Challenging the Kryptonite-n Dataset and its Claims on ML Limits

Faiza Jalil, Malika Dixit, Leili Barekatain, Yanjinlkham Temuujin [1]

Abstract

This paper challenges the claims made about the Kryptonite-n dataset, which suggests that machine learning models are limited in their predictive capabilities. We argue that these reported limitations are not inherent to the models but rather stem from methodological flaws in the original study. We demonstrate that the reported performance barriers can be overcome using more suitable modeling techniques, including Extra Trees, Neural Networks, and Residual Networks.

1. Introduction

Machine learning (ML) has become an indispensable tool across various industries. Central to ML's progress are **challenge datasets**, specifically designed to push the boundaries of existing algorithms. These datasets are crucial for identifying the limitations of current models, fostering innovation, and ensuring that ML advancements are both robust and generalizable.

In response to perceived overinflated claims about ML capabilities, the paper *"Kryptonite-n: A Simple End to Machine Learning Hype?"* introduces **Kryptonite-n**, a binary classification dataset with an n-dimensional feature space. The authors contend that Kryptonite-n challenges foundational theoretical claims, such as the Universal Function Approximation Theorem, by demonstrating that even advanced ML models achieve only 60% accuracy on this task. If validated, this finding significantly undermines the credibility of current ML advancements and the narratives surrounding artificial general intelligence.

This paper aims to critically evaluate and refute the claims presented *Kryptonite-n* study. We argue that the reported ML model failures result from flawed experimental methodologies and theoretical oversights rather than inherent model limitations. We reassess the experimental setups and theoretical assumptions, demonstrating that alternative models such as **Extra Trees** and **Neural Networks** effectively address the Kryptonite-n challenge. Consequently, this study addresses the following research questions:

RQ1: Can logistic regression models achieve higher accuracy on Kryptonite-n with improved preprocessing and hyperparameter tuning?

RQ2: Which alternative models can effectively solve the Kryptonite-n tasks, and what enables their success?

Contrary to the *Kryptonite-n* study, our experiments demonstrate that well-configured ML models achieve significantly higher accuracy on the Kryptonite-n dataset. Specifically, logistic regression reaches the target accuracy at $n = 9$ with proper preprocessing and hyperparameter tuning. Extra Trees and Neural Networks meet target accuracy thresholds up to $n = 15$, while ResNets successfully handle $n = 18$ and $n = 24$. These findings suggest that the original study's limitations arise from suboptimal experimental setups and theoretical oversights, rather than inherent flaws in the models or the dataset.

2. Methodology

2.1. Methodological Critique

2.1.1. Logistic Regression

The original study employs logistic regression with polynomial basis expansion, achieving approximately 60% accuracy on Kryptonite-9, which falls short of the targeted performance. Notably, the authors opt for the Stochastic Average Gradient (SAG) optimizer instead of methods such as Stochastic Gradient Descent (SGD) or Adam. Unlike SGD, which updates parameters based on a single sample's gradient, SAG reduces gradient variance by maintaining an average of gradients from all samples (Schmidt et al., 2017) (Definition in Appendix A.1), potentially leading to faster convergence when the loss function's Hessian is well-conditioned. However, this methodological choice may have inadvertently contributed to the suboptimal accuracy observed. The convergence rate of the SAG optimizer is heavily influenced by the condition number $\kappa(H)$ of the

[1]Department of Computing, Imperial College London, London, United Kingdom. Correspondence to: Leili Barekatain <leili.barekatain24@imperial.ac.uk>, Malika Dixit <malika.dikshit24@imperial.ac.uk>, Yanjinlkham Temuujin <yanjinlkham.temuujin24@imperial.ac.uk>.

Coursework. For review by instructors of Mathematics for Machine Learning (70015) . Do not distribute.

Hessian matrix H, defined as:

$$\kappa(H) = \frac{\lambda_{\max}(H)}{\lambda_{\min}(H)}, \tag{1}$$

where $\lambda_{\max}(H)$ and $\lambda_{\min}(H)$ are the largest and smallest eigenvalues of H, respectively. A high condition number indicates a poorly conditioned Hessian matrix, which significantly slows the convergence of SAG. In logistic regression with L_2 regularization, the Hessian matrix H is given by:

$$H = X^\top D X + \lambda I, \tag{2}$$

where X is the design matrix with polynomial feature expansions, D is a diagonal matrix with entries $D_{ii} = \sigma(x_i^\top \theta)(1 - \sigma(x_i^\top \theta))$, and λ is the regularization parameter. Polynomial basis expansion increases feature space dimensionality, leading to features with vastly different scales. This disparity causes the eigenvalues of H to spread widely, inflating the condition number $\kappa(H)$ and rendering the Hessian poorly conditioned. Dominance of higher-degree terms creates an imbalance, worsening the optimization landscape, slowing SAG convergence, and undermining performance. The lack of feature scaling exacerbates these issues, allowing polynomial features to vary widely in magnitude and further inflating the Hessian's eigenvalue spread, resulting in suboptimal convergence.

