# Simplifying Machine Learning Models: Balancing Complexity and Performance

Zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften

**(Dr. rer. pol.)**

von der KIT-Fakultät für
Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

M.Sc. Nils Koster

Tag der mündlichen Prüfung: 17. März 2025

Referent: Prof. Dr. Fabian Krüger
Korreferentin: Prof. Dr. Melanie Schienle

Karlsruhe 2025

# Acknowledgments

Zuerst möchte ich mich bei Prof. Krüger für die Möglichkeit zur Promotion bei ihm und seine hervorragende Betreuung bedanken. Mit seiner Geduld, seinem Verständnis für jegliche Situation, privat oder beruflich, und Demut vor dem Fach war er mir immer ein Vorbild in der wissenschaftlichen Welt. Mit viel Hilfe, Unterstützung und toll harmonierender Zusammenarbeit hat er mich durch mein Doktorandenleben begleitet und hoffentlich auch darüber hinaus. Weiterhin möchte ich bei Prof. Grothe bedanken, der mir seit meiner Masterarbeit immer mit gutem Rat und Witz zur Seite stand. Auch möchte ich Prof. Schienle danken, die mich stets gefördet und in langatmigen Zeiten mit ihrer lockeren Art motiviert hat. Bei Martin Stražar, Adit Radhakrishnan, und Caroline Uhler vom MIT und Broad Institute möchte ich mich für die einmalige Möglichkeit bedanken, mit ihnen an einer der besten wissenschaftlichen Einrichtungen zusammengearbeitet haben zu dürfen. Die Erfahrungen und Erkenntnisse, die ich in dieser Zeit gewonnen habe, sind für mich von unschätzbarem Wert. Dem Karlsruhe House of Young Scientists (KHYS) danke ich für die großzügige Förderung des Aufenthalts. Weiterhin möchte ich mich bei allen Kollegen und Kolleginnen für die freundliche, lockere und unterstützende Atmosphäre im Büro bedanken. Besonders erwähnt seien hier Lisa Leimenstoll, Tobias Biegert, Lotta Rüter, Friederike Becker und Andreas Eberl in denen ich Freunde gefunden habe. Sie haben mir die Zeit im Büro mit anregenden Diskussionen über die Welt und sehr humorvollen Gesprächen in der Mittagspause stets versüßt (wortwörtlich). Ihre Unterstützung bei meiner Arbeit, schätze ich sehr. Sebastian Lerch und Johannes Bracher möchte ich für die gute Zusammenarbeit in verschiedensten Formaten und insbesondere Sebastian für seine geduldige Unterstützung danken.

Besonderer Dank gilt auch meiner Familie und Freunden, besonders meinen Eltern. Sie haben mich unterstützt wo und wie es ging, von der Grundschule (damals hakte es bei der Umrechnung von Einheiten) bis zum heutigen Tag. Sie hatten Verständnis für stressige Zeiten und entsprechende Nerven. Auch bei meinen Großeltern bedanke ich mich ausdrücklich für die liebevolle Unterstützung. Fabian Kächele möchte ich für viele interessante und inspirierende fachliche und nicht-fachliche Gespräche danken. Ich möchte mich auch bei Cassandra bedanken, die mich lange Zeit durch mein Studium und die Promotion begleitet hat. Sie hat mich mit aller Kraft unterstützt, auch wenn ihre Last selbst am größten war. Zuletzt danke ich von Herzen Clara. Ohne dich wäre ich nicht wo ich jetzt stehe. Mit allumfassender Geduld, Freundlichkeit und Liebe fürs Detail hast du mich in allen Lebenssituationen gestützt, ob die Arbeit frustrierend war, die kleinsten Fehler gefunden werden wollten oder das Privatleben durcheinandergewürfelt war. Danke!

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **AE** | absolute error |
| **BNN** | binarized neural networks |
| **CDF** | cumulative distribution function |
| **CNN** | convolutional neural network |
| **CRPS** | continuously ranked probability score |
| **DAX** | German stock index (Deutscher Aktienindex) |
| **DWD** | German weather service (Deutscher Wetterdienst) |
| **ELU** | exponential linear unit initialization |
| **ELUS** | exponential linear unit scaled initialization |
| **EMOS** | ensemble model output statistics |
| **FNR** | false negative rate |
| **FPR** | false positive rate |
| **HMDB** | Human Metabolome Database |
| **KIT** | Karlsruhe Institute of Technology |
| **LC-MS/MS** | liquid chromatography – tandem mass spectrometry |
| **MS/MS** | tandem mass spectrometry |
| **NN** | neural network |
| **NTK** | neural tangent kernel |
| **NWP** | numerical weather processing |
| **QRF** | quantile regression forest |
| **RF** | random forest |
| **SE** | squared error |
| **SOEP** | German socioeconomic panel |
| **TNN** | ternarized neural networks |
| **TNR** | true negative rate |
| **TPR** | true positive rate |

# 1 Introduction

The increased popularity and success of machine learning models were significantly driven by a recent surge in computing power and related technological advancements (Nordhaus, 2007; Denning and Lewis, 2016). Most prominently, neural networks (NNs) have achieved remarkable results by being able to extract unique information from huge data sets in numerous fields of application such as natural language processing (OpenAI et al., 2023), weather forecasting (Bi et al., 2023; Kochkov et al., 2024), or life sciences (Jumper et al., 2021). The prevailing consensus is that, with abundant data, larger, more complex and flexible models outperform simpler, conventional statistical or machine learning models (Belkin et al., 2018a). While the 'bigger is better'-approach may work well for some problems, it remains unclear whether large models generally outperform simpler ones on small or mid-sized datasets (Grinsztajn et al., 2022; Hollmann et al., 2025). Additionally, NNs' high parameter count and complexity make them difficult to interpret (Szegedy et al., 2014), leading to issues like vulnerability to adversarial attacks (Moosavi-Dezfooli et al., 2017; Ghorbani et al., 2019). Although post-hoc techniques aim to shed light on the inner workings of models, they themselves can be challenging to understand (e.g., Lundberg and Lee, 2017; Mahendran and Vedaldi, 2015; Molnar, 2022). These considerations lead to the following question: *Is it possible to simplify models while maintaining predictive performance?*

Recent literature has analyzed this question from a theoretical perspective, deriving convergence rates or error bounds based on parameter count (e.g., Schmidt-Hieber, 2020; Kohler and Krzyżak, 2021; Braun et al., 2024, and literature cited therein). Methodologically, pruning is an established technique for creating more parsimonious models and has recently gained renewed interest, especially in NNs (Cheng et al., 2024). However, pruning is less common in other model classes like gradient-boosted trees (Friedman, 2001) or random forests (RFs) (Breiman, 2001). This work contributes to the growing body of literature focused on reducing model complexity from a methodological perspective without compromising performance. Defining model complexity – or its counterpart,

simplicity – in a way that is both concise and universally applicable is challenging due to the many factors involved. For example, the parameter count of NNs is often used to quantify model complexity, whereas other models, such as tree-based methods, do not have comparable parameters to count.

In this work, we either simplify or work with simple models from different perspectives, loosely focusing on four broad aspects. Usually, *performance* is heavily prioritized in model development. We argue that, while good performance is important and necessary for any model, there are additional aspects that should be considered. Models should further be *accessible and applicable*, i.e., being user-friendly, and minimizing hardware and monetary constraints. Also, models should be *interpretable*, meaning that it should be easy to comprehend how a prediction is formed, which inputs are most influential, or which training samples influence a prediction. A fourth facet regarding the success and practical application of simple models is an effective *communication* of the fact that simple models can be more useful than their complex counterparts, especially given the ongoing trend towards increased complexity. Both complex and simple models are tools that must be used appropriately depending on the task at hand. These four aspects will be relevant to varying degrees throughout this work.

This thesis is divided into two parts. The following two chapters focus on developing methods to simplify existing machine learning models in two different ways. The subsequent two chapters focus on the application of straightforward models and the communication of the 'choose your model as simple as possible'-approach. More specifically, Chapter 2 introduces signed Supermasks, which train neural networks through sparsification, effectively facilitating a better understanding of their workings. In Chapter 3, we present a method to enhance interpretability in RFs and extensions thereof, which are already easily applicable and widely accessible models, by simplifying the forests' forecast. Motivated by a problem in computational biology, in Chapter 4 we apply kernel ridge regression, a simple and well-understood machine learning method, to efficiently predict mass spectra of molecules outperforming existing literature. Finally, in Chapter 5, we illustrate both the concept and insights of an innovative lecture in which students learn about easily applicable probabilistic forecasting and evaluation methods in a fun, competitive, and real-time setting.

The contributions of each chapter are outlined and presented in more detail in the following paragraphs. In Chapter 2, we introduce signed Supermasks, which involve training an NN by deciding whether a particular weight in the network is important or not. The algorithm is based on the lottery ticket hypothesis, which posits that an NN contains a smaller subnetwork that performs as well as or better than the original network (Frankle and Carbin, 2019). Previous work (Zhou et al., 2019) suggests that the signs of weights may be important for good Supermask performance. Therefore, we enable the network to flip the respective signs of weights. Further, we develop a fitting initialization scheme to account for the degree of sparsity and the activation function. Results on well-known data sets show that the resulting sparsity exceeds 90% for most architectures while matching the performance of the original network, thereby empirically showing that only few parameters of an NN are needed. Furthermore, results indicate that the actual parameter values are less relevant than the combination of features and their subsequent interactions. These results provide a foundation for subsequent investigations into understanding why and how NNs work by simplifying their structure. The chapter is joint work with Oliver Grothe and Achim Rettinger *published* at the *International Conference on Learning Representations 2022*.

In Chapter 3, we present an algorithm which simplifies (probabilistic) predictions of RFs and quantile regression forests (Meinshausen, 2006). The algorithm is based on the fact that an RF's mean prediction can be represented as a weighted sum of training observations. Similarly, a probabilistic prediction can be represented as a weighted empirical cumulative distribution function (CDF) using the same weights. Oftentimes, an RF's predictions are dominated by a few observations with large weights accompanied by many observations with small weights. By restricting the prediction to use the $k$ largest weights, this method is similar to pruning an RF as a whole, but not its individual trees. This allows us to interpret and communicate the prediction from an unconventional but valuable perspective: Which training observations contribute to the final prediction, and to which training sample is the current case being compared to? We can interpret a weighted empirical CDF with $k$ support points as a list of scenarios with predicted probabilities, making the model's prediction easier to understand. We test this idea on various data sets and find that even for values as small as $k = 10$, performance is comparable, sometimes even improved. We support our empirical results with a stylized

analytical model, characterizing cases where our method performs well. This chapter is based on joint work with Fabian Krüger.

Chapter 4 addresses a problem in computational biology related to compound identification. Tandem mass spectrometry (MS/MS) is a method used to show the presence of compounds in a biological sample, such as blood, by fragmenting the detected compounds. The result is a mass spectrum unique to each detected compound. However, MS/MS can only record the abundance and mass of fragments; the compounds themselves remain unknown. The accepted method to identify the corresponding compound is often complicated, time-consuming, and costly. Motivated by this problem, we develop models to predict a mass spectrum for a given compound and compare it to the observed spectrum. We apply kernel ridge regression, a popular machine learning method, on a dataset containing spectra produced by MS/MS and corresponding compounds. By utilizing the so-called neural tangent kernel (NTK) (Jacot et al., 2018), we emulate an infinitely wide NN. With a closed-form solution available, this method is faster, simpler to train, less demanding on computer hardware, and more accessible compared to existing NN approaches. We test various modeling approaches and outperform related work in all considered tasks. The chapter is based on joint work with Adityanarayanan Radhakrishnan, Julian Avila-Pacheco, Clary B. Clish, Martin Stražar, Ramnik J. Xavier, and Caroline Uhler.

The applicability, accessibility, and interpretability of forecasting models become increasingly important when dealing with problems in real-time. Therefore, we developed a university course format where students learn about these concepts in probabilistic forecasting in a playful, real-time, and real-world setting, which we describe in Chapter 5. The task was to produce quantile predictions for two target variables. These predictions were part of a challenge where each participant's prediction performance was evaluated weekly. An analysis of their results shows that simple, easy-to-use models, which are not prone to user errors, performed best over the entire 14-week period. The findings presented in this chapter are based on joint work with Johannes Bracher, Fabian Krüger, and Sebastian Lerch *published* at the *The American Statistician* in 2024.

# 2 Signing the Supermask: Keep, Hide, Invert

This chapter is based on joint work with Oliver Grothe and Achim Rettinger, *published* at the *International Conference on Learning Representations* (Koster et al., 2022). It presents an algorithm that trains an NN by sparsifying it with fixed weights. Permission to reuse the article, as well as the figures, in this work is granted by the copyright holder.

## 2.1 Introduction

In recent years, NNs have established themselves as versatile problem solvers far beyond the field of machine learning. However, the performance increases achieved are obtained by an even larger growth in their size; hundreds of billions of parameters as of today. This not only leads to growing energy costs (Strubell et al., 2019), but also complicates interpretability considerably. There are different approaches to reduce various aspects of complexity within an NN, which broadly can be categorized either as pruning, i.e., the technique of trimming down NNs, or as low-precision training, i.e., representing the weights by low precision numbers. Pruning has shown to significantly reduce the size of an NN (in terms of 'active' weights) without affecting performance considerably once the training phase has ended. Compared to the original network, the pruned network is sparse, since many of the original, 'useless' weights are set to zero. In contrast, replacing weights by low precision numbers reduces memory size, but does not reveal a sparse, likely easier to interpret network structure.

Many approaches of pruning exist to identify 'useless' weights (e.g., Janowsky, 1989; LeCun et al., 1989; Han et al., 2015; Li et al., 2017; Molchanov et al., 2019, among others). This raises the question of why not train a smaller network from scratch, since the training of a large network is the computationally most expensive step and pruning is another additional step. Frankle and Carbin (2019) present a possible solution to this

question by formulating the lottery ticket hypothesis (LTH), which states that any densely connected NN contains smaller subnetworks that, when trained in isolation, perform just as well or even better than the original network. They are able to identify such subnetworks (*winning tickets*) by iteratively pruning the network based on the magnitude of each weight. For deeper architectures, Frankle et al. (2019) show that pruning after a few epochs of normal training works significantly better over pruning at initialization.

Zhou et al. (2019) follow the seminal idea of the LTH and are able to train NNs by only selecting *untrained* weights (i.e., weights are frozen after initialization), a concept they call *Supermasks*. In other words, they find a smaller subnetwork during training without adjusting the weights themselves. Although this approach did not match the performance of their baselines and the pruning rate is inconsistent, it revealed a startling insight: weight values do not seem to be as important as the connection itself; a single, well-initialized value for each layer is sufficient. Ramanujan et al. (2020) further develop Supermasks by modifying the way masks are calculated, which leads to significant performance improvements compared to Zhou et al. (2019). However, the number of parameters is still high. Recently, Chijiwa et al. (2021) modified the approach of Ramanujan et al. (2020) by randomizing the scores, but are only able to prune their networks more than 60%.

