_t was calculated to be 7.364.

7.2.6 Monitor dataset

To mimic an attack, the M_t is queried using the 20,000 unused images from the CIFAR-10 dataset. This set of images is denoted \mathcal{D}_{query} . They are evenly distributed over the same ten categories as the target dataset. Before their use, the order is shuffled.

The input-output pairs of images and labels from querying M_t API constitute the monitor dataset, denoted \mathcal{D}_m , and used for training M_m . Each M_m is trained on queries in the same order. However, when labeled by M_t , the distribution of the images changes as M_t accuracy for this task is 79.3 percent. This redistribution is presented later in results, Table 8.4.

7.2.7 Monitor models

Six monitor models stemming from three families of convolutional neural networks are evaluated. All are implemented using the default, not pre-trained version from the Torchvision model library. For practical reasons, all are implemented using the same transformation, criterion, optimizer, and hyperparameters as M_t .

Two monitors are residual network models, previously described in Section 2.9.7. One is ResNet-18 (S2E1), which shares its architecture with M_t , taking advantage of white-box knowledge. The other is ResNet-152 (S2E2). The main difference is the number of layers, where one uses 18 and the other 152.

Two monitors are based on the very deep convolutional neural network architecture, introduced in Section 2.9.6. One is VGG-11 (S2E3), using eight convolutional layers. The other is the deeper VGG-19 (S2E4), using 16 convolutional layers.

Finally, two monitors are DenseNets, a family of densely connected convolutional network models, introduced in Section 2.9.8; DenseNet-161 (S2E5) and DenseNet-121 (S2E6).

The implementation for this ensemble of models is based on the same code, where only a few lines of code are changed. The prerequisite is that the torch and the torchvision packages are installed. Using the code below as a template, the term USED_MODEL is changed below for the name of the imported model: resnet18, resnet152, vgg11, vgg19, densenet161, or densenet121.

```

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torchvision.models import resnet18, resnet152, vgg11,
                             vgg19, densenet161, densenet121

class NeuralNetwork(nn.Module):
    def __init__(self, num_classes=10):
        super(NeuralNetwork, self).__init__()
        self.model = USED_MODEL(pretrained=False,

```

```

num_classes=num_classes)

def forward(self, x):
    return self.model(x)

# Data transformation used
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

n_epochs = 50
batch_size = 64

monitor_model = NeuralNetwork()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(monitor_model.parameters(),
                       lr=0.01, momentum=0.9, weight_decay=1e-4)

```

All monitor models are trained queries from the same \mathcal{D}_m , described in Section 7.2.6, and in the same order. \mathcal{D}_m mimics intercepted input-output pairs from M_t . Each model can train on an additional 1,000 queries at a time, up to 20,000 queries.

Following the worst-case paradigm, each model is trained from scratch whenever it accesses more queries. This choice is less efficient than adding training incrementally to a model. However, preliminary experiments show that incremental training gives a poorer accuracy under the conditions given by this scenario.

7.3 Experimental environment

An environment has been built for the purpose of running the experiments. It could be described as a simple game engine that keeps track of occurring events. The work progress can be observed and presented as feedback on-screen and logged. When the scenario is concluded, the resulting models and datasets are saved to provide material for the report.



8 Results

In this chapter, the results from the experimental work are presented. The presentation starts with a short introduction to each scenario, designed in Chapter 7. The results presented in this chapter are later discussed in Chapter 9.

8.1 Scenario 1 results: Mechanical system

To recapitulate, the target model, M_t , mimics a mechanical system model with the behavior of a sine function with some systematic measurement error. M_t is a Ridge regression model, with degree 11 and $\alpha = 0.001$, trained on the target dataset, \mathcal{D}_{train} . Mean square error (MSE) on \mathcal{D}_{test} is $3.61 \cdot 10^{-8}$.

Two monitor models, M_m , are trained on input-output pairs that result from querying M_t with 0-100 random floating point values in the interval $[0, 2\pi]$:

- S1E1 M_m , based on a Ridge regression model with the same properties used for M_t .
- S1E2 M_m , using a Gaussian process model with default properties.

Queries generated by an adversary using M_t are stored in a monitor dataset, \mathcal{D}_m .

Unless otherwise stated, all results have been obtained by averaging 20,000 experimental runs of each monitor to mitigate the randomness involved in the experiments. For practical reasons, each experiment's maximum number of possible queries has been set to 100.

The monitors are evaluated on behavior similarity, using prediction consistency by calculating MSE loss on the target test data, \mathcal{D}_{test} .

Numerical data from the experiments are available in Appendix A.

8.1.1 Aggregated results

Aggregated results for the two evaluated monitor models are presented in Figure 8.1 and Table 8.1. Based on MSE loss for \mathcal{D}_{test} , prediction consistency is plotted for each experiment alongside the mean for each M_m . The M_t is included as a reference, at which point the behavior of the target model can be considered stolen.

As shown by Figure 8.1, S1E2 M_m outperforms S1E1 M_m , learning quicker and achieving a prediction consistency visually indistinguishable from M_t , included as a reference.

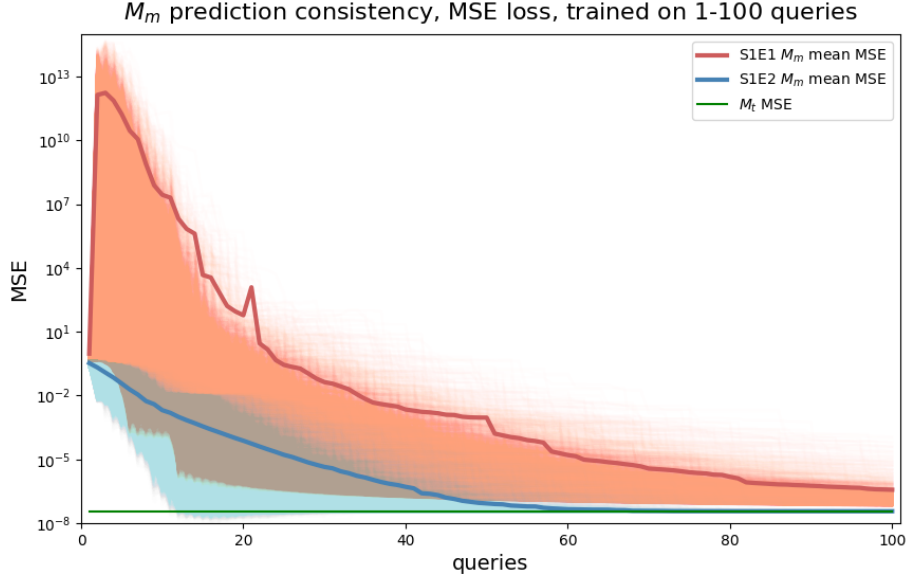


Figure 8.1: Prediction consistency for the two M_m , trained on 1-100 queries from the M_t API. MSE loss on \mathcal{D}_{test} is used as a metric for comparison. All experiments are plotted alongside the mean as a dashed curve for each M_m . S1E2 M_m learns quicker and achieves MSE almost indistinguishable from M_t , which is included as a reference.

For the S1E1 M_m , model parameter similarity to M_t is calculated using normalized Euclidean distance and cosine similarity and presented in Table 8.1. For a larger setup, and depending on the architecture, it could be computationally beneficial to calculate the similarity between model parameter vectors instead of MSE loss on a test dataset.

The complete similarity between the two vectors is equivalent to a 0 distance for the (normalized) Euclidean distance. For the cosine similarity metric, the complete similarity is equivalent to $\cos(0) = 1$, while complete orthogonality means $\cos(1) = 0$. Shown in Table 8.1, both metrics show that S1E1 M_m is similar to M_t and show promise for use as a similarity metric in this setup.

Since S1E2 M_m is non-parametric, these metrics do not apply.

Table 8.1: Scenario 1 M_m prediction consistency and parametric similarity to M_t after 100 queries and averaged over 20,000 experimental runs. The target model was added for reference.

Monitors	Prediction consistency		Parametric similarity to M_t	
	MSE	Euclid	Euclid	Cosine sim.
S1E1 M_m RR	$3.71 \cdot 10^{-7}$	$6.15 \cdot 10^{-2}$		0.997
S1E2 M_m GPR	$3.64 \cdot 10^{-8}$	n/a		n/a
M_t , target model	$3.61 \cdot 10^{-8}$	0.000		1.000

For the monitor results to be valid for comparison, the respective \mathcal{D}_m used should, on average, be similar. If not, that could in itself explain different outcomes. Since each \mathcal{D}_m is the result of randomness, metrics for the two datasets are also presented in Table 8.2 and Figure 8.2 to substantiate that the two monitors have been trained on equivalent datasets.

Regardless of using basic statistical metrics or calculating the Wasserstein-1 distance, presented in Section 3.9.2, the input-output pairs in the two \mathcal{D}_m must be considered equivalent on

average. This is unsurprising since their mean, the standard deviation, and the Wasserstein-1 distance presented in Table 8.2 are the input-output pairs averages over 20,000 experiments.

By visual inspection of Figure 8.2, it is impossible to tell the plotted Wasserstein-1 distances apart. Turning to the numerical results, available in Table A.3, it can be confirmed that they are also numerically indistinguishable in relation to \mathcal{D}_t . Identical datasets have a zero distance, while completely dissimilar datasets have a distance of 1.

It should be noted that the dataset metrics for only one experimental run could differ significantly from results averaged over 20,000 experiments. This is addressed in Section 8.1.4.

Table 8.2: Scenario 1 \mathcal{D}_m metrics after 100 queries and averaged over 20,000 experimental runs. The \mathcal{D}_t metrics are added for reference. The respective \mathcal{D}_m are equivalent and somewhat mirror the properties of \mathcal{D}_t .

Datasets	Statistical metrics		Similarity to \mathcal{D}_t
	$\mu(x,y)$	$\sigma(x,y)$	Wasserstein-1 distance
S1E1 \mathcal{D}_m	3.142, 0.088	1.802, 0.703	0.013
S1E2 \mathcal{D}_m	3.140, 0.089	1.803, 0.703	0.013
target dataset $\mathcal{D}_t = \{\mathcal{D}_{train}, \mathcal{D}_{test}\}$	3.142, 0.088	1.850, 0.700	0.000

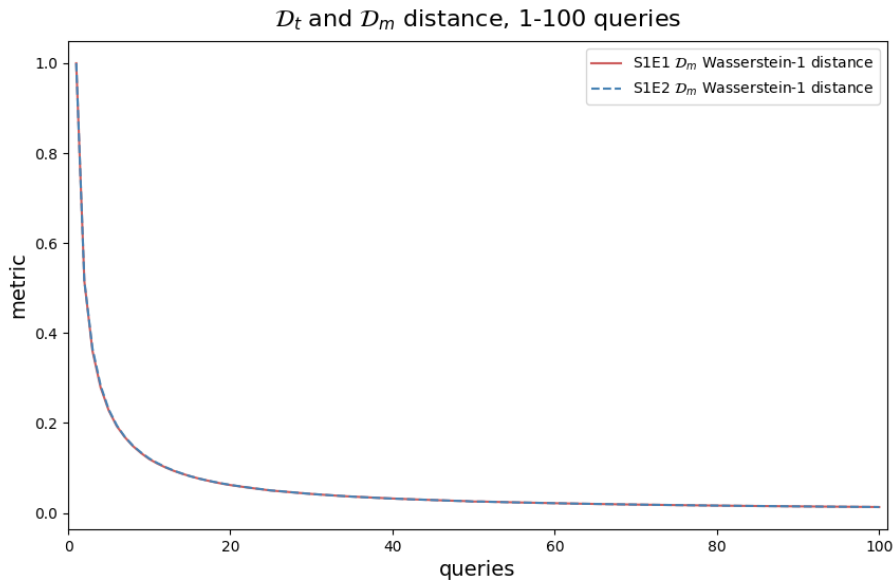


Figure 8.2: Scenario 1 Wasserstein-1 distance between \mathcal{D}_t and \mathcal{D}_m , collected from 1-100 queries and averaged over 20,000 experimental runs. The results for S1E1 and S1E2 overlap, which is natural considering the law of great numbers.

8.1.2 Monitor details S1E1

Turning attention to the details of S1E1 M_m , all possible outcomes become clear, as shown in Figure 8.3. During the initial 10-20 queries, the monitor struggles with limited training data, spiking MSE loss. When trained on all 100 queries, the prediction consistency becomes capable, measured as MSE loss on \mathcal{D}_{test} . Even if it never reaches the results of M_t , it would provide good enough predictions in many cases.

