Journal of Software Engineering Research and Development, 2026, 14:16, doi: 10.5753/jserd.2026.6399

On the Effectiveness of Feedforward Neural Networks for Performance Prediction in Highly Configurable Systems: The Case of Linux Kernel Size

João Marcello Bessa [**Pontifical Catholic University of Rio de Janeiro (PUC-Rio)** | ***jrodrigues@inf.puc-rio.br***]
Heraldo Borges [**Univ Rennes, Inria, CNRS, IRISA** | ***heraldo.pimenta-borges-filho@irisa.fr***]
Pedro Lopes [**Pontifical Catholic University of Rio de Janeiro (PUC-Rio)** | ***pedrolopes@aise.inf.puc-rio.br***]
Lucas Lopes [**Pontifical Catholic University of Rio de Janeiro (PUC-Rio)** | ***lucasfalopes@aise.inf.puc-rio.br***]
Mathieu Acher [**Univ Rennes, INSA, Inria, CNRS, IRISA** | ***mathieu.acher@irisa.fr***]
Juliana Alves Pereira [**Pontifical Catholic University of Rio de Janeiro (PUC-Rio)** | ***juliana@inf.puc-rio.br***]

Abstract

Highly configurable software systems, such as the Linux kernel, expose thousands of configuration options whose combinations can drastically influence non-functional properties. In embedded or resource-constrained environments, kernel binary size is particularly important because it directly affects memory footprint, boot time, storage usage, and overall system responsiveness. Accurately predicting binary size from configuration options enables developers to evaluate trade-offs early, in the development cycle, reducing the need for costly trial-and-error compilation and measurement processes. This study investigates the use of Feedforward Neural Networks (FNNs) for predicting Linux kernel binary size and compares their performance against strong classical Machine Learning (ML) baselines, including Random Forest and Gradient Boosting Trees. In addition, we analyze the impact of feature selection (FS) strategies on predictive accuracy and computational cost. We evaluate five FS approaches, including supervised tree-based importance rankings and a semantic, label-free method based on Word2Vec embeddings extracted from Linux kernel documentation. Our experiments were conducted on a dataset containing 9,670 Linux kernel configuration options. The results show that classical tree-based ensemble methods outperformed FNNs in predictive accuracy, with Gradient Boosting Trees achieving the best overall results (MAPE of 5.21%). Although FNNs combined with feature selection achieved reasonable accuracy (best MAPE of 8.26%) and benefited from reduced training times, they did not surpass the traditional ML baselines. These findings provide important empirical evidence that more complex neural architectures do not necessarily yield superior performance for SPL prediction tasks, even in high-dimensional configuration spaces. We also show that the Word2Vec-based semantic FS method offers a practical label-free alternative for early-stage scenarios where measured NFP data are unavailable or expensive to obtain, although it generally underperforms supervised feature selection strategies in both accuracy and efficiency. Overall, our findings provide practical guidance for SPL practitioners and researchers in selecting prediction models and preprocessing strategies that balance accuracy, interpretability, and computational cost in highly configurable systems.

Keywords: *Software Product Lines, Configurable Systems, Deep Learning, Performance Prediction*

1 Introduction

Modern software systems are often designed as highly configurable platforms, exposing hundreds or thousands of features that influence both functional and non-functional properties (NFPs) such as memory footprint, boot time, and energy consumption (Ochoa et al., 2017; Pereira et al., 2021). This flexibility, central to Software Product Lines (SPLs), enables product customization for diverse requirements but also leads to a combinatorial explosion in configuration space, making exhaustive evaluation infeasible.

The Linux kernel is a prominent example, offering over 9,000 configuration options (many tristate), resulting in billions of potential variants. In this work, we focus on predicting a specific and practically relevant NFP: the *binary size* of the kernel. This property directly impacts deployability, memory usage, and startup time, particularly in embedded, IoT, and cloud environments. In resource-constrained deployment scenarios, exceeding a given size budget may render a kernel variant unusable. For instance, an embedded gateway with limited flash storage cannot accommodate a kernel image above a strict threshold. Currently, developers navigate the trade-off between required functionality and size constraints through costly trial-and-error compilation cycles: they select a tentative set of configuration options, compile the kernel, measure the resulting binary size, and iteratively adjust their selection until size and functionality requirements are met. This manual loop becomes prohibitively expensive in practice, given the billions of possible configurations. As an example, consider a developer configuring a Linux-based firmware for an IoT sensor node with 32 MB of flash memory. The developer needs networking, USB support, and a specific filesystem driver, but must keep the kernel image under 15 MB. Instead of compiling dozens of candidate configurations, a trained predictive model can instantly estimate the binary size of any proposed configuration, enabling the developer to efficiently explore the configuration space and identify the most size-efficient variant that satisfies all requirements. Accurately estimating the kernel size before compilation enables faster design iterations and

reduces costly trial-and-error build cycles.

We base our evaluation on the TuxKConfig dataset (Borges et al., 2025), which contains 95,854 compiled kernel variants and their measured sizes (ranging from 7 MB to 1,698 MB). The dataset, built over 15,000 computation hours, provides a robust benchmark for studying predictive modeling in large configuration spaces.

Prior work on kernel size prediction includes linear models (Guo et al., 2013), decision tree ensembles, and deep learning (DL) approaches such as DeepPerf (Ha and Zhang, 2019a) and PerLasso (Ha and Zhang, 2019b). While these approaches successfully capture non-linear interactions, they rely on *implicit* feature selection via L_1 regularization, which can be sensitive to hyperparameters and yield unstable feature subsets. This limits transparency and interpretability, which are critical factors for practitioners in SPL engineering. In contrast, our study investigates the effect of *explicit feature selection* applied before model training, which selects the most relevant input variables for prediction. We evaluated five feature selection strategies: four based on tree-derived importance scores and one semantic method using Word2Vec embeddings from kernel documentation. Decoupling feature selection from model training allows for better interpretability, modular experimentation, and potential reuse across prediction tasks. Notably, the Word2Vec-based method requires zero compiled samples to perform feature selection, representing a valuable accuracy-versus-data-scarcity trade-off: while it yields higher error rates than tree-based methods, it operates entirely without labeled NFP data, making it applicable in early-stage SPL engineering scenarios where compiling thousands of configurations to generate ground truth for supervised feature ranking is computationally prohibitive.

Our motivation for comparing *Feedforward Neural Networks (FNNs)* with classical *Machine Learning* (ML) stems from the need to better understand the practical trade-offs between model complexity, predictive accuracy, and computational cost in SPL performance prediction. On the one hand, tree-based ensemble methods such as Random Forest and Gradient Boosting Trees have consistently demonstrated strong predictive performance in configurable-system tasks. On the other hand, FNNs are often considered promising alternatives for high-dimensional binary spaces because of their representational flexibility and scalability. In this context, we empirically assess the conditions under which they become viable alternatives, particularly when combined with feature selection techniques.

By benchmarking FNNs against well-established baselines under the same experimental conditions, we provide a rigorous evaluation of their strengths and limitations for Linux kernel size prediction. Our results ultimately show that, despite the flexibility of neural architectures, classical tree-based ensemble methods remain more accurate and robust for this prediction task, while feature selection primarily benefits training efficiency rather than reversing the relative superiority of the baselines.

This work contributes to the SPL community by providing an empirical assessment of the trade-offs between deep learning and classical ML approaches for performance prediction in highly configurable systems, as well as insights into the role of feature selection in balancing predictive accuracy, interpretability, and computational efficiency. In summary, this paper makes the following contributions:

- We conduct a large-scale empirical evaluation of FNNs for predicting Linux kernel binary size, comparing them against strong classical ML baselines (*e.g.*, Random Forest and Gradient Boosting Trees).
- We show that, despite FNNs flexibility for high-dimensional spaces, they do not consistently outperform tree-based ensemble methods for this task, highlighting important practical limitations of DL approaches in SPL performance prediction.
- We systematically investigate five feature selection strategies, including supervised tree-based rankings and a novel semantic-based approach using Word2Vec embeddings extracted from Linux documentation. We analyze how these strategies affect predictive accuracy, dimensionality reduction, and training efficiency.
- We show that semantic-based feature selection using Word2Vec provides a viable label-free preprocessing alternative for early-stage scenarios where measured NFP data are unavailable or expensive to obtain, although supervised feature selection methods generally achieve superior predictive performance.
- We provide a comprehensive experimental infrastructure with publicly available datasets, configuration files, preprocessing scripts, and training pipelines to ensure full reproducibility and support future research in SPL performance prediction at `https://github.com/aisepucrio/DL2SPL_FeatureSelection`.

2 Background and Related Work

Highly configurable systems, such as Software Product Lines (SPLs), expose large sets of configuration options (*a.k.a.* features) that impact both functional behavior and non-functional prop- erties (NFPs) ochoa2017survey,pereira2021learning. Examples of NFPs include binary size, memory footprint, and execution time. Due to the combinatorial growth of configuration spaces, exhaustive measurement of NFPs is impractical. As a result, predictive modeling using Machine Learning (ML) has become a standard approach to estimate NFPs from a representative subset of configurations pereira2021learning, ochoa2018survey.