To achieve efficient convergence, we impose a target condition number $\kappa_{\text{target}} = 1$ on the Hessian matrix H. This requirement leads to the inequality (detailed in Appendix A.2)

$$C \leq \frac{1}{\lambda_{\max}(X^\top D X)}. \tag{3}$$

This upper bound on C ensures that the condition number of H remains close to 1, optimizing the convergence rate of the SAG optimizer.

Our experiments aim to demonstrate that, through appropriate feature scaling and selection of the regularization parameter C, the polynomial logistic regression model performs effectively on smaller datasets.

2.1.2. Using GPT Embeddings

The use of GPT embeddings for structured data classification, as applied in the original study, is methodologically flawed. GPT models, designed for NLP, excel at capturing linguistic patterns but lack the inherent capacity to handle structured numerical data. Converting structured input $\mathbf{x} \in \mathbb{R}^n$ into text prompts introduces ambiguities that GPT embeddings are not optimized to resolve.

2.1.3. Justification for Exploring Alternative Models

Polynomial basis expansions capture complex functions through higher-order feature interactions, but as the polynomial degree p increases, the number of basis terms $N = \binom{d+p}{p}$ grows combinatorially with feature dimension d. While the Stone-Weierstrass Theorem (Stone, 1948) ensures that polynomials can approximate any continuous function over a compact domain, achieving high precision in high-dimensional spaces requires large p, causing rapid growth in N. This combinatorial explosion imposes significant computational demands and increases the risk of overfitting.

Therefore, although polynomial expansions may be feasible for logistic regression on smaller datasets, they become impractical as p and d increase. This motivates exploring alternative models that capture complex feature interactions without the scalability challenges of high-dimensional polynomial expansions.

2.2. Extremely Randomized Trees

Extra Trees are ensemble methods for classification and regression (Geurts et al., 2006), introducing additional randomness by selecting both split points and features for each split. This approach enhances model diversity and reduces variance compared to traditional Random Forests. We integrate Extra Trees with polynomial feature expansion, enabling the capture of non-linear interactions through threshold-based partitioning, which partially mitigates the curse of dimensionality. The ensemble nature and inherent randomization of Extra Trees also reduce the risk of overfitting, making them more robust to the complexities introduced by polynomial expansions than logistic regression. Consequently, Extra Trees with polynomial feature expansion will outperform logistic regression on smaller-dimensional datasets.

However, as discussed in Section 2.1.3, the combinatorial growth of polynomial features poses challenges for larger datasets. In such cases, Extra Trees, while more powerful than logistic regression, encounter significant computational demands and an increased risk of overfitting due to the exponential increase in feature dimensionality. Below, we outline the mathematical foundation of Extra Trees, including key hyperparameters that influence their performance.

2.2.1. Mathematical Basis:Extra Trees

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ with features $x_i \in \mathbb{R}^d$ and corresponding target values $y_i \in \mathbb{R}$ (or $y_i \in \{1, \ldots, C\}$ for classification), Extra Trees aim to learn a function $f(x)$ to predict y.

The model constructs an ensemble of T independent decision trees $\{f_t(x)\}_{t=1}^T$, and the final prediction $\hat{y}$ is obtained by averaging the outputs of the trees by majority voting for classification tasks:

$$\hat{y} = \text{mode}\{f_t(x)\}_{t=1}^T \quad \text{(classification)}. \tag{4}$$

Each decision tree in Extra Trees is built using a subset of the data, but unlike Random Forests, Extra Trees do not use bootstrapped samples. Instead, they use the entire dataset $\mathcal{D}$ to build each tree. The key differences lie in the selection of splitting features and thresholds:

Step 1: Random Feature Selection
For each split in the tree, a subset of K features is randomly selected from the total d features. This introduces randomness and controls overfitting. Let $S \subseteq \{1, \ldots, d\}$ denote the set of randomly selected features.

Step 2: Random Threshold Selection
For each selected feature $x_j \in S$, rather than computing the optimal split point as in traditional decision trees, Extra Trees randomly choos a split threshold τ_j from the range of values observed in the dataset for that feature:

$$\tau_j \sim \text{Uniform}(\min(x_j), \max(x_j)). \tag{5}$$

The split that maximizes a predefined criterion (e.g., Gini index for classification or variance reduction for regression) is chosen.

Step 3: Recursive Splitting
This process is recursively applied to each node until a stopping criterion is met, such as reaching a minimum number of samples per node n_{min}.

Classification: The Gini impurity is used to evaluate splits:

$$G(\mathcal{D}) = 1 - \sum_{c=1}^{C} p_c^2, \tag{6}$$

where p_c is the proportion of class c samples in dataset $\mathcal{D}$. Some of the key hyperparameters of the model include `n_estimators` (number of trees in the ensemble), `min_samples_split` (minimum number of samples required to split a node), `min_samples_leaf` (minimum number of samples required to be a leaf node).