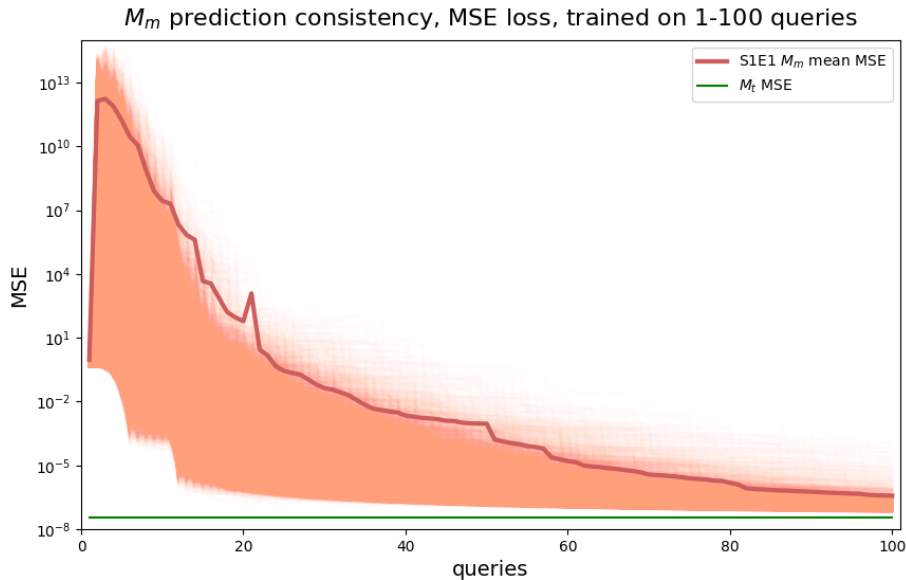


Figure 8.3: Prediction consistency for S1E1 M_m , trained on 1-100 queries from the M_t API. MSE loss on \mathcal{D}_{test} is used as a metric. All experiments are plotted alongside the mean as a dashed curve for M_m . After all 100 queries, the monitor proves capable; however, it never reaches the results of M_t .

Since the monitor is based on a parametric Ridge regression model, parameter vector similarity between M_m and M_t has been tracked, as plotted in Figure 8.4. Both normalized Euclidean distance and Cosine similarity behave as expected and could be used to track model parameter similarity.

8.1.3 Monitor details S1E2

Reviewing the details of the S1E2 M_m , the monitor performs well even with limited training data, as shown in Figure 8.5. The outcomes of all individual experiments are evenly centered around the mean, with few extreme outliers. On average, the MSE loss is monotonically decreasing and visually indistinguishable from M_t after 100 queries. In some individual experiments, M_m surpasses M_t results early on.

There are no other metrics implemented for this monitor since it is non-parametric.

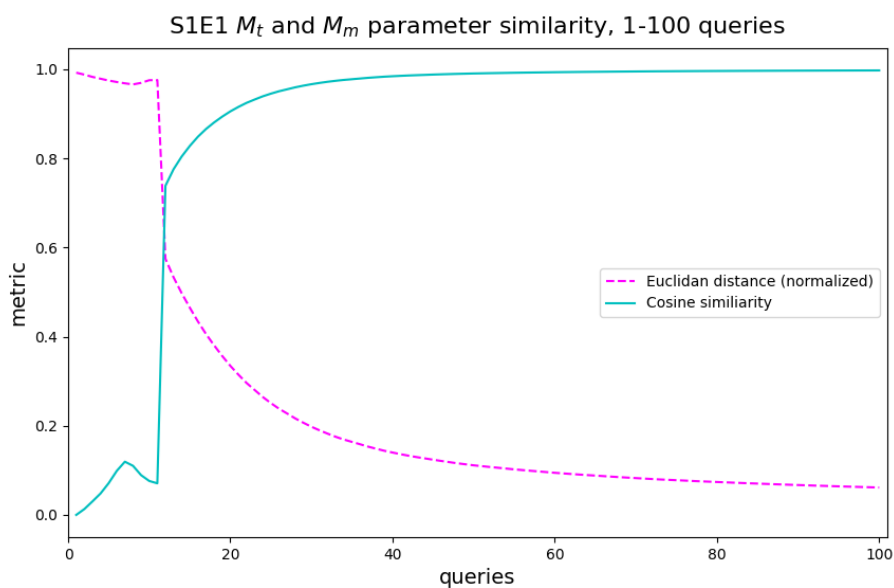


Figure 8.4: Parametric model similarity between S1E1 M_m and M_t , trained on 1-100 queries from the M_t API.

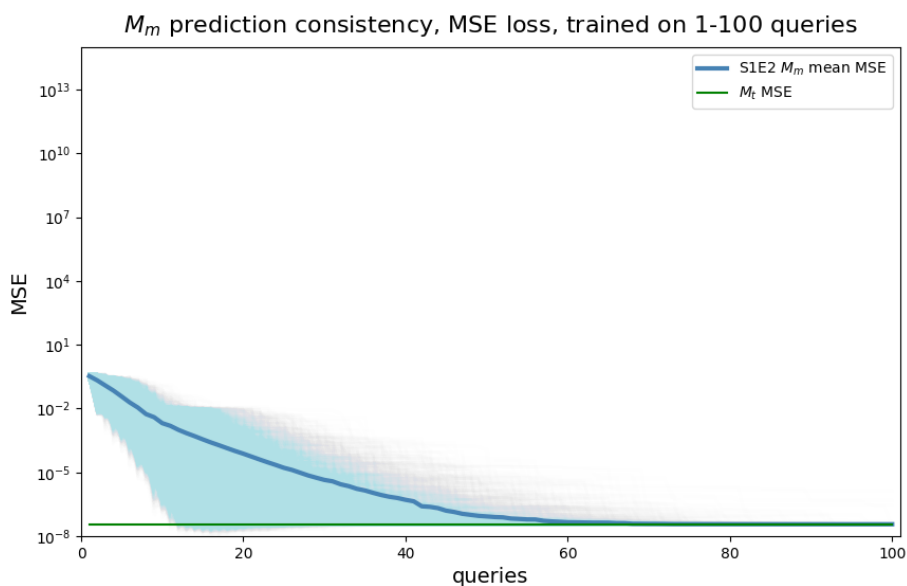


Figure 8.5: Prediction consistency for S1E2 M_m , trained on 1-100 queries from the M_t API. MSE loss on \mathcal{D}_{test} is used as a metric. All experiments are plotted alongside the mean for the M_m as a dashed curve. After only 10-20 queries, M_m surpasses the results of M_t in some cases. After 100 queries, M_m and M_t results are on average indistinguishable.

8.1.4 Monitor datasets

Previously, in Table 8.2, it has been shown that the average properties of \mathcal{D}_m used for respective monitors are equivalent, even if they are not the same. They also share properties with \mathcal{D}_t . Tracking the Wasserstein-1 distance also shows overlapping results, as presented in Figure 8.2. This enables comparability between the two monitors.

However, a few words of caution should be noted on the topic of comparability. Since a random D_m is used in every experiment, an individual D_m could differ substantially from the average over all 20,000 D_m . To visualize the difference, one random D_m has been drawn from experiments on S1E1 M_m and one from experiments on S1E2 M_m . Their contents are clearly differently distributed, as presented in Figure 8.6.

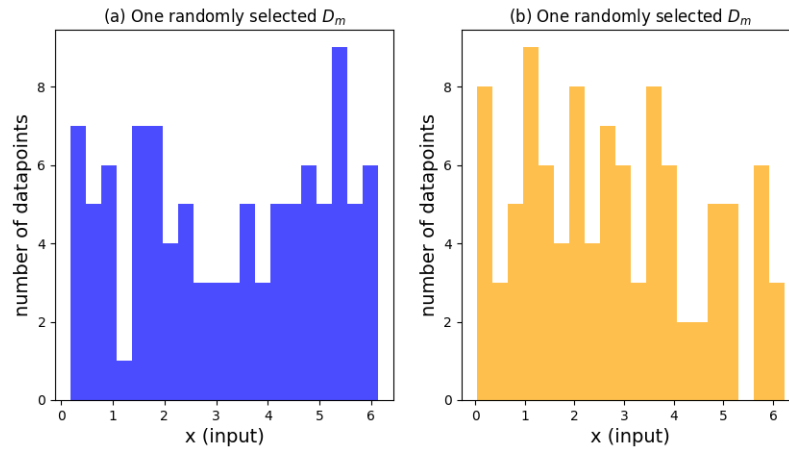


Figure 8.6: (a)-(b) Distributions of data samples in two random D_m from the experiments, illustrating that individual D_m can be very dissimilar to their average over 20,000 experimental runs. Averaged results are not the same as the outcome for each experiment since the data is random.

8.2 Scenario 2 results: Image classification

In this scenario, the target model is a system for image classification trained to classify images into ten categories: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The target model accuracy on the target test dataset is 77.23 %, and has a rank score of 7.364.

The scenario tests the proposed method for neural networks while evaluating more possible monitor model architectures. Six standard image classification architectures from the Torchvision model library are implemented and trained on input-output pairs from querying M_t :

- S2E1 M_m is a ResNet-18 model with the same properties as M_t .
- S2E2 M_m is based on the ResNet-152 model.
- S2E3 M_m is based on the VGG-11 model.
- S2E4 M_m is based on the VGG-19 model.
- S2E5 M_m is based on the DenseNet-161 model.
- S2E6 M_m is based on the DenseNet-121 model.

Queries generated by an adversary using M_t are stored in a monitor dataset, \mathcal{D}_m . The monitors are evaluated on behavior similarity, using prediction consistency by calculating accuracy on \mathcal{D}_{test} . The rank score is evaluated for each M_m as a metric for approximating model rank similarity to M_t .

Numerical data from the experiments are available in Appendix B.

8.2.1 Aggregated results

Aggregated results for the ensemble of evaluated monitors are presented in Figure 8.7 and Table 8.3. Incrementally, each M_m increases the volume of training data. A new model is trained from scratch for each additional batch of 1,000 queries. The prediction consistency is plotted for each experiment using accuracy as a metric.

For practical reasons, the number of possible queries in each experiment is limited to the 20,000 images available in \mathcal{D}_m .