In this chapter, we propose a technique called **signed Supermask**, a natural extension of Zhou et al. (2019) and Ramanujan et al. (2020). We not only determine the importance of a weight by masking, but also learn the respective sign of a weight. Signed Supermasks aim at very sparse structures and are able to uncover subnetworks with far fewer parameters in the range of 0.5% to 4% of the original network, requiring little additional computational effort without sacrificing performance.

This differs substantially from low precision training. In its most extreme form, low-precision training quantizes the weight values of an NN to three constants, including zero or two constants, excluding zero. There, binarized neural networkss (BNNs) (Courbariaux et al., 2015) reduce complexity but not the size of the networks in terms of sparsity of weights. Ternarized neural networkss (TNNs), introduced by Li et al. (2016), allow in principle for sparse subnetworks, however this literature focuses almost exclusively on optimizing computational costs while maintaining predictive performance. For TNNs to reduce computational complexity, sparsity (and interpretability) is of no importance, as they are focused on reducing the computational footprint only. The literature presents diverse approaches, for example Shayer et al. (2018) utilize the local reparameterization

trick (Kingma et al., 2015) to learn ternarized weights stochastically, Zhu et al. (2017) ternarize the weights but scale them layer-wise by two learned real-valued scalars, and Alemdar et al. (2016) employ a teacher-student approach. Deng and Zhang (2020) train their networks normally with an additional regularization term in the loss function and only ternarize at the end of training by rounding. To the best of our knowledge, this is the only work on TNNs also reporting on sparsity. More recently, Diffenderfer and Kailkhura (2021) combined the edge-popup algorithm by Ramanujan et al. (2020) and the binarization of weights and achieve good results. Thus, although TNNs and Supermasks in general appear similar at first glance, their goals are different: while the former attempt to reduce computational complexity, the latter attempt to find the smallest possible subnetworks within an NN to better understand NNs in general and work towards facilitating interpretability.

This chapter takes on the Supermasks perspective and our experiments show that signing the Supermask matches or outperforms baselines and state-of-the-art approaches on Supermasks and leads to very sparse representations. A convenient side effect of signed Supermask is the ternarization of weights, with implied reduced memory requirements (Li et al., 2016) and further significant speedup in inference (Hidayetoglu et al., 2020; Brasoveanu et al., 2020). In summary, signed Supermasks provide two major advantages. First, the network structure is simplified while maintaining or improving the performance compared to their dense counterpart. Second, the reduction facilitates a better understanding of the inner mechanics of NNs. Based on that, we might be able to build smaller but equally powerful models a priori. Additionally, once trained, signed Supermask models can be stored more efficiently and have the potential for faster inference.

## 2.2 Signed Supermask

The idea of a signed Supermask is simple, but elegant: the network may not only switch off a given weight, but may also flip its sign if deemed beneficial. We do this by multiplying each weight with a mask $m \in \{-1, 0, 1\}$, in contrast to $m \in \{0, 1\}$ as in previous work (Zhou et al., 2019; Ramanujan et al., 2020).

For example, if a single weight is initialized with a negative sign but the performance of the network would increase if it was positive, the signed Supermask has the additional degree of freedom to flip the weight's sign. This step decreases the impact of random

initialization, which matters specifically for a *signed constant initialization*, as introduced by Zhou et al. (2019) in the context of Supermasks. Despite including -1 in $m$, we have no additional information to store for the final sparse weight matrices (i.e., the final mask multiplied with the weight matrix), since it already includes the weight value, the sign and the zero. In this regard, the additional flexibility in the training phase comes for free.

Note that signed Supermasks are applicable to any NN architecture. By masking, we neither alter the information flow, nor any architecture specific properties. For the sake of brevity, however, we focus in the following on the application on feed-forward NNs only.

The general setup is as follows: Let $\boldsymbol{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$ denote the frozen weight matrix of layer $l \in \{1, \dots, L\}$ of a feed-forward NN with $L$ layers and $n_l$ denoting the number of neurons in layer $l$. We initialize $\boldsymbol{W}_l \sim \mathcal{D}_w$. Let further $\boldsymbol{M}_l \in \mathbb{R}^{n_l \times n_{l-1}}$, $\boldsymbol{M}_l \sim \mathcal{D}_m$ be a learnable, real-valued matrix that is transformed to values $\{-1, 0, 1\}$ by a element-wise function $g$, where

$$g(\boldsymbol{M}) = \begin{cases} -1, & \boldsymbol{M} \leq \tau_n \\ 0, & \tau_n < \boldsymbol{M} < \tau_p \\ 1, & \boldsymbol{M} \geq \tau_p \end{cases} \tag{2.1}$$

for some thresholds $\tau_n$ and $\tau_p$ as well as appropriate distributions $\mathcal{D}_w$ and $\mathcal{D}_m$ that will be specified below. We now define the signed Supermask as $\bar{\boldsymbol{M}} \coloneqq g(\boldsymbol{M})$, i.e., a Supermask as in Zhou et al. (2019) with the additional degree of freedom of a negative sign. Further note that, compared to Zhou et al. (2019), the function $g(\cdot)$ is deterministic. In order to make full use of the *signed* Supermask, we utilize the ELU activation function (Clevert et al., 2016) with hyperparameter $\alpha$. Hereafter, we always set $\alpha = 1$. Then, with $\odot$ symbolizing the Hadamard product, each layer $l$'s output $\boldsymbol{o}_l$ with $\boldsymbol{z}_l = (\boldsymbol{W}_l \odot \bar{\boldsymbol{M}}_l)^T \boldsymbol{o}_{l-1}$ is computed as follows:

$$\boldsymbol{o}_l = \begin{cases} \boldsymbol{z}_l, & \boldsymbol{z}_l > 0 \\ e^{\boldsymbol{z}_l} - 1, & \boldsymbol{z}_l \leq 0. \end{cases} \tag{2.2}$$

The parameters $\tau_{n,p}$ are hyperparameters that need to be tuned: the larger they are in absolute value, the more weights are pruned at initialization and the larger values in $\boldsymbol{M}$ need to grow during the course of training to 'activate' its responding weight. The values of $\tau_{n,p}$ might differ depending on the task and network architecture at hand. The gradient is estimated using the straight-through estimator (Hinton, 2012; Bengio et al.,

2013) as in previous work (Zhou et al., 2019; Ramanujan et al., 2020). That is, in the backward-pass, we regard the threshold function $g(\cdot)$ as the identity function. Thus, the gradient is going 'straight through' $g(\cdot)$.

Zhou et al. (2019) generate their masks by sampling the entries from Bernoulli distributions with learnable Bernoulli probabilities. Their $\boldsymbol{M}$ therefore resembles the probability, that weight $w_{ij}$ in some layer $l$ is active. Sampling from a Bernoulli distribution can also be interpreted as a stochastic threshold function. Our **fixed threshold** approach follows the same intuition with the difference of not using a stochastic sampling method. That is, we define two thresholds $\tau_n$ and $\tau_p$ after the initialization phase which remain fixed during the course of training in contrast to Ramanujan et al. (2020) who update $\tau_n$ and $\tau_p$ after each epoch such that the pruning rate stays constant. Having fixed thresholds has three main advantages which are in line with the argumentation of Zhou et al. (2019): first, the network can choose the optimal number of remaining weights. Second, it can choose the optimal distribution of 1's and $-1$'s. Third, this approach is fast, as we only have to compute $\tau_n$ and $\tau_p$ once. Moreover, this method, with some modifications, is frequently used in the TNN literature, which underlines its legitimacy. As a result, an exact pruning rate cannot be determined a priori, but is learned. However, we show in Section 2.3.2, that the pruning rate can be indirectly influenced by adjusting other hyperparameters.

### 2.2.1 Thoughts on Initialization

Signed Supermasks do not alter the value of any weight during training. Therefore, it is especially important to initialize weights appropriately. Common weight initialization methods such as He (He et al., 2015) or Xavier (Glorot and Bengio, 2010) neither take masking into account nor the ELU activation function.

**Weights** We propose a weight initialization scheme that considers both aforementioned alterations, namely, the signed masking process and a differing activation function. In simple terms, since each layer holds fewer weights, we can scale the weight value to counteract masking. Considering that the goal of initialization methods is to keep the variance in the forward- and backward-pass constant, we suggest to use the following variance:

$$\mathrm{Var}[\boldsymbol{W}_l] = \frac{1.5}{n_l(1-p_0^l)}, \tag{2.3}$$

with fan-out $n_l$ and layer-wise equal, initial probabilities $p_0^l = P(\bar{\boldsymbol{M}}_{ij}^l = 0)$ for layer $l \in \{1, ..., L\}$ and $\forall i \in 1, \ldots, n_l, j \in 1, \ldots, n_{l-1}$. A detailed formal motivation of Equation 2.3 under specified assumptions can be found in Appendix A.1. The result is still valid without masking, i.e., $p_0^l = 0$. He et al. (2015) come to a similar conclusion for their parametric ReLU (PReLU) activation function.
Henceforth, we refer to this initialization method as exponential linear unit scaled initialization (ELUS) with masking or simply exponential linear unit initialization (ELU) in case of no masking. Notice that ELUS is simply a scaled version of He initialization (He et al., 2015), where the scalar is only dependent on the hyperparameter $\alpha$ of the ELU activation function which is known before training (and set to 1 in this work).

**Mask** Our assumptions and observations regarding the initialization of the real-valued mask $\boldsymbol{M}$ are as follows: We assume that $\mathcal{D}_m$ is symmetric around zero, a standard assumption for common initialization distributions. Given the properties of $\boldsymbol{M}$, $\bar{\boldsymbol{M}}$ and the gradient straight-through estimator, it is advantageous if the values in $\boldsymbol{M}$ and $\boldsymbol{W}$ are roughly of the same magnitude, such that the gradient, which is independent of $\boldsymbol{M}$, is neither too small nor too large. In other words, $\mathcal{D}_m$ should be chosen such that it is symmetrical around zero and has a similar variance as $\mathcal{D}_w$. In the case of mask initialization, we advice against a signed constant approach but recommend a normal or uniform distribution instead.

## 2.3 Empirical Assessment

The following paragraphs present the experiments conducted in order to assess the validity and effectiveness of our fixed threshold signed Supermasks. The code for the experiments can be found here: `https://github.com/kosnil/signed_supermasks`.

**Experimental Setup** To ensure comparability, we utilize the same experimental setup as Zhou et al. (2019) and Frankle and Carbin (2019) i.e., models FCN, Conv2, Conv4 and Conv6, VGG-like convolutional neural network (CNN) architectures (Simonyan and Zisserman, 2015), with the additional Conv8 model of Ramanujan et al. (2020) without batch normalization (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014). Since we want to study the functionality of signed Supermasks in particular, any extra functionalities would disturb from the essential topic even though it might bump

performance. Details about the aforementioned model architectures can be found in Table A.1 in Appendix A.2. Furthermore, we study signed Supermasks on ResNet20, ResNet56 and ResNet110 (He et al., 2016) with the exception of replacing ReLU with ELU as the activation function (similar to Shah et al. (2016)). The batch normalization layers in the ResNet models are frozen for signed Supermask training and are only used because its standard practice. For all experiments the same architectures as the respective signed Supermask models but trained conventionally, act as baselines.

The fully-connected model FCN is trained on MNIST (LeCun et al., 2010) and all CNNs plus ResNet20 on CIFAR-10 (Krizhevsky, 2009). We only apply per image standardization on both of those data sets. ResNet56 and ResNet110 are trained on CIFAR-100 (Krizhevsky, 2009), where we apply per image standardization, pad the image with 4 pixels on each side and randomly crop a 32x32 image out of the original or its horizontally flipped counterpart.

We ran each experiment 50 times (ResNet56/ResNet110: 3 runs) with the same setup but different weights and masks drawn from the respectively same specified distribution. This ensures that our experimental results are not affected by the randomness of the initialization step. To obtain an intuition of the scale of memory reduction, we provide a simple metric by comparing the average network size in bytes stored in `numpy` arrays and scipy's `csr_matrix`[1] sparse matrices after training. For matrices with more than 2 dimensions, we reshape the matrix to two dimensions. This does not give an exact result but provides a good heuristic. Since we attempt to reduce complexity of a neural network, we also provide the average time needed to train a model for a single epoch to further assess practicability.

**Initialization** To initialize the weights of the signed Supermask models, we use signed constants (i.e., the respective standard deviation) in combination with Xavier, He and ELUS. Thus, the respective weight value has a 50% probability of being negative, where the absolute value of the constant is influenced by the respective initialization method. We initialize the baseline models with the same method within a uniform distribution as the respective signed Supermask model. The idea of using only one (well-chosen) signed constant is particularly compelling from the perspective of complexity reduction. For the ELUS initialization, we scale a He initialization by $\sqrt{3}$ for FCN and Conv architectures

[1] The choice of `csr` is arbitrary. The purpose of this metric is not to show how we can most efficiently store the networks but to give an intuition on added value.

**Table 2.1: FCN: Test accuracy and training time.** We report the mean, 5% and 95% quantiles of 50 runs for each initialization method as well as the results of Zhou et al. (2019) and Ramanujan et al. (2020). Significance of test accuracy with a p-value $< 0.01$ is indicated with '**' and a p-value $< 0.05$ with '*'. Mean training time per epoch is reported in the last column.

| | Init | Test Accuracy [%] | Rem. Weights [%] | TT / Epoch [s] |
|---|---|---|---|---|
| *Baseline* | He | 97.40 [97.3, 97.5] | 100 | 1.23 |
| | ELU | 97.42 [97.3, 97.5] | 100 | 1.21 |
| | Xavier | 97.43 [97.4, 97.5] | 100 | 1.21 |
| *Signed Supermask* | He | 97.12 [97.0, 97.3] | 4.93 [4.9,5.0] | 1.36 |
| | ELUS | 97.48 [97.3, 97.7]* | 3.77 [3.7,3.8] | 1.27 |
| | Xavier | 96.89 [96.8, 97.0] | 5.51 [5.5,5.6] | 1.32 |
| *Zhou et al.* | Xavier (S.C.) | 98.0 | 11 - 93 | ? |

and $\sqrt{1.5}$ for our ResNets, which returned best results in the preliminary testing phase. All masks are initialized with Xavier uniform initialization. Results of experiments with different mask initializations and distributions can be found in Appendix A.3.