2.3. Neural Networks

Another alternative model we explore is neural networks, which capture complex, non-linear patterns through layers of neurons performing linear transformations followed by non-linear activations:

$$\mathbf{a} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{7}$$

where $\mathbf{W}$ represents weights, $\mathbf{b}$ biases, and σ is a non-linear activation function. Non-linear activations are crucial as they enable the network to learn complex patterns that multiple linear transformations alone cannot capture. The final layer produces output $\hat{\mathbf{y}}$ using a task-specific function - typically sigmoid for binary classification.

To measure the difference between the network's predictions $\hat{\mathbf{y}}$ and the actual labels $\mathbf{y}$, a loss function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ is defined. Binary Cross-Entropy (BCE) Loss is a common choice for binary classification. The network's parameters are optimized by minimizing the loss function using backpropagation (See A.3). The **Adam** (Adaptive Moment Estimation) optimizer combines the benefits of momentum and adaptive learning rates. By adjusting learning rates for each parameter and smoothing updates, Adam ensures faster convergence and robust training stability (Goodfellow et al., 2016). Detailed mathematical formulations are provided in the appendix A.4.

Key hyperparameters for neural networks include the **optimizer**, the **the learning rate** ((η), which controls the step size of each parameter update, with small values leading to slow convergence and large values potentially causing the model to miss the optimal solution by overshooting; **the batch size**, representing the number of samples processed before updating the model's parameters, where larger batch sizes provide more stable updates and smaller batches allow more frequent updates; and **regularization parameters** which introduce constraints to prevent overfitting.

2.3.1. Overfitting

Overfitting occurs in neural networks when the model learns not only the underlying patterns in the training data but also the noise and specific details that do not generalize to new data (Goodfellow et al., 2016). (See A.6 for more details.)

To address overfitting, early stopping is a practical technique. It controls the validation error during training and stops the optimization process when the validation error begins to increase.

2.3.2. Residual Networks

Residual Networks (ResNets) extend traditional neural networks by introducing residual connections, which enable the effective training of deeper architectures(He et al., 2016). ResNets are well-suited for high-dimensional datasets due to their capacity to learn complex representations, with key hyperparameters including the number of residual blocks, neurons per layer, and other neural network parameters. Detailed mathematical formulation are provided in the appendix A.9.

3. Experimental Design

3.1. Hypotheses

This study evaluates the following hypotheses:

H1: Logistic regression with polynomial feature expansion, combined with feature scaling and appropiate regularization, will improve accuracy on the Kryptonite dataset.
H2: The Extra Trees model will outperform logistic regression by capturing complex feature interactions and non-

linear relationships through its ensemble-based structure.
H3: Neural networks will achieve higher accuracy than logistic regression and Extra Trees. ResNets are expected to enhance performance on higher dimensional datasets.

3.2. Experimental Setup

The following experimental setup outlines the common methodology applied to both the Extra Trees and Neural Network models to address H2 and H3. The dataset was scaled using StandardScaler to standardize each feature. For each fold in cross-validation (CV), separate scalers were fitted on the training and validation sets to avoid data leakage, ensuring that the validation data remains unseen during training. Grid Search was employed for Extra Trees and Neural Networks to select the best model. The hyperparametrs inluded in the tuning process are presented in Sections 3.4 and 3.5.

3.2.1. Validity of Experiments

Cross Validation: A 5-fold cross-validation procedure was used to evaluate model performance. The dataset was divided into five distinct folds, with four used for training and one for validation in each iteration. To enhance **robustness**, the procedure was repeated for multiple random seeds, reducing the influence of any particular fold configuration. By aggregating results across these repetitions, we obtained a more reliable estimate of the models' generalization performance.

Concentration Inequalities and Risk Estimation

To validate the reliability of our experimental results, we apply Hoeffding's Inequality (Hoeffding, 1963) to establish statistical bounds on our risk estimates. For each dataset D_i with sample size n_i, the empirical risk $\hat{R}_{D_i}(f)$ represents the average loss of our model f over D_i. Hoeffding's Inequality bounds the probability that $\hat{R}_{D_i}(f)$ deviates from the true risk $R(f)$ by more than $\epsilon > 0$:

$$P\big(|\hat{R}_{D_i}(f) - R(f)| \geq \epsilon\big) \leq 2\exp(-2n_i\epsilon^2). \quad (8)$$

This ensures that our empirical risk reliably estimates the true risk within margin ϵ with high probability. We determine the minimum sample size required to achieve a desired confidence level $1-\delta$ and allowable error ϵ:

$$n_i \geq \frac{\ln(2/\delta)}{2\epsilon^2}. \quad (9)$$

3.2.2. Performance Metrics

Our primary performance metric is **classification accuracy**. Additionally, we evaluate sustainability using metrics calculated with the Python package `CodeCarbon` (Courty et al., 2024), including **Emissions** (the carbon footprint of model training in kilograms of CO_2-equivalents based on energy usage and carbon intensity), **Duration** (total training time in seconds), **Emissions Rate** (emissions normalized by training duration in kilograms per second), and **Energy Consumed** (total energy usage of CPU, GPU, and RAM during training in kilowatt-hours). These sustainability metrics are utilized in the analysis presented in Section 5.