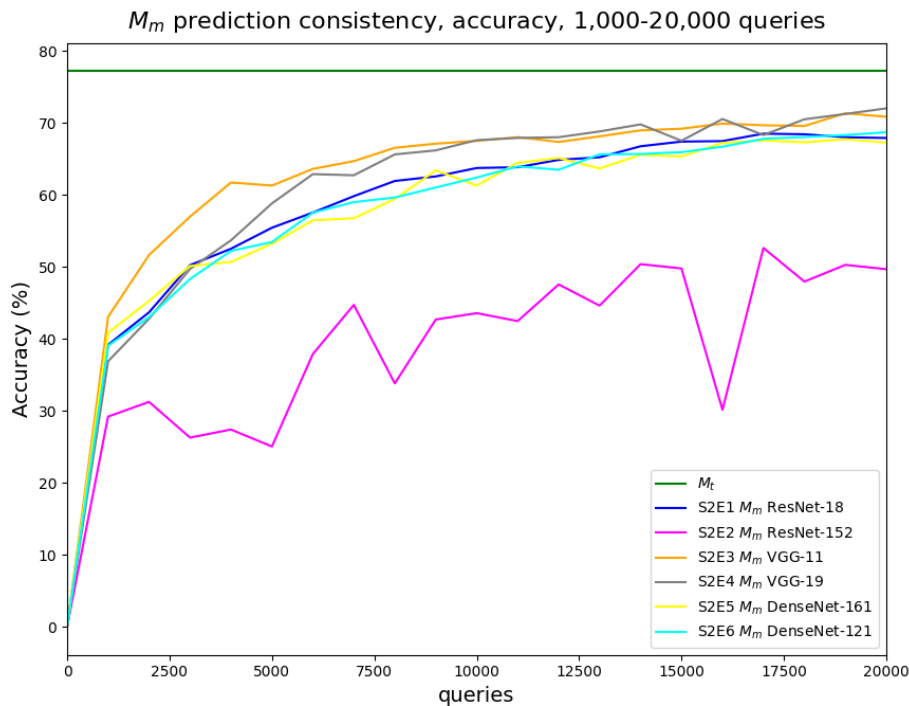


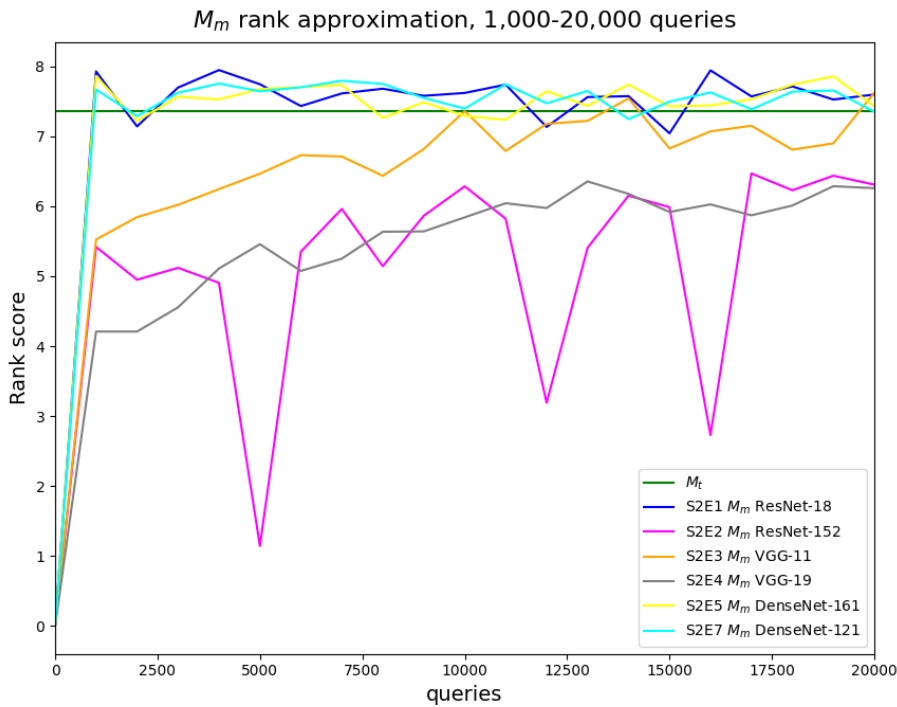
Figure 8.7: Scenario 2 prediction consistency on \mathcal{D}_{test} for six M_m , trained on 1,000-20,000 queries from the M_t API. Accuracy on \mathcal{D}_{test} is used as a metric for comparison.

As shown in Figure 8.7, S2E3 M_m and S2E4 M_m archive the best overall performance. They share a the VGG architecture, reaching accuracy on \mathcal{D}_{test} of 70.88 % and 72.05 %, respectively, after 20,000 queries. Considering that M_t has an accuracy of 77.23 %, the behavior is greatly stolen at that point. Factoring in added uncertainty, M_t could, however, be in jeopardy far sooner.

For each iteration of the models in the ensemble, the model rank score is tracked using the method presented in Section 7.2.3. The idea is that the rank score approximates the model output complexity, working as a proxy for model similarity. The results are presented in Figure 8.8. For some models, the score increases with the volume of available training data. For others, the score consistently oscillates around the score of the M_t .

Table 8.3: Scenario 2 M_m prediction consistency and model rank similarity to M_t for 20,000 queries and for peak accuracy. The target model was added for reference.

Monitors	Trained on 20,000 queries		Best accuracy at...		
	Acc. (%)	Rank score	Queries	Acc. (%)	Rank score
S2E1 M_m RN-18	67.93	7.599	17,000	68.53	7.571
S2E2 M_m RN-152	49.70	6.310	17,000	52.65	6.470
S2E3 M_m VGG-11	70.88	7.623	19,000	71.37	6.900
S2E4 M_m VGG-19	72.05	6.260	20,000	72.05	6.260
S2E5 M_m DN-161	67.30	7.433	19,000	67.75	7.857
S2E6 M_m DN-121	68.72	7.352	20,000	68.72	7.352
M_t , trained on 30,000 images in \mathcal{D}_{train}			-	77.23	7.364

Figure 8.8: Monitor rank approximation using the rank score metric, trained on 1,000-20,000 queries to the M_t API. M_t rank score is included as a reference.

8.2.2 Monitor dataset

To ensure comparability, all models should be trained on similar \mathcal{D}_m . For the experiments in this scenario, all models have been trained on the same \mathcal{D}_m .

The Fréchet Inception Distance can measure quality similarities for images in datasets. A small distance indicates similar datasets; the distance between $\mathcal{D}_t = \{\mathcal{D}_{train}, \mathcal{D}_{test}\}$ and itself is zero. For \mathcal{D}_m and \mathcal{D}_t , the distance is 1.904. Using research by Heusel et al. [44] as a benchmark, this distance must be considered a small distance, approximately zero.

The contents of \mathcal{D}_m are 20,000 images from the CIFAR-10 dataset, originally distributed evenly between the ten categories, denoted \mathcal{D}_{query} . However, when these images are used

to query M_t , the labels change. M_t has a 79.3 % accuracy on this task, resulting in each M_m being a somewhat distorted dataset. The dataset distributions in this scenario are described by Table 8.4.

Table 8.4: Scenario 2: Distribution of images in used datasets

Category distribution	Dataset size			
	\mathcal{D}_{train}	\mathcal{D}_{test}	\mathcal{D}_{query}	$\xrightarrow{M_t} \mathcal{D}_m$
airplane	3,000	1,000	2,000	2,174
automobile	3,000	1,000	2,000	2,116
bird	3,000	1,000	2,000	1,838
cat	3,000	1,000	2,000	1,849
deer	3,000	1,000	2,000	2,090
dog	3,000	1,000	2,000	2,025
frog	3,000	1,000	2,000	1,979
horse	3,000	1,000	2,000	1,889
ship	3,000	1,000	2,000	2,074
truck	3,000	1,000	2,000	1,966
sum	30,000	10,000	20,000	20,000



9 Discussion

This chapter starts by discussing the results presented in Chapter 8. A discussion also follows on the method of the experiments, described in Chapter 7. Finally, this work is put into a wider context.

9.1 Results

For the experimental work, the proposed method was evaluated using two scenarios. The first scenario was also used to improve the second scenario's design. Notes on this are available in Section 7.2.1, but will also be covered in the discussion provided in this chapter.

9.1.1 Scenario 1: Mechanical system

The aim of scenario 1 was to show that the approximation-based monitor approach could work for a limited and straightforward regression problem. Within the scenario, two very different ML models were evaluated as a basis for a monitor model. The scenario also explored the usefulness of Euclidean distance and cosine similarity as model similarity metrics for parametric models.

Monitor behavior

A Ridge regression model that uses the target model properties, M_t , is implemented for the first monitor (S1E1 M_m). This choice takes advantage of white-box knowledge about M_t , which performs well on the target dataset of 40 samples. However, as noted as a concern in Section 7.1.5, Ridge regression is a slow learner. It does not perform well on small volumes of training data, as shown in Figure 8.3.

During the first available queries to train on, MSE spikes for most of the 20,000 experimental runs. Even in the best experimental outcomes, it does not perform well. This behavior should not be surprising. Having very little data to train on, each additional training sample generates great shifts in the parameter vector. As the volume increases, the behavior calms and the results converge towards the one of M_t .

Considering the randomness of how M_t is queried and how \mathcal{D}_m is generated, the experiments have a wide range of possible outcomes. Observing the mean curve, the outcomes

above the curve are highly scattered and, therefore, not clearly visible. Outcomes below the mean curve are closer together, and for some experiments, the model is almost completely copied after training on 10-20 queries, depending on the desired prediction consistency. Here, there are also visible steps in the prediction accuracy. This indicates shifts in the parameter vector and the associated penalty function when reaching a certain number of queries to train M_m .

For the second monitor, a Gaussian process regression model is implemented (S1E2 M_m). The motivation for this choice is to utilize the strength of Bayesian methods for tackling problems involving uncertainty. As shown by Figure 8.5, the monitor performs well from the start. It does not exhibit the same initial issues as S1E1 M_m ; instead, the monitor has a monotonically decreasing function behavior on average. Overall, all possible outcomes seem to have a smoother behavior than S1E1 M_m .

The same randomness in how \mathcal{D}_m is generated is present for this experiment. However, the span of possible outcomes is tighter for S1E2 M_m . The distribution of outcomes is also more evenly divided over this span. Early on, from 10 available queries to train on, M_m outperforms M_t in some cases. As the volume of training data increases, S1E2 converges onto itself since the number of possible outcomes becomes more limited. It also converges towards M_t and is practically indistinguishable from M_t from 60 queries and onward.

Parametric similarity metrics

Since S1E1 M_m was a parametric model, the Euclidean distance and cosine similarity were tracked for parametric model similarity to M_t . For larger systems, tracking a metric other than MSE on a test dataset could make computational sense.

As for the prediction consistency of S1E1 M_m , these metrics have a considerable initial flutter, as shown by Figure 8.4. However, after about 10-15 queries, both have a smooth and monotonic behavior. Under the constraints of this experiment, where two Ridge regression parameter vectors are compared, these metrics could be harnessed for a monitor instead of MSE loss. Using these metrics for other parametric models or across a combination of parametric models could perhaps be possible, depending on their properties.

Dataset similarity metrics

For the scenario, dataset similarity metrics are mainly used for ascertaining that the models are trained on equivalent data. Otherwise, differences in behavior could have to do with data and not model properties.

The datasets are entirely numerical and based on the behavior of a sine function. This makes using basic statistical metrics, such as the arithmetic mean and standard deviation, useful, and confirming their validity is intuitively easy.

The Wasserstein-1 distance, also called the Earth movers distance, was implemented for this scenario. It calculates the cost of transforming the distribution of one dataset into another. If one were to base a monitor on analyzing data similarity to the target dataset rather than a continuously trained model, the metric could be useful for the task, even if not examined further for this work.

After 100 queries averaged over 20,000 experimental runs, the dataset collected, \mathcal{D}_m , is almost equivalent to the target dataset, \mathcal{D}_t , based on mean and standard deviation. That is expected due to the law of great numbers. However, this would not necessarily be true for a single experiment, illustrated by Figure 8.6. Differences in datasets between each experimental run explain the span of all possible outcomes for the two monitors, discussed above and illustrated in Figure 8.1.

Usefulness of monitors

Within the confines of scenario 1, the experiments show that the proposed method could work. Implemented with an appropriate threshold, it could protect against query-based attacks by cut-off or some other defensive measure.

The two evaluated M_m behave differently, illustrated by Figure 8.1 and Table 8.1. S1E1 M_m has the benefit of sharing model architecture with M_t , opening up more options to track monitor similarity. This could benefit the task of monitoring a more complex target model than the one used for this experiment. S1E2 M_m , based on a Gaussian process regression, has the upper hand on prediction consistency. Using white-box knowledge to make M_m into an exact copy of M_t properties is clearly unsuccessful.

For an attack, there is black-box knowledge both ways. The adversary knows little about the target model. The owner of the target model knows very little about an adversary. Based on this realization, the natural choice is to test all possible models as a basis for a monitor, picking the worst-case scenario from a defense perspective. This result has been allowed to influence the design of scenario 2.

For this experimental setup, continued work using the S1E2 M_m would be a natural choice since it outperforms S1E1 M_m - and sometimes even M_t .

9.1.2 Scenario 2: Image classification

For scenario 2, the aim was to show that the approximation-based monitor approach could work in a complex setup with neural network models. It also explored using rank scores as a model similarity metric for neural networks.

Monitor behavior

An ensemble of six monitors stemming from three convolutional neural network families is implemented for the scenario. For practical reasons, the models were default installations from the Torchvision models library. The aggregated results of the monitors are illustrated in Figure 8.7 and summarized in Table 8.3.

Of the six models, the two based on the VGG architecture perform best, reaching a prediction accuracy of over 70 %. S2E3 M_m achieves 70.88 %, while S2E4 M_m achieves 72.05 %. This can be compared to M_t , reaching an accuracy of 77.23 %.

S2E1 M_m , S2E5 M_m , and S2E6 M_m all achieve a prediction accuracy just below 70 %. After 20,000 queries have been made available to the models, all discussed M_m converge around 70 % accuracy.

The outlier is the S2E2 M_m , which generally shows bad results. Its prediction accuracy must be considered unstable. Of the six evaluated monitors, it is the only one not improving at a somewhat constant pace. Clearly, the implementation of S2E2 M_m was not successful using the same properties as for all other models.

Rank similarity metric

For work on neural network architectures, the scenario explored whether a rank score could be used as a monitor similarity metric. If successful, this could be used in a similar way as parametric model similarity did for S1E1 M_m .