### 2.3.1 Fully-Connected Network

This section evaluates the performance of the FCN signed Supermask model. Hyperparameters used for training the FCN are listed in Tables A.2 and A.4 in Appendix A.2. Note that compared to Zhou et al. (2019), the parameter choices are in line with standard practice.

Table 2.1 summarizes the average test accuracy, remaining weights and training time per epoch with the respective 5% and 95% quantiles for both baseline and signed Supermask models with the different weight initializations. All models achieve similar test accuracy. Best performing is the ELUS signed Supermask model. Furthermore, all signed Supermask models have a wider confidence interval, indicating higher variance. In contrast, confidence intervals for the ratio of remaining weights are very small, demonstrating constant pruning rates throughout all runs. Furthermore, ELUS' ratio of remaining weights of 3.77% is considerably lower than for the other two initializations. This indicates a superior initialization as the network is able to achieve a higher performance with fewer weights remaining. For another angle of observation, Figure A.3 in Appendix A.3 visualizes the average ratio of remaining weights over the course of training for each weight initialization. Regarding efficiency, ELUS signed Supermask models require about 5% more training time. However, the obtained compression rate for the ELUS model

after training is 92.01% on average, which makes up for the additional training effort. Although the performance of the ELUS signed Supermask is trailing 0.5pp behind the FCN model of Zhou et al. (2019), we argue that considering the pruning rate (they state the pruning rate only imprecisely, ranging from 7% to 89%) and performance, signed Supermasks are advantageous.

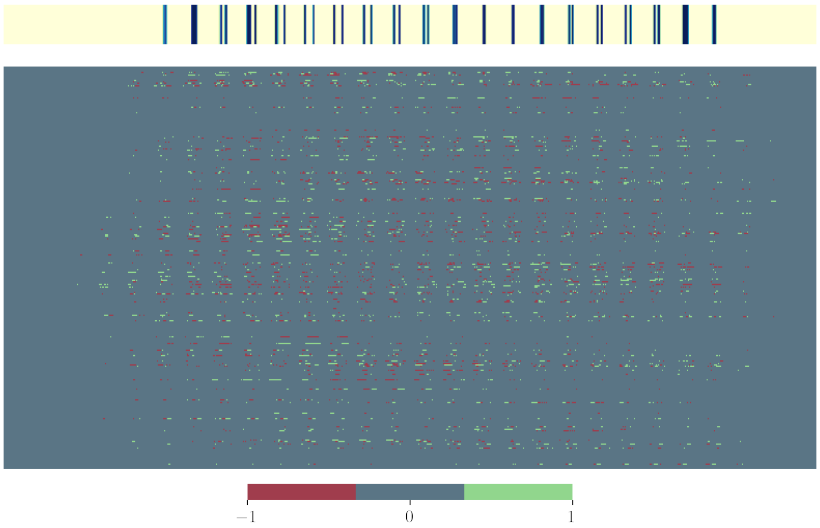



**Figure 2.1: FCN signed Supermask: First layer mask.** The upper, yellow-tinted figure shows a randomly chosen (in this case the digit '8') flattened MNIST sample. The blue-tinted heatmap visualizes the first-layer mask of a FCN. Each line represents a single pixel where non-yellow lines represent the pixels containing a part of the handwritten digit. The lower figure visualizes the first mask of a randomly picked and trained ELUS FCN model. We can clearly see the alignment of the 'digit-pixels' in the MNIST sample and the non-zero mask elements.

To assess how the model interprets a given data set, we examine the first masking layer of the network, visualized in Figure 2.1. The top image represents an input vector (i.e., a flattened MNIST sample), with each vertical line representing one pixel. The bottom shows the mask of the first layer of a randomly selected trained ELUS FCN model. The signed Supermask has learned which of the input nodes are more important than others. Those that match a background pixel are completely masked, while the input nodes that receive actual information are allowed to let the information pass through. Thus, the first layer acts as a filter and feature selector. We further analyze the behavior of the masks and draw comparisons to binary Supermasks in Appendix A.3.

The results so far show the potential of signed Supermasks in relatively small architectures. Not only does the ELUS model outperform the respective baselines with only 3.8% of the original weights, it also saves 92% of memory once trained. Furthermore, we gain insights into layer interpretability as pruning is not evenly distributed across the network.

### 2.3.2 Convolutional Networks

Here, we present the main experimental results of CNN signed Supermasks. Overfitting was a major issue in the development, especially for the baseline models. To counter that, the baseline models are only trained for 50 epochs. In contrast, signed Supermask models were not as prone to overfitting. However, once we anticipated overfitting tendencies of Conv2, we reduced its learning rate compared to the other CNN models. Apart from those alterations, the hyperparameter choices are the same throughout all CNN architectures. All models are optimized with SGD with weight decay. Learning rate for Conv models are exponentially reduced over the course of training. Tables A.2 and A.4 in Appendix A.2 summarize all training parameters. We have found in initial experiments that $\pm 0.01$ is a solid choice for the threshold parameters $\tau_n$ and $\tau_p$. This corresponds to an initial overall pruning rate of about 11% to 30%.Thus, larger layers initially experience a slightly higher pruning rate than smaller layers, following the intuition that weights in smaller layers impact the output of the network to a greater extent than weights in larger layers.

Table 2.2 presents test accuracy for our baselines, the models of Zhou et al. (2019) and Ramanujan et al. (2020) and signed Supermasks as well as remaining weights (i.e., $1 -$ pruning rate) for the latter three. For a broader comparison to literature, see Table A.8 in Appendix A.3. While we were able to reproduce the results of Zhou et al. (2019), this was not the case for Ramanujan et al. (2020) which not only delivered worse results than reported but also demanded a multiple of the training time. The interested reader can find additional results, experiments on different weight and initialization methods as well as investigations on extreme pruning rates in Appendices A.3 A.4, A.5 and A.6. First, apart from Conv2, all CNN models significantly outperform their respective baselines. Conv2 reaches similar accuracy to its baseline, the difference only being 0.4pp. The more our models grow in depth, the greater the benefit of signed Supermasks over the baselines. When considering the ratio of remaining weights, a likely reason for the slight

**Table 2.2: CNN signed Supermask: Test accuracy and remaining weights.** We report the mean, 5% and 95% quantiles of 50 runs for the ELUS signed Supermasks and baselines. The respective best results of Zhou et al. (2019), Ramanujan et al. (2020) (abbreviated 'Ram') and Diffenderfer and Kailkhura (2021) (abbreviated 'Diff') are shown as well. Evidently, signed Supermasks outperform the previous literature on Supermasks on all CNN architectures, taking our extreme pruning rates of at least 97% into account. Apart from Conv2, we also gain higher performance than the baseline models.

| | Accuracy [%] | | | | | Rem. Weights [%] | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Baseline | Zhou | Ram. | Diff. | Sig. Supermask | Zhou | Ram. | Diff. | Sig. Supermask |
| Conv2 | 68.79 [68.4, 69.2] | 66.0 | 65 | 70 | 68.37 [67.7, 69.0] | 11 - 93 | 10 | 10 | 0.60 [0.58, 0.62] |
| Conv4 | 74.50 [74.7, 75.3] | 72.5 | 74 | 79 | 77.40 [76.7, 78.3] | 11 - 93 | 10 | 10 | 2.91 [2.9, 3.0] |
| Conv6 | 75.91 [75.4, 76.4] | 76.5 | 77 | 82 | 79.17 [78.1, 80.5] | 11 - 93 | 10 | 10 | 2.36 [1.9, 2.6] |
| Conv8 | 72.24 [71.4, 73.0] | - | 70 | 85 | 80.91 [79.9, 81.7] | - | 10 | 10 | 1.17 [1.1, 1.2] |

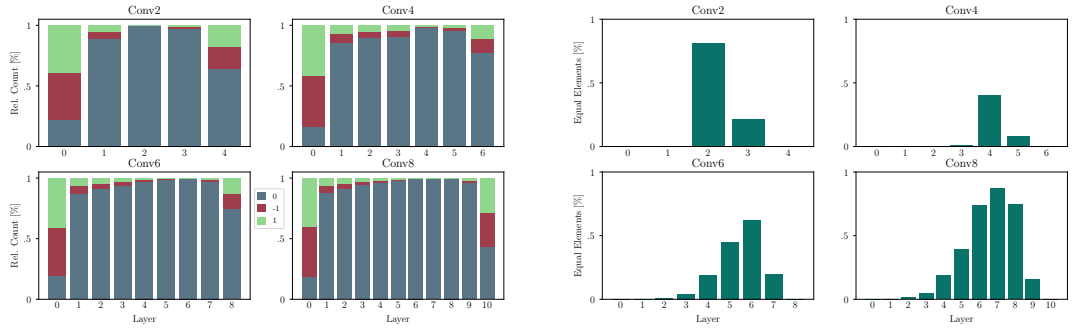
performance glitch of Conv2 becomes apparent: it utilizes only 0.6% of the original weights. In other words, Conv2 reaches similar performance with 99.4% of the original weights unused. Conv4, the smallest model in terms of absolute parameter count, achieves its performance while keeping the highest percentage of weights in relation to the original count. As the networks grow in size and depth, we can once again note a reduction of the weights remaining in relation to the original number. Except for Conv6, there is hardly any variance at all in this metric, indicating a robust behavior.

**Table 2.3: CNN signed Supermask: Additional training time and compression rate** compared to the respective baselines. Signed Supermask CNN models train roughly 6 to 10% longer, however, once trained, their required memory can be reduced by at least 93.8%. Training time is abbreviated 'TT' below.

| | Add. TT/Epoch [%] | Compression Rate [%] |
|---|---|---|
| Conv2 | 10.14 | 98.41 |
| Conv4 | 6.02 | 93.82 |
| Conv6 | 6.25 | 95.07 |
| Conv8 | 6.00 | 97.6 |

In terms of efficiency, we see in Table 2.3 that the signed Supermask models require 6% to 10% more training time, which can be accounted to the additional calculations necessary. However, once the networks are trained the required memory can, on average, be compressed by 94% to 98%. Signed Supermasks outperform not only their respective baselines but also the methods proposed in the related work. Compared to Zhou et al. (2019), our Conv2, Conv4 and Conv6 models achieve a higher performance of at least 2pp. Diffenderfer and Kailkhura (2021) do not report performance over a 90% pruning

rate, however, their models achieve higher accuracy than ours. We account this to their lower pruning rate. Ramanujan et al. (2020) do not report model performance below a pruning rate of 90% as well. Indeed, in this case, all our Conv models outperform their equivalent models by 3 to 10 pp, while uncovering much smaller subnetworks.




**(a)** Average ELUS mask distribution. Sparsity is very high in very wide layers, indicating that only few of those weights are needed.

**(b)** Equality of all 50 final ELUS masks. Masks show high element-wise equality where sparsity is highest.

**Figure 2.2: CNN signed Supermask: Mask distribution and equality.**

Figure 2.2a helps to interpret the masks by visualizing the average layer-wise mask distribution of each architecture. It can be clearly seen that the first layers are least affected by pruning, while all other layers reach a significantly higher pruning rate of over 99%. The effect in the first layer can be explained by the concept of weight sharing in convolutional layers. As the layer size increases, the pruning rate also increases and peaks in the largest layer of the network. The output layers are also relatively sparse, again indicating proper initialization. This finding confirms our previous assumption that larger layers contain far fewer weights that are essential to the network. The relative element-wise equality of the final masks over all 50 runs for all layers and architectures is shown in Figure 2.2b. This provides an intuition about the similarity of the masks and therefore subnetworks through different runs. We observe that pairwise equality is strongly correlated with the size and sparsity of a layer: the larger, the sparser, the more alike. To some degree this is expected as very sparse layers are alike by chance. However, measuring similarity in matrices is a very difficult task (for an overview on this literature from a graph-based perspective see e.g., Emmert-Streib et al., 2016), which is why we provide visualizations in Figure A.7 in Appendix A.3 that emphasize especially the similarity in structure. This shows, that signed Supermasks repeatedly find strikingly

similar subnetworks for different runs, meaning that it might be possible to identify certain neurons that are more important than others and hence better understand neural networks in general.

### 2.3.3 Residual Networks

To further investigate the effectiveness of signed Supermasks, we turn our attention towards residual networks which allow for deeper architectures. The investigate residual networks (ResNets) consisting of {9,27,54} residual blocks, each in equal parts with 16, 32 and 64 filters, respectively. This sums up to a comparatively small parameter count of roughly 0.25M, 0.84M and 1.7M. In the following, we refer to those as ResNet20, ResNet56 and ResNet110, respectively. As hypothesized above, signed Supermasks work better in deeper architectures that contain enough parameters to find a good subnetwork. To investigate this, we multiply the number of filters by 1.5, 2, 2.5 and 3 to create wider variants of ResNet20. Note that even the largest of those variants is smaller than any of the studied VGG-like architectures above, suggesting less overparameterization, especially for the more complex CIFAR-100 data set. Hyperparameters for baseline and signed Supermask models are depicted in Tables A.3 and A.5 in Appendix A.2. Compared to the other investigated architectures, we only reduce the learning rate once the train loss hits a plateau. Furthermore, since there is no large deviation between layer sizes, we set the threshold parameters $\tau_n$ and $\tau_p$ such that only 25% of the weights are left for ResNet20 and 15% for ResNet56 and ResNet110.

Table 2.4 shows the test accuracy and remaining weights of both baselines and signed Supermasks trained on CIFAR-10. Notably, signed Supermask always trail 2 to 3 pp behind their respective baseline. However, once the signed Supermask model gets wide enough it can reach the performance of the original ResNet20 with only a third of the parameter count. This finding is partially in line with Ramanujan et al. (2020) and their scaling of ResNet50. As expected, the pruning rate increases as the network grows in depth. We suspect that weight count could be further dropped in the deep architectures by adapting the weight decay parameter, for the sake of clarity however, we decided to keep all learning parameters the same. A possible reason for the slightly lower performance of signed Supermasks might be the frozen batch normalization layers, which would benefit learning greatly. We investigate this aspect in the next section. The performance and remaining weights of ResNet56 and ResNet110 on CIFAR-100 is

**Table 2.4: ResNet20 signed Supermask: Test accuracy and remaining weights on CIFAR-10.** We report the mean, 5% and 95% quantiles for the ELUS signed Supermasks and ELUbaselines. ResNet20 signed Supermasks are not able reach the performance of the baselines. However, the wider the ResNet20s become, the closer they get to the respective baseline. Furthermore, with a multiplier of 2.5 and 3, the signed Supermasks reach the performance of the original baseline with a third of the original weights. The wider the architectures get, the higher the pruning rate.

| | Accuracy [%] | | Rem. Weights | |
|---|---|---|---|---|
| | Baseline | Sig. Supermask | Baseline | Sig. Supermask |
| ResNet20 | 84.91 [84.39, 85.56] | 81.68 [81.03, 82.62] | 248 624 | 53 530 (21.13%) |
| ResNet20x1.5 | 86.18 [85.59, 86.74] | 83.76 [83.01, 84.31] | 558 600 | 64 223 (11.93%) |
| ResNet20x2.0 | 86.80 [86.39, 87.21] | 84.42 [83.67, 85.26] | 992 352 | 77 280 (7.69%) |
| ResNet20x2.5 | 87.08 [86.72, 87.42] | 84.71 [84.26, 85.32] | 1 549 880 | 84 446 (5.42%) |
| ResNet20x3.0 | 87.32 [87.00, 87.67] | 84.89 [84.36, 85.48] | 2 231 184 | 91 798 (4.06%) |

depicted in Table 2.5. Again, signed Supermask show that they successfully learn from the data set, even in deep and relatively small environments. However, they cannot reach the same test accuracy as their respective baseline. This can be partially explained by the relatively small original network sizes which were reduced even more. Furthermore, as hypothesized above, batch normalization is of great importance especially in deep architectures like ResNet110, which thereby particularly suffered.