3.3. Specific Setup: Logistic Regression

To test Hypothesis H_1, we conducted experiments on the *kryptonite-9* dataset using logistic regression with polynomial feature expansion. The dataset was split into training (60%), validation (20%), and test (20%) sets. Feature scaling was applied to mitigate convergence issues caused by varying feature magnitudes. We employed a fixed polynomial degree of 6 and varied the regularization parameter C across $\{0.001, 0.01, 0.1, 0.5, 0.85\}$, where $C = 0.85$ replicates the original study's value. While polynomial expansion effectively approximates complex functions, as discussed in Section 2, it becomes computationally infeasible for larger datasets.

3.4. Specific Setup: Extra Trees

Apart from the steps in Section 3.2, the steps of the experiment specific to Extra Trees are outlined below:

1. Data Preprocessing: In addition to the standard preprocessing described in 3.2, the Extra Trees model incorporated polynomial feature expansion. Degree 4 was selected as the control variable based on empirical performance (comparing model performance on a held out dataset for various degrees), balancing model accuracy and overfitting risk. After the polynomial transformation, the datasets were standardized with StandardScaler to ensure that all features were on a comparable scale.

2. Hyperparameter Tuning: Grid search was conducted specifically to optimize the Extra Trees model's parameters, focusing on the number of trees in the forest, the minimum number of samples required to split an internal node, and the minimum number of samples required at a leaf node.

3.5. Specific Setup: Neural Networks

Apart from the steps in Section 3.2, the steps of the experiment specific to Neural Networks are outlined below:

1. Model Architecture: Simple Neural Networks The neural network is structured with an input layer matching the feature dimensions of the dataset, followed by two hidden layers containing 128 and 64 neurons respectively. ReLU activation is applied to each hidden layer, and a final output layer with a sigmoid activation produces probabilities for binary classification.

2. Hyperparameter Settings: Key hyperparameters for the neural network include learning rate, optimizer, and mini-batches size. The hyperparameters were selected using grid search to find the optimal configuration for the model, and the results are shown in Section 4. The model is trained for a maximum of 100 epochs, with early stopping terminating training if the validation loss does not improve for 10 consecutive epochs in order to prevent overfitting.

3. Deep ResNet Architecture/Training Settings:
To address $n = 18$ and $n = 24$, we employ a ResNet architecture starting with a fully connected layer that maps input features to a 512-dimensional space, followed by four residual blocks. Each residual block comprises two fully connected layers with 512 units, batch normalization, and ReLU activation functions, with a dropout rate of 0.5 applied afterward to prevent overfitting. We trained models on $n = 18$ for 500 epochs with a batch size of 256 and on $n = 24$ for 80 epochs with a batch size of 1024, using the Adam optimizer with a learning rate of 0.001. These hyperparameters were not fully optimized, as model performance is highly sensitive to hyperparameter settings, and further optimization is beyond the scope of this work.

4. Experimental Results

4.1. H1: Logistic Regression

In Figure A.7, we observe that logistic regression with feature scaling achieves significantly higher accuracy (for all C) than the 60% reported in the original paper, thereby validating our hypothesis (**H1**). Additionally, we find that the regularization parameter C directly affects the condition number of the Hessian matrix H; specifically, as the condition number approaches 1, test accuracy increases, reaching the target accuracy of 95% when $C = 0.01$.

4.2. H2: Extra Trees

From the results of Grid Search, we determined that the optimal hyperparameters were `n_estimators` = 200 and `min_samples_leaf` = 2 for all datasets. For $n = 9$, the optimal value for `min_samples_split` was 3, whereas for $n = 12$ and $n = 15$, the optimal value was 2.

Random Seed	n = 9	n = 12	n = 15
42	95.8±0.003	85.9±0.008	52.48±0.006
99	95.8±0.004	85.4±0.02	52.35±0.0075
123	95.8±0.003	86.33±0.0229	52.29±0.0036

Table 1. Accuracies of the Best Extra Tree Model across diff. seeds

4.2.1. Results of the Best Model

We performed a cross-validation experiment using three different random seeds: 42, 99, and 123. For each seed, we conducted 5-fold cross-validation with the best model identified through hyperparamter tuning and recorded the mean accuracy and standard deviation of the model's performance. Table 1 present the results. The results indicate that the model achieves a high and consistent accuracy above the desired boundary, across different random seeds, with mean accuracies around 95.9% only for $n = 9$. However, for $n = 12$ and $n = 15$, the model accuracies do not cross the required threshold.