For the ensemble of the six monitors, presented in Figure 8.8, there is no clear indication that the rank score could be used as a general proxy for model similarity. For the VGG-based S2E3-S2E4 M_m , there is an indication that it could be used for those specific models. However, for S2E1 M_m , S2E5 M_m , and S2E6 M_m , the rank score oscillates at the same level as M_t .

In implementing a rank score for this experiment, the image embedding is collected from each model output layer. In hindsight, this might not be a sufficiently complex layer to approximate model rank complexity for most cases.

The idea of a rank score metric shows some promise for comparing model similarity. However, the results of this work are not clear enough to draw any definite conclusions.

Dataset similarity metrics

As for scenario 1, dataset similarity metrics have mainly been used to ascertain that the models are trained on equivalent data. In this case, all monitors were trained on the same dataset and fed queries in the same order.

Fréchet Inception Distance was used to compare the target and monitor datasets. It is a metric well established in the field of ML. The distance, or the difference between the datasets, is very small - perhaps unrealistically small for a more realistic attack scenario. Since the images in both datasets originate from CIFAR-10, they are more similar than random images scraped from the internet and fed to M_t .

Usefulness of monitors

Scenario 2 broadens the results from scenario 1, showing that the proposed method could work for monitoring more complex neural networks. Implemented with an appropriate threshold, it could protect against query-based attacks.

For the setting of scenario 2, choosing one of the two VGG-based models would be a natural choice since they perform best. For this experiment, S2E4 M_m performs slightly better. It should be noted that neither in scenario 2, it was a successful strategy to let M_m be an exact copy of M_t .

9.2 Method discussion

This section discusses and criticizes the proposed method and experimental work.

9.2.1 Monitor limitations

The proposed method is interesting for monitoring attacks against ML models. However, it is not difficult to identify issues with such an approach for a real-world case.

The experimental work is conducted on models of limited size and complexity. The necessary calculations are, therefore, also limited. This makes it possible to re-do metric calculations often, more often than what could be practical for a commercially available model.

Delimitations made for this work also create a highly controlled and predictable environment. If queries to the model API were made from several user accounts, this falls outside the scope of this work. This would be a useful approach in a real-world attack, working around the monitoring proposed in this work and for most monitors from related research.

9.2.2 Grey-box knowledge

For the experimental work, the focus has been on the target side of the API. Actions taken on a potential attack side are unknown, except for the inputs to the API. Making this delimitation for the experiments should make for more realistic results by adhering to the concept of black-box knowledge.

To evaluate the monitors, an attack has to be simulated, i.e., the target model has to be queried in some structured way. In the design of both scenarios, there is more knowledge about the target model and dataset than what would normally be the case. This is a good thing since it creates a harsher environment for the target model. For example, in querying the target model in scenario 2, images that originate from the same dataset as the target training data are used. Hence, images are equal in quality and only belong to one of the ten categories. This is not realistic, making the simulations into a desired worst-case evaluation.

9.2.3 Reference methods

Although other monitor methods are referenced in related work, Chapter 4, none of them are implemented and used to evaluate the relative effectiveness of the proposed method. This work explores model approximation-based monitors and compares the behavior of different model architectures within this method. However, it does not put it into the context of other solutions.

9.2.4 Compromises

For the design of scenario 2, a compromise was made based on the experience of scenario 1. This led to more neural network models being implemented and evaluated but at the price of less attention to the detailed design of those models. A set of proven, standard Torchvision library models was used, but no adaptations were made to choose hyperparameters, optimizers, etc. It can, therefore, not be claimed that each model was given optimal conditions for performance. For this reason, the performance of individual neural networks might be less relevant than the bigger picture represented by the ensemble of monitors.

9.2.5 Research sources

The research area of MEA has gained traction as the dependence of ML has increased. This means that much of the research referenced in this work is fairly recent. On the plus side, this means that the material is current. On the negative side, there are gaps in knowledge and concepts that other researchers have not confirmed. Some source material is not peer-reviewed.

Following the research material in chronological order, the difficulty of protecting ML models against attacks clearly emerges. When a problem is examined, and a solution is proposed, it seldom takes long for a new hack to emerge, making the solution obsolete. This is a fast-moving field of research, making an overview difficult. One example is that a standard set of notations and terminology has clearly not yet been settled within the community, causing unnecessary confusion.

A large part of the research into MEA is focused on commercial ML platforms, funded and made in co-operation with their operators [15, 23]. Taking model extraction attacks seriously is good, but it also puts into question whether the research findings are comprehensively presented. Of course, there is no clear incentive to inform about all vulnerabilities found, and too many details are included in defensive actions taken.

For theory on ML fundamentals, in Chapter 2, two of the sources used are textbooks, by Lindholm et al. [4] and Goodfellow et al. [17]. This was a deliberate choice, combining sources that are meant to be accessible with more research-oriented source material. The ambition of this chapter is accessibility to the area of ML without much prior knowledge.

9.3 The work in a wider context

Deployed in all parts of society, research into machine learning is of great importance. Using AI and ML can facilitate a positive societal change, helping humanity and individuals reach new insights.

Considering the UN Sustainable Development Goals¹, technological breakthroughs could greatly advance several of the 17 goals. For instance, machine learning in healthcare has been shown to advance the development of new diagnostics and treatments, making good health and well-being accessible to more people [52]. Equally, it is fairly easy to imagine the positive impact that technology could have on access to education, industry innovation, equality, and

¹<https://sdgs.un.org/>

sustainable communities. Research by Asadikia et al. [52] shows that using ML also could help prioritize between goals and individual efforts to achieve a greater impact.

Inevitably, and rightly so, there is also skepticism and fear of how technology will impact societies and people's lives. It is not self-evident that only good comes from technological achievements. One growing concern within the field of ML is the safety, integrity, and protection of the models and the data they are built upon. The models used for providing our healthcare might also contain some of our most personal information, directly impacting the integrity of individuals. On a societal level, model theft or leaks of proprietary and sensitive information might seriously impact national interests - and erode trust in political leaders and institutions [7].

Learning more about the possibilities of machine learning brings necessary tools for society. However, building knowledge about its weaknesses is just as important. In *Everyday Ethics for Artificial Intelligence* [53], a report by IBM Research, states five pillars for trustworthy AI: Explainability, fairness, robustness, transparency, and privacy. Research into these five areas is just as important to mitigate risks, build robustness, and achieve societal trust necessary for the continued, safe deployment of more useful services and models. This work is a small and humble contribution.



10 Conclusion

This chapter contains a summarization of the purpose and the research questions. Finally, it also contains ideas for continued and future work.

10.1 Conclusion

This thesis explores an approach for approximation-based monitoring of model extraction attacks (MEAs), utilizing interactions between an adversary and the targeted model. The aim has been to approximate an ongoing attack by continuously evaluating the similarity between a targeted model and a monitor model. Based on a theoretical foundation and experimental scenarios, this work has sought to answer three research questions.

The first research question, '**How could black-box interactions with a targeted model be used for approximating an adversary substitute model?**', centers around using queries put to a model API for approximating how far a potential theft has progressed. To answer this question, a method for an approximation-based monitor is proposed. It uses the queries from a user and the output from the targeted model to train a worst-case monitor model that can indicate an adversary's success. The method has been evaluated during two experimental scenarios, showing that it could work under the delimitations of the thesis.

Results from both scenarios show that query-based retraining attacks can be successful early on. For the right choice of architecture, the prediction accuracy for a model copy can be close to the targeted model. In both scenarios, the monitor model that was best at copying the targeted model was of another architecture than was used for the targeted model. An approach to determine the worst-case scenario, providing the best monitor, could be to identify the model architecture that performs best at copying the targeted model, using as few queries as possible to train on, given a desired performance.

The limited size of the problems posed for the experiments certainly affects the outcome, but it is also an example of how difficult it can be to protect against an MEA. The balance between the usability of a model and its protection is not straightforward.

The proposed method can be questioned on the grounds of efficiency. Clearly, the approach is not feasible for large-scale MLaaS. It could, however, be useful for some models and as a part of a larger protection setup.

The second research question, '**What are relevant model and dataset similarity metrics?**', deals with measuring similarities between ML models and datasets.

For model similarity, the thesis discusses metrics for three categories: Behavioral similarity, parametric similarity, and rank similarity. For experimental evaluation, mean square error and accuracy have been used to track behavior similarity. These are well-proven metrics but can be computationally heavy for large test datasets. For comparing parametric models, Euclidean distance and Cosine similarity have been used to track model parametric similarity. These could be good monitoring metrics when computationally easier to calculate than the alternative. However, they are limited for use on comparable model architectures. Rank similarity is evaluated to compare similarity for neural networks. The metric shows some promise for certain architectures, but results are inconclusive.

For dataset similarity metrics, the main purpose has been to ascertain that the monitor evaluations have been made on equivalent data. Well-proven statistical measures, such as the arithmetic mean and the standard deviation, have been used for numerical datasets. In this setting, optimal transport has been implemented using Wasserstein-1 distance, also called Earth movers distance. This metric works well for the problems it is used for, comparing datasets that can be described as distributions. For comparing datasets of images, Fréchet inception distance is used in a limited capacity. It is a well-proven metric, albeit computationally costly for large datasets.

The third research question, **'When an attempted model extraction attack is detected, what are possible defenses?'**, deals with possible defensive measures that a monitor could take. For this work, the topic of possible defensive strategies and tactics is only addressed in theory.

The most basic method is to cut off the user's access to the targeted model when the monitor triggers based on some metric threshold. This action prevents the user from collecting enough queries to retrain a substitute model. Easy to implement, it has the downside of revealing information to the adversary. The attack has obviously been successful when the user is cut off from use.

Output perturbations, slightly distorting or truncating values, have been the topic of much research. It has the benefit of not revealing when an adversary is getting too close to copying a model, and small perturbations might not make a difference for a benign user. However, even small perturbations have been shown to cause substantial drift when using queries to train a substitute model. A similar approach is a mathematical framework for differential privacy, initially focused on protecting the privacy of individuals in datasets.

Watermarking data is another defensive measure focused on asserting ownership after an attack. This could add a distinguishing feature to images and texts. Another approach is to add fabricated data, such as paper towns, that should not exist. If occurring in another model, it indicates that it has been trained on stolen data.

Cut-off, output perturbations, and watermarking focus on protecting model behavior by controlling the data. However, other methods focus more on protecting the model properties, such as model obfuscation and model distillation.

10.2 Future work

From the experimental work of this thesis, it is clear that the choice of model on which to base a monitor affects the efficiency of the monitoring. If the monitor model learns very slowly, it might not trigger until too late. Hence, for maximum protection, it is important to test several models and pick the worst-case scenario. For this work, the scenarios barely scratch the surface of possible combinations. Neither have all models been given optimal conditions to perform well. A more comprehensive and systematic study of possible combinations for monitor models could be interesting to continue work on this method.

In researching MEA, it is noted that there is a lack of knowledge of extraction attacks aimed at regression models. This was one of the reasons that this was made a part of an experimental scenario. However, the experiment is limited, and the targeted model is low-

dimensioned. Continued work on more complex regression models, both from a monitoring and attack standpoint, should make for interesting work and for covering a gap in current research.

The work of this thesis uses intercepted queries to train a monitor model that is then used as a proxy for approximating an adversary's substitute model. Since training models are computationally costly, and their outcome can depend heavily on the choice of parameters, another approach for designing a monitor would be to directly analyze the intercepted data. When examining related work, there is research into analyzing patterns in the inputs to determine if an attack is ongoing. Using similar methods for analyzing both the input and the output could be a more powerful tool, yet still more effective than parallel training a monitor model.



Bibliography

- [1] Sergei Treil. *Linear Algebra Done Wrong*. 2017. URL: <https://www.math.brown.edu/streil/papers/LADW/LADW.html>.
- [2] Nist. *Security and Privacy Controls for Information Systems and Organizations*. Tech. rep. Gaithersburg, MD: National Institute of Standards and Technology, Sept. 2020. DOI: 10.6028/NIST.SP.800-53r5.
- [3] Nist. *Computer Security Resource Center: Glossary*. Jan. 2024. URL: <https://csrc.nist.gov/glossary/>.
- [4] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön. *Machine Learning*. Cambridge University Press, Mar. 2022. ISBN: 9781108919371. DOI: 10.1017/9781108919371.
- [5] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. “I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences”. In: *ACM Computing Surveys* 55.14 (July 2023). ISSN: 15577341. DOI: 10.1145/3595292.
- [6] Didem Genç, Mustafa Özuysal, and Emrah Tomur. “A Taxonomic Survey of Model Extraction Attacks”. In: Institute of Electrical and Electronics Engineers (IEEE), Aug. 2023, pp. 200–205. ISBN: 9798350311709. DOI: 10.1109/csr57506.2023.10224959.
- [7] Säkerhetspolisen. “Årsbok 2022 – 2023”. In: (2023). URL: <https://sakerhetspolisen.se/om-sakerhetspolisen/publikationer/sakerhetspolisens-arsberattelse/sakerhetspolisen-2022-2023/sammanfattning/lagesbilden-som-pdf.html>.
- [8] IBM. *Cost of a Data Breach Report 2023*. Tech. rep. 2023. URL: <https://www.ibm.com/reports/data-breach>.
- [9] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006, p. 248. ISBN: 026218253X.
- [10] Y LeCun, B Boser, J S Denker, D Henderson, R E Howard, W Hubbard, and L D Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.
- [11] K He, X Zhang, S Ren, and J Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. ISBN: 1063-6919. DOI: 10.1109/CVPR.2016.90.