2.1 Machine Learning Fundamentals

Machine Learning (ML) is a branch of artificial intelligence that enables systems to learn patterns from data without being explicitly programmed for each specific task. In the context of this study, ML techniques are used to build predictive models that estimate NFPs (*e.g.* binary size) from a combination of configuration options pereira2020machine.

ML approaches are broadly categorized into *supervised learning*, where models are trained on labeled input-output pairs to learn a mapping function, and *unsupervised learning*, where patterns are discovered in unlabeled data (Nasrabadi, 2007). In this work, all used tree-based models (Random Forest, Gradient Boosting, XGBoost, Decision Tree) and

Feedforward Neural Networks operate under the supervised paradigm, while the Word2Vec-based feature selection method is unsupervised.

Key concepts used throughout this study include ((Srivastava et al., 2014; Nasrabadi, 2007)): *(i) Activation functions*, which introduce non-linearity into neural networks (we evaluate ReLU, Leaky ReLU, PReLU, and ELU; see Section 3.4.2); *(ii) Dropout*, a regularization technique that randomly deactivates a fraction of neurons during training to prevent overfitting; *(iii) Loss functions*, which quantify the discrepancy between predictions and actual values (we employ MAPE, MSE, and Huber Loss/SmoothL1); *(iv) Early stopping*, a training strategy that halts the learning process when validation performance ceases to improve, thus preventing overfitting; and *(v) Feature selection*, the process of identifying and retaining only the most relevant input variables, thereby reducing dimensionality, improving generalization, and lowering computational cost.

2.2 Predictive Modeling in SPLs

Classical models, such as linear regression and decision tree-based regressors (*e.g.*, CART/ DECART (Guo et al., 2013)), offer interpretability and can effectively capture low-order interactions. However, they often struggle with the high-order, non-linear feature interactions common in SPLs. Ensemble methods like Random Forest (RF) and Gradient Boosting Tree (GBT) extend this capability, but their reliance on tree structures may limit expressiveness in extremely high-dimensional binary spaces.

Deep learning (DL) methods, and in particular *Feedforward Neural Networks (FNNs)*, have emerged as promising alternatives (Ha and Zhang, 2019a; Gong and Chen, 2025; Cheng et al., 2023). FNNs propagate information in one direction from input to output, with each neuron applying a non-linear activation (*e.g.*, ReLU (Nair and Hinton, 2010) or ELU (Clevert et al., 2016)) to a weighted sum of its inputs. Training is performed via backpropagation (Rumelhart et al., 1986) to minimize a loss function, such as the Mean Absolute Percentage Error (MAPE) (Chai and Draxler, 2014). Their flexibility allows them to model complex dependencies between configuration options without relying on pre-defined interaction structures.

Several works illustrate the application of neural networks to configurable systems. DeepPerf (Ha and Zhang, 2019a) uses FNNs with multiple hidden layers to implicitly model high-order interactions. PerLasso (Ha and Zhang, 2019b) incorporates a Fourier decomposition with L_1 regularization, performing implicit feature selection during training. Although these methods have shown promising results, they depend heavily on hyperparameter tuning and provide limited control over the selected features. More recent work by Cheng et al. (2023) proposed a hierarchical interaction FNN to explicitly model feature interactions, achieving improvements in certain SPL benchmarks. Similarly, Gong and Chen (2025) investigated hybrid neural architectures for configurable systems, suggesting that combining feature selection with neural models can improve both efficiency and interpretability.

In contrast, our study does not introduce new neural architectures or hybrid models. Instead, we emphasize a systematic evaluation of *feature selection strategies* and their role in shaping predictive performance when combined with traditional FNNs. Unlike Cheng et al. (2023), who designed architectures tailored to hierarchical feature interactions, our focus lies on assessing how reducing the dimensionality of configuration spaces through syntactic and semantic selection impacts prediction accuracy, computational cost, and scalability. Complementing previous hybrid approaches (Gong and Chen, 2025), we directly compare tree-based feature rankings (RF/GBT/XGBoost) with a Word2Vec-based semantic filter. This positioning highlights our contribution: bridging classical and semantics-driven feature selection with FNNs to provide practical guidance for predictive modeling in high-dimensional SPLs. Furthermore, whereas Gong and Chen (2025) integrated feature selection mechanisms into hybrid models, we investigate the comparative effectiveness of tree-based feature rankings and a novel Word2Vec-based semantic filtering approach. This highlights the originality of our contribution: bridging classical and semantic-driven feature selection with FNNs to provide practical insights into predictive modeling for high-dimensional SPLs.

2.3 Feature Selection in SPL Prediction

In high-dimensional spaces like the Linux kernel (over 9,000 configuration options), many features are irrelevant or redundant for a given prediction task. *Feature Selection* refers to the process of identifying the most informative input variables before model training. Feature selection can reduce training cost, improve generalization, and enhance interpretability (Acher et al., 2022). While some neural models, such as DeepPerf (Ha and Zhang, 2019a) and PerLasso (Ha and Zhang, 2019b), apply *implicit* feature selection via L_1 regularization, this approach is sensitive to hyperparameter settings and can yield unstable subsets. In contrast, *explicit* feature selection allows transparent control over the selected features and enables fairer comparisons across different models.

Tree-based methods, such as RF and XGBoost, are commonly used for supervised feature selection by ranking features according to their importance scores. Alternatively, semantic-based approaches can exploit natural language descriptions of configuration options. In this work, we explore a Word2Vec-based feature selection method that leverages kernel documentation to select features semantically related to the prediction target (see Section 3.4).

Word2Vec addresses a fundamentally different question from what can be derived directly from KConfig. KConfig encodes structural constraints between configuration options, such as dependencies, selections, defaults, and conflicts. These constraints determine which option combinations are valid and how features interact syntactically. However, they do not express how relevant a given option is to a specific NFP. The Word2Vec-based method, in contrast, targets semantic relevance. We train Word2Vec embeddings on Linux kernel documentation and compute a cosine similarity score between each configuration option and a target query vector constructed from performance-related seed terms (*e.g.*, “performance”, “speed”, “efficiency”, “optimiza-

tion"). The resulting ranking reflects the conceptual proximity of each option to performance concerns. This information is orthogonal to structural dependency graphs and cannot be inferred from KConfig alone.

2.4 Linux Kernel Prediction Studies

The Linux kernel has served as a benchmark for SPL research due to its vast configuration space and real-world relevance. Martin et al. (2022) investigated transfer learning across kernel versions for predicting binary size, demonstrating the impact of configuration variability on predictive performance. Acher et al. (2022, 2019) compared several ML models for kernel size prediction in combination with a tree-based feature selection pipeline. They used a single feature selection approach (*i.e.*, the Random Forest algorithm) and showed that feature selection improves efficiency without significantly compromising accuracy. Our study builds on these works by systematically evaluating FNNs under multiple feature selection methods, including both tree-based and semantic approaches, and by benchmarking against Random Forest and Gradient Boosting Tree as strong classical baselines.

2.5 Gap and Positioning of This Work

While tree-based feature selection has been shown to be effective for kernel size prediction (Acher et al., 2022), and neural models have demonstrated the potential to capture complex interactions (Ha and Zhang, 2019a,b), there has been no systematic evaluation of FNNs combined with diverse feature selection strategies, including semantic methods. Our work addresses this gap by:

1. Benchmarking FNNs against strong classical tree-based baselines (RF and GBT) under multiple feature selection strategies.
2. Introducing a novel Word2Vec-based semantic feature selection method that does not require labeled importance scores.
3. Analyzing both predictive accuracy and computational efficiency in the context of the Linux kernel, with implications for other SPLs.

3 Study Design

This section presents the design of our study, detailing the research questions, methodology, experimental setup, and the rationale behind our choices.

3.1 Research Questions

Highly configurable systems, such as the Linux kernel, expose thousands of configuration options (*a.k.a.*, features), which greatly expand the variability space and also make the prediction of NFPs a challenging task. Because the number of variants grows exponentially, only a subset can be measured in practice. Consequently, labeled data are scarce relative to dimensionality; interactions among options induce complex, non-linear, higher-order effects; and large portions of the configuration space may remain unobserved in the collected data. For predictive studies, this means each feature adds little useful information; we have too few examples relative to the number of model parameters, it becomes harder to learn how features relate to the target, and exploring the configuration space requires much more computation. (Kotsiantis, 2013; Ma et al., 2022; Ochoa et al., 2017; Pereira et al., 2021).

Previous work has shown that traditional ML models, such as decision trees and ensemble methods, can achieve promising results in this context (Kim et al., 2007). More recently, deep learning (DL) methods have been investigated, and in this study, we focus on a specific DL class, namely feedforward neural networks (FNNs, multilayer perceptrons), to model non-linear dependencies and capture intricate feature interactions in structured configuration data (Li et al., 2022).

Despite their modeling power, neural networks face challenges with high-dimensional data. As the number of input features grows, model size and the required amount of training data also increase, elevating the risk of overfitting and substantially increasing training time (Ma et al., 2022). This phenomenon, often referred to as the *curse of dimensionality* (Kotsiantis, 2013), can affect generalization and make neural networks computationally expensive in SPL scenarios. Feature selection methods are a common strategy to mitigate these issues, aiming to retain only the most relevant features and discard noisy or redundant ones (Li et al., 2022; Acher et al., 2022; Bessa and Pereira, 2023).