4.3. H3: Neural Networks

We chose the **Adam optimizer** for training as explained in 2.3. We conducted a grid search over learning rates (0.001, 0.01, 0.1) and batch sizes (32, 64, 128) to optimize model performance across Kryptonite-n datasets ($n = 9, 12, 15$). Each combination was evaluated using 5-fold cross-validation. By systematically varying these parameters, we aimed to identify the optimal configuration that minimizes the model's validation loss and enhances generalization on unseen data.

From the results, we determined that the optimal hyperparameters were **Learning rate = 0.001** and **Batch size = 128**. These values were consistently optimal across all datasets (Kryptonite-n, $n = 9, 12, 15$), demonstrating the robustness and generalizability of this configuration. Illustration of the plots for hyperparameter tuning are provided in A.5

4.3.1. Results of the Best Model

Similar to Extra Trees, we conducted a 5-fold cross-validation experiment using three different random seeds: 42, 99, and 123. Table 2 presents the results, indicating that the model achieves high and consistent accuracy above the desired threshold for $n = 9, 12, 15$ across different random seeds. However, for $n = 18$, the model accuracy does not exceed the required threshold. Additionally, Figure 1 illustrates the learning curves for the Kryptonite datasets with varying dimensions, highlighting the model's behavior during training. For $n = 9, 12, 15$, both training and validation losses reach acceptable levels, demonstrating minimal overfitting. In contrast, for $n = 18$, the training loss continues to decrease while the validation loss fluctuates and plateaus, indicating struggles in generalization and significant overfitting with higher-dimensional data.

Random Seed	n = 9	n = 12	n = 15	n = 18
42	95.75±0.0032	95.75±0.0022	93.71±0.0062	82.02±0.1547
99	95.77±0.0036	95.85±0.0018	93.81±0.0047	66.80±0.1882
123	95.77±0.0027	95.85±0.0029	93.87±0.0033	81.78±0.1463

Table 2. Accuracies of the Best Neural Network Model across different seeds

Kryptonite-n	Sample Size n_i	Extra Trees (%)	Neural Networks (%)	ResNets (%)	Hoeffding Bound (%)
9	18,000	95.80 ± 0.003	95.75 ± 0.0032	—	0.91
12	24,000	85.90 ± 0.008	95.75 ± 0.0022	—	0.79
15	30,000	52.48 ± 0.006	93.71 ± 0.0062	—	0.71
18	36,000	—	82.02 ± 0.16	90.47 ± 0.27	0.65
24	298,000	—	49.97 ± 0.01	86.28 ± 18.12	0.22

Table 3. Classification accuracies and Hoeffding bounds for models on Kryptonite datasets at 95% confidence level (random seed 42)

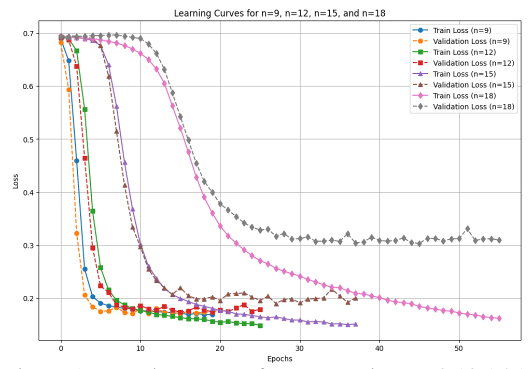

Figure 1. Learning curves for Kryptonite n = 9,12,15,18

4.4. Validation of Hypothesis

To verify **H2** and **H3**, we present the final results of each model applied to the respective datasets in Table 3. Based on the sample size of each dataset, we also calculate the Hoeffding bounds at the 95% confidence level.

These findings validate our hypotheses. The Extra Trees model effectively classifies the lower-dimensional dataset (*Kryptonite-9*), achieving an accuracy of 95.80% . With 95% confidence, the true accuracy lies within $\pm 0.91\%$ of the observed accuracy, indicating high reliability. Neural Networks, capable of modeling non-linear relationships and capturing complex patterns, effectively classify datasets across a wider range of dimensions (*Kryptonite-9*, 12, and 15), achieving higher accuracies than logistic regression and Extra Trees, and meeting the threshold accuracies with low Hoeffding bounds. ResNets further improve performance on higher-dimensional datasets (*Kryptonite-18* and *Kryptonite-24*), meeting the threshold accuracies and exhibiting even lower Hoeffding bounds, thus validating **H3**.

We provide confirmation of model stability across different random seeds in A.8. The low standard deviations and error bars indicate that the models' performance is consistent and not due to random fluctuations.

5. Discussion

5.1. Performance Comparison

Our results validate our hypotheses by demonstrating that our models attain the threshold accuracy on datasets up to n = 24, refuting the arguments of the original paper. For smaller datasets ($n = 9$), both Logistic Regression and Extra Trees achieve comparable accuracies due to simpler data structure, where the linear decision boundary of Logistic Regression, enhanced with higher-order features, suffices to capture underlying relationships. As dimensionality increases ($n = 12, 15$), Extra Trees significantly outperform Logistic Regression. This is because Extra Trees can handle high-dimensional data better. In contrast, Logistic Regression, relies on a global linear decision boundary, which becomes inefficient in high-dimensional settings.