- [12] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: (Sept. 2014). URL: <http://arxiv.org/abs/1409.1556>.
- [13] G Huang, Z Liu, L Van Der Maaten, and K Q Weinberger. “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. ISBN: 1063-6919. DOI: 10.1109/CVPR.2017.243.
- [14] Quentin Garrido, Randall Balestriero, Laurent Najman, and Yann Lecun. “RankMe: Assessing the downstream performance of pretrained self-supervised representations by their rank”. In: *Proceedings of Machine Learning Research*. Oct. 2023. URL: <http://arxiv.org/abs/2210.02885>.
- [15] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. *Stealing Machine Learning Models via Prediction APIs*. Tech. rep., p. 601. URL: https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_tramer.pdf.
- [16] Yang Liu, Ji Luo, Yi Yang, Xuan Wang, Mehdi Gheisari, and Feng Luo. “ShrewdAttack: Low Cost High Accuracy Model Extraction”. In: *Entropy* 25.2 (Feb. 2023). ISSN: 10994300. DOI: 10.3390/e25020282.
- [17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [18] Lennart Råde, Bertil Westergren, and Frank Wikström. *Mathematics Handbook For Science and Engineering*. 6:1. Lund: Studentlitteratur, 2019.
- [19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *Springer Series in Statistics The Elements of Statistical Learning Data Mining, Inference, and Prediction, 2nd*. ISBN: 978-0-387-84857-0.
- [20] S Amari. “A Theory of Adaptive Pattern Classifiers”. In: *IEEE Transactions on Electronic Computers* EC-16.3 (1967), pp. 299–307. ISSN: 0367-7508. DOI: 10.1109/PGEC.1967.264666.
- [21] Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. “Efficiently stealing your machine learning models”. In: *Proceedings of the ACM Conference on Computer and Communications Security*. Association for Computing Machinery, Nov. 2019, pp. 198–210. ISBN: 9781450368308. DOI: 10.1145/3338498.3358646.
- [22] Ben Cottier. *Trends in the Dollar Training Cost of Machine Learning Systems*. 2023. URL: <https://epochai.org/blog/trends-in-the-dollar-training-cost-of-machine-learning-systems>.
- [23] Yugeng Liu, Rui Wen, Xinlei He, Ahmed Salem, Zhikun Zhang, Michael Backes, Emiliano De Cristofaro, Mario Fritz, and Yang Zhang. *ML-DOCTOR: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models*. Tech. rep. 2021. URL: <https://github.com/liuyugeng/ML-Doctor>.
- [24] Daniel Lowd and Christopher Meek. *Adversarial Learning*. ACM Press, 2005, p. 826. ISBN: 159593135X.
- [25] Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. “Efficiently stealing your machine learning models”. In: *Proceedings of the ACM Conference on Computer and Communications Security*. Association for Computing Machinery, Nov. 2019, pp. 198–210. ISBN: 9781450368308. DOI: 10.1145/3338498.3358646.
- [26] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. “Model Extraction Warning in MLaaS Paradigm”. In: (Nov. 2017). URL: <http://arxiv.org/abs/1711.07221>.

- [27] Soham Pal, Yash Gupta, Aditya Kanade, and Shirish Shevade. “Stateful Detection of Model Extraction Attacks”. In: (July 2021). URL: <http://arxiv.org/abs/2107.05166>.
- [28] Xinjing Liu, Zhuo Ma, Yang Liu, Zhan Qin, Junwei Zhang, and Zhuzhu Wang. “SeInspect: Defending Model Stealing via Heterogeneous Semantic Inspection”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 13554 LNCS. Springer Science and Business Media Deutschland GmbH, 2022, pp. 610–630. ISBN: 9783031171390.
- [29] Zhanyuan Zhang, Yizheng Chen, and David Wagner. “SEAT: Similarity Encoder by Adversarial Training for Detecting Model Extraction Attack Queries”. In: *AISec 2021 - Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, co-located with CCS 2021*. Association for Computing Machinery, Inc, Nov. 2021, pp. 37–48. ISBN: 9781450386579. DOI: 10.1145/3474369.3486863.
- [30] Adam Dzedzic, Muhammad Ahmad Kaleem, Yu Shen Lu, and Nicolas Papernot. “Increasing the Cost of Model Extraction with Calibrated Proof of Work”. In: (Jan. 2022). URL: <http://arxiv.org/abs/2201.09243>.
- [31] A M Sadeghzadeh, A M Sobhanian, F Dehghan, and R Jalili. “HODA: Hardness-Oriented Detection of Model Extraction Attacks”. In: *IEEE Transactions on Information Forensics and Security* 19 (2024), pp. 1429–1439. ISSN: 1556-6021. DOI: 10.1109/TIFS.2023.3320609.
- [32] Hengrui Jia, Christopher A Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. *Entangled Watermarks as a Defense against Model Extraction*. Tech. rep. 2021.
- [33] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. “A Watermark for Large Language Models”. In: (Jan. 2023). URL: <http://arxiv.org/abs/2301.10226>.
- [34] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. “Prediction Poisoning: Towards Defenses Against DNN Model Stealing Attacks”. In: *[International Conference on Machine Learning Logo] Proceedings of Machine Learning Research*. June 2020. URL: <http://arxiv.org/abs/1906.10908>.
- [35] Cynthia Dwork. “Differential Privacy”. In: *Automata, Languages and Programming*. Ed. by Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12. ISBN: 978-3-540-35908-1.
- [36] Huadi Zheng, Qingqing Ye, Haibo Hu, Changfang Fang, and Jie Shi. *BDPL: A Boundary Differentially Private Layer Against Machine Learning Model Extraction Attacks*. Ed. by Kazue Sako, Steve Schneider, and Peter Y. A. Ryan. Vol. 11735. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-29958-3. DOI: 10.1007/978-3-030-29959-0.
- [37] Kálmán Szentannai, Jalal Al-Afandi, and András Horváth. *Intelligent Computing: Preventing Neural Network Weight Stealing via Network Obfuscation*. Ed. by Kohei Arai, Supriya Kapoor, and Rahul Bhatia. Vol. 1230. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2020, pp. 1–11. ISBN: 978-3-030-52242-1. DOI: 10.1007/978-3-030-52243-8.
- [38] Kuan-Chieh Wang, Y A N FU, Ke Li, Ashish Khisti, Richard Zemel, and Alireza Makhzani. “Variational Model Inversion Attacks”. In: *Advances in Neural Information Processing Systems*. Ed. by M Ranzato, A Beygelzimer, Y Dauphin, P S Liang, and J Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 9706–9719. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/50a074e6a8da4662ae0a29edde722179-Paper.pdf.
- [39] Olivier Roy and Martin Vetterli. “The effective rank: A measure of effective dimensionality”. In: *2007 15th European Signal Processing Conference*. 2007, pp. 606–610.

- [40] Gaspard Monge. *Mémoire sur la Théorie des Déblais et des Remblai*. Hist. de l'Acad. des Sciences de Paris, 1781, pp. 666–704.
- [41] Gabriel Peyré and Marco Cuturi. “Computational Optimal Transport”. In: *Foundations and Trends in Machine Learning* 11 (5-6) (2019), pp. 355–602.
- [42] Cédric Villani. “Optimal transport: old and new”. In: *Bulletin of the american mathematical society* 47.4 (2010), pp. 723–727.
- [43] Cédric Villani. *Topics in Optimal Transportation*. Vol. 58. American Mathematical Society, 2003. ISBN: 9780821833124.
- [44] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. “GANs trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 6629–6640. ISBN: 9781510860964.
- [45] Sayanton V. Dibbo. “SoK: Model Inversion Attack Landscape: Taxonomy, Challenges, and Future Roadmap”. In: Institute of Electrical and Electronics Engineers (IEEE), Aug. 2023, pp. 439–456. ISBN: 9798350321920. DOI: 10.1109/csf57540.2023.00027.
- [46] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. “Membership Inference Attacks Against Machine Learning Models”. In: *Proceedings - IEEE Symposium on Security and Privacy*. Institute of Electrical and Electronics Engineers Inc., June 2017, pp. 3–18. ISBN: 9781509055326. DOI: 10.1109/SP.2017.41.
- [47] Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T H Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. “POT: Python Optimal Transport”. In: *Journal of Machine Learning Research* 22.78 (2021), pp. 1–8. URL: <http://jmlr.org/papers/v22/20-451.html>.
- [48] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. *CINIC-10 Is Not ImageNet or CIFAR-10*. Tech. rep. 2018. URL: <https://datashare.is.ed.ac.uk/>.
- [49] Jia Deng, Wei Dong, Richard Socher, Li Li-Jia, Kai Li, and Fei-Fei Li. “ImageNet: A Large-Scale Hierarchical Image Database”. In: IEEE, 2009. ISBN: 9781424439911.
- [50] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. 2009.
- [51] Hargurjeet Singh Ganger. *7 Best Techniques To Improve The Accuracy of CNN W/O Overfitting*. URL: <https://medium.com/mllearning-ai/7-best-techniques-to-improve-the-accuracy-of-cnn-w-o-overfitting-6db06467182f>.
- [52] Atie Asadikia, Abbas Rajabifard, and Mohsen Kalantari. “Systematic prioritisation of SDGs: Machine learning approach”. In: *World Development* 140 (Apr. 2021). ISSN: 18735991. DOI: 10.1016/j.worlddev.2020.105269.
- [53] *Everyday Ethics for Artificial Intelligence*. Tech. rep. URL: www.ibm.com/legal/us/en/copytrade.shtml.


A

Scenario 1: Experimental data

The results presented in Chapter 8 are a condensate of the experimental data produced for this work. More detailed data for monitor performance and other metrics are presented as an addendum.

Table A.1: S1E1 M_m prediction consistency and parametric similarity to M_t for 1-100 queries available for training over 20,000 experimental runs.