To address these challenges and opportunities, we investigate the following three research questions (RQs):

RQ_1. How do FNNs compare to traditional ML models in predicting Linux kernel binary size? This question investigates whether FNNs can achieve competitive predictive accuracy compared to strong, well-established ML baselines such as Random Forest and Gradient Boosting Tree when applied to highly configurable systems. The focus here is on understanding the modeling potential of FNNs in an SPL context with high-dimensional variability, such as the Linux kernel, where feature interactions can be very complex. While tree-based methods have a long track record of strong performance in kernel size prediction, they may be limited in capturing certain non-linear patterns and higher-order interactions. In contrast, FNNs offer greater modeling flexibility and can approximate complex functional relationships. This RQ evaluates accuracy under controlled conditions, exploring whether FNNs can surpass the performance of traditional models, and identifying the circumstances in which they may be a viable alternative.

RQ_2. What is the impact of different feature selection strategies on the predictive performance of FNNs? High-dimensional configuration spaces, such as the 9,670 options in the Linux kernel (represented as binary features in the TuxKConfig dataset (Borges et al., 2025)), can increase training time, increase the risk of overfitting, and harm generalization. We therefore compare alternative feature selection strategies: four supervised, tree-based methods (Random Forest, Decision Tree, Gradient Boosting Tree, and

XGBoost) and one semantic (Word2Vec-based), label-free method that uses kernel documentation to pick conceptually related options. We evaluate each method at three retention levels (top 30%, 50%, and 70%), where retention denotes the fraction of features kept after ranking. Accuracy is measured using MAPE, with identical preprocessing, splits, and seeds across methods to ensure a fair comparison. This setup reveals how much accuracy FNNs can recover when trained on smaller, targeted subsets compared to using all 9,670 features.

RQ$_3$. How do different approaches affect training time? Even models with strong predictive accuracy may be impractical in real-world SPL scenarios if their training cost is high, particularly in industrial contexts that require frequent retraining due to evolving configurations or workloads. This question quantifies the computational cost of alternative pipelines by measuring end-to-end time for preprocessing, feature selection, and model training. We also examine accuracy versus wall-clock time and report speedup relative to the all-features FNN baseline, defined as $\text{Speedup} = T_{\text{all-features}}/T_{\text{model}}$. It provides empirical guidance when computational resources are limited or rapid iteration is necessary.

3.2 Experimental Process

As illustrated in Figure 1, the study comprises three main stages: (1) baseline and setup based on previous studies, (2) feature selection and model training, and (3) results and evaluation.

We follow a multi-factorial experimental design, enabling a systematic evaluation of multiple interacting factors that may influence predictive performance. Specifically, we vary three main dimensions: *(i)* the learning algorithm (FNNs, Gradient Boosting, Random Forest, and related baselines), *(ii)* the feature selection strategy (no feature selection, tree-based importance ranking, and Word2Vec-based semantic filtering), and *(iii)* the dimensionality of the input space resulting from each selection strategy. This design allows us to isolate the individual and combined effects of model architecture and preprocessing choices on both predictive accuracy (MAPE) and training efficiency. Thus, we ensure that comparisons are fair, controlled, and reproducible, while also enabling the identification of interaction effects, for example, whether certain learning models benefit more from specific feature selection techniques. This systematic variation strengthens the internal validity of our findings and provides clearer practical guidance for selecting modeling pipelines in highly configurable systems.

3.3 Baseline and Setup

This initial stage establishes the study's foundations by defining the dataset and the reference models. We used the TuxKconfig dataset (Borges et al., 2025), which contains 95,854 Linux kernel configuration variants along with their corresponding compiled binary sizes, the latter being the prediction target. The dataset was constructed using `make randconfig`, a standard Linux Kernel utility that generates

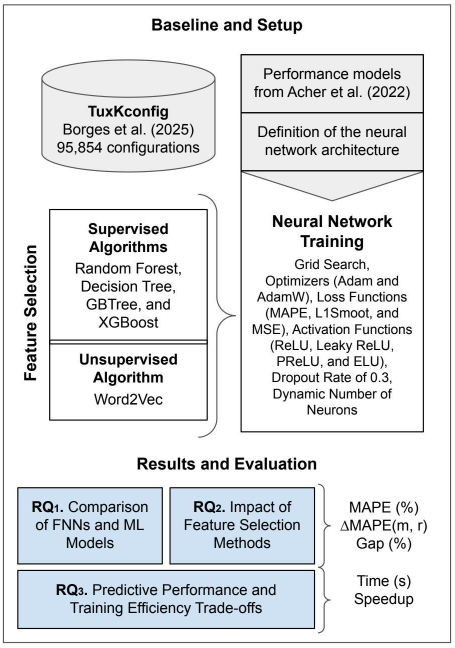

Figure 1. Study design overview.

random configurations while explicitly enforcing all Kconfig dependency constraints, including option dependencies, mutual exclusions, and selections. Because `randconfig` operates directly on the Kconfig constraint graph, every generated configuration is guaranteed to be valid by construction, without requiring any post-hoc filtering for constraint violations (Borges et al., 2025). The design choice ensures full coverage of the valid configuration space while eliminating the need for a separate step for resolving constraints.

Following the preprocessing pipeline described in Borges et al. (2025), non-tristate options (*e.g.*, integer and string-valued options, which account for approximately 300–320 options per kernel version and have limited influence on binary size) were removed. The remaining tristate options, whose values are `y` (built-in), `m` (module), and `n`, were encoded as 0, since prior work has shown that these two states exhibit similar effects on NFPs such as binary size (Borges et al., 2025). Configuration options that remained constant across all sampled variants were subsequently discarded to eliminate zero-variance features. The resulting dataset, covering kernel version 4.13, retains 9,670 binary features and 95,854 valid configuration instances, with binary sizes ranging from 7 MB to 1,698 MB. This preprocessing pipeline, together with the constraint-aware sampling strategy, ensures that all configurations used in our study are both valid and informative, providing a sound empirical basis for evaluating predictive models in the Linux Kernel's highly constrained configuration space.

As model baselines, we adopted Random Forest (RF) and Gradient Boosting Tree (GBT), both of which have shown strong results in previous studies (Acher et al., 2022). The hyperparameters of these models were tuned through small grid searches around the default values, following the same training and validation protocol applied to the FNNs to ensure comparability. As a third step, we performed the definition of the neural network architecture. We defined the search space covering different optimizers (Adam and AdamW), loss functions (MAPE, L1Smoot, and MSE), activation func-

tions (ReLU, Leaky ReLU, PReLU, and ELU), and a fixed dropout rate of 0.3. The hidden layers were progressively reduced in size after the input layer to promote hierarchical abstraction and reduce the risk of overfitting. Training incorporated adaptive early stopping and checkpointing, ensuring consistency and improved generalization. In Section 3.4, we detail each decision made in this step.

3.4 Feature Selection and Model Training

This section details the core modeling pipeline of our study. We first describe the feature selection methods used to reduce the high-dimensional configuration space of the Linux kernel, covering both supervised, model-based importance rankings and an unsupervised semantic approach. We then present the neural network training setup and hyperparameter exploration strategy adopted to ensure a systematic comparison across feature subsets. Together, these components allow us to isolate the impact of different feature selection methods on predictive accuracy and computational cost.

3.4.1 Feature Selection

Feature importance was estimated using five feature selection methods representing diverse strategies. Four are supervised, model-based approaches that produce feature-importance rankings: Random Forest, Decision Tree, GBTree, and XGBoost. The fifth is an unsupervised semantic method (Word2Vec) that leverages kernel documentation to infer conceptual similarity between features and the prediction target NFP (*i.e.*, kernel size).

Next, we briefly describe the tree-based feature selection methods used:

- *(1) Decision Trees (DT):* Supervised models that split data hierarchically, ranking featuaares by their ability to reduce impurity (*e.g.*, entropy or Gini index) (Rokach, 2015).
- *(2) Random Forest (RF):* Ensembles of randomized decision trees, computing average feature importance across trees to mitigate overfitting and produce robust rankings (Biau and Scornet, 2016).
- *(3) Gradient Boosting Tree (GBT):* Sequentially builds trees that correct the residual errors of the ensemble, prioritizing features that consistently reduce the prediction error (Natekin and Knoll, 2013).
- *(4) XGBoost:* Optimized gradient boosting implementation with parallelization, cache optimization, and efficient handling of missing values (Chen and Guestrin, 2016).

Next, we describe the proposed *Word2Vec-based semantic* method.