Neural Networks outperform Extra Trees due to their ability to model complex, high-dimensional relationships through layered non-linear transformations. Neural Networks adapt to diverse patterns, making them better suited for tasks like *Kryptonite-n*. ResNets further improve performance by introducing residual connections that alleviate the vanishing gradient problem and enable training of deeper architectures. However, ResNets exhibited higher standard deviations and were less stable. Optimizing ResNet models for these high-dimensional datasets remains a subject for future work, with additional efforts focusing on consistently reaching the threshold accuracy for $n \gg 30$.

5.2. Sustainability Analysis

This section evaluates the sustainability of our model training experiments, with detailed metrics in A.10. Neural Networks and Extra Trees were trained with 5-fold CV, resulting in longer training times. In contrast, ResNet did not utilize CV due to its high computational demands, and for $n = 18$, extended training epochs further increased resource usage. For smaller datasets, Neural Networks showed greater computational efficiency compared to Extra Trees. At $n = 15$, Extra Trees emitted 1.02×10^{-5} kg/s—nearly double that of Neural Networks—due to the construction of 200 decision trees, which scales poorly with higher-dimensional data. Despite their higher resource consumption, Extra Trees didn't achieve comparable performance, establishing Neural Networks as more sustainable and accurate for smaller datasets. DeepResNet proved to be the most accurate model for larger datasets but incurred higher computational costs, emitting between 1.73×10^{-5} and 1.78×10^{-5} kg/s for $n = 18$ and $n = 24$. Although cross-validation wasn't performed for ResNet, it still exhibited higher emissions compared to Neural Networks and Extra Trees.

References

Courty, B., Schmidt, V., Luccioni, S., Goyal-Kamal, MarionCoutarel, Feld, B., Lecourt, J., LiamConnell, Saboni, A., Inimaz, supatomic, Léval, M., Blanche, L., Cruveiller, A., ouminasara, Zhao, F., Joshi, A., Bogroff, A., de Lavoreille, H., Laskaris, N., Abati, E., Blank, D., Wang, Z., Catovic, A., Alencon, M., Stechły, M., Bauer, C., de Araújo, L. O. N., JPW, and MinervaBooks. mlco2/codecarbon: v2.4.1, May 2024. URL https://doi.org/10.5281/zenodo.11171501.

Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. *Machine learning*, 63:3–42, 2006.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

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

Hoeffding, W. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Schmidt, M., Le Roux, N., and Bach, F. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

Stone, M. H. The generalized weierstrass approximation theorem. *Mathematics Magazine*, 21(4):167–184, 1948.

A. Appendix

A.1. SAG Definition

The SAG optimizer updates θ by averaging gradients over all samples at each iteration:

$$\theta_{t+1} = \theta_t - \alpha \left(\frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L_i(\theta_t) \right), \tag{10}$$

where α is the learning rate, N is the total number of samples, and $\nabla_\theta L_i(\theta_t)$ denotes the gradient of the loss function with respect to θ for the i-th sample.

A.2. Derivations of Regularization Parameter Bound

To derive the upper bound for the regularization parameter C, we start with the condition number constraint set for the Hessian matrix H of the optimization problem. The condition number $\kappa(H)$ is defined as:

$$\kappa(H) = \frac{\lambda_{\text{max}}(H)}{\lambda_{\text{min}}(H)} \leq \kappa_{\text{target}} = 1.$$

Given that $H = X^\top DX + \lambda I$, where I is the identity matrix, the eigenvalues of H satisfy:

$$\lambda_{\text{max}}(H) = \lambda_{\text{max}}(X^\top DX) + \lambda,$$

$$\lambda_{\text{min}}(H) = \lambda.$$

Substituting these into the condition number expression:

$$\kappa(H) = \frac{\lambda_{\text{max}}(X^\top DX) + \lambda}{\lambda} \leq 1.$$

This simplifies to:

$$\lambda \geq \lambda_{\text{max}}(X^\top DX).$$

Since $\lambda = \frac{1}{C}$, we substitute to obtain:

$$\frac{1}{C} \geq \lambda_{\text{max}}(X^\top DX) \implies C \leq \frac{1}{\lambda_{\text{max}}(X^\top DX)}.$$

Thus, the upper bound for C is established as:

$$C \leq \frac{1}{\lambda_{\text{max}}(X^\top DX)}.$$

This bound ensures that the Hessian matrix H is well-conditioned, facilitating optimal convergence of the SAG optimizer.

A.3. Mathematical Expressions for Gradient Descent

These gradients are used in an optimization algorithm such as SGD to update the parameters. The gradient descent update rule for each weight $\mathbf{W}$ and bias $\mathbf{b}$ is:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}}, \quad \mathbf{b} \leftarrow \mathbf{b} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}}, \tag{11}$$

where η is the learning rate, controlling the step size in the parameter update.