S2E1 M_m RR	Prediction consistency MSE			Parametric similarity to M_t	
	Low	Mean	High	Euclid.	Cosine sim.
1	$4.323 \cdot 10^{-1}$	$9.403 \cdot 10^{-1}$	$1.607 \cdot 10^0$	0.9924	1.0000
2	$4.203 \cdot 10^{-1}$	$1.394 \cdot 10^{12}$	$2.353 \cdot 10^{14}$	0.9881	0.0132
3	$2.989 \cdot 10^{-1}$	$1.752 \cdot 10^{12}$	$5.197 \cdot 10^{14}$	0.9829	0.0303
4	$1.038 \cdot 10^{-1}$	$7.562 \cdot 10^{11}$	$4.204 \cdot 10^{14}$	0.9788	0.0482
5	$8.796 \cdot 10^{-3}$	$1.747 \cdot 10^{11}$	$3.941 \cdot 10^{14}$	0.9748	0.0713
6	$7.938 \cdot 10^{-5}$	$2.893 \cdot 10^{10}$	$1.870 \cdot 10^{14}$	0.9715	0.0989
7	$6.421 \cdot 10^{-5}$	$1.147 \cdot 10^{10}$	$2.499 \cdot 10^{13}$	0.9685	0.1195
8	$3.801 \cdot 10^{-5}$	$7.747 \cdot 10^8$	$3.038 \cdot 10^{12}$	0.9661	0.1102
9	$3.707 \cdot 10^{-5}$	$7.815 \cdot 10^7$	$4.994 \cdot 10^{11}$	0.9696	0.0892
10	$4.358 \cdot 10^{-5}$	$2.788 \cdot 10^7$	$1.947 \cdot 10^{11}$	0.9755	0.0762
11	$3.049 \cdot 10^{-5}$	$2.049 \cdot 10^7$	$2.188 \cdot 10^{11}$	0.9759	0.0709
12	$4.590 \cdot 10^{-7}$	$2.101 \cdot 10^6$	$5.246 \cdot 10^9$	0.5748	0.7386
13	$5.717 \cdot 10^{-7}$	$6.966 \cdot 10^5$	$4.374 \cdot 10^9$	0.5332	0.7758
14	$2.500 \cdot 10^{-7}$	$4.135 \cdot 10^5$	$3.069 \cdot 10^9$	0.4977	0.8042
15	$4.044 \cdot 10^{-7}$	$4.727 \cdot 10^3$	$2.525 \cdot 10^7$	0.4651	0.8279
16	$2.328 \cdot 10^{-7}$	$3.572 \cdot 10^3$	$3.216 \cdot 10^7$	0.4343	0.8486
17	$2.344 \cdot 10^{-7}$	$7.487 \cdot 10^2$	$8.079 \cdot 10^6$	0.4059	0.8660
18	$2.241 \cdot 10^{-7}$	$1.638 \cdot 10^2$	$5.612 \cdot 10^5$	0.3805	0.8807
19	$2.150 \cdot 10^{-7}$	$9.165 \cdot 10^1$	$5.216 \cdot 10^5$	0.3566	0.8938

Continued on next page

Queries	Prediction consistency MSE			Parametric similarity to M_t	
	Low	Mean	High	Euclid. (%)	Cosine sim.
20	$1.969 \cdot 10^{-7}$	$6.021 \cdot 10^1$	$5.043 \cdot 10^5$	0.3345	0.9055
21	$1.639 \cdot 10^{-7}$	$1.240 \cdot 10^3$	$7.043 \cdot 10^4$	0.3147	0.9156
22	$1.596 \cdot 10^{-7}$	$2.871 \cdot 10^0$	$1.083 \cdot 10^4$	0.2961	0.9247
23	$1.633 \cdot 10^{-7}$	$1.464 \cdot 10^0$	$8.141 \cdot 10^3$	0.2803	0.9322
24	$1.709 \cdot 10^{-7}$	$4.665 \cdot 10^{-1}$	$5.284 \cdot 10^3$	0.2646	0.9394
25	$1.484 \cdot 10^{-7}$	$2.794 \cdot 10^{-1}$	$6.432 \cdot 10^2$	0.2509	0.9453
26	$1.332 \cdot 10^{-7}$	$2.240 \cdot 10^{-1}$	$2.412 \cdot 10^2$	0.2383	0.9507
27	$1.380 \cdot 10^{-7}$	$1.870 \cdot 10^{-1}$	$2.209 \cdot 10^2$	0.2276	0.9551
28	$1.279 \cdot 10^{-7}$	$1.103 \cdot 10^{-1}$	$2.485 \cdot 10^2$	0.2166	0.9595
29	$1.271 \cdot 10^{-7}$	$6.276 \cdot 10^{-2}$	$2.644 \cdot 10^0$	0.2069	0.9633
30	$1.670 \cdot 10^{-7}$	$4.314 \cdot 10^{-2}$	$1.028 \cdot 10^0$	0.1979	0.9666
31	$1.381 \cdot 10^{-7}$	$3.635 \cdot 10^{-2}$	$1.025 \cdot 10^2$	0.1898	0.9695
32	$1.307 \cdot 10^{-7}$	$2.710 \cdot 10^{-2}$	$9.826 \cdot 10^1$	0.1823	0.9720
33	$1.181 \cdot 10^{-7}$	$1.986 \cdot 10^{-2}$	$1.065 \cdot 10^2$	0.1753	0.9742
34	$1.175 \cdot 10^{-7}$	$1.150 \cdot 10^{-2}$	$1.082 \cdot 10^2$	0.1691	0.9762
35	$1.125 \cdot 10^{-7}$	$7.032 \cdot 10^{-3}$	$9.967 \cdot 10^1$	0.1638	0.9777
36	$1.115 \cdot 10^{-7}$	$4.712 \cdot 10^{-3}$	$1.019 \cdot 10^2$	0.1586	0.9793
37	$1.071 \cdot 10^{-7}$	$3.984 \cdot 10^{-3}$	$8.953 \cdot 10^1$	0.1532	0.9809
38	$1.125 \cdot 10^{-7}$	$3.467 \cdot 10^{-3}$	$2.317 \cdot 10^1$	0.1484	0.9822
39	$1.106 \cdot 10^{-7}$	$3.056 \cdot 10^{-3}$	$2.253 \cdot 10^1$	0.1439	0.9834
40	$1.182 \cdot 10^{-7}$	$2.220 \cdot 10^{-3}$	$2.314 \cdot 10^1$	0.1400	0.9844
41	$1.271 \cdot 10^{-7}$	$1.963 \cdot 10^{-3}$	$7.934 \cdot 10^0$	0.1363	0.9853
42	$1.263 \cdot 10^{-7}$	$1.727 \cdot 10^{-3}$	$7.326 \cdot 10^0$	0.1328	0.9861
43	$1.324 \cdot 10^{-7}$	$1.629 \cdot 10^{-3}$	$1.740 \cdot 10^0$	0.1297	0.9868
44	$1.251 \cdot 10^{-7}$	$1.479 \cdot 10^{-3}$	$9.119 \cdot 10^{-1}$	0.1268	0.9875
45	$1.234 \cdot 10^{-7}$	$1.254 \cdot 10^{-3}$	$7.073 \cdot 10^{-1}$	0.1239	0.9882
46	$1.242 \cdot 10^{-7}$	$1.222 \cdot 10^{-3}$	$6.097 \cdot 10^{-1}$	0.1211	0.9887
47	$1.175 \cdot 10^{-7}$	$1.009 \cdot 10^{-3}$	$6.112 \cdot 10^{-1}$	0.1185	0.9893
48	$1.158 \cdot 10^{-7}$	$9.446 \cdot 10^{-4}$	$6.219 \cdot 10^{-1}$	0.1159	0.9898
49	$1.157 \cdot 10^{-7}$	$9.241 \cdot 10^{-4}$	$4.816 \cdot 10^{-1}$	0.1136	0.9902
50	$1.127 \cdot 10^{-7}$	$9.181 \cdot 10^{-4}$	$4.730 \cdot 10^{-1}$	0.1113	0.9906
51	$1.115 \cdot 10^{-7}$	$1.626 \cdot 10^{-4}$	$4.739 \cdot 10^{-1}$	0.1094	0.9910
52	$1.112 \cdot 10^{-7}$	$1.338 \cdot 10^{-4}$	$4.925 \cdot 10^{-1}$	0.1076	0.9913
53	$1.091 \cdot 10^{-7}$	$1.115 \cdot 10^{-4}$	$5.008 \cdot 10^{-1}$	0.1058	0.9916
54	$1.050 \cdot 10^{-7}$	$9.948 \cdot 10^{-5}$	$4.285 \cdot 10^{-1}$	0.1041	0.9919
55	$1.039 \cdot 10^{-7}$	$8.066 \cdot 10^{-5}$	$1.716 \cdot 10^{-1}$	0.1022	0.9922
56	$1.019 \cdot 10^{-7}$	$7.261 \cdot 10^{-5}$	$1.730 \cdot 10^{-1}$	0.1006	0.9925
57	$1.020 \cdot 10^{-7}$	$6.121 \cdot 10^{-5}$	$1.728 \cdot 10^{-1}$	0.0991	0.9927
58	$9.906 \cdot 10^{-8}$	$2.347 \cdot 10^{-5}$	$1.028 \cdot 10^{-1}$	0.0976	0.9930
59	$9.925 \cdot 10^{-8}$	$1.942 \cdot 10^{-5}$	$9.958 \cdot 10^{-2}$	0.0961	0.9932
60	$9.742 \cdot 10^{-8}$	$1.594 \cdot 10^{-5}$	$9.950 \cdot 10^{-2}$	0.0945	0.9935

Continued on next page

Queries	Prediction consistency MSE			Parametric similarity to M_t	
	Low	Mean	High	Euclid. (%)	Cosine sim.
61	$9.542 \cdot 10^{-8}$	$1.402 \cdot 10^{-5}$	$3.415 \cdot 10^{-2}$	0.0932	0.9937
62	$9.262 \cdot 10^{-8}$	$9.686 \cdot 10^{-6}$	$3.386 \cdot 10^{-2}$	0.0919	0.9939
63	$9.020 \cdot 10^{-8}$	$8.931 \cdot 10^{-6}$	$3.322 \cdot 10^{-2}$	0.0906	0.9940
64	$9.249 \cdot 10^{-8}$	$8.242 \cdot 10^{-6}$	$3.358 \cdot 10^{-2}$	0.0894	0.9942
65	$8.887 \cdot 10^{-8}$	$7.373 \cdot 10^{-6}$	$3.237 \cdot 10^{-2}$	0.0883	0.9944
66	$8.744 \cdot 10^{-8}$	$6.768 \cdot 10^{-6}$	$2.288 \cdot 10^{-2}$	0.0871	0.9946
67	$8.593 \cdot 10^{-8}$	$6.018 \cdot 10^{-6}$	$2.159 \cdot 10^{-2}$	0.0859	0.9947
68	$8.415 \cdot 10^{-8}$	$5.428 \cdot 10^{-6}$	$1.702 \cdot 10^{-2}$	0.0847	0.9949
69	$8.509 \cdot 10^{-8}$	$4.691 \cdot 10^{-6}$	$1.713 \cdot 10^{-2}$	0.0837	0.9950
70	$8.358 \cdot 10^{-8}$	$3.667 \cdot 10^{-6}$	$3.274 \cdot 10^{-3}$	0.0826	0.9952
71	$7.951 \cdot 10^{-8}$	$3.490 \cdot 10^{-6}$	$3.305 \cdot 10^{-3}$	0.0816	0.9953
72	$7.929 \cdot 10^{-8}$	$3.276 \cdot 10^{-6}$	$3.316 \cdot 10^{-3}$	0.0806	0.9954
73	$7.917 \cdot 10^{-8}$	$3.059 \cdot 10^{-6}$	$3.328 \cdot 10^{-3}$	0.0796	0.9956
74	$7.853 \cdot 10^{-8}$	$2.753 \cdot 10^{-6}$	$3.341 \cdot 10^{-3}$	0.0787	0.9957
75	$7.740 \cdot 10^{-8}$	$2.472 \cdot 10^{-6}$	$3.218 \cdot 10^{-3}$	0.0779	0.9958
76	$7.721 \cdot 10^{-8}$	$2.293 \cdot 10^{-6}$	$2.104 \cdot 10^{-3}$	0.0771	0.9959
77	$7.537 \cdot 10^{-8}$	$2.189 \cdot 10^{-6}$	$2.050 \cdot 10^{-3}$	0.0762	0.9960
78	$7.577 \cdot 10^{-8}$	$1.910 \cdot 10^{-6}$	$2.076 \cdot 10^{-3}$	0.0754	0.9961
79	$7.411 \cdot 10^{-8}$	$1.855 \cdot 10^{-6}$	$2.054 \cdot 10^{-3}$	0.0746	0.9961
80	$7.517 \cdot 10^{-8}$	$1.511 \cdot 10^{-6}$	$2.006 \cdot 10^{-3}$	0.0739	0.9962
81	$7.128 \cdot 10^{-8}$	$1.275 \cdot 10^{-6}$	$2.039 \cdot 10^{-3}$	0.0732	0.9963
82	$7.141 \cdot 10^{-8}$	$8.292 \cdot 10^{-7}$	$2.034 \cdot 10^{-3}$	0.0724	0.9964
83	$7.025 \cdot 10^{-8}$	$7.769 \cdot 10^{-7}$	$1.950 \cdot 10^{-3}$	0.0717	0.9965
84	$7.105 \cdot 10^{-8}$	$7.382 \cdot 10^{-7}$	$1.987 \cdot 10^{-3}$	0.07010	0.9965
85	$6.998 \cdot 10^{-8}$	$6.991 \cdot 10^{-7}$	$1.982 \cdot 10^{-3}$	0.0703	0.9966
86	$6.996 \cdot 10^{-8}$	$6.649 \cdot 10^{-7}$	$2.013 \cdot 10^{-3}$	0.0696	0.9967
87	$6.991 \cdot 10^{-8}$	$6.515 \cdot 10^{-7}$	$6.041 \cdot 10^{-4}$	0.0690	0.9968
88	$6.973 \cdot 10^{-8}$	$6.186 \cdot 10^{-7}$	$5.967 \cdot 10^{-4}$	0.0683	0.9968
89	$6.931 \cdot 10^{-8}$	$6.006 \cdot 10^{-7}$	$5.825 \cdot 10^{-4}$	0.0677	0.9969
90	$6.879 \cdot 10^{-8}$	$5.735 \cdot 10^{-7}$	$5.499 \cdot 10^{-4}$	0.0671	0.9970
91	$6.816 \cdot 10^{-8}$	$5.536 \cdot 10^{-7}$	$1.626 \cdot 10^{-4}$	0.0665	0.9970
92	$6.738 \cdot 10^{-8}$	$5.221 \cdot 10^{-7}$	$1.582 \cdot 10^{-4}$	0.0658	0.9971
93	$6.608 \cdot 10^{-8}$	$5.056 \cdot 10^{-7}$	$1.585 \cdot 10^{-4}$	0.0653	0.9971
94	$6.371 \cdot 10^{-8}$	$4.885 \cdot 10^{-7}$	$1.590 \cdot 10^{-4}$	0.0647	0.9972
95	$6.449 \cdot 10^{-8}$	$4.687 \cdot 10^{-7}$	$1.562 \cdot 10^{-4}$	0.0641	0.9972
96	$6.378 \cdot 10^{-8}$	$4.592 \cdot 10^{-7}$	$1.568 \cdot 10^{-4}$	0.0636	0.9973
97	$6.433 \cdot 10^{-8}$	$4.138 \cdot 10^{-7}$	$1.572 \cdot 10^{-4}$	0.0631	0.9973
98	$6.321 \cdot 10^{-8}$	$3.922 \cdot 10^{-7}$	$1.239 \cdot 10^{-4}$	0.0626	0.9974
99	$6.184 \cdot 10^{-8}$	$3.831 \cdot 10^{-7}$	$1.233 \cdot 10^{-4}$	0.0620	0.9974
100	$6.222 \cdot 10^{-8}$	$3.719 \cdot 10^{-7}$	$1.152 \cdot 10^{-4}$	0.0615	0.9975