**Table 2.5: ResNet56/110: Test accuracy and remaining weights on CIFAR-100.** We report the mean, estimated 5% and 95% quantiles for the ELUS signed Supermasks and ELUbaselines as well as the utilized weights. Signed Supermask models trail behind their baseline, however, utilizing a maximum of 29% of the original weights.

| | Accuracy [%] | | Rem. Weights | |
|---|---|---|---|---|
| | Baseline | Sig. Supermask | Baseline | Sig. Supermask |
| ResNet56 | 68.04 [67.84, 68.24] | 60.01 [59.51, 60.52] | 834 992 | 245 116 (29.39%) |
| ResNet110 | 62.70 [54.80, 68.61] | 46.42 [46.31, 46.51] | 1 668 272 | 346 757 (20.64%) |

### The Role of Batch Normalization

ResNets excel compared to conventional CNNs because of their depth. A big role in being able to learn very deep architectures is batch normalization. Here, we want to study the influence of batch normalization on the ResNet signed Supermask architectures. For those experiments, we explicitly turn on the batch normalization layers in the respective architecture. This parts a bit from the minimalist idea of Supermasks in

general, however, weights are still frozen at initialization and this step might be necessary for deep architectures.

Table 2.6 reports the results of the baseline models (as displayed above) and the signed Supermask with learnable batch normalization layers. Compared to the pure signed Supermask models, batch normalization helps to push performance while increasing pruning rate for all models. However, they are unable to reach the performance of the baselines. Nevertheless, taking the remaining weights into account, the predictive performance still impresses. More research is needed to investigate the performance gap. Besides this, the results underline that batch normalization helps signed Supermasks to gain better performance in very deep architectures while utilizing fewer weights.

**Table 2.6: ResNet signed Supermask: Influence of batch normalization**. Test accuracy and remaining weights on CIFAR-10 for ResNet 20 and CIFAR-100 for ResNets 56 and 110. We report the mean, estimated 5% and 95% quantiles for the ELUS signed Supermasks and ELUbaselines as well as the utilized weights. Batch normalization is abbreviated 'BN' in the following.

| | Accuracy [%] | | Rem. Weights | |
|---|---|---|---|---|
| | Baseline | Sig. Supermask BN | Baseline | Sig. Supermask BN |
| ResNet20 BN | 84.91 [84.39, 85.56] | 83.38 [82.87, 83.92] | 248 624 | 37 992 (15.54%) |
| ResNet56 BN | 68.04 [67.84, 68.24] | 64.24 [63.90, 64.58] | 834 992 | 231 769 (27.59%) |
| ResNet110 BN | 62.70 [54.80, 68.61] | 53.67 [53.63, 53.72] | 1 668 272 | 64 663 (3.86%) |

## 2.4 Summary

In this work, we introduced the concept of signed Supermasks. Extending the original work on Supermasks (Zhou et al., 2019), we include the sign in the masking process and utilize a simple step-function to calculate the effective masks. Additionally, we gave theoretical and experimental justification for an adapted weight initialization scheme to the masking process, called **ELUS**.

In contrast to TNNs (Li et al., 2016), signed Supermasks heavily focus on sparsity, which allows them to train and simultaneously prune the network with little additional training effort, but also gain valuable insights into the general functionality of NNs. As a side effect, the trained signed Supermask models' memory footprint can be compressed by up to 98%. Our experiments on MNIST and CIFAR-10 show, that signed Supermask models at best hold as little as 0.6% of the original weights, while performing on an equal

or superior level compared to our baselines and current literature (Zhou et al., 2019; Ramanujan et al., 2020). Further analyses on CIFAR-100 with ResNet56 and ResNet110 suggest that signed Supermasks are generally able to work on complex data sets and architectures as well. Moreover, by allowing the networks to learn batch normalization parameters, signed Supermasks improve their performance while increasing the pruning rate.

# 3 Simplifying Random Forests' Probabilistic Forecasts

This chapter is based on joint work with Fabian Krüger and was already presented at the *International Symposium on Forecasting 2024* in Dijon (France). It presents an algorithm that simplifies the (mean and probabilistic) forecast's of an RF such that it is easier to interpret while maintaining predictive performance.

## 3.1 Introduction

Many statisticians agree that forecast distributions are preferable to mere point forecasts. Correspondingly, an active literature is concerned with making statistical forecast distributions in meteorology (e.g. Rasp and Lerch, 2018), economics (e.g. Krüger et al., 2017), energy (e.g. Taieb et al., 2021), epidemiology (e.g. Cramer et al., 2022), and other fields. Nevertheless, point predictions still dominate in many practical settings in policy, business, and society. As argued by Raftery (2016), the cognitive load that forecast distributions impose upon their users may be an important bottleneck impeding their adoption. Motivated by this possibility, we consider simplifying the forecast distributions produced by random forests (RFs; Breiman, 2001; Meinshausen, 2006), and study how simplification affects statistical forecasting performance. While our main focus is on probabilistic forecasting, the method we propose can also be used to simplify point forecasts for the mean.

More specifically, we approximate an RF forecast distribution for a continuous scalar outcome by a discrete distribution with $k$ support points, where $k \in \{1, 2, \ldots, n\}$ is a user-determined parameter and $n$ denotes the number of training samples. The resulting forecast distribution can be cast as a collection of $k$ scenarios, each occurring with a specified probability. If $k$ is sufficiently small, this setup can effectively be communicated to non-statisticians. For example, Altig et al. (2022) use it to survey business executives

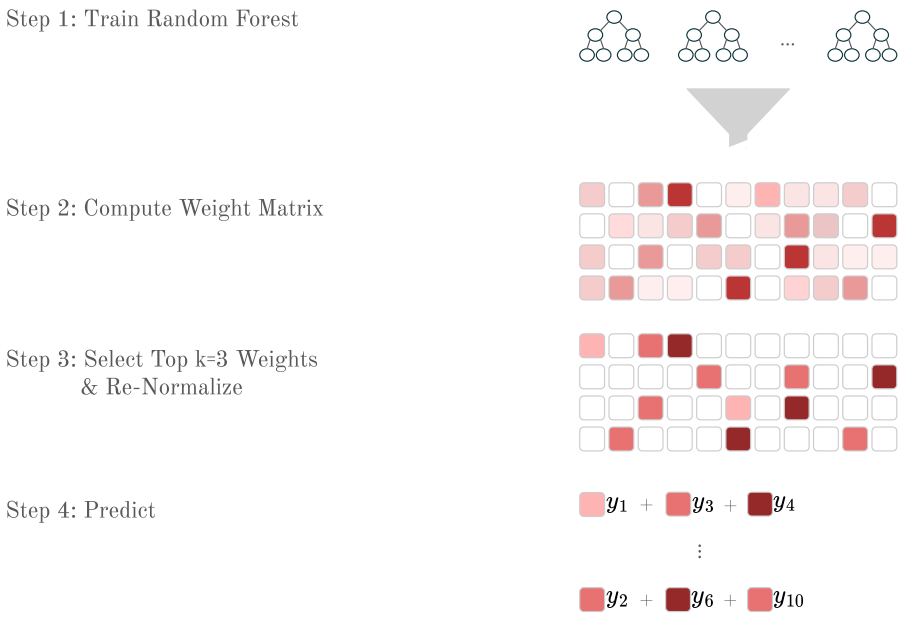



**Figure 3.1: Summary of proposed method.** This schematic description summarizes the method proposed in this chapter. First, a standard RF is trained. Second, from the trained RF, we compute a weight vector for each new test case. Collecting the weight vectors for multiple test cases results in a matrix, with rows corresponding to test cases and columns corresponding to training cases. Next, we select the top $k$ (in this case $k = 3$) weights for each test case, re-normalize such that the weights again sum up to one and set the remaining weights to zero. These weights can now be used for prediction, as illustrated here for the first and fourth test cases.

about firm outcomes like future sales growth and employment. Abbas and Howard (2015, Chapter 35) provide a textbook discussion from a decision analysis perspective.

Our approximation builds upon the fact that RFs can be cast as a nearest-neighbor-like method (Lin and Jeon, 2006; Meinshausen, 2006): The forecast distribution for a test sample observation with feature vector $\boldsymbol{x}_0$ is a discrete distribution with support points given by the training sample outcomes $(y_i)_{i=1}^n$ and corresponding probability weights $(w_i(\boldsymbol{x}_0))_{i=1}^n$. Meinshausen (2006) called these forecast distributions quantile regression forests (QRFs). In the following, we do not distinguish between QRFs and RFs since the former can be constructed from the latter without any additional model fitting. As detailed in Section 3.2.1, the weight $w_i(\boldsymbol{x}_0)$ reflects the similarity between $\boldsymbol{x}_0$ and $\boldsymbol{x}_i$, the feature vector of the $i$-th training sample observation. Consider a given test case $\boldsymbol{x}_0$ and the set $\mathcal{I}_k(\boldsymbol{x}_0)$ containing the indices of the $k$ largest weights for this test case, where

$k \in \{1, 2, \ldots, n\}$. We then set

$$\tilde{w}_i(\boldsymbol{x}_0) = \begin{cases} \frac{w_i(\boldsymbol{x}_0)}{\sum_{j \in \mathcal{I}_k(\boldsymbol{x}_0)} w_j(\boldsymbol{x}_0)} & \text{if } i \in \mathcal{I}_k(\boldsymbol{x}_0) \\ 0 & \text{else.} \end{cases}$$

That is, we retain only the $k << n$ largest weights, and re-scale them such that they sum to one. All other weights are set to zero. Figure 3.1 provides a schematic description. Setting $k = n$ recovers the weights of the initial RF's mean and the QRF's distribution forecast, which is described in more detail in Section 3.2.[1] Larger values of $k$ correspond to a larger number of scenarios. This increases the complexity of the forecast distribution, but may be beneficial in terms of statistical forecasting performance.

**Practical Illustration** We provide a practical illustration of our method by considering labor market survey data provided by DIW (2022). The data set is an openly available subsample of the German socioeconomic panel (SOEP; Goebel et al., 2019). It is recorded between 2015 and 2019, covering 21 851 observations corresponding to 5847 distinct persons after preprocessing. Many participants were asked repeatedly over the survey years, so that the data has a panel structure. Our goal here is to predict a person's annual salary in Euros based on socioeconomic regressors such as age, education, family status, and the industry in which the person is employed. We use data from 2015–2018 for training and 2019 for testing, and consider a Top$k$ model with $k = 5$ along with a full RF. Compared to the full RF, the Top$k$ model is about 6% worse in terms of distribution forecasting performance (as measured by the CRPS, introduced in Section 3.2.2), and about 3% better in terms of point forecasting performance (as measured by the SE). Appendix B.1 provides details on the data set and RF forecasting performance.

To illustrate, consider a fully employed 47-year-old male, living alone and working in the financial sector. We aim to predict his salary in 2019. A full RF predicts a mean income of 84 705, placing non-zero weight on 171 training sample observations. The simplified Top5 model utilizes only the five largest of these 171 weights which (before re-normalization) sum up to 0.47. To re-normalize, we simply multiply the five remaining weights by 1/0.47. The five support points (the training sample observations that correspond to the remaining weights) are listed in Table 3.1, together with their weights. With those,

[1] Prediction as in Equations 3.3 and 3.4 is unaltered, except for the substitution of $w_i(\boldsymbol{x}_0)$ by $\tilde{w}_i(\boldsymbol{x}_0)$.

**Table 3.1: SOEP test prediction.** The table shows the five support points $\boldsymbol{x}_{s1}$ to $\boldsymbol{x}_{s5}$ for a test point (first row, $\boldsymbol{x}_0$) in the Top5 model. 'Empl.' abbreviates the variable 'employed', 'F' abbreviates 'full employment' and 'Unk.' stands for 'unknown'. Sector ID 64 corresponds to 'Provision of financial services'. The corresponding (rounded) weight of each support point is depicted in the penultimate column, while we report the respective income in the last column.

| | Survey Year | Female | Age | No. Per. | No. Child. | Years Educ. | Empl. | Sector ID | $w_i(\boldsymbol{x}_0)$ [%] | Income |
|---|---|---|---|---|---|---|---|---|---|---|
| $\boldsymbol{x}_0$ | 2019 | False | 47 | 1 | 0 | 14.5 | F | 64 | - | 83 279 |
| $\boldsymbol{x}_{s1}$ | 2017 | False | 45 | 1 | 0 | 14.5 | F | 64 | 30.6 | 94 903 |
| $\boldsymbol{x}_{s2}$ | 2018 | False | 45 | 1 | 0 | 18 | F | 64 | 24.6 | 79 206 |
| $\boldsymbol{x}_{s3}$ | 2017 | False | 44 | 1 | 0 | 15 | F | 64 | 17.3 | 100 433 |
| $\boldsymbol{x}_{s4}$ | 2016 | False | 44 | 1 | 0 | 14.5 | F | 64 | 16.0 | 51 608 |
| $\boldsymbol{x}_{s5}$ | 2015 | False | 43 | 1 | 0 | 14.5 | F | 64 | 11.6 | 87 173 |

the Top5 model predicts a mean income of 84 184. For communicating these results, the five remaining weights and support points can be cast as scenarios, spanning an income range from 51 608 to 100 433 Euros. The most likely scenario, occurring with a probability of 30.6%, involves an income of 94 903 Euros. Due to the structure of RF forecast distributions, the selected scenarios are directly associated with five actual observations occurring in our training sample. In particular, three of the five scenarios (number 1, 4 and 5) listed in Table 3.1 involve responses of the same individual in previous years. This situation is natural since some of the regressors are nearly constant over time, so that the test point $\boldsymbol{x}_0$ is likely to be similar to training data $\boldsymbol{x}_i$ corresponding to the same individual. Using the same individual's past incomes also seems plausible from a practical perspective.[2] Table 3.1 also yields relevant (albeit implicit) substantive information: For example, the fact that all five scenarios involve individuals of the same gender, family status and industry sector indicate that the RF model considers these regressors highly relevant for prediction. Conversely, the five scenarios cover all survey years present in the training data (2015-2018), indicating that time variation in incomes is less important.