A.4. Mathematical Description of Adam

The **Adam** (Adaptive Moment Estimation) optimizer is an advanced optimization algorithm that combines the advantages of momentum and adaptive learning rates, making it effective for deep learning models. Momentum helps smooth and accelerate optimization by accumulating a moving average of past gradients, which reduces oscillations and allows faster convergence. At each time step t, the gradient of the loss function $\mathcal{L}$ with respect to the parameters θ is computed as

$$g^{(t)} = \nabla\mathcal{L}\left(\theta^{(t-1)}\right).$$

Adam then maintains two moving averages of the gradients: the first moment (mean) $m^{(t)}$ and the second moment (uncentered variance) $v^{(t)}$, updated as

$$m^{(t)} = \beta_1 m^{(t-1)} + (1-\beta_1)g^{(t)}$$

and

$$v^{(t)} = \beta_2 v^{(t-1)} + (1-\beta_2)(g^{(t)} \odot g^{(t)}),$$

where β_1 and β_2 are decay rates for the moment estimates, typically set to 0.9 and 0.999, respectively. Here, $m^{(t)}$ acts as a momentum term by accumulating a weighted average of past gradients, helping the optimizer to maintain consistent progress in relevant directions, even in the presence of noisy updates.

Bias-corrected estimates $\hat{m}^{(t)}$ and $\hat{v}^{(t)}$ are then computed as

$$\hat{m}^{(t)} = \frac{m^{(t)}}{1-\beta_1^t}, \quad \hat{v}^{(t)} = \frac{v^{(t)}}{1-\beta_2^t}.$$

An adaptive learning rate $\alpha^{(t)}$ is then computed as

$$\alpha^{(t)} = \frac{\alpha^{(t-1)}\sqrt{1-\beta_2^t}}{1-\beta_1^t}.$$

Finally, the parameters θ are updated using the adaptive learning rate:

$$\theta^{(t)} = \theta^{(t-1)} - \frac{\alpha^{(t)}\hat{m}^{(t)}}{\sqrt{\hat{v}^{(t)}}+\epsilon},$$

where ϵ is a small constant (typically 10^{-8}) added to prevent division by zero and improve numerical stability. By adjusting learning rates for each parameter and leveraging momentum, Adam accelerates convergence and enhances training stability (Goodfellow et al., 2016).

A.5. Hyperparamter Selection for Neural Networks

To better understand the training dynamics, we plotted learning curves for each combination of parameters for Kryptonite-15 as an example (with similar trends observed for Kryptonite-9 and Kryptonite-12). Figures 2 and 3 show the training and validation losses for varying learning rates and batch sizes, respectively.

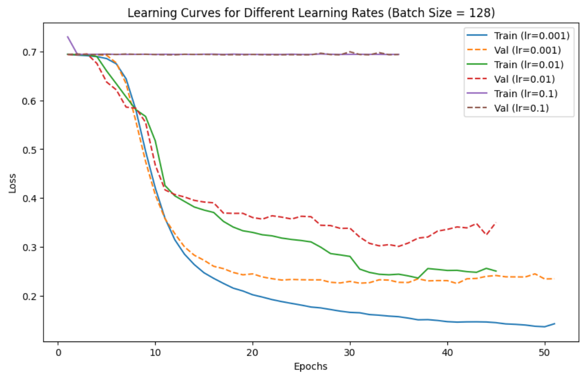

Figure 2. Learning curves for different learning rates for Kryptonite-15 (fixed batch size = 128).

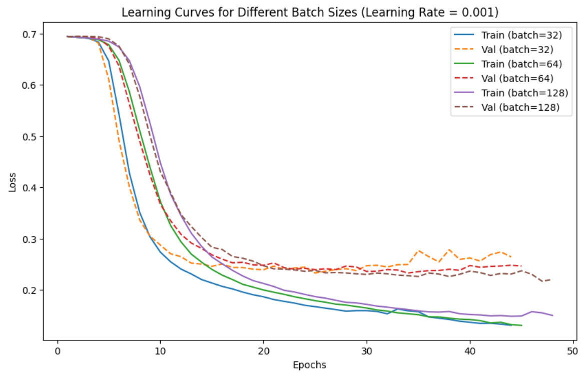

Figure 3. Learning curves for different batch sizes for Kryptonite-15 (fixed learning rate = 0.001).

In Figure 2, we observe that a learning rate of 0.001 achieves a steady decrease in validation loss, outperforming higher rates (0.01 and 0.1) that show more oscillation, especially in the validation loss, indicating potential instability in training. Similarly, Figure 3 demonstrates that a batch size of 128 provides a smoother convergence in the validation loss compared to smaller batch sizes, which tend to exhibit higher variability in the validation phase. These visualizations confirm that a learning rate of 0.001 and a batch size of 128 yield the most stable and lowest validation loss, validating our choice of optimal hyperparameters for this model.

A.6. Overfitting

Mathematically, overfitting can be expressed as:

$$\mathrm{Err}(x_t) = \sigma_\epsilon + \mathrm{Bias}^2 + \mathrm{Variance}. \quad (12)$$

Here, σ_ϵ represents the noise inherent in the data.