Table A.2: S1E2 M_m prediction consistency for 1-100 queries available for training over 20,000 experimental runs.

S2E2 M_m GPR	Prediction consistency MSE		
	Low	Mean	High
1	$2.005 \cdot 10^{-1}$	$3.385 \cdot 10^{-1}$	$4.492 \cdot 10^{-1}$
2	$5.076 \cdot 10^{-3}$	$2.110 \cdot 10^{-1}$	$4.495 \cdot 10^{-1}$
3	$6.874 \cdot 10^{-4}$	$1.211 \cdot 10^{-1}$	$4.384 \cdot 10^{-1}$
4	$2.280 \cdot 10^{-4}$	$6.984 \cdot 10^{-2}$	$3.441 \cdot 10^{-1}$
5	$8.985 \cdot 10^{-5}$	$3.705 \cdot 10^{-2}$	$3.356 \cdot 10^{-1}$
6	$1.616 \cdot 10^{-5}$	$1.940 \cdot 10^{-2}$	$3.119 \cdot 10^{-1}$
7	$4.979 \cdot 10^{-6}$	$1.115 \cdot 10^{-2}$	$3.041 \cdot 10^{-1}$
8	$2.847 \cdot 10^{-6}$	$5.530 \cdot 10^{-3}$	$2.366 \cdot 10^{-1}$
9	$1.068 \cdot 10^{-7}$	$3.935 \cdot 10^{-3}$	$1.819 \cdot 10^{-1}$
10	$6.445 \cdot 10^{-8}$	$2.067 \cdot 10^{-3}$	$1.255 \cdot 10^{-1}$
11	$2.027 \cdot 10^{-8}$	$1.545 \cdot 10^{-3}$	$1.282 \cdot 10^{-1}$
12	$1.405 \cdot 10^{-8}$	$1.001 \cdot 10^{-3}$	$6.794 \cdot 10^{-2}$
13	$9.169 \cdot 10^{-9}$	$6.986 \cdot 10^{-4}$	$4.725 \cdot 10^{-2}$
14	$1.313 \cdot 10^{-8}$	$5.041 \cdot 10^{-4}$	$3.943 \cdot 10^{-2}$
15	$1.152 \cdot 10^{-8}$	$3.564 \cdot 10^{-4}$	$3.388 \cdot 10^{-2}$
16	$7.728 \cdot 10^{-9}$	$2.564 \cdot 10^{-4}$	$1.935 \cdot 10^{-2}$
17	$8.075 \cdot 10^{-9}$	$1.887 \cdot 10^{-4}$	$1.847 \cdot 10^{-2}$
18	$1.066 \cdot 10^{-8}$	$1.374 \cdot 10^{-4}$	$1.189 \cdot 10^{-2}$
19	$1.496 \cdot 10^{-8}$	$1.008 \cdot 10^{-4}$	$1.134 \cdot 10^{-2}$
20	$1.211 \cdot 10^{-8}$	$7.555 \cdot 10^{-5}$	$1.101 \cdot 10^{-2}$
21	$7.585 \cdot 10^{-9}$	$5.526 \cdot 10^{-5}$	$1.098 \cdot 10^{-2}$
22	$1.336 \cdot 10^{-8}$	$4.040 \cdot 10^{-5}$	$1.097 \cdot 10^{-2}$
23	$1.444 \cdot 10^{-8}$	$2.978 \cdot 10^{-5}$	$1.083 \cdot 10^{-2}$
24	$1.891 \cdot 10^{-8}$	$2.230 \cdot 10^{-5}$	$9.711 \cdot 10^{-3}$
25	$1.898 \cdot 10^{-8}$	$1.640 \cdot 10^{-5}$	$9.567 \cdot 10^{-3}$
26	$1.892 \cdot 10^{-8}$	$1.301 \cdot 10^{-5}$	$9.480 \cdot 10^{-3}$
27	$1.984 \cdot 10^{-8}$	$9.643 \cdot 10^{-6}$	$9.260 \cdot 10^{-3}$
28	$2.091 \cdot 10^{-8}$	$7.156 \cdot 10^{-6}$	$9.257 \cdot 10^{-3}$
29	$2.001 \cdot 10^{-8}$	$5.595 \cdot 10^{-6}$	$9.192 \cdot 10^{-3}$
30	$1.433 \cdot 10^{-8}$	$4.412 \cdot 10^{-6}$	$9.153 \cdot 10^{-3}$
31	$1.430 \cdot 10^{-8}$	$3.793 \cdot 10^{-6}$	$9.077 \cdot 10^{-3}$
32	$1.424 \cdot 10^{-8}$	$2.750 \cdot 10^{-6}$	$5.661 \cdot 10^{-3}$
33	$1.421 \cdot 10^{-8}$	$2.245 \cdot 10^{-6}$	$3.756 \cdot 10^{-3}$
34	$2.781 \cdot 10^{-8}$	$1.686 \cdot 10^{-6}$	$3.751 \cdot 10^{-3}$
35	$2.781 \cdot 10^{-8}$	$1.407 \cdot 10^{-6}$	$3.696 \cdot 10^{-3}$
36	$2.796 \cdot 10^{-8}$	$1.089 \cdot 10^{-6}$	$9.599 \cdot 10^{-4}$
37	$3.270 \cdot 10^{-8}$	$8.619 \cdot 10^{-7}$	$4.106 \cdot 10^{-4}$
38	$3.271 \cdot 10^{-8}$	$7.123 \cdot 10^{-7}$	$4.026 \cdot 10^{-4}$
39	$3.304 \cdot 10^{-8}$	$6.235 \cdot 10^{-7}$	$3.854 \cdot 10^{-4}$
40	$3.304 \cdot 10^{-8}$	$5.125 \cdot 10^{-7}$	$3.600 \cdot 10^{-4}$

Continued on next page

Prediction consistency MSE

Queries	Low	Mean	High
41	$3.364 \cdot 10^{-8}$	$4.361 \cdot 10^{-7}$	$3.478 \cdot 10^{-4}$
42	$3.363 \cdot 10^{-8}$	$2.539 \cdot 10^{-7}$	$3.487 \cdot 10^{-4}$
43	$3.364 \cdot 10^{-8}$	$2.390 \cdot 10^{-7}$	$2.353 \cdot 10^{-4}$
44	$3.390 \cdot 10^{-8}$	$2.076 \cdot 10^{-7}$	$2.330 \cdot 10^{-4}$
45	$3.390 \cdot 10^{-8}$	$1.582 \cdot 10^{-7}$	$2.338 \cdot 10^{-4}$
46	$3.389 \cdot 10^{-8}$	$1.373 \cdot 10^{-7}$	$2.336 \cdot 10^{-4}$
47	$3.389 \cdot 10^{-8}$	$1.127 \cdot 10^{-7}$	$1.766 \cdot 10^{-4}$
48	$3.389 \cdot 10^{-8}$	$1.006 \cdot 10^{-7}$	$1.766 \cdot 10^{-4}$
49	$3.392 \cdot 10^{-8}$	$9.007 \cdot 10^{-8}$	$1.703 \cdot 10^{-4}$
50	$3.392 \cdot 10^{-8}$	$8.467 \cdot 10^{-8}$	$1.239 \cdot 10^{-4}$
51	$3.392 \cdot 10^{-8}$	$7.928 \cdot 10^{-8}$	$4.240 \cdot 10^{-5}$
52	$3.392 \cdot 10^{-8}$	$7.670 \cdot 10^{-8}$	$4.233 \cdot 10^{-5}$
53	$3.390 \cdot 10^{-8}$	$6.734 \cdot 10^{-8}$	$4.235 \cdot 10^{-5}$
54	$3.391 \cdot 10^{-8}$	$6.450 \cdot 10^{-8}$	$4.237 \cdot 10^{-5}$
55	$3.390 \cdot 10^{-8}$	$6.125 \cdot 10^{-8}$	$3.217 \cdot 10^{-5}$
56	$3.390 \cdot 10^{-8}$	$6.085 \cdot 10^{-8}$	$3.213 \cdot 10^{-5}$
57	$3.390 \cdot 10^{-8}$	$5.278 \cdot 10^{-8}$	$1.848 \cdot 10^{-5}$
58	$3.390 \cdot 10^{-8}$	$4.944 \cdot 10^{-8}$	$1.842 \cdot 10^{-5}$
59	$3.390 \cdot 10^{-8}$	$4.731 \cdot 10^{-8}$	$1.843 \cdot 10^{-5}$
60	$3.401 \cdot 10^{-8}$	$4.493 \cdot 10^{-8}$	$1.708 \cdot 10^{-5}$
61	$3.401 \cdot 10^{-8}$	$4.452 \cdot 10^{-8}$	$1.702 \cdot 10^{-5}$
62	$3.401 \cdot 10^{-8}$	$4.382 \cdot 10^{-8}$	$1.697 \cdot 10^{-5}$
63	$3.400 \cdot 10^{-8}$	$4.311 \cdot 10^{-8}$	$1.688 \cdot 10^{-5}$
64	$3.401 \cdot 10^{-8}$	$4.300 \cdot 10^{-8}$	$1.685 \cdot 10^{-5}$
65	$3.403 \cdot 10^{-8}$	$4.267 \cdot 10^{-8}$	$1.685 \cdot 10^{-5}$
66	$3.403 \cdot 10^{-8}$	$4.085 \cdot 10^{-8}$	$1.292 \cdot 10^{-5}$
67	$3.403 \cdot 10^{-8}$	$4.074 \cdot 10^{-8}$	$1.289 \cdot 10^{-5}$
68	$3.413 \cdot 10^{-8}$	$3.818 \cdot 10^{-8}$	$1.285 \cdot 10^{-5}$
69	$3.413 \cdot 10^{-8}$	$3.802 \cdot 10^{-8}$	$1.283 \cdot 10^{-5}$
70	$3.413 \cdot 10^{-8}$	$3.771 \cdot 10^{-8}$	$5.091 \cdot 10^{-6}$
71	$3.415 \cdot 10^{-8}$	$3.767 \cdot 10^{-8}$	$5.091 \cdot 10^{-6}$
72	$3.421 \cdot 10^{-8}$	$3.729 \cdot 10^{-8}$	$5.083 \cdot 10^{-6}$
73	$3.424 \cdot 10^{-8}$	$3.686 \cdot 10^{-8}$	$2.228 \cdot 10^{-6}$
74	$3.429 \cdot 10^{-8}$	$3.680 \cdot 10^{-8}$	$2.190 \cdot 10^{-6}$
75	$3.429 \cdot 10^{-8}$	$3.674 \cdot 10^{-8}$	$2.188 \cdot 10^{-6}$
76	$3.429 \cdot 10^{-8}$	$3.673 \cdot 10^{-8}$	$2.186 \cdot 10^{-6}$
77	$3.429 \cdot 10^{-8}$	$3.671 \cdot 10^{-8}$	$2.146 \cdot 10^{-6}$
78	$3.425 \cdot 10^{-8}$	$3.661 \cdot 10^{-8}$	$2.139 \cdot 10^{-6}$
79	$3.427 \cdot 10^{-8}$	$3.660 \cdot 10^{-8}$	$2.137 \cdot 10^{-6}$
80	$3.426 \cdot 10^{-8}$	$3.657 \cdot 10^{-8}$	$2.138 \cdot 10^{-6}$