(5) Word2Vec-based semantic feature selection. The goal of the Word2Vec-based semantic feature selection method is to provide a label-free mechanism for ranking configuration options based on their conceptual relevance to the prediction target, namely binary size. Unlike supervised importance rankings, which rely on measured NFP values, this approach leverages textual knowledge embedded in Linux kernel documentation to infer semantic proximity between configuration options and performance-related concepts. The underlying intuition is that documentation written by maintainers encodes domain knowledge about the purpose, impact, and intended usage of features-knowledge that may indirectly signal their influence on NFPs, such as kernel size. These data may provide broad coverage of rarely used options that may be underrepresented in sampling data from measured configurations. Moreover, it does not require any compiled configurations or measured NFP values. As such, it provides a low-cost preprocessing strategy applicable in early design phases, before any sampling or benchmarking has taken place. In contexts where labeled performance data are unavailable or expensive to obtain, this semantic filtering approach offers a practical alternative for preliminary dimensionality reduction and can complement later data-driven feature selection methods. In the following, we describe the step-by-step procedure for implementing the Word2Vec-based semantic feature selection method.

Step 1: Collect and preprocess the documentation corpus. The documentation corpus was constructed by first cloning the official Linux kernel source code repository from its public GitHub[1]. To efficiently gather relevant textual data, we developed a custom, high-performance Rust script to traverse the entire directory tree and extract content from all reStructuredText (.rst) files, the standard format for kernel documentation. The collected raw text was then subjected to a comprehensive preprocessing pipeline to prepare it for model training. This process involved several standard NLP techniques: filtering out any non-English text to maintain a consistent vocabulary, removing all punctuation, normalizing the text by lemmatizing words to their base morphological form, and eliminating common English stopwords (e.g., ”the”, ”a”, ”is”) that provide little semantic value. The resulting clean, processed corpus served as the basis for training the Word2Vec model.

Step 2: Train and persist embeddings. A Word2Vec model is trained in a documentation corpus to learn distributed vector representations (embeddings) (Mikolov et al., 2013), where semantically related terms are placed close in the vector space. The trained model is saved for later queries and ranking steps.

Step 3: Define the target concept. To represent the concept of performance, a seed vocabulary is specified with explicit terms that express the target NFP concept (*e.g.*, “performance”, “speed”, “efficiency”, “optimization”). Using the learned embeddings, we form a target query vector by averaging seed-term vectors.

Step 4: Optional refinement. To reduce noise, we optionally refine the candidate terms via unsupervised clustering and expert curation. We applied K-Means to the embeddings and retained the cluster containing the anchor term “performance“, as it concentrates the semantically closest items. We treat this step as optional because, depending on corpus coverage and vocabulary cleanliness, the raw nearest neighbors may already be sufficiently focused; in our corpus, we *did* use clustering and selected the cluster containing “perfor-

[1] `https://github.com/torvalds/linux.git`

mance“, without a numeric similarity threshold beyond the cluster membership.

Step 5: Map configuration options to embeddings (preprocessing). We map each kernel option name to an embedding. Because option names in the dataset (here, 9,469 configuration options in the Parquet file for this Word2Vec run) rarely matched the vocabulary verbatim at first, we normalize names (lowercasing and replacing “_”/“-” with spaces) and then split composite identifiers into tokens (e.g., `cpu_usage` → “cpu”, “usage”). This increased coverage from 1 match initially to 340 after normalization and to ∼95% after token splitting. The final vector for each feature is obtained by averaging the embeddings of its tokens present in the vocabulary.

Step 6: Score and rank. The semantic relevance of each feature is calculated as the cosine similarity between its vector and the target query vector, which represents the performance concept. We then retain the top fraction p_{sem} of options as an *unsupervised pre-filter* for downstream predictors. In our experiments, we set $p_{\text{sem}} \in \{0.3, 0.5, 0.7\}$ to study the performance of the use of the top-30%, top-50%, and top-70% features for training, *i.e.* the highest-scoring $p_{\text{sem}} \times N$ options among the 9,469 candidates. The score and rank are computed based on text-only similarity scores, without using NFP labels.

3.4.2 Neural Network Training

FNNs were constructed to systematically assess the influence of architecture choices and regularization techniques. Each network had an input layer corresponding to the number of selected features, followed by progressively smaller hidden layers to promote hierarchical data abstraction (Li et al., 2022; Ma et al., 2022). Next, we summarize the main settings experimented with in this study.

- *Grid Search:* We employed an exhaustive grid search to explore combinations of neural network hyperparameters, including optimizer type, loss function, activation function, and dropout rate. This systematic approach ensured that each configuration was evaluated under controlled conditions, allowing for a fair comparison across different model settings (Bergstra and Bengio, 2012).
- *Optimizer:* Both Adam and AdamW optimizers were evaluated. Adam adaptively scales the learning rate for each parameter, supporting fast convergence in complex, sparse-gradient problems (Kingma and Ba, 2014). AdamW decouples weight decay from gradient updates, often yielding better generalization, especially when combined with dropout and large batch sizes (Loshchilov and Hutter, 2019).
- *Loss Functions:* We employed Mean Absolute Percentage Error (MAPE), Mean Squared Error (MSE), and Huber Loss (L1 Smooth). We opt for the robust variant proposed by Barron (Barron, 2019), which unifies several robust losses and provides stable gradients for outlier-prone data.
- *Activation Functions:* Four rectifier-based functions were evaluated: ReLU (baseline, for simplicity and gradient resilience (Nair and Hinton, 2010)), Leaky ReLU and PReLU (to address the dying-ReLU problem (Maas et al., 2013; He et al., 2015)), and ELU (to accelerate convergence and regularization (Clevert et al., 2016)).
- *Dropout Value:* Dropout layers with a rate of 30% were used, based on literature recommendations to balance regularization and model performance, especially in large-scale, high-dimensional contexts (Srivastava et al., 2014; Baldi and Sadowski, 2013).
- *Number of Neurons:* The input layer matched the number of selected features (up to 9 670). Subsequent layers were gradually reduced in size to foster abstraction and mitigate overfitting, consistent with empirical analyses on depth–width trade-offs and double-descent behaviour in modern networks (Zhang et al., 2021; He et al., 2016).

To ensure a robust and methodologically sound evaluation, we partitioned the dataset into 80% training and 20% validation subsets using a fixed random seed to guarantee reproducibility and preserve the original data distribution. The 80/20 split is widely adopted in ML practice and has become a de facto standard in SPL performance prediction research, including large-scale configuration modeling studies (Acher et al., 2022). In high-dimensional configurable systems such as the Linux kernel, this proportion provides a balanced trade-off: the training set remains sufficiently large to capture complex feature interactions, while the validation set is representative enough to reliably estimate generalization error. Importantly, this strict separation between training and validation data prevents data leakage, ensuring that performance estimates reflect true predictive capability rather than memorization effects. This consideration is particularly critical in SPL contexts, where configuration samples may share structural similarities, and inadvertent overlap between training and validation sets could artificially inflate reported accuracy.

Given the high dimensionality of the feature space and the complexity of DL models, we implemented a *custom early stopping* mechanism that dynamically adapts to the training process. Instead of relying on static patience thresholds, our strategy responds to recent trends in validation loss, stopping training when improvement plateaus. This approach strikes a balance between prematurely halting promising models and wasting resources on negligible gains, a particularly relevant concern in SPLs, where training time can scale significantly with feature count. Our early stopping process was structured around five main parameters:

First, the *baseline patience* was set to 10 epochs, meaning training would continue for at least ten consecutive iterations before early stopping could be considered. This value was chosen based on systematic investigations of early stopping behavior in DL models, which suggest that patience windows in the range of 5 to 15 epochs tend to offer optimal trade-offs between accuracy and efficiency (Huang et al., 2021).

To better align this mechanism with the stochastic nature of validation loss, we introduced a *minimum delta* criterion: the smallest change in validation loss that qualifies as an improvement. We defined this delta as 0.4 times the standard deviation of the validation loss over the past five epochs. This adaptive threshold makes the early stopping mechanism

more resilient to noise and prevents it from reacting to insignificant fluctuations. Similar noise-aware strategies have been shown to improve early-stopping reliability in noisy or high-variance learning settings (Huang et al., 2021).

The standard deviation was computed over a *rolling window of five epochs*, reflecting a deliberate compromise between reactivity and stability. A shorter window would increase sensitivity to transient noise, whereas a longer window could obscure meaningful trends. Our choice aligns with recommendations from the literature, which emphasizes the value of recent validation behavior for adaptive decision-making (Huang et al., 2021).

In addition to these adaptive controls, we set a *maximum patience limit of 20 epochs* to constrain the total training time when no improvement is observed. This static ceiling serves as a fail-safe against indefinitely long training periods, especially in configurations where the validation signal becomes flat or noisy. Such safeguards are particularly important in high-dimensional models with dropout regularization, where temporary instability in validation performance can delay convergence (Loshchilov and Hutter, 2019).

Lastly, we enforced a *hard training cap of 300 epochs* across all experiments to ensure consistent runtime boundaries and fair comparability between different experimental settings of the algorithms, including variations in hyperparameters (*e.g.*, optimizer type, activation function, loss function), network architecture, and feature selection methods. This ceiling also reflects best practices in structured hyperparameter tuning protocols, where limiting the training horizon helps control variance in results and avoid overfitting through prolonged exposure to the same training set (Smith, 2018; Raschka et al., 2022).