**Related Literature** We approach the interpretability of an RF from a different angle compared to most other existing literature on the topic: In Breiman's original work on RFs, feature permutation was introduced, effectively quantifying the influence of each feature on the overall performance of the model. This method is still widely used and implemented in standard software packages (Pedregosa et al., 2011). Biau and Scornet

[2] If one wanted to avoid the use of data from the same individual, one could reduce the data set to a single observation per individual, and otherwise use the same methodology.

(2016, Section 5) provide further discussion. This perspective on interpretation is also commonly used for other forecasting models, such as NNs, see e.g. Lundberg and Lee (2017). Visualizations of RFs have also been proposed (Zhao et al., 2019; Haddouchi and Berrado, 2019). In addition, alternative notions of sparsity have been considered in the context of tree-type models. For example, Nan et al. (2016) use the notion of sparsity in terms of feature usage in order to speed up predictions. Several other studies consider combinations of simple prediction rules, building upon ideas in Friedman and Popescu (2008). The 'node harvest' method by Meinshausen (2010) is particularly interesting since it can be cast as a weighted empirical distribution, and can thus be used for probabilistic prediction. Node harvest first generates a large number of 'nodes', each of which defines a simple subspace of the predictor space. It then uses a constrained optimization problem to select the nodes that are most useful for prediction. Although sparsity is not enforced, Meinshausen (2010) finds that the optimal solution often involves a small number of nodes only. However, this type of sparsity (with respect to nodes) is different from the type of sparsity we consider, which is with respect to the number of training sample observations being considered for prediction. Indeed, these two notions of sparsity seem highly complementary: A forecast distribution constructed via node harvest may involve a large number of training sample observations, i.e. be far from sparse in our sense. Conversely, a forecast distribution constructed using our proposed methodology implicitly involves many regression trees, thus being far from sparse in Meinshausen's sense.

Finally, many extensions and modifications of Breiman's original RF have been proposed in the literature, typically with the aim of improving statistical performance (either empirically, or in terms of theoretical properties). See Biau and Scornet (2016) for a survey, and for example Beck et al. (2023) for a forecast combination perspective, Cevid et al. (2022) for a multivariate model that links RFs to kernel-based methods, and Wager and Athey (2018) for an adaption to causal inference. By contrast, we focus on a standard, univariate RF implementation, and post-process its forecast distribution in a way that makes it easier to interpret. The procedure we propose can easily be applied to other RF variants (or even to prediction methods other than RFs), provided that they can be represented as discrete distributions of the type described above.

**Roadmap of the Chapter** The remainder of this chapter is organized as follows: Section 3.2 introduces our methodological setup, including RFs and methods for evaluating forecast distributions. In Section 3.3, we study the performance of our 'Top$k$'

simplification in a series of empirical experiments based on 18 data sets for which RFs have been found to perform well, as compared to NN models (Grinsztajn et al., 2022). Our findings indicate that already for $k = \{5, 10\}$, the simplified RFs can perform on a similar level compared to the full counterpart for both probabilistic and point forecasts, depending on the data set. Considering probabilistic forecasts, for $k = 20$, the median performance across all data sets is equivalent to the full RFs and considering $k = 50$ even increases the median performance slightly. For point (mean) forecasts, $k = 20$ even increases performance slightly. Even though very sparse choices like $k \in \{3, 5\}$ may come with a significant performance decrease, we argue that they may be worthwhile if ease of communication is a main concern. We further show that our results are qualitatively robust to different hyperparameter choices. In order to rationalize the empirical results, Section 3.4 then considers a detailed analytical example that models the weights estimated by RFs as a random draw from a Dirichlet-type distribution. The example also features a true vector of weights that may be either similar to, or different from, the estimated weights. This setup is useful to study how a simplifying approximation similar to Top$k$ affects statistical forecasting performance. In a nutshell, the amount of noise in the estimated weights determines whether or not the simplification comes at a high cost in terms of performance. Perhaps surprisingly, one can construct examples in which simplification improves performance: this result arises when the largest weights are estimated precisely, whereas smaller weights are more noisy. Focusing on the largest weights then constitutes a beneficial form of shrinkage. When the small weights are estimated precisely, however, simplification is harmful in terms of performance. The appendix contains further empirical results and detailed derivations for Section 3.4. Replication materials are available at `https://github.com/kosnil/simplify_rf_dist`.

## 3.2 Methodological Setup

In this section, we describe RFs based forecasting as well as the forecast evaluation methods we consider.

### 3.2.1 Forecasting Methods

Here we describe RFs and their probabilistic cousins, QRFs. We follow Lin and Jeon (2006) and Meinshausen (2006) who emphasize the perspective of RFs predictions as a

weighted sum over training observations. We refer to Hastie et al. (2009) for a textbook presentation of RFs.

Our goal is to fit a univariate forecasting model. That is, we have some data set $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^n$ of training set size $n$, where $\boldsymbol{x}_i$ is a $p$-dimensional vector of features, and $y_i \in \mathbb{R}$ is a real-valued outcome. We use this data set to train our model $\hat{f}$, such that some choice of loss function $\mathcal{L}_n$ is minimized:

$$\min_{\hat{f}} \mathcal{L}_n(\hat{f}) = \frac{1}{n} \sum_{i}^{n} \mathcal{L}(\hat{f}(\boldsymbol{x}_i), y_i).$$

In a typical regression context, the function $\hat{f} : \mathbb{R}^p \to \mathbb{R}$ maps $\boldsymbol{x}_i$ to the real line in order to estimate the conditional expectation functional. A popular loss function is given by SE, with $\mathcal{L}(z, y) = (y - z)^2$. Let further $\mathcal{B} \subseteq \mathbb{R}^p$ denote the feature space, i.e., the space in which the individual input samples $\boldsymbol{x}_i$ exist.

**Random Forest** An RF is an ensemble of individual regression trees, each denoted by $\mathcal{T}(\xi)$, where $\xi$ describes the configuration of the tree. At each node, a single tree greedily splits $\mathcal{B}$ and rectangular subspaces thereof into two further rectangular subspaces, such that the loss $\mathcal{L}_n$ is minimized. Each resulting subspace corresponds to a leaf $\mathcal{R}_l \subseteq \mathcal{B},\ l = 1, \ldots, L$ where $L$ is the total number of leaves. Each sample $\boldsymbol{x}_i$ can only occur in one leaf or, put differently, when dropping a sample $\boldsymbol{x}_i$ down the tree, it can only fall into one leaf. This leaf is denoted $\ell(\boldsymbol{x}_i, \xi) \in \{1, \ldots, L\}$ for tree $\mathcal{T}(\xi)$. For a single tree $\mathcal{T}(\xi)$, a prediction $\hat{\mu}_{\mathcal{T}}(\boldsymbol{x}_0)$ for a new sample $\boldsymbol{x}_0$ is obtained by taking the mean of all training samples within leaf $\ell(\boldsymbol{x}_0, \xi)$. This can be expressed as

$$\hat{\mu}_{\mathcal{T}}(\boldsymbol{x}_0) = \sum_{i=1}^{n} w_i(\boldsymbol{x}_0, \xi) y_i,$$

where the weight $w_i(\boldsymbol{x}_0, \xi)$ is equal to zero for all training samples $i$ that fall into leaves other than $\ell(\boldsymbol{x}_0, \xi)$, and is equal to one over the leaf size for all training samples that fall into $\ell(\boldsymbol{x}_0, \xi)$:

$$w_i(\boldsymbol{x}_0, \xi) = \frac{\mathbb{1}\{\boldsymbol{x}_i \in \mathcal{R}_{\ell(\boldsymbol{x}_0, \xi)}\}}{\sum_{j=1}^{n} \mathbb{1}\{\boldsymbol{x}_j \in \mathcal{R}_{\ell(\boldsymbol{x}_0, \xi)}\}}. \tag{3.1}$$

Motivated by the lack of stability and tendency to overfit of individual trees, RFs build $B$ trees $(\mathcal{T}(\xi_b))_{b=1}^B$, based on $B$ bootstrap samples of $\mathcal{D}$, and consider their average

prediction. Moreover, in each split within each tree, it is common to consider only a random subsample of $\tilde{p}$ out of $p$ regressors. This step aims to diversify the ensemble of trees by avoiding excessive use of the same regressors for splitting. Common choices for $\tilde{p}$ are $\lfloor\sqrt{p}\rfloor$ or $\lfloor\frac{p}{3}\rfloor$ (Probst et al., 2019), where $\lfloor z\rfloor$ floors the real number $z$ to the nearest integer. The RFs mean prediction can thus be expressed as

$$\begin{aligned}\hat{\mu}_{\mathrm{RF}}(\boldsymbol{x}_0) &= \frac{1}{B}\sum_{b=1}^{B}\sum_{i=1}^{n} w_i(\boldsymbol{x}_0,\xi_b)y_i \\ &= \sum_{i=1}^{n} w_i(\boldsymbol{x}_0)y_i, \end{aligned} \tag{3.2}$$

where

$$w_i(\boldsymbol{x}_0) = \frac{1}{B}\sum_{b=1}^{B} w_i(\boldsymbol{x}_0,\xi_b) \tag{3.3}$$

is the weight for training sample $i$, averaged across all $B$ trees. By construction, the weights $(w_i(\boldsymbol{x}_0))_{i=1}^n$ are non-negative and sum to one. Thus, $w_i(\boldsymbol{x}_0)$ can be interpreted as the empirically estimated probability that the new test sample observation is equal to $y_i$.

**Quantile Regression Forest (QRF)** Conceptually, $\hat{\mu}_{\mathrm{RF}}(\boldsymbol{x}_0)$ is an estimate of the conditional mean $\mathbb{E}[Y|X=\boldsymbol{x}_0]$. As described above, it is obtained as a weighted sum over all training observations. Meinshausen (2006) extends this framework to estimating the cumulative distribution function (CDF) of $Y$, which is given by $\mathbb{E}[\mathbb{1}(Y\leq t)|X=\boldsymbol{x}_0] = \mathbb{P}(Y\leq t|X=\boldsymbol{x}_0)$, where $t\in\mathbb{R}$ is a threshold value. The similarity to RFs becomes apparent in the last expression. Utilizing the weights from Equation 3.3, one can approximate the CDF by the weighted mean over the binary observations $\mathbb{1}(y_i\leq t)$:

$$\widehat{\mathbb{P}}(Y\leq t|X=\boldsymbol{x}_0) = \sum_{i=1}^{n} w_i(\boldsymbol{x}_0)\mathbb{1}(y_i\leq t). \tag{3.4}$$

That is, QRFs estimate the CDF of $Y$ via the weighted empirical CDF of the training sample outcomes $(y_i)_{i=1}^n$, using the weights $(w_i(\boldsymbol{x}_0))_{i=1}^n$ produced by RFs. This estimator is practically appealing as it arises as a byproduct of standard RFs software implementation. Furthermore, its representation in terms of a weighted empirical CDF enables a theoretical understanding of its properties by leveraging tools from nonparametric statistics (Lin and Jeon, 2006; Meinshausen, 2006). In this chapter, we consider the standard variant of

QRFs which uses the squared error as a criterion for finding splits (and thus growing the forest's individual trees). Various other splitting criteria have been analyzed in the literature. In particular, Cevid et al. (2022) propose to use a splitting criterion based on distributional similarity. Since their RF variant retains the weighted empirical CDF representation (see their Section 2.2), our Top$k$ method can be applied to it as well.

### 3.2.2 Forecast Evaluation

Since we generate probabilistic forecasts, we need a tool to evaluate them. For this, we use the continuously ranked probability score (CRPS), a strictly proper scoring rule. Scoring rules are loss functions for probabilistic forecasts. We use them in negative orientation, so that smaller scores indicate better forecasts. When evaluated using a proper scoring rule, a forecaster minimizes their expected score by stating what they think is the true forecast distribution. Under a strictly proper scoring rule, this minimum is unique within a suitable class of forecast distributions. Conceptually, strictly proper scoring rules incentivize careful and honest forecasting. See Gneiting and Raftery (2007) for a comprehensive technical treatment, and Winkler (1996) and Gneiting and Katzfuss (2014) for further discussion and illustration. The CRPS (Matheson and Winkler, 1976) is defined as

$$\text{CRPS}(\hat{F}, y) = \int_{-\infty}^{\infty} \left(\hat{F}(z) - \mathbb{1}\{z \geq y\}\right)^2 dz, \tag{3.5}$$

where $\hat{F}$ denotes the CDF implied by the forecast distribution and $y$ denotes the true outcome. For various forms of $\hat{F}(z)$, closed-form expressions of the CRPS are available, so that there is no need for costly numerical evaluation of the integral in Equation 3.5.

The CRPS allows for very general types of forecast distributions $\hat{F}$. In particular, the forecast distribution may be discrete, that is, it need not possess a density. This allows for evaluating forecast distributions based on (weighted) empirical CDFs, which arise in the case of QRFs. In the special case that the forecast distribution is deterministic, i.e., it places point mass on a single outcome, the CRPS reduces to the absolute error (AE). Thus, numerical values of the AE and CRPS can meaningfully be compared to each other.

Jordan et al. (2019) provide an efficient implementation of the CRPS for weighted empirical distributions in their `R`-package `scoringRules`. Let $(y_i)_{i=1}^n$ denote the response values from the training data, and denote by $y_{(i)}$ their $i$th ordered value, with $y_{(1)} \leq$

$y_{(2)} \leq \ldots \leq y_{(n)}$. Furthermore, let $w_{(i)}$ denote the weight corresponding to $y_{(i)}$. Then the CRPS for a realization $y \in \mathbb{R}$ is given by

$$\mathrm{CRPS}(\hat{F}, y) = 2 \sum_{i=1}^{n} w_{(i)} (y_{(i)} - y) \left( \mathbb{1}\{y < y_{(i)}\} - \left( \sum_{j=1}^{i} w_{(j)} \right) + \frac{w_{(i)}}{2} \right), \tag{3.6}$$

where we dropped the dependence of $w_{(i)}$ on a vector $\boldsymbol{x}_0$ of covariates at this point for ease of notation. Equation 3.6 extends Jordan et al.'s Equation 3 to the case of non-equal weights, based on their implementation in the function `crps_sample`. In the case of sparse weights, one may omit the indices $i$ with $w_{(i)} = 0$ from the sum at (3.6) in order to speed up the computation.