Continued on next page

Prediction consistency MSE

Queries	Low	Mean	High
81	$3.426 \cdot 10^{-8}$	$3.654 \cdot 10^{-8}$	$2.141 \cdot 10^{-6}$
82	$3.426 \cdot 10^{-8}$	$3.652 \cdot 10^{-8}$	$1.856 \cdot 10^{-6}$
83	$3.425 \cdot 10^{-8}$	$3.652 \cdot 10^{-8}$	$1.854 \cdot 10^{-6}$
84	$3.425 \cdot 10^{-8}$	$3.651 \cdot 10^{-8}$	$1.709 \cdot 10^{-6}$
85	$3.425 \cdot 10^{-8}$	$3.651 \cdot 10^{-8}$	$1.710 \cdot 10^{-6}$
86	$3.428 \cdot 10^{-8}$	$3.648 \cdot 10^{-8}$	$1.708 \cdot 10^{-6}$
87	$3.428 \cdot 10^{-8}$	$3.647 \cdot 10^{-8}$	$1.710 \cdot 10^{-6}$
88	$3.428 \cdot 10^{-8}$	$3.646 \cdot 10^{-8}$	$1.698 \cdot 10^{-6}$
89	$3.428 \cdot 10^{-8}$	$3.644 \cdot 10^{-8}$	$1.688 \cdot 10^{-6}$
90	$3.428 \cdot 10^{-8}$	$3.644 \cdot 10^{-8}$	$1.358 \cdot 10^{-6}$
91	$3.428 \cdot 10^{-8}$	$3.644 \cdot 10^{-8}$	$1.358 \cdot 10^{-6}$
92	$3.433 \cdot 10^{-8}$	$3.643 \cdot 10^{-8}$	$1.356 \cdot 10^{-6}$
93	$3.433 \cdot 10^{-8}$	$3.641 \cdot 10^{-8}$	$1.326 \cdot 10^{-6}$
94	$3.433 \cdot 10^{-8}$	$3.640 \cdot 10^{-8}$	$1.326 \cdot 10^{-6}$
95	$3.433 \cdot 10^{-8}$	$3.641 \cdot 10^{-8}$	$1.325 \cdot 10^{-6}$
96	$3.433 \cdot 10^{-8}$	$3.640 \cdot 10^{-8}$	$1.319 \cdot 10^{-6}$
97	$3.433 \cdot 10^{-8}$	$3.640 \cdot 10^{-8}$	$1.297 \cdot 10^{-6}$
98	$3.433 \cdot 10^{-8}$	$3.640 \cdot 10^{-8}$	$1.297 \cdot 10^{-6}$
99	$3.433 \cdot 10^{-8}$	$3.640 \cdot 10^{-8}$	$1.280 \cdot 10^{-6}$
100	$3.433 \cdot 10^{-8}$	$3.64 \cdot 10^{-8}$	$1.280 \cdot 10^{-6}$

Table A.3: Wasserstein-1 distance between \mathcal{D}_t and S1E1 \mathcal{D}_m and S1E2 \mathcal{D}_m respectively. 1-100 queries averaged over 20,000 experimental runs.

Dataset similarity	Wasserstein-1 distance to M_t		
	Queries	S1E1 \mathcal{D}_m	S1E2 \mathcal{D}_m
1	1.0000000	1.0000000	0.0
2	0.5156710	0.5156710	0.0
3	0.3607940	0.3607940	0.0
4	0.2793900	0.2793900	0.0
5	0.2281590	0.2281590	0.0
6	0.1929180	0.1929180	0.0
7	0.1683090	0.1683090	0.0
8	0.1485550	0.1485550	0.0
9	0.1329870	0.1329870	0.0
10	0.1198150	0.1198150	0.0
11	0.1098180	0.1098180	0.0
12	0.1012130	0.1012130	0.0
13	0.0937596	0.0937596	0.0
14	0.0874530	0.0874530	0.0
15	0.0819789	0.0819789	0.0
16	0.0769470	0.0769470	0.0
17	0.0727037	0.0727037	0.0
18	0.0686272	0.0686272	0.0
19	0.0650214	0.0650214	0.0
20	0.0619327	0.0619327	0.0
21	0.0593517	0.0593517	0.0
22	0.0568153	0.0568153	0.0
23	0.0544815	0.0544815	0.0
24	0.0521148	0.0521148	0.0
25	0.0497541	0.0497541	0.0
26	0.0483167	0.0483167	0.0
27	0.0468191	0.0468191	0.0
28	0.0451626	0.0451626	0.0
29	0.0437123	0.0437123	0.0
30	0.0422153	0.0422153	0.0
31	0.0409558	0.0409558	0.0
32	0.0397063	0.0397063	0.0
33	0.0385050	0.0385050	0.0
34	0.0373522	0.0373522	0.0
35	0.0364041	0.0364041	0.0
36	0.0354745	0.0354745	0.0
37	0.0346382	0.0346382	0.0
38	0.0337370	0.0337370	0.0
39	0.0329123	0.0329123	0.0
40	0.0320725	0.0320725	0.0

Continued on next page

Prediction consistency MSE

Queries	S1E1 D_m	S1E2 D_m	Difference
41	0.0313497	0.0313497	0.0
42	0.0305852	0.0305852	0.0
43	0.0299490	0.0299490	0.0
44	0.0293046	0.0293046	0.0
45	0.0286362	0.0286362	0.0
46	0.0280102	0.0280102	0.0
47	0.0274254	0.0274254	0.0
48	0.0268390	0.0268390	0.0
49	0.0263182	0.0263182	0.0
50	0.0253286	0.0253286	0.0
51	0.0253537	0.0253537	0.0
52	0.0248460	0.0248460	0.0
53	0.0244455	0.0244455	0.0
54	0.0239910	0.0239910	0.0
55	0.0235424	0.0235424	0.0
56	0.0231522	0.0231522	0.0
57	0.0227902	0.0227902	0.0
58	0.0224492	0.0224492	0.0
59	0.0220476	0.0220476	0.0
60	0.0216520	0.0216520	0.0
61	0.0212912	0.0212912	0.0
62	0.0209079	0.0209079	0.0
63	0.0205859	0.0205859	0.0
64	0.0202509	0.0202509	0.0
65	0.0199425	0.0199425	0.0
66	0.0196609	0.0196609	0.0
67	0.0194129	0.0194129	0.0
68	0.0190957	0.0190957	0.0
69	0.0188418	0.0188418	0.0
70	0.0185817	0.0185817	0.0
71	0.0183350	0.0183350	0.0
72	0.0181108	0.0181108	0.0
73	0.0178801	0.0178801	0.0
74	0.0176533	0.0176533	0.0
75	0.0173657	0.0173657	0.0
76	0.0171946	0.0171946	0.0
77	0.0170030	0.0170030	0.0
78	0.0167665	0.0167665	0.0
79	0.0165567	0.0165567	0.0
80	0.0163410	0.0163410	0.0

Continued on next page

Prediction consistency MSE

Queries	S1E1 \mathcal{D}_m	S1E2 \mathcal{D}_m	Difference
81	0.0161476	0.0161476	0.0
82	0.0159438	0.0159438	0.0
83	0.0157597	0.0157597	0.0
84	0.0155595	0.0155595	0.0
85	0.0153851	0.0153851	0.0
86	0.0152167	0.0152167	0.0
87	0.0150641	0.0150641	0.0
88	0.0149002	0.0149002	0.0
89	0.0147555	0.0147555	0.0
90	0.0145972	0.0145972	0.0
91	0.0144384	0.0144384	0.0
92	0.0142891	0.0142891	0.0
93	0.0141265	0.0141265	0.0
94	0.0139713	0.0139713	0.0
95	0.0138501	0.0138501	0.0
96	0.0137141	0.0137141	0.0
97	0.0135673	0.0135673	0.0
98	0.0134396	0.0134396	0.0
99	0.0133127	0.0133127	0.0
100	0.0130607	0.0130607	0.0



B Scenario 2: Experimental data

The results presented in Chapter 8 are a condensate of the experimental data produced for this work. More detailed data for monitor performance and other metrics are presented as an addendum.

Table B.1: S2E1-E2 M_m prediction consistency and model rank similarity to M_t for each additional 1,000 queries available for training.

Queries	S2E1 M_m ResNet-18		S2E1 M_m ResNet-152	
	Accuracy (%)	Rank score	Accuracy (%)	Rank score
1,000	39.19	7.930	29.22	5.419
2,000	43.71	7.142	31.26	4.950
3,000	50.29	7.698	26.30	5.121
4,000	52.54	7.948	27.42	4.905
5,000	55.46	7.746	25.06	1.143
6,000	57.57	7.434	37.92	5.352
7,000	59.85	7.614	44.74	5.962
8,000	61.96	7.682	33.83	5.143
9,000	62.59	7.580	42.71	5.865
10,000	63.76	7.621	43.59	6.286
11,000	63.87	7.740	42.49	5.822
12,000	64.89	7.135	47.58	3.191
13,000	65.25	7.562	44.64	5.408
14,000	66.78	7.575	50.40	6.152
15,000	67.43	7.044	49.80	5.988
16,000	67.50	7.944	30.17	2.730
17,000	68.53	7.571	52.65	6.470
18,000	68.45	7.715	47.98	6.229
19,000	68.04	7.527	50.30	6.438
20,000	67.93	7.599	49.70	6.310

Table B.2: S2E3-E4 M_m prediction consistency and model rank similarity to M_t for each additional 1,000 queries available for training.

Queries	S2E3 M_m VGG-11		S2E4 M_m VGG-19	
	Accuracy (%)	Rank score	Accuracy (%)	Rank score
1,000	43.08	5.526	36.90	4.209
2,000	51.67	5.845	42.81	4.209
3,000	57.00	6.021	49.73	4.555
4,000	61.75	6.246	53.71	5.110
5,000	61.32	6.466	58.84	5.458
6,000	63.64	6.731	62.91	5.076
7,000	64.71	6.712	62.75	5.251
8,000	66.57	6.436	65.65	5.636
9,000	67.14	6.818	66.21	5.641
10,000	67.53	7.361	67.63	5.842
11,000	68.05	6.792	67.94	6.044
12,000	67.38	7.179	68.04	5.975
13,000	68.16	7.222	68.84	6.356
14,000	68.99	7.543	69.82	6.177
15,000	69.22	6.828	67.55	5.919
16,000	69.93	7.072	70.57	6.028
17,000	69.70	7.152	68.34	5.871
18,000	69.60	6.810	70.53	6.011
19,000	71.37	6.900	71.27	6.287
20,000	70.88	7.623	72.05	6.260

Table B.3: S2E5-E6 M_m prediction consistency and model rank similarity to M_t for each additional 1,000 queries available for training.

Queries	S2E5 M_m DenseNet-161		S2E6 M_m DenseNet-121	
	Accuracy (%)	Rank score	Accuracy (%)	Rank score
1,000	40.89	7.862	39.06	7.670
2,000	45.29	7.221	43.05	7.289
3,000	50.15	7.567	48.34	7.624
4,000	50.70	7.529	52.24	7.755
5,000	53.24	7.677	53.48	7.645
6,000	56.49	7.704	57.55	7.702
7,000	56.78	7.735	59.02	7.795
8,000	59.46	7.264	59.66	7.750
9,000	63.43	7.489	61.05	7.549
10,000	61.32	7.294	62.44	7.398
11,000	64.48	7.236	63.98	7.744
12,000	65.15	7.646	63.52	7.473
13,000	63.70	7.435	65.66	7.649
14,000	65.59	7.741	65.71	7.247
15,000	65.37	7.430	65.96	7.497
16,000	67.25	7.442	66.72	7.628
17,000	67.61	7.532	67.82	7.387
18,000	67.33	7.739	68.07	7.640
19,000	67.75	7.857	68.34	7.656
20,000	67.30	7.433	68.72	7.352