Upon termination, we restored the model weights from the epoch that achieved the lowest validation loss. This final checkpoint selection ensures that the retained model exhibits the best observed generalization capacity during training, thereby avoiding overfitting and maximizing the utility of each training cycle.

3.5 Results and Evaluation

This stage implements the analysis aligned with RQ_1–RQ_3 by defining how we computed metrics to compare FNNs against baselines, quantified the effect of feature selection strategies on performance, and measured training time to analyze computational cost. Before detailing the evaluation procedure, we clarify that the experimental unit in our study is a single valid Linux kernel configuration, represented as a specific combination of selected configuration options, together with its associated compiled binary size. Each observation in our dataset, therefore, corresponds to one concrete configuration instance, encoded as a high-dimensional binary feature vector indicating enabled/disabled options, and a continuous target variable representing the resulting kernel binary size. All reported metrics (*e.g.*, MAPE and training time) are computed over predictions made at the level of these individual configuration instances.

We evaluated performance using Mean Absolute Percentage Error (MAPE) as the primary metric. Given ground truth values y_i and predictions $\hat{y}_i$ for n configurations, we compute

$$\mathrm{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|.$$

MAPE is reported for each trained model and summarized in tables to answer RQ_1 and RQ_2.

Compare MAPE across models (RQ_1). We report the best FNN setting and place it alongside Random Forest (RF) and Gradient Boosting Tree (GBT), which we treat as baselines, all trained with the same train/validation split and preprocessing. For FNNs, the point of reference is the FNN trained with all features (*FNN-all*). For each FNN configuration, we compute: (i) $\Delta\mathrm{MAPE} = \mathrm{MAPE}_{\text{model}} - \mathrm{MAPE}_{\text{FNN-all}}$, (ii) $\mathrm{Gap}(\%) = 100 \cdot \Delta\mathrm{MAPE}/\mathrm{MAPE}_{\text{FNN-all}}$, and (iii) $\mathrm{Speedup} = T_{\text{FNN-all}}/T_{\text{model}}$. Thus, all deltas, gaps, and speedups are defined with respect to *FNN-all*. Note that ΔMAPE represents the *absolute* percentage point difference between a given model and the reference FNN (*i.e.*, $\mathrm{MAPE}_{\text{model}} - \mathrm{MAPE}_{\text{FNN-all}}$), whereas Gap (%) represents the *relative* change normalized by the baseline ($100 \times \Delta\mathrm{MAPE}/\mathrm{MAPE}_{\text{FNN-all}}$). While these two metrics are related, they provide complementary views: ΔMAPE captures the magnitude of the difference in absolute terms, while Gap (%) conveys the proportional significance of that difference relative to the baseline performance. Both are reported in our tables to support practitioners in interpreting the practical impact of each configuration. We also present the absolute MAPE of RF and GBT to contextualize how close the FNNs are to these baselines.

$$\Delta\mathrm{MAPE} = \mathrm{MAPE}_{\text{model}} - \mathrm{MAPE}_{\text{FNN,all}}, \tag{1}$$

$$\text{Gap (\%)} = 100 \cdot \frac{\Delta\mathrm{MAPE}}{\mathrm{MAPE}_{\text{FNN,all}}}. \tag{2}$$

Analyze the influence of feature selection (RQ_2). To isolate the effect of dimensionality reduction, we repeat the evaluation for each feature selection method and retention level (top 30%, 50%, and 70%). For every feature selection method m and retention r, we compute the change in error relative to using all features:

$$\Delta\mathrm{MAPE}(m, r) = \mathrm{MAPE}_{\text{FNN, }(m,r)} - \mathrm{MAPE}_{\text{FNN, all}}.$$

We summarize results per feature selection strategy by reporting the best configuration and the distribution across the top configurations. This addresses RQ_2 by showing when feature selection helps FNNs and which methods are the most effective.

Computational cost analysis (RQ_3) We quantify training cost as wall-clock time measured between the start and the end of the `fit` procedure, including data loading on the same hardware and software stack described in the experimental setup. Times are reported in seconds.

For each configuration, we record the training time and summarize it together with MAPE. We also report speedup relative to the best all-features FNN:

$$\mathrm{Speedup} = \frac{T_{\text{FNN, all}}}{T_{\text{model}}},$$

where T_{model} is the training time for the configuration under analysis. This addresses RQ_3 by quantifying the cost impact of feature selection and of different FNN settings, *i.e.* optimizers, losses, and activations. All experiments were conducted on dedicated hardware with no concurrent workloads, thereby minimizing external sources of timing noise (see Section 3.6).

To analyze accuracy versus cost, we produce scatter plots with MAPE on the vertical axis and training time on the horizontal axis. We highlight the Pareto frontier to identify configurations that are not dominated in both dimensions. Points on the frontier represent the most attractive trade-offs for practitioners. We discuss which feature selection methods tend to place models closer to this frontier and how FNNs compare to RF and GBT in the joint space of performance and time.

3.6 Instrumentation

The experiments were conducted on a server equipped with an NVIDIA RTX 3090 GPU (24GB GDDR6X), an Intel Xeon W-2275 CPU (8 cores/16 threads at 3.3 GHz), and 64 GB DDR4 ECC RAM. A 1 TB NVMe SSD stores datasets and logs, while Ubuntu 20.04 LTS provides the operating system environment. Our software stack included Python 3.12, PyTorch 2.x, PyTorch Lightning 1.x, scikit-learn, NumPy, Pandas, and Matplotlib/Seaborn.

4 Results

This section reports the experimental results, structured around the three research questions (RQs) introduced in Section 3.1. First, we evaluate the predictive performance of Feedforward Neural Networks (FNNs) in comparison with strong classical ML baselines—Random Forest and Gradient Boosting Tree—when trained on the full feature set without feature selection (RQ_1). Second, we assess the effect of feature selection (FS) strategies on FNN performance (RQ_2). In particular, we contrast four supervised, tree-based FS methods with a semantic, Word2Vec-based filtering technique that operates without labeled data. Third, we analyze the computational cost of FNN training under different FS methods and model settings (RQ_3). Here, *model settings* refers to the complete experimental setup, including the FS method and retention level of features (top 30%, 50%, or 70%), network architecture (layer depth and widths), and key hyperparameters such as optimizer, activation function, loss function, learning rate, and batch size.

Across all experiments, we report Mean Absolute Percentage Error (MAPE) as the primary measure of predictive accuracy, complemented by wall-clock training time to evaluate computational efficiency. All results are supported by tables and figures, organized by research question, to provide a clear and systematic evaluation.

4.1 Comparison of FNNs and ML Models (RQ_1)

We first evaluate whether FNNs can achieve predictive performance comparable to traditional ML models in the task of predicting Linux kernel *binary size*, focusing exclusively on the case *without* feature selection. In this setting, all models are trained on the complete set of 9,670 binary configuration options, which represents the most challenging scenario due to the high dimensionality of the feature space.

The "FNNs: Without FS" and "ML baselines: Without FS" sections of Table 1 summarize the results. The first column presents the name of the best performance feature selection method and the top-N% of *features* (*e.g.*, "XGBoost 50%" keeps the top half of the ranked features, about $4{,}835$ of the $9{,}670$ original kernel options). The second column ("Setting") reports the training *loss function* (*e.g.*, MAE, SmoothL1/Huber, MSE), the *optimizer* (Adam or AdamW), and the hidden-layer *activation* (ReLU, LeakyReLU, PReLU, ELU). The third and fourth columns report the *MAPE (%)* (lower values are better) and $\Delta\mathbf{MAPE}(m, r)$. The next columns report the *Speedup* and computational *time* in seconds (lower is better). Here, Speedup is the factor relative to the all-features top-1 FNN (2,775s). For example, *XGBoost 50%* runs in 2,311s and shows *1.20*$\times$ speedup; *XGBoost 30%* runs in 1,285s and shows *2.16*$\times$ speedup.

As shown in Table 1, the best-performing FNN configuration—using MAE loss, AdamW optimizer, and ELU activation—achieved a MAPE of 8.55%. This value serves as the reference baseline for FNNs without feature selection. Other FNN configurations yielded similar results, ranging from 8.60% to 9.70%, with only minor variations depending on the choice of optimizer, activation, and loss function.

When compared against the traditional ML baselines, RF obtained an error of 8.60%, essentially matching the best FNN (a marginal +0.05% difference). This suggests that FNNs can be at least as effective as tree-based ensemble methods in capturing non-linear relationships in the kernel configuration space, despite the large number of input features. In contrast, Gradient Boosting Tree (GBTree) achieved a substantially lower error of 7.02%, clearly outperforming all FNN configurations by more than 1.5 percentage points in MAPE.

This conclusion highlights two important insights: (1) FNNs can serve as a viable alternative to RF in raw prediction accuracy, even when operating directly on thousands of binary features. (2) The inability of FNNs to surpass GBTree suggests that high-dimensional inputs may dilute their modeling power, reinforcing the need for feature selection or dimensionality reduction to unlock their scalability benefits. Thus, these findings show that FNNs are competitive but not state-of-the-art, motivating subsequent investigations into feature selection strategies (RQ_2) and computational trade-offs (RQ_3).