Bias captures the error due to the expected value from the estimated model being different from the expected value of the true model, and Variance reflects the model's sensitivity to small fluctuations in the training data.Specifically, the bias and variance terms are defined as:

$$\text{Bias}^2 = \mathbb{E}\left[f(x_t) - \hat{f}(x_t)\right]^2, \tag{13}$$

$$\text{Variance} = \mathbb{E}\left[\hat{f}(x_t) - \mathbb{E}[\hat{f}(x_t)]\right]^2, \tag{14}$$

where $f(x_t)$ is the true function, and $\hat{f}(x_t)$ is the model's prediction.

A.7. Effect of Condition Number C with Logistic Regression

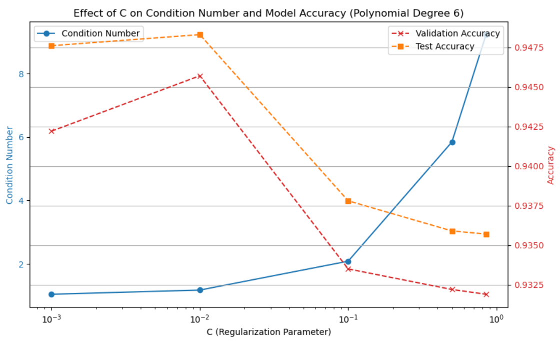

Figure 4. Effect of Condition Number C with Logistic Regression

A.8. Model Stability

Figure 5 and 6 provide a detailed visualization of the stability and consistency of the Extra Trees and Neural Network models across varying random seeds

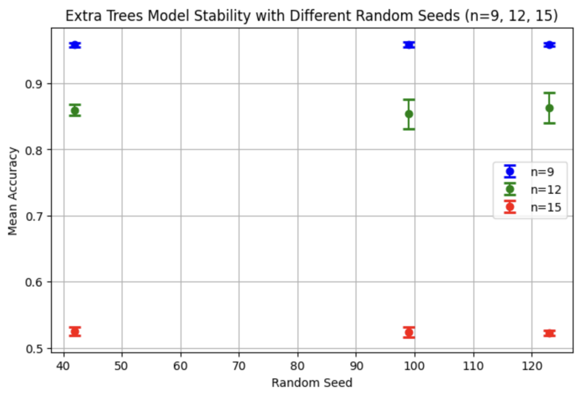

Figure 5. Extra Trees mean accuracy with error bars across different random seeds

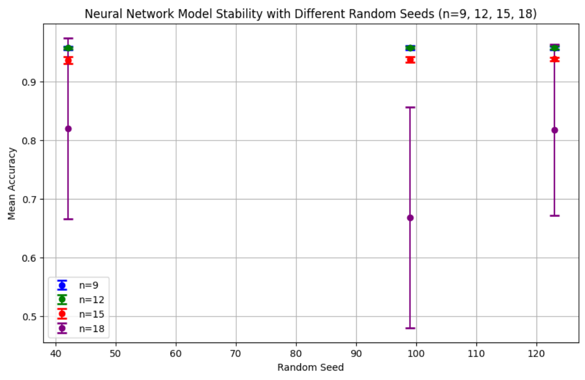

Figure 6. Neural Network mean accuracy with error bars across different random seeds

A.9. Residual Networks

In a residual network, a residual block is defined as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{\mathbf{W}_i\}) + \mathbf{x}, \tag{15}$$

where $\mathbf{x}$ is the input, $\mathcal{F}(\mathbf{x}, \{\mathbf{W}_i\})$ represents the residual mapping (comprising layers and activations), and $\mathbf{y}$ is the output. This improve gradient flow, smooth the loss landscape, and enhance feature reuse. Additionally, ResNets incorporate batch normalization and ReLU activations in each block, further stabilizing and accelerating training.

A.10. Sustainability

The table A.10 summarizes the sustainability metrics for our model training experiments, providing a detailed comparison of resource usage, emissions, and computational efficiency across different models and dataset sizes.

n	Model	Duration (s)	Emissions CO_2eq (Kg)	Emissions rate (Kg/s)	Energy (kWh)
9	Neural Networks	107.80	6.40e-04	5.94e-06	1.41e-03
	Extra Trees	430.18	1.23e-03	2.87e-06	5.64e-03
12	Neural Networks	177.6	1.05e-03	5.94e-06	2.33e-03
	Extra Trees	1546.53	5.79e-03	3.75e-06	2.03e-02
15	Neural Networks	299.15	1.78e-03	5.94e-06	3.92e-03
	Extra Trees	2469.56	2.52e-02	1.02e-05	3.92e-02
18	DeepResNet	198.88	3.45e-03	1.73e-05	7.32e-03
24	DeepResNet	173.11	3.08e-03	1.78e-05	6.55e-03

Table 4. Sustainability metrics of the model training experiments.