Additionally to the CRPS, we also report results for the squared error (SE). In the present context, the SE is given by

$$\mathrm{SE}(\hat{F}, y) = (y - \sum_{i=1}^{n} w_i y_i)^2. \tag{3.7}$$

Hence, the SE depends on the forecast distribution $\hat{F}$ via its mean $\sum_{i=1}^{n} w_i y_i$ only.

## 3.3 Experimental Results

In order to assess the statistical performance of the simplified forecast distributions, we conduct experiments on 18 data sets considered by Grinsztajn et al. (2022) in the context of numerical regression. The authors demonstrate that tree-based methods compare favorably to neural networks for these data sets. Their selection of data sets aims to represent real-world, 'clean' data sets with medium size as well as heterogeneous data types and fields of applications. If deemed necessary, some basic preprocessing was applied by Grinsztajn et al. (2022). Details can be found in Section 3.5 and Appendix A.1 in their paper, and in Table B.3 of Appendix B.2 below. The data set `delays_zurich_transport` contains about 5.6 million data points in its original form. For computational reasons, we reduced the size of this data set through random subsampling, to approximately 1.1 million data points (20% of the original observations). We did not apply further preprocessing of any of the data sets in order to retain comparability. We allocate 70% of each data set for training our models and reserve the remaining 30% for testing. For each data set, we train an RF, and then evaluate its performance by computing the average

CRPS and SE, as introduced in Equations 3.6 and 3.7 over the test data set. Our main interest is in studying the impact of the parameter $k$ which governs the number of support points of the sparsified forecast distribution. We consider a grid of choices $k = 1, \ldots, 50$ and denote these sparse RFs as 'Top$k$'. Our standard choice of RF hyperparameters is a combination of default values of the machine learning software packages `scikit-learn` (Pedregosa et al., 2011) and `ranger` (Wright and Ziegler, 2017) which are popular choices in the `Python` (van Rossum et al., 2011) and `R` (R Core Team, 2022) programming languages, respectively. In summary, we consider a random selection of $\sqrt{p}$ out of $p$ possible features at each split point, do not impose any form of regularization in terms of tree growth restriction, and set the number of trees to 1000 in order to obtain a large and stable ensemble. These choices, which are also listed in the first row (entitled 'standard') of Table B.5 in Appendix B.2, are used for the analysis in Sections 3.3.1 and 3.3.2. In Section 3.3.3, we further consider the effects of tuning hyperparameters.

### 3.3.1 Probabilistic Forecasts

Table 3.2 presents the CRPS for the full RF and the relative CRPS of Top$k$, for $k \in \{3, 5, 10, 20, 50\}$, compared to the full RF. For example, a relative CRPS of 1.5 indicates a 50% larger CRPS for the Top$k$ version compared to the full RF. The three smallest values for $k$ seem especially attractive in terms of simplicity and ease of communication. While the two larger choices $k \in \{20, 50\}$ are less attractive in terms of simplicity, we also consider them in order to assess the trade-off between simplicity and performance.

Using only $k = 3$ support points performs worse than the full RF, with a median performance cost of 25% across data sets. While this result is unsurprising from a qualitative perspective, its magnitude is interesting, and gives a first indication of the performance cost of using a rather drastic simplification of the original forecast distribution. For each single data set, we find that the performance of Top$k$ improves monotonically when increasing $k$ from 3 to 5, from 5 to 10, and for all data sets but one when increasing $k$ from 10 to 20. This pattern is plausible, given that we move from a drastic simplification ($k = 3$) to less drastic versions. Compared to the full RF, Top5 implies a median loss increase of 14%, whereas Top10 yields a median loss increase of 5%. Top20 performs equally well as the full RF in the median. For most data sets, Top50 slightly enhances predictive performance compared to the full RF. Whether $k = 50$ support points remain worth interpreting depends on the application at hand. Only for

**Table 3.2: Results for forecast distributions.** The table reports the CRPS of the full RF as well as the CRPS for Top{3,5,10,20,50} relative to the full RF, i.e., $^{\text{CRPS}_{\text{Top}k}}/_{\text{CRPS}_{\text{Full}}}$. A value smaller than 1 means that Top$k$ outperforms the full RF. The last row lists the median relative CRPS across data sets.

| | Absolute CRPS | CRPS relative to Full | | | | |
|---|---|---|---|---|---|---|
| Data set | Full | Top3 | Top5 | Top10 | Top20 | Top50 |
| cpu_act | 1.2508 | 1.21 | 1.08 | 1.00 | 0.96 | 0.95 |
| pol | 1.4203 | 1.50 | 1.28 | 1.10 | 1.01 | 0.96 |
| elevators | 0.0014 | 1.22 | 1.10 | 1.01 | 0.96 | 0.95 |
| wine_quality | 0.2565 | 1.35 | 1.20 | 1.09 | 1.02 | 0.99 |
| Ailerons | 0.0001 | 1.23 | 1.11 | 1.01 | 0.97 | 0.96 |
| houses | 0.1229 | 1.19 | 1.08 | 0.99 | 0.96 | 0.95 |
| house_16H | 0.1984 | 1.32 | 1.17 | 1.06 | 1.01 | 0.98 |
| diamonds | 0.1320 | 1.34 | 1.21 | 1.11 | 1.05 | 1.01 |
| Brazilian_houses | 0.0256 | 0.88 | 0.82 | 0.78 | 0.79 | 0.85 |
| Bike_Sharing_Demand | 46.0292 | 1.26 | 1.15 | 1.06 | 1.02 | 1.00 |
| nyc-taxi-green-dec-2016 | 0.1505 | 1.39 | 1.24 | 1.12 | 1.06 | 1.02 |
| house_sales | 0.0995 | 1.25 | 1.14 | 1.04 | 0.99 | 0.97 |
| sulfur | 0.0123 | 1.05 | 0.97 | 0.94 | 0.95 | 0.97 |
| medical_charges | 0.0376 | 1.35 | 1.21 | 1.11 | 1.05 | 1.01 |
| MiamiHousing2016 | 0.0778 | 1.20 | 1.10 | 1.02 | 0.98 | 0.97 |
| superconduct | 3.6341 | 1.14 | 1.08 | 1.02 | 1.00 | 0.99 |
| yprop_4_1 | 0.0140 | 1.38 | 1.25 | 1.14 | 1.08 | 1.03 |
| delays_zurich_transport | 1.6174 | 1.33 | 1.19 | 1.10 | 1.05 | 1.02 |
| Median | - | 1.25 | 1.14 | 1.05 | 1.00 | 0.98 |

a few data sets, Top50 is not sufficient to reach the performance of the full RF, but performance costs are small even for these data sets.

In order to contextualize the magnitude of our presented results (such as Top3's median CRPS increase of 25% compared to the full RF), we next present results on two simple benchmark methods.[3] First, we consider a deterministic point forecast which assumes that the full RF's median forecast materializes with probability one.[4]

This is a very optimistic point (rather than probabilistic) forecast, containing no uncertainty. As noted earlier, its CRPS is the same as its AE. We consider this benchmark in order to quantify the costs of ignoring uncertainty altogether. We clearly expect these costs to be positive, i.e., we expect the point forecast to perform worse than the full RF.

[3] We refer to Grinsztajn et al. (2022) for detailed comparisons of RF point forecasts to other tree-based models and neural networks.

[4] Formally, this forecast is characterized by the CDF $\hat{F}_\delta(z) = \mathbb{1}(z \geq \text{med}(\hat{F}_{\text{Full}}(z))$, where $\text{med}(\hat{F}_{\text{full}})$ denotes the median implied by the CDF of the full RF's forecast distribution. That is, the CDF $\hat{F}_\delta$ is a step function with a single jump point at the median forecast of the full RF. We choose the median functional here because the latter is the optimal point forecast under absolute error loss, to which the CRPS reduces in the case of a deterministic forecast.

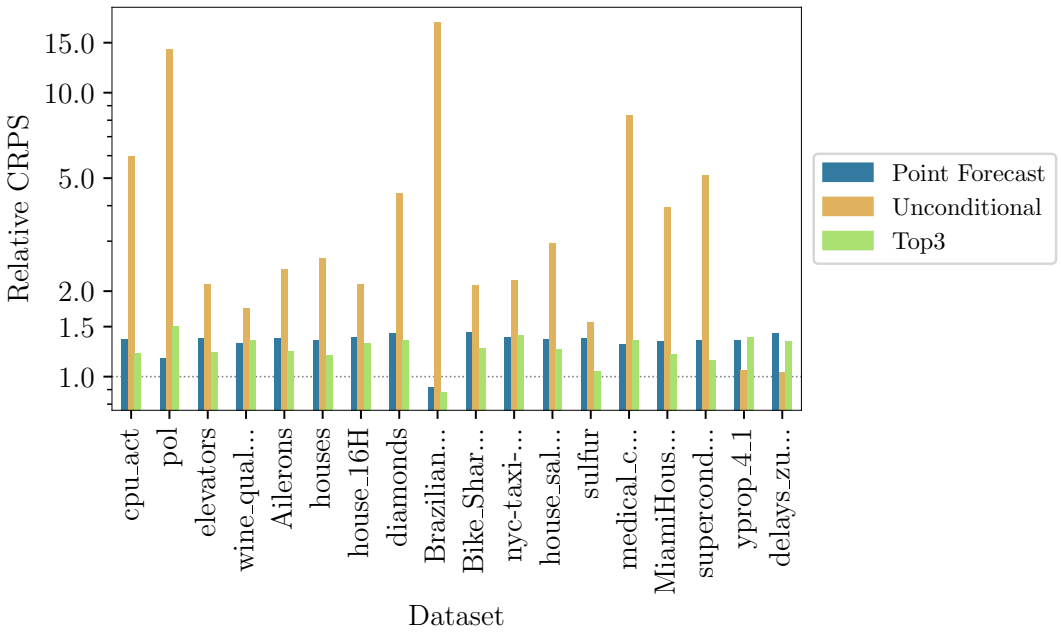



**Figure 3.2: Benchmarking performance.** Blue bars: Relative CRPS of a deterministic point forecast (median) as a benchmark that ignores uncertainty. Ochre bars: Relative CRPS of the unconditional forecast distribution, yielding a conservative benchmark that ignores features. Green bars: Relative CRPS of Top3, as listed in Table 3.2. In each case, relative CRPS results are compared to the full RF. That is, a relative CRPS smaller than 1 (represented by horizontal line) indicates that the method performs better than the full RF.

Second, we use the CRPS of the unconditional empirical distribution of the response variable in the training sample. This distribution places a uniform weight of $1/n$ on each training sample observation, in contrast to the QRF weights $w(\boldsymbol{x}_0)$ that depend on the feature vector $\boldsymbol{x}_0$. The unconditional distribution is a very conservative forecast, as no information about the features is used whatsoever. Qualitatively, we clearly expect this forecast to perform worse than the full RF. Quantitatively, the difference in performance of the unconditional versus conditional forecast distributions captures the predictive content of the features (see e.g. Gneiting and Resin, 2023, Section 2.5). Both benchmarks are visualized jointly with Top3 results in Figure 3.2.

As expected, the relative CRPS of the point forecast (shown in blue) exceeds one for most data sets, indicating that it is generally inferior to the full RF model. Notably, an exception is observed for `Brazilian_houses`, where the point forecast's relative CRPS

is smaller than one. This result appears to be due to prediction uncertainty being very small for this data set; see Figure 13 in the supplemental material for Grinsztajn et al. (2022). Indeed, for this data set, the response variable seems to mostly be a linear combination of a subset of the features. Furthermore, Top3 (shown in green) performs similar to or better than the point forecast for all data sets, demonstrating the usefulness of incorporating additional uncertainty in the forecast. Note that the point forecast has access to the full RF forecast distribution, using the median of this distribution as a point forecast. By contrast, Top3 only has access to the three most important support points of the RF forecast distribution.

The relative CRPS of the unconditional forecast distribution (displayed in ochre in Figure 3.2) exceeds two for most data sets, indicating that the features are generally very useful for prediction. An exception occurs for the last two data sets (`yprop_4_1` and `delays_zurich_transport`). In these cases, the RF appears unable to learn meaningful connections between the features and the target variable. Figure 13 in the online supplement of Grinsztajn et al. (2022) supports this interpretation, reporting low predictability (in terms of low out-of-sample $R^2$) for these data sets. In this situation, we cannot expect a Top$k$-model to perform well compared to the full RF. To see this, consider the stylized case of the features being entirely uninformative. Subsequently, the unconditional distribution (placing a weight of $1/n$ on all training sample responses) is the best possible forecast, which is in sharp contrast to Top$k$ (for which $k$ weights are non-zero and large by construction, whereas the remaining $n - k$ weights are forced to zero).