> **(RQ_1, FNN performance without FS)** When trained on the full feature set, FNNs are competitive with RF

but underperform compared to GBTrees. These results indicate that traditional boosting ensembles remain the strongest choice in this context.

4.2 Impact of Feature Selection Methods on FNN Performance (RQ_2)

Here, we examine the effect of feature selection (FS) methods on the predictive accuracy of FNNs. The motivation arises from the high dimensionality of SPL configuration spaces, where noisy or redundant features can impair learning. By filtering out such features, FS can potentially improve generalization, mitigate overfitting, and reduce the computational burden of training.

To this end, we compared four supervised, tree-based methods (Decision Tree, Random Forest, GBTree, and XGBoost) with a semantic filtering strategy based on Word2Vec embeddings (see Section 3.4). The semantic approach operates without labeled data, relying instead on kernel documentation to select conceptually related configuration options.

Table 1 summarizes the top-performing FNNs settings with and without FS (*i.e.*, "All features"). Each row corresponds to a trained setup with the same preprocessing, split, and seeds. The results indicate that feature selection consistently improves FNN performance. In particular, the XGBoost-FS method at the 50% retention level produced the strongest FNN model (SmoothL1 loss, Adam optimizer, PReLU activation), achieving a MAPE of 8.26%. This represents an absolute improvement of 0.29 percentage points (3.39% relative) compared to the best all-features FNN (8.55%). Importantly, this gain in predictive accuracy is accompanied by reduced training time (see RQ_3), demonstrating that selective dimensionality reduction can yield more efficient models without sacrificing accuracy.

The observed hyperparameter patterns across the top FNN configurations provide additional insight into their suitability for the Linux kernel's high-dimensional configuration space. In Table 1, PReLU activations recur in the leading XGBoost-based feature-selection runs-*e.g.*, the top three setups at 50% retention (8.26%, 8.29%, 8.30% MAPE), which is consistent with PReLU's mitigation of the "dying ReLU" issue and its improved gradient flow for subtle non-linear interactions among options. SmoothL1 also appears in multiple top entries and often edges MAE among the best 50% retention runs, aligning with its stable gradients under large target variation. By contrast, among the all-features FNN settings, ELU dominates the best configurations, likely due to its exponential behavior that can speed up convergence in dense input spaces. Overall, these trends reinforce the importance of adaptive activations and robust losses when training FNNs for SPL prediction with complex, non-linear feature interactions.

The advantage of boosting-based selectors is consistent with prior work: ensemble boosting methods are particularly effective at ranking features that capture subtle dependencies and non-linear interactions, which appears to benefit neural models in high-dimensional configuration spaces. Among tree-based methods, XGBoost proved the most effective, contributing multiple runs among the top-5 FNN (with the lowest MAPE) configurations with FS.

The Word2Vec-based semantic approach, while not outperforming tree-based methods, still produced competitive FNN results. Its key advantage is that it does not require labeled performance data, making it a practical alternative when annotated datasets are scarce or costly to obtain.

Table 2 presents the top-10 *FNN* configurations obtained with Word2Vec-based feature selection, following the same layout as Table 1. The first column indicates the feature selection method and retention threshold, which in this case always corresponds to "Word2Vec 70%"—meaning that approximately 6,769 out of 9,670 configuration options were retained based on their semantic relevance in kernel documentation.

Overall, the Word2Vec-based method yielded higher error rates, with MAPE values ranging from 11.65% to 12.84%, which were noticeably lower than those of the tree-based approaches. Nonetheless, its distinctive strength lies in leveraging semantic relationships between configuration options, derived from natural-language descriptions, to drive the selection process. This makes it fundamentally different from supervised, performance-driven methods.

The best result was achieved with the setting using SmoothL1Loss, AdamW, and PReLU with Word2Vec, with the top-70% selected features (MAPE 11.65%). PReLU activations systematically present the top positions and outperform LeakyReLU and ReLU by roughly 0.9 to 1.2 percentage points in MAPE. Within the PReLU family, AdamW tends to edge out Adam, suggesting slightly better convergence behavior. For the loss function, SmoothL1 is marginally superior to MAE, reflecting its robustness to outliers.

It is also worth noting that, even after FS, FNNs did not surpass the best traditional ML baselines: RF reached a MAPE of 7.53%, while GBTree achieved 5.21%, the best overall performance. However, the results confirm that FS improves the robustness and efficiency of FNN, particularly when using boosting-based selection. Moreover, from a practical point of view, the Word2Vec-based approach offers an interpretable and computationally lightweight alternative, particularly useful in scenarios where domain knowledge is extensively documented and where labeled training data are scarce. Although it does not outperform supervised tree-based methods in predictive accuracy, it introduces a promising avenue for semantic-driven analysis in highly configurable systems. Word2Vec can serve as a label-free prefiltering step, providing a meaningful initial reduction of the feature space before applying sampling heuristics and more sophisticated prediction approaches.

(RQ_2, Influence of Feature Selection on FNNs) Feature selection (FS) reduced the dimensionality of the Linux kernel configuration space and, in several cases, improved the training efficiency of FNNs. Among the evaluated methods, supervised tree-based approaches, particularly XGBoost-based ranking, yielded the most effective feature subsets, resulting in the best FNN predictive performance. However, even with FS, FNNs did not surpass the accuracy of classical tree-based

Table 1. Summary of predictive performance and training efficiency across model configurations (RQ_1–RQ_3). Top-5 FNNs with and Top-5 FNNs without feature selection (FS), plus traditional ML baselines with and without FS. The "FNNs: Without FS" and "ML baselines: Without FS" sections address RQ_1 (FNN vs. ML comparison). The "FNNs: With FS" and "ML baselines: With FS" sections address RQ_2 (impact of feature selection). Training time and Speedup columns support RQ_3 (computational cost). $\Delta\mathrm{MAPE}(m, r)$ and Gap are relative to the best FNN without FS (8.55%). $\mathrm{Speedup} = 2775\,\mathrm{s}/T_{\text{model}}$.

	FS Method	Setting	MAPE (%)	$\Delta\mathrm{MAPE}(m,r)$	Gap (%)	Speedup	Time (s)
		FNNs: With FS					
1	XGBoost 50%	SmoothL1 / Adam / PReLU	8.26	-0.29	-0.29	1.20	2311
2	XGBoost 50%	SmoothL1 / AdamW / PReLU	8.29	-0.26	-0.26	1.21	2304
3	XGBoost 50%	MAE / AdamW / PReLU	8.30	-0.25	-0.25	1.17	2366
4	XGBoost 30%	SmoothL1 / AdamW / PReLU	8.36	-0.19	-0.19	2.16	1285
5	XGBoost 50%	MAE / Adam / PReLU	8.36	-0.19	-0.19	1.38	2014
		ML baselines: With FS					
1	RF (Top 300 features)	–	7.53	-1.02	-1.02	21.51	129
2	GBTree (Top 1,500 features)	–	5.21	-3.34	-3.34	0.74	3755
		FNNs: Without FS					
1	allFeatures	MAE / AdamW / ELU	8.55	0.00	0.00	1.00	2775
2	allFeatures	MAE / Adam / ELU	8.60	+0.05	+0.05	1.09	2543
3	allFeatures	SmoothL1 / AdamW / ELU	8.64	+0.09	+0.09	1.01	2757
4	allFeatures	SmoothL1 / Adam / ELU	9.09	+0.54	+0.54	1.02	2728
5	allFeatures	SmoothL1 / AdamW / PReLU	9.70	+1.15	+1.15	1.30	2135
		ML baselines: Without FS					
1	RF	–	8.60	+0.05	+0.05	21.51	4030
2	GBTree	–	7.02	-1.53	-1.53	0.74	19302

Table 2. Word2Vec-based feature selection results (RQ_2 and RQ_3). Top-10 FNNs using Word2Vec-based FS (70%). Same format as Table 1. $\Delta\mathrm{MAPE}(m, r)$ is relative to the best FNN without FS (8.55%). $\mathrm{Speedup} = 2775\,\mathrm{s}/T_{\text{model}}$. Gap (%) is relative to all-features FNN.

	FS Method	Setting	MAPE (%)	$\Delta\mathrm{MAPE}(m,r)$	Gap (%)	Speedup	Time (s)
1	Word2Vec 70%	SmoothL1Loss / AdamW / PReLU	11.65	+3.10	+3.10	2.01	1378
2	Word2Vec 70%	MAE / AdamW / PReLU	11.72	+3.17	+3.17	1.84	1511
3	Word2Vec 70%	MAE / Adam / PReLU	11.81	+3.26	+3.26	1.89	1467
4	Word2Vec 70%	SmoothL1Loss / Adam / PReLU	11.92	+3.37	+3.37	1.99	1393
5	Word2Vec 70%	SmoothL1Loss / AdamW / LeakyReLU	12.73	+4.18	+4.18	1.38	2005
6	Word2Vec 70%	MAE / Adam / LeakyReLU	12.74	+4.19	+4.19	1.52	1831
7	Word2Vec 70%	MAE / AdamW / LeakyReLU	12.79	+4.24	+4.24	1.58	1760
8	Word2Vec 70%	SmoothL1Loss / Adam / LeakyReLU	12.81	+4.26	+4.26	1.52	1824
9	Word2Vec 70%	SmoothL1Loss / AdamW / ReLU	12.82	+4.27	+4.27	1.44	1927
10	Word2Vec 70%	MAE / AdamW / ReLU	12.84	+4.29	+4.29	1.50	1848

ensemble learners, which consistently remained the strongest predictors overall. The Word2Vec-based semantic FS method provided a label-free alternative for early-stage scenarios without measured NFP data, but generally achieved lower accuracy and smaller efficiency gains than supervised FS approaches.