In order to further study the properties of Top$k$ forecast distributions, Table 3.3 reports the average weight sums across test cases, for different values of $k$. The weight sums are computed before our normalization step (see Section 3.1) which re-scales all weight sums to one. For a given choice of $k$, the unnormalized weight sums can vary in magnitude, both across data sets and from test case to test case. By construction, the weight sums increase with $k$. Interestingly, many data sets yield a large weight sum for Top3, exceeding 10% for all but four data sets. For Top50, the average weight sum exceeds 50% for most data sets. These numbers are remarkable, given that the data sets include thousands of training samples ($n$, see rightmost column of Table 3.3) that could potentially be used as support points for the RF forecast distributions. If the weights were uniform, we would hence observe Top$k$ weight sums of $k/n$. This is in sharp contrast to our empirical finding that a few large weights dominate for most data sets. Lin and Jeon (2006) find similar

**Table 3.3: Top$k$ weight sums.** Average sum of un-normalized Top{3,5,10,20,50} weights. The last column reports the number of observations in the training set.

| | Average Sum | | | | | |
|---|---|---|---|---|---|---|
| Data set | Top3 | Top5 | Top10 | Top20 | Top50 | $n$ |
| cpu_act | 0.094 | 0.135 | 0.212 | 0.318 | 0.504 | 5734 |
| pol | 0.073 | 0.104 | 0.159 | 0.233 | 0.359 | 10 500 |
| elevators | 0.119 | 0.166 | 0.252 | 0.366 | 0.555 | 11 619 |
| wine_quality | 0.150 | 0.189 | 0.258 | 0.350 | 0.513 | 4547 |
| Ailerons | 0.119 | 0.168 | 0.257 | 0.374 | 0.566 | 9625 |
| houses | 0.143 | 0.201 | 0.304 | 0.435 | 0.634 | 14 447 |
| house_16H | 0.071 | 0.102 | 0.162 | 0.247 | 0.402 | 15 948 |
| diamonds | 0.257 | 0.348 | 0.494 | 0.656 | 0.851 | 37 758 |
| Brazilian_houses | 0.202 | 0.278 | 0.406 | 0.558 | 0.763 | 7484 |
| Bike_Sharing_Demand | 0.248 | 0.334 | 0.473 | 0.628 | 0.821 | 12 165 |
| nyc-taxi-green-dec-2016 | 0.172 | 0.234 | 0.337 | 0.461 | 0.642 | 407 284 |
| house_sales | 0.116 | 0.160 | 0.240 | 0.345 | 0.522 | 15 129 |
| sulfur | 0.212 | 0.296 | 0.438 | 0.602 | 0.811 | 7056 |
| medical_charges | 0.223 | 0.304 | 0.438 | 0.590 | 0.789 | 114 145 |
| MiamiHousing2016 | 0.196 | 0.267 | 0.385 | 0.520 | 0.704 | 9752 |
| superconduct | 0.390 | 0.497 | 0.617 | 0.708 | 0.805 | 14 884 |
| yprop_4_1 | 0.111 | 0.149 | 0.216 | 0.304 | 0.456 | 6219 |
| delays_zurich_transport | 0.015 | 0.024 | 0.048 | 0.095 | 0.230 | 765 180 |

results in their work, where they consider RFs as adaptive nearest-neighbor methods and investigate the influence of the minimum number of samples per leaf. Figures 1c,d and 5 show few large weights for synthetic data sets. The presence of a small number of important weights explains why the simplification pursued by Top$k$ often results in modest (if any) performance costs as compared to the full RF.

### 3.3.2 Mean Forecasts

Let us turn our attention towards conditional mean forecasts, for which results in terms of SE are shown in Table 3.4. We notice a similar pattern as in the probabilistic scenario: apart from Top3 outperforming the full RF for one data set (`sulfur`), Top3 performs worse than the full RF for the remaining data sets, with a maximum performance cost of 61% for `pol`. This results in a median SE increase of 22% for Top3. Top5 still shows a 12% median increase and for Top10, the median performance almost matches the full RF's performance. Top20 and Top50 even outperform the full RF, yielding

**Table 3.4: Results for conditional mean forecasts.** The table reports the SE of the full RF as well as the SE for Top{3,5,10,20,50} relative to the full RF, i.e., $\text{SE}_{\text{Top}k}/\text{SE}_{\text{Full}}$. The last row lists the median relative SE for each choice of $k$ across data sets.

| | Absolute SE | SE relative to Full | | | | |
|---|---|---|---|---|---|---|
| Data set | Full | Top3 | Top5 | Top10 | Top20 | Top50 |
| cpu_act | 6.5359 | 1.11 | 0.99 | 0.93 | 0.91 | 0.91 |
| pol | 38.5033 | 1.61 | 1.37 | 1.16 | 1.03 | 0.95 |
| elevators | $9.22 \times 10^{-6}$ | 1.09 | 0.98 | 0.89 | 0.85 | 0.86 |
| wine_quality | 0.3485 | 1.42 | 1.24 | 1.11 | 1.03 | 1.00 |
| Ailerons | $3.22 \times 10^{-8}$ | 1.15 | 1.02 | 0.94 | 0.91 | 0.91 |
| houses | 0.0590 | 1.16 | 1.03 | 0.94 | 0.91 | 0.92 |
| house_16H | 0.3011 | 1.49 | 1.31 | 1.15 | 1.06 | 1.00 |
| diamonds | 0.0563 | 1.37 | 1.24 | 1.12 | 1.06 | 1.02 |
| Brazilian_houses | 0.0072 | 1.05 | 1.12 | 0.99 | 0.94 | 0.94 |
| Bike_Sharing_Demand | 10 126.7983 | 1.22 | 1.11 | 1.03 | 0.99 | 0.99 |
| nyc-taxi-green-dec-2016 | 0.1528 | 1.38 | 1.23 | 1.12 | 1.05 | 1.02 |
| house_sales | 0.0382 | 1.22 | 1.11 | 1.02 | 0.96 | 0.94 |
| sulfur | 0.0015 | 0.69 | 0.69 | 0.73 | 0.81 | 0.90 |
| medical_charges | 0.0073 | 1.36 | 1.22 | 1.11 | 1.05 | 1.01 |
| MiamiHousing2016 | 0.0243 | 1.15 | 1.05 | 0.97 | 0.94 | 0.94 |
| superconduct | 87.2592 | 1.05 | 1.02 | 0.97 | 0.95 | 0.95 |
| yprop_4_1 | 0.0010 | 1.26 | 1.16 | 1.10 | 1.05 | 1.02 |
| delays_zurich_transport | 9.3282 | 1.33 | 1.19 | 1.10 | 1.05 | 1.02 |
| Median | - | 1.22 | 1.12 | 1.02 | 0.98 | 0.95 |

median improvements of 2% and 5%, respectively. Compared to the results for forecast distributions (Table 3.2), the results in Table 3.4 indicate that the performance costs of simplicity are comparatively lower in the case of mean forecasts, with median relative losses being somewhat smaller for a given value of $k$.

### 3.3.3 Varying Hyperparameters

Compared to other modeling algorithms, RFs have relatively few hyperparameters. Nevertheless, tuning its hyperparameters can improve the performance of RFs (Probst et al., 2019). The most important hyperparameters control the depth of each individual tree, as well as the number of randomly selected regressors considered for splitting. The depth of a tree can be restricted directly ('maximum depth') or indirectly by restricting leaf and split sizes ('minimum leaf size' and 'minimum split size', respectively). The number of regressors considered for splitting is often denoted as 'max features' or 'mtry' (the latter term is used, e.g. in the `R` packages `randomForest` (Liaw and Wiener, 2002) and `ranger` (Wright and Ziegler, 2017)). In what follows, we study how different

hyperparameter sets influence the Top$k$ prediction and whether a hyperparameter set that optimizes the full RF is also beneficial for Top$k$. We therefore investigate the influence of 'max features' and one of the depth-regularizing hyperparameters, 'minimum leaf size', on the Top$k$ approach. To do so, we consider a grid search for both, the full RF and Top3, with 5-fold cross-validation on the training set of each data set. Due to the size of `medical_charges`, `nyc-taxi-green-dec-2016` and `delays_zurich_transport`, we use a validation set which contains 25% of the training set instead. Further, the latter two are down-sampled to 30% and 15% of their original training set size, respectively. For brevity, our analysis of hyperparameter tuning focusses on the case $k = 3$, which is the most drastic simplification we consider.

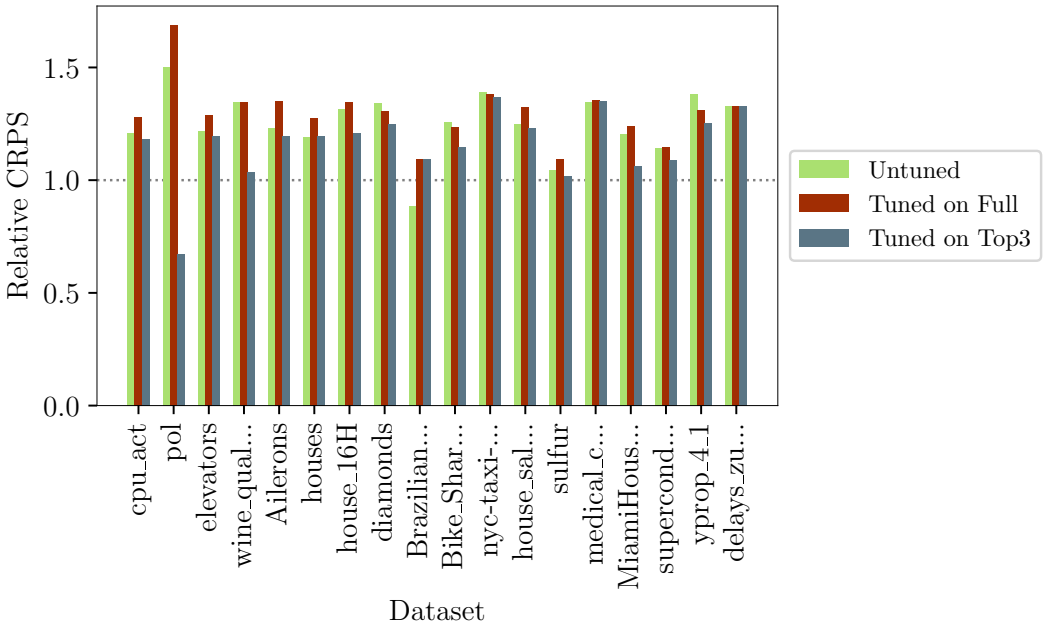



**Figure 3.3: Hyperparameter tuning performance (probabilistic forecasts).** The figure shows the relative CRPS of Top3 compared to the full RF, for different hyperparameter settings. Green bars: standard hyperparameters, as listed in Table 3.2. Red bars: hyperparameters that optimize the full RF. Blue bars: hyperparameters that optimize Top3. Hyperparameter tuning is based on CRPS.

Figure 3.3 visualizes the CRPS of Top3 relative to the full RF's CRPS for three different hyperparameter sets. In each case, we consider the same hyperparameter set for Top3 and the full RF. The green bars show the results with standard hyperparameters, as listed in Table 3.2. The red bars indicate the relative performance with the respective

hyperparameter set that optimizes the full RF, while the blue bars visualize the relative performance with the hyperparameters that optimize the CRPS of Top3. When hyperparameters are tuned on the full RF, Top3 performance tends to slightly decrease overall. Across the data sets, the median relative CRPS is 1.31, compared to 1.25 for the standard setting. Conversely, if hyperparameters are tuned on Top3, the relative performance of Top3 is mostly better than in the standard version, reaching a median relative CRPS of 1.20.

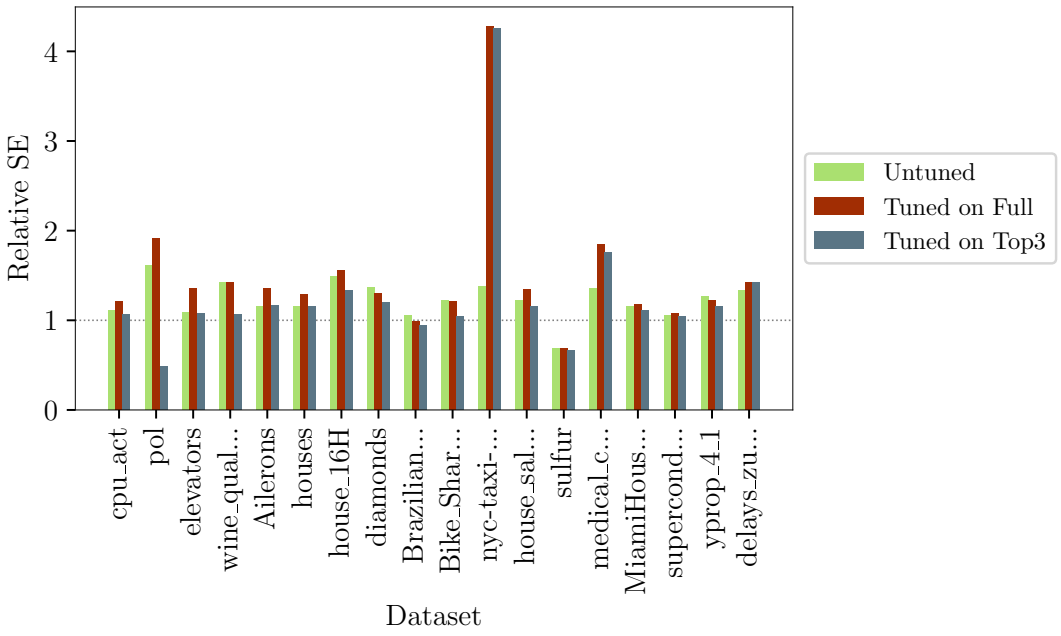



**Figure 3.4: Hyperparameter tuning performance (point forecasts).** The figure shows the relative SE of Top3 compared to the full RF, for different hyperparameter settings. Green bars: standard hyperparameters, as listed in Table 3.2. Red bars: hyperparameters that optimize the full RF. Blue bars: hyperparameters that optimize Top3. Hyperparameter tuning is based on SE.

Figure 3.4 presents results for point forecasting performance, which are similar to the probabilistic case. Tuning on the full RF hurts Top3, with a median relative SE of 1.32, compared to 1.22 in the standard case. By contrast, tuning on Top3 benefits Top3, with a median relative SE of 1.13. In a small minority of cases, tuning is not beneficial in terms of the relative loss.

Hence, despite its simplicity, Top3 can perform similar to the full RF given suitable hyperparameter choices. Overall, the relative performance of Top3 obtained under the

standard setting is between the performance obtained under the two other hyperparameter settings. Furthermore, the full RF and Top3 require different sets of hyperparameters in order to perform well. Ideally, users of the Top$k$ method should thus choose hyperparameters based on the performance of Top$k$ itself, rather than the performance of the full RF.

## 3.4 Stylized Analytical Model

In this section, we construct a stylized analytical framework which helps explain our experimental findings presented in Section 3.3: For many data sets, simplified RFs perform similar to full RFs even for relatively small choices of $k$. This finding, and especially the possibility that simplified RFs may even outperform full RFs, deserves further investigation. Motivated by the structure of RF forecast distributions (see Section 4.2), we consider a model in which the true forecast distribution is discrete with support points $\mathbf{u} = (u_i)_{i=1}^n$ and corresponding (true) probabilities $\omega^* = (\omega_i^*)_{i=1}^n$ that are positive and sum to one. Specifically, we let

$$\omega_i^* = \begin{cases} \theta^*/k & \text{if } i \in \mathcal{I} \\ (1-\theta^*)/(n-k) & \text{if } i \notin \mathcal{I}\,, \end{cases} \tag{3.8}$$

where $\mathcal{I} \subseteq \{1, 2, \ldots, n\}$ is a subset of 'important' indexes with $|\mathcal{I}| = k$. The corresponding 'important' probabilities $(\omega_i^*)_{i\in\mathcal{I}}$ sum to $\theta^* \in [0, 1]$, whereas the other, 'unimportant', probabilities sum to $1 - \theta^*$. To justify the notion of 'important' probabilities, we will focus on choices of $\theta^*$ and $k$ that satisfy $\theta^*/k > (1-\theta^*)/(n-k)$.