4.3 Predictive Performance and Training Efficiency Trade-offs (RQ_3)

In addition to predictive performance, computational efficiency is an important consideration in real-world SPL settings, where retraining may be frequent, and resources are constrained. Therefore, we analyzed the training time of the top-20 FNNs (see Tables 1 and 2) across different feature selection strategies and model settings to determine the associated computational costs.

Note that in our experimental protocol, each configuration was executed multiple times under the same hardware and software environment, using fixed random seeds and controlled preprocessing settings. However, the observed variance in training time was consistently low across repetitions, due to the controlled execution environment, deterministic preprocessing pipeline, fixed train/validation splits, and stable early-stopping settings. Thus, the values reported in Tables 1 and 2 correspond to the average measurements obtained from these repeated executions.

To aid in interpretation, Tables 1 and 2 report *Speedup (×)* and *Time (s)* side by side, where $\mathrm{Speedup} = T_{\text{all-features}}/T_{\text{model}}$ and $T_{\text{all-features}} = 2775\,\mathrm{s}$ are the results for the best FNN without FS. Models trained on all 9,670 features consistently required longer durations than those trained on selected subsets. For example, the best FNN XGBoost with the top-50% features ran in $2311\,\mathrm{s}$ ($\mathrm{Speedup} = 1.20\times$), while XGBoost with the top-30% features reached $1285\,\mathrm{s}$ ($\mathrm{Speedup} = 2.16\times$). In our top runs, reducing the feature set to 30% or 50% cuts wall-clock time by about 17% to 54% compared to the FNN with all features. Note that minor fluctuations are expected in any ML training process (*e.g.*,

due to GPU scheduling, cache effects, or stochastic optimization dynamics), while the relative differences between configurations remained stable across runs. Consequently, the comparative speedup trends reported here were preserved even when accounting for execution-to-execution variability.

Among the FS methods, XGBoost, while slightly more computationally demanding due to its complexity, offered the best balance of accuracy and time. Word2Vec-based models, in Table 2, despite their lower accuracy, consistently exhibited the lowest time cost, indicating their suitability for rapid real-time prediction and efficient deployment in resource-constrained environments.

With feature selection, the Random Forest was trained in 129 seconds, including both fitting the selector on the training split and training the Random Forest on the selected subset of features using the same splits and seeds adopted for the FNNs. For kernel size prediction, RF with FS is a strong baseline in terms of accuracy and cost. FNNs with XGBoost-based FS outperform all-feature FNNs and provide a viable neural alternative, although they did not surpass Gradient Boosting in our results.

Finally, regarding the optimizer choice, the results in our tables do not indicate a universal winner between Adam and AdamW. The best FNN with a supervised, tree-based selector (XGBoost at 50% retention) uses Adam and attains the lowest MAPE among the neural models, whereas the best configuration under the unsupervised, Word2Vec-based selection uses AdamW. In our runs, the training-time difference between Adam and AdamW was small and typically favored AdamW (12 times from a total of 20).

> **(RQ_3, FNN computational cost)** Feature selection (FS) reduced FNN training time and also improved accuracy slightly. Training in selected subsets yielded 1.2x to 2.16x speedup compared to the all-features FNN (2,311s at 50% and 1,285s at 30% vs. 2,775s). Among FS methods, XGBoost offered the best balance between accuracy and time for FNNs; Word2Vec had the lowest time cost but higher error. Random Forest with FS was the fastest baseline (129s), while GBTree remained the most accurate overall.

4.4 Threats to Validity

In this section, we identified potential threats to validity that could affect the study's results and conclusions according to Wohlin et al. (2012).

Construct Validity. Construct validity concerns whether the metrics used in the study accurately measure the concepts they are intended to represent. Our primary metric, MAPE, measures the average relative prediction error and is widely adopted in SPL performance prediction studies (Acher et al., 2022; Ha and Zhang, 2019a). However, MAPE can be sensitive to configurations with very small actual values, where small absolute errors produce disproportionately large percentage deviations. In our dataset, the minimum kernel size is 7 MB, which limits this effect but does not eliminate it entirely. We chose MAPE over alternatives such as RMSE or MAE because it provides an intuitive, scale-independent measure of prediction quality that facilitates comparison across studies. Nevertheless, future work could complement MAPE with additional metrics (*e.g.*, RMSE, MAE, or R^2). Another construct consideration is the use of wall-clock training time as a proxy for computational cost, which can be influenced by hardware variability and background processes. We mitigated this by running all experiments on the same dedicated server under controlled conditions.

Internal Validity. Results might depend on the choice of hyperparameters, *e.g.,* dropout rate and optimizer configuration. However, to mitigate this threat, we used the dropout rate suggested in the literature and well-known optimizers such as Adam and AdamW. Moreover, we used grid search for hyperparameter optimization. Another aspect is the potential overfitting during model training. Overfitting occurs when the model learns not only the underlying patterns but also the noise within the training data, leading to poor generalization of unseen data. While dropout was used as a regularization technique, the use of other regularization methods, such as weight decay and cross-validation, could further strengthen our analysis, which we plan to investigate in future studies.

External Validity. A primary threat to external validity is that our results may not generalize beyond the Linux kernel. We mitigate this threat by using the kernel as our object of study because it is a long-standing benchmark in SPL research, with thousands of options and rich real-world constraints, and it has been repeatedly employed in prediction studies and surveys of configurable systems (Ochoa et al., 2017; Pereira et al., 2021; Ha and Zhang, 2019a,b; Acher et al., 2022; Martin et al., 2022). This diversity and complexity make the kernel a robust platform for evaluating FNNs in configuration-driven prediction tasks, which provides a reasonable basis for exploring other systems.

Conclusion Validity. Conclusion validity concerns the reliability and reproducibility of the statistical conclusions drawn from the data. In our study, all models were trained and evaluated under identical conditions, including the same dataset splits, random seeds, preprocessing pipelines, and hardware, to ensure fair and reproducible comparisons. The use of a fixed 80/20 train/validation split with a deterministic seed controls for sampling variability across runs. However, we did not employ repeated random subsampling or k-fold cross-validation, which could provide more robust estimates of model performance variability. We partially address this limitation by reporting training time variability and by evaluating a large number of hyperparameter configurations through grid search, which provides insight into how sensitive the results are to modeling choices.

5 Conclusion and Future Work

This study investigated the effectiveness of *Feedforward Neural Networks* (FNNs) for performance prediction in

highly configurable systems, using the Linux kernel as a case study. We developed and evaluated FNNs using best practices in architecture design, regularization, and early stopping, aiming to balance predictive accuracy and training efficiency. Our analysis was structured around three research questions covering model accuracy, the role of feature selection, and computational cost.

The results from our first research question (RQ_1) demonstrated that traditional ML models achieved the highest predictive accuracy. However, FNNs combined with feature selection approached these results closely (RQ_2), with the best FNN reaching a MAPE of 8.26%, compared to 5.21% for Gradient Boosting. XGBoost for feature selection with FNN model used for prediction consistently delivered the best results among the evaluated strategies. Meanwhile, our exploration of Word2Vec as a semantic selection method highlighted its practicality: although less accurate than tree-based methods, it produced competitive results with lower training cost and without requiring labeled data, making it a viable alternative for contexts where semantic structure or data scarcity is a concern. This represents a noteworthy accuracy-versus-data-scarcity trade-off: Word2Vec-based feature selection requires zero compiled samples to rank features, whereas tree-based methods depend on a pre-existing dataset of measured NFP values to compute importance scores. In early-stage SPL engineering, where compiling thousands of kernel configurations to generate labeled ground truth is computationally prohibitive, the Word2Vec approach provides a practical and cost-effective starting point for dimensionality reduction. This confirms that DL models, while not outperforming ML models, remain competitive and offer modeling flexibility for large configuration spaces.

Our third research question (RQ_3) emphasized the impact of feature selection on computational efficiency. Models trained with all 9,670 features exhibited training times between 1,900 and 2,800 seconds, while those with selected subsets typically trained in nearly half the time. Random Forest remained the fastest overall (~120 seconds), but FNNs trained on selected features offered a more flexible trade-off between cost and model complexity. Optimizer choice (*e.g.*, Adam vs. AdamW) also presented a measurable influence on training time.

Overall, this study highlights the role of feature selection not only in improving predictive accuracy but also in enabling practical, scalable training of FNNs in high-dimensional spaces. By combining FNNs with structured selection strategies, such as XGBoost or Word2Vec, FNNs can achieve competitive performance for configuration-related prediction tasks.

As future work, we plan to investigate a two-stage feature selection pipeline in which Word2Vec acts as a semantic pre-filter before applying more sophisticated sampling, selection or learning methods. First, Word2Vec would serve as a label-free pass, ranking configuration options by their conceptual proximity to the target NFP using only documentation, without requiring compiled configurations or measured performance data. In a second stage, we envision integrating a graph-based representation of the configuration space, derived from the structural constraints encoded in KConfig (*e.g.*, dependencies, selections, and exclusions). Software product line management tools (Pereira et al., 2016a, 2014, 2013) and strategies (Pereira et al., 2018a,b, 2017, 2016b) could be leveraged to ensure that the reduced feature set respects structural validity while also limiting the space of candidate configurations to those involving semantically relevant options. Thus, new sampling techniques could be proposed and compared with other sampling techniques from Bessa et al. (2025) and Pereira et al. (2020).

Supervised feature ranking or performance modeling could then be applied within this constrained subspace. Because the search space is already pruned semantically and structurally, this stage would require significantly fewer labeled configurations. This two-stage design makes the accuracy-versus-data-scarcity trade-off actionable: practitioners with no compiled samples start with Word2Vec alone, and progressively incorporate supervised ranking as labeled data for configurations including relevant features become available. Such a progressive strategy aligns with real-world SPL workflows, where measurement budgets are limited.

Moreover, we aim to explore transfer learning techniques to improve generalization across different systems, system variants, and software versions. Performance models are often trained on a specific version of a system and may degrade significantly when applied to newer releases or related systems due to evolving feature sets, shifting dependencies, and changes in implementation. This phenomenon, sometimes referred to as concept drift or configuration drift, limits the long-term utility of learned models and increases the need for repeated costly measurement. To mitigate this issue, we plan to investigate whether shared semantic representations of configuration options obtained from documentation or learned embeddings can provide a stable cross-version feature space that facilitates knowledge reuse even when individual options are added, removed, or renamed.

ARTIFACT AVAILABILITY

We have made all artifacts from our study publicly available in the study's repository at `https://github.com/aisepucrio/jserd2026-FNN4SPL`.

Acknowledgements

This research was partially funded by the Brazilian funding agencies CAPES/COFECUB (Grants 88881.879016/2023-01 and Ma1036/24) and FAPESP (Grant 2023/00811-0). We also acknowledge the Brazilian company Stone[2] for their financial support.

References

Acher, M., Martin, H., Lesoil, L., Blouin, A., Jézéquel, J.-M., Khelladi, D. E., Barais, O., and Pereira, J. A. (2022). Feature subset selection for learning huge configuration spaces: The case of Linux kernel size. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference (SPLC)*, pages 85–96. ACM.

[2] `https://www.stone.com.br/`

Acher, M., Martin, H., Pereira, J. A., Blouin, A., Jézéquel, J.-M., Khelladi, D. E., Lesoil, L., and Barais, O. (2019). Learning very large configuration spaces: What matters for Linux kernel sizes. Technical Report RR-9286, Inria.

Baldi, P. and Sadowski, P. J. (2013). Understanding dropout. *Advances in Neural Information Processing Systems*, 26:2814–2822.

Barron, J. T. (2019). A general and adaptive robust loss function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4331–4339. IEEE.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Bessa, J. M., Cavalcanti, M., Acher, M., Endler, M., and Pereira, J. A. (2025). Unveiling the impact of sampling on feature selection for performance prediction in configurable systems. In *Proceedings of the International Conference on Software and Systems Reuse (ICSR)*, pages 1–12. IEEE.

Bessa, J. M. and Pereira, J. A. (2023). A unified repository of product lines measurements. GitHub repository. `https://github.com/jualvespereira/Dataset_ML4ConfigSys`. Accessed: 15 Jun. 2023.

Biau, G. and Scornet, E. (2016). A random forest guided tour. *TEST*, 25(2):197–227.

Borges, H., Pereira, J. A., Khelladi, D. E., and Acher, M. (2025). Linux kernel configurations at scale: A dataset for performance and evolution analysis. In *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–10. ACM.

Chai, T. and Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? Arguments against avoiding RMSE in the literature. *Geoscientific Model Development*, 7(3):1247–1250.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–794. ACM.

Cheng, J., Gao, C., and Zheng, Z. (2023). HINNPerf: Hierarchical interaction neural network for performance prediction of configurable systems. *ACM Transactions on Software Engineering and Methodology*, 32(2):1–30.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs). In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*.

Gong, J. and Chen, T. (2025). Deep configuration performance learning: A systematic survey and taxonomy. *ACM Transactions on Software Engineering and Methodology*, 34(1):1–52.

Guo, J., Czarnecki, K., Apel, S., Siegmund, N., and Wąsowski, A. (2013). Variability-aware performance prediction: A statistical learning approach. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 301–311. IEEE.

Ha, H. and Zhang, H. (2019a). DeepPerf: Performance prediction for configurable software with deep sparse neural network. In *Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 1095–1106. IEEE.

Ha, H. and Zhang, H. (2019b). Performance-influence model for highly configurable software with Fourier learning and Lasso regression. In *Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 470–480. IEEE.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034. IEEE.

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

Huang, L., Zhang, B., and Liu, Y. (2021). A systematic study of early stopping for deep learning. *IEEE Access*, 9:120501–120512.

Kim, M., Notkin, D., and Grossman, D. (2007). Automatic inference of structural changes for matching across program versions. In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 333–343. IEEE.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kotsiantis, S. B. (2013). Decision trees: A recent overview. *Artificial Intelligence Review*, 39(4):261–283.

Li, Z., Zhang, Z., Xu, R., Ding, Z., and Tao, D. (2022). DeepFS: An end-to-end deep feature selection model for tabular data.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

Ma, Y., Huang, Q., Wang, J., Xu, K., Goldblum, M., and Goldstein, T. (2022). Overfitting in adversarially robust deep learning.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML), Workshop on Deep Learning for Audio, Speech and Language Processing*.

Martin, H., Acher, M., Pereira, J. A., Lesoil, L., Jézéquel, J.-M., and Khelladi, D. E. (2022). Transfer learning across variants and versions: The case of Linux kernel size. *IEEE Transactions on Software Engineering*, 48(11):4274–4290.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26 (NeurIPS)*, pages 3111–3119.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814.

Nasrabadi, N. M. (2007). Pattern recognition and machine learning. *Journal of Electronic Imaging*, 16(4):049901.

Natekin, A. and Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7:21.

Ochoa, L., Pereira, J. A., González-Rojas, O., Castro, H., and Saake, G. (2017). A survey on scalability and performance concerns in extended product lines configuration. In *Proceedings of the 11th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 5–12. ACM.

Pereira, J. A., Acher, M., Martin, H., and Jézéquel, J.-M. (2020). Sampling effect on performance prediction of configurable systems: A case study. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 277–288. ACM.

Pereira, J. A., Acher, M., Martin, H., Jézéquel, J.-M., Botterweck, G., and Ventresque, A. (2021). Learning software configuration spaces: A systematic literature review. *Journal of Systems and Software*, 182:111044.

Pereira, J. A., Constantino, K., and Figueiredo, E. (2014). A systematic literature review of software product line management tools. In *Proceedings of the 13th International Conference on Software Reuse (ICSR)*, pages 73–89. Springer.

Pereira, J. A., Krieter, S., Meinicke, J., Schröter, R., Saake, G., and Leich, T. (2016a). FeatureIDE: Scalable product configuration of variable systems. In *Proceedings of the 15th International Conference on Software Reuse (ICSR)*, pages 397–401. Springer.

Pereira, J. A., Maciel, L., Noronha, T. F., and Figueiredo, E. (2017). Heuristic and exact algorithms for product configuration in software product lines. *International Transactions in Operational Research*, 24(6):1285–1306.

Pereira, J. A., Matuszyk, P., Krieter, S., Spiliopoulou, M., and Saake, G. (2016b). A feature-based personalized recommender system for product-line configuration. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE)*, pages 120–131. ACM.

Pereira, J. A., Matuszyk, P., Krieter, S., Spiliopoulou, M., and Saake, G. (2018a). Personalized recommender systems for product-line configuration processes. *Computer Languages, Systems & Structures*, 54:451–471.

Pereira, J. A., Schulze, S., Krieter, S., Ribeiro, M., and Saake, G. (2018b). A context-aware recommender system for extended software product line configurations. In *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 97–104. ACM.

Pereira, J. A., Souza, C., Figueiredo, E., Abilio, R., Vale, G., and Costa, H. A. X. (2013). Software variability management: An exploratory study with two feature modeling tools. In *Proceedings of the 7th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, pages 20–29. IEEE.

Raschka, S., Liu, Y., and Mirjalili, V. (2022). *Machine Learning with PyTorch and Scikit-Learn*. Packt Publishing.

Rokach, L. (2015). *Data Mining with Decision Trees: Theory and Applications*. World Scientific, 2nd edition.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1 — learning rate, batch size, momentum, and weight decay.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer.

Zhang, C., Bengio, S., and Singer, Y. (2021). Understanding deep learning requires rethinking generalization. *Communications of the ACM*, 64(3):107–115.