In addition to the true forecast distribution just described, we consider an estimated forecast distribution that uses the same support points $\mathbf{u}$, together with possibly incorrectly estimated probabilities $\omega = (\omega_i)_{i=1}^n$. We assume that the estimated probabilities can be described by the following model:

$$\omega_i = \begin{cases} \theta\ Z_{1,i} & \text{if } i \in \mathcal{I} \\ (1-\theta)\ Z_{2,i} & \text{if } i \notin \mathcal{I}, \end{cases} \tag{3.9}$$

where $\theta \in [0, 1]$, $Z_1$ is a draw from a Dirichlet distribution with $k$-dimensional parameter vector $(d_1, \ldots, d_1)$, with $d_1 > 0$, such that each element of $Z_1$ has expected value $1/k$ and

variance $(k-1)/(k^2(kd_1+1))$. Similarly, $Z_2$ is a draw from another, independent Dirichlet distribution with $(n-k)$-dimensional parameter vector $(d_2, \ldots, d_2)$, where $d_2 > 0$. This means that the expected probabilities are given by

$$\mathbb{E}[\omega_i] = \begin{cases} \theta/k & \text{if } i \in \mathcal{I} \\ (1-\theta)/(n-k) & \text{if } i \notin \mathcal{I}\,. \end{cases} \tag{3.10}$$

In the following, we assume that $2 \leq k \leq n-2$, which ensures that there are at least two 'important' and 'unimportant' support points, respectively. This restriction ensures that weight estimation within both sets is a non-trivial problem.[5]

Thus, if $\theta \neq \theta^*$, the forecast model's expected probabilities differ from the true ones in Equation 3.8. The parameters $d_1$ and $d_2$ represent the precision of the forecast model's probabilities around their expected values. Small values for $d_1, d_2$ indicate noisy probabilities, whereas large values for $d_1, d_2$ correspond to probabilities close to their expected values. This is a property of the variance of Dirichlet-distributed random variables as noted above for the case of $Z_1$. Conceptually, the above model provides a stylized probabilistic description of the estimated probabilities $\omega$ produced by a forecasting method like RFs. Thereby, the model does not aim to specify the mechanism by which the forecasting method generates these probabilities.

Figure 3.5 illustrates this setup, with $n = 20$ and the important indexes given by $\mathcal{I} = \{1, 2, 3, 4, 5\}$. The left panel shows a situation in which the estimated probabilities are quite noisy ($d_1 = d_2 = 1$) but are correct in expectation ($\theta = \theta^*$). In the right panel, the estimated probabilities are less noisy ($d_1 = d_2 = 10$) but are false in expectation ($\theta = 0.4 \neq 0.8 = \theta^*$).

In the special case that $\theta = \theta^*, d_1 \to \infty, d_2 \to \infty$, the forecast model coincides with the true model. Furthermore, in the case $\theta = 1$, the forecast model is very similar to the 'Top$k$' strategy (retaining the $k$ most important probabilities, rescaling them to sum to one, and setting all other probabilities to zero). The somewhat subtle difference between the analytical model and our practical implementation of Top$k$ is that the indexes of the important weights are fixed in the analytical model (given by the set $\mathcal{I}$), whereas they are chosen as the $k$ largest empirical weights in practice. Exact modeling of our practical procedure would seem to complicate the analysis substantially without

[5] If there was only one important support point, for example, the probability of this support point would necessarily be equal to $\theta$, rendering weight estimation trivial.

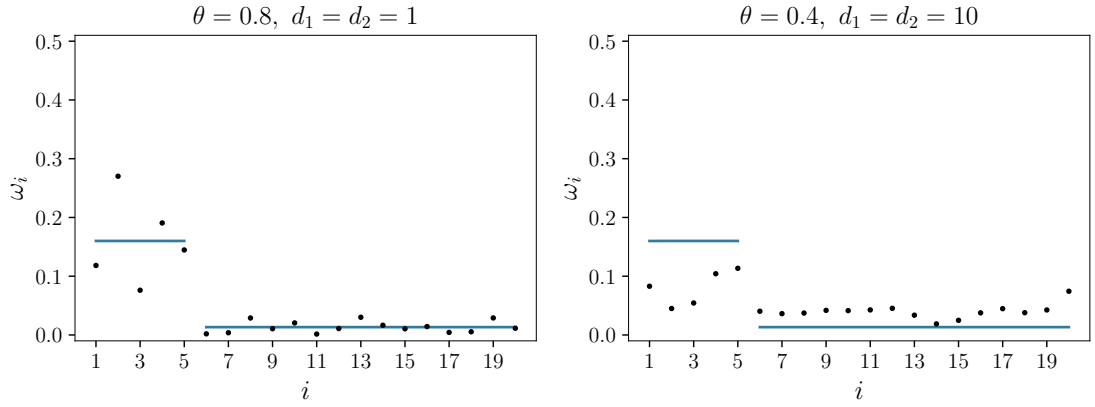



**Figure 3.5: Simulated probability estimates.** In both panels, we set $n = 20, k = 5$, and $\theta^* = 0.8$. The other parameters ($\theta, d_1$ and $d_2$) are as indicated in the header. Horizontal segments represent true probabilities, dots represent estimated probabilities.

necessarily yielding further insights. While stylized, the analytical model described above is flexible enough to cover various situations of applied interest. For example, the relation between 'important' versus 'unimportant' values of the true probabilities can be governed flexibly via the parameters $n, k$ and $\theta^*$. While we assume that the set $\mathcal{I}$ of important indexes is known to the forecast model, the possibility of a poor forecasting model can be represented by a value $\theta$ that differs substantially from $\theta^*$, and/or small values of $d_1, d_2$ that correspond to noisy estimates. Thus, the forecast model could even yield estimates of the 'unimportant' probabilities that greatly exceed those of the 'important' ones.[6] For given support points $\mathbf{u}$ and estimated probabilities $\omega$, the expected SE and expected CRPS implied by the analytical framework are given by

$$\mathbb{E}[\mathrm{SE}(\omega, \mathbf{u})] = \sum_{i=1}^{n} \omega_i^* (u_i - \sum_{j=1}^{n} \omega_j u_j)^2, \tag{3.11}$$

$$\mathbb{E}[\mathrm{CRPS}(\omega, \mathbf{u})] = \sum_{i=1}^{n} \sum_{j=1}^{n} \omega_i (\omega_j^* - \omega_j/2)\ |u_i - u_j|; \tag{3.12}$$

the expression for the CRPS follows from adapting the representation in Equation 2 of Jordan et al. (2019). In both cases, the expected value is computed with respect to the discrete distribution with support points $\mathbf{u}$ and associated true probabilities $\omega^*$. As noted in Equation 3.9, we cast the predicted probabilities $\omega$ as scaled draws from two

[6] In practice, where the set of important indexes $\mathcal{I}$ is not known, this situation corresponds to one in which the largest empirical weights are not helpful for predicting new test sample cases.

Dirichlet distributions. We further assume that the support points $\mathbf{u}$ are $n$ draws from a standard normal distribution; these are mutually independent and independent of $\omega$. Using these assumptions, we obtain the following expressions for the (unconditionally) expected SE and CRPS:

$$\begin{aligned}\mathbb{E}[\mathrm{SE}] &= \int\int \mathbb{E}[\mathrm{SE}(\omega, \mathbf{u})] d\mathrm{F}_\omega(\omega) dF_{\mathbf{u}}(\mathbf{u}) \\ &= 1 - 2\left\{\frac{\theta^*\theta}{k} + \frac{(1-\theta^*)(1-\theta)}{n-k}\right\} \\ &\quad + \frac{\theta^2}{k} + \frac{(1-\theta)^2}{n-k} + \frac{\theta^2(k-1)}{k(d_1 k+1)} \\ &\quad + \frac{(1-\theta)^2(n-k-1)}{(n-k)(d_2(n-k)+1)}, \end{aligned} \tag{3.13}$$

$$\begin{aligned}\mathbb{E}[\mathrm{CRPS}] &= \int\int \mathbb{E}[\mathrm{CRPS}(\omega, \mathbf{u})] d\mathrm{F}_\omega(\omega) dF_{\mathbf{u}}(\mathbf{u}) \\ &= \frac{1}{\sqrt{\pi}} \mathbb{E}[\mathrm{SE}], \end{aligned} \tag{3.14}$$

where $F_\omega$ is the distribution of the estimated probabilities that is implied by our model setup, and $F_{\mathbf{u}}$ is the joint distribution of $n$ independent standard normal variables. The proof can be found in Appendix B.3. The result that the expressions for $\mathbb{E}[\mathrm{SE}]$ and $\mathbb{E}[\mathrm{CRPS}]$ are identical up to a factor of $\sqrt{\pi}$ is a somewhat idiosyncratic implication of our model setup.

In order to interpret the implications of these formulas, we compare a forecasting method with $\theta < 1$ (representing standard RFs) to a method with $\theta = 1$ (representing Top$k$) in the following.

For given values of $n, k$ and $\theta^*$, both $\mathbb{E}[\mathrm{SE}]$ and $\mathbb{E}[\mathrm{CRPS}]$ attain their theoretical minimum at $\theta = \theta^*, d_1 \to \infty$ and $d_2 \to \infty$.[7] This result is unsurprising: Under the stated conditions, the forecast model coincides with the true model, i.e., $\omega = \omega^*$ with probability one. Since the SE is strictly consistent for the mean (and, similarly, the CRPS is a strictly proper scoring rule), the true model must yield the smallest possible expected score.[8] As both expected score functions are continuous in $\theta, d_1$ and $d_2$, this implies that

[7] Proof: $\frac{\partial \mathbb{E}[\mathrm{SE}]}{\partial d_i} < 0$ for $i = 1, 2$; this holds for all values of $\theta, \theta^*, n, k, d_1$ and $d_2$. It is hence optimal to let $d_1, d_2$ go to infinity. Next consider the limiting expression of $\mathbb{E}[\mathrm{SE}]$ as $d_1, d_2 \to \infty$. Minimizing this expression with respect to $\theta$ yields the solution $\theta = \theta^*$.

[8] While the possibility of exactly matching the true model is unrealistic in practice, the requirement that the true model perform best is conceptually plausible, and is the main idea behind forecast evaluation via proper scoring rules and related tools.

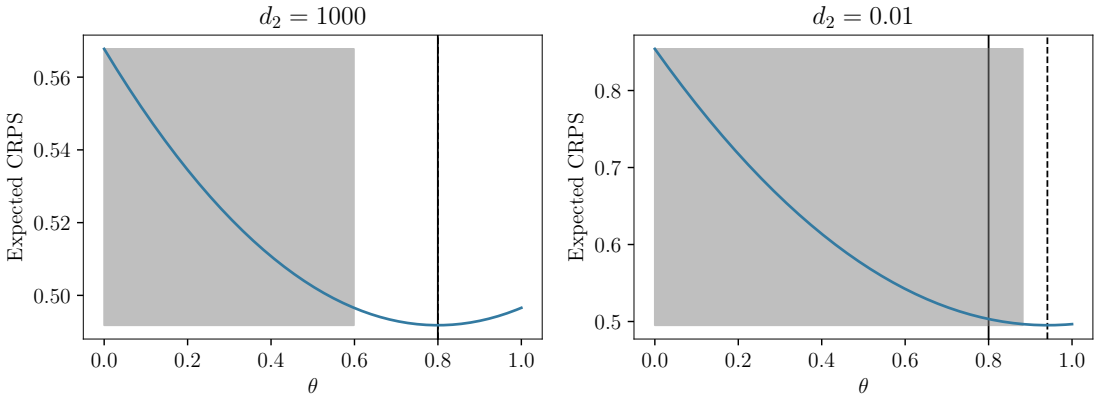



**Figure 3.6: Expected CRPS as a function of $\theta$.** The left panel refers to $d_2 = 1000$ (i.e., precise estimates of 'unimportant' probabilities), whereas the right panel assumes $d_2 = 0.01$ (i.e., noisy estimates). The other parameters are set as follows: $n = 100, k = 5, \theta^* = 0.8, d_1 = 1000$. Solid vertical line marks $\theta^*$, dashed vertical line marks best value for $\theta$. Shaded area marks range of values for $\theta$ that perform worse than $\theta = 1$ (corresponding to the Top$k$ method).

if $\theta$ is sufficiently close to $\theta^*$, and $d_1, d_2$ are sufficiently large, then the standard approach will outperform the Top$k$ method.

Conversely, the following conditions favor the Top$k$ method over the standard approach:

- $|\theta - \theta^*| > |1 - \theta^*|$, i.e. the standard method's implicit assumption that $\theta^* = \theta$ is worse than Top$k$'s implicit assumption that $\theta^* = 1$
- $d_1$ is large, i.e. important probabilities are estimated precisely
- $d_2$ is small, i.e. estimates of unimportant probabilities are noisy

If these conditions, or an appropriate combination thereof, hold, then the Top$k$ approach can be expected to perform well.

Figure 3.6 illustrates the above discussion. In the left panel (with $d_2 = 1000$), the 'unimportant' probabilities are estimated very precisely. Here the Top$k$ method is superior only to values $\theta \leq 0.6$ that are clearly smaller than the true parameter $\theta^* = 0.8$. In the right panel ($d_2 = 0.01$), the estimates of the unimportant probabilities are very noisy. Hence it is beneficial to focus on the important probabilities which are estimated precisely (since $d_1 = 1000$). Accordingly, the Top$k$ method - which focuses on the important probabilities exclusively - is superior to a wide range of values for $\theta$. Interestingly, this range includes the true parameter $\theta = \theta^*$, i.e., the Top$k$ method can be beneficial even if the probability estimates are correct in expectation.

## 3.5 Conclusion

This chapter has considered simplified RF forecast distributions that consist of a small number $k$ of support points, in contrast to thousands of support points (possibly equal to $n$, the size of the training set) of the original forecast distribution. The Top$k$ forecast distribution can be viewed as a collection of $k$ scenarios with attached probabilities. It hence simplifies communication and improves interpretability of the probabilistic forecast. Our empirical results in Tables 3.2 and 3.4 imply that simplified distributions using five or ten support points often attain similar performance as the original forecast distribution, while larger choices of $k$, e.g., 20 or 50, even increase performance slightly in many cases. Our analytical framework in Section 3.4 offers a theoretical rationale for these results. Our empirical analysis further shows that when tuning hyperparameters to the target value for $k$, even $k = 3$ can yield very good results.

While we have focused on the trade-off between simplicity (as measured by $k$) and statistical performance, the optimal choice of $k$ depends on the subjective preferences of the audience to which forecast uncertainty is communicated. In order to choose $k$ in practice, we recommend that communicators first assess the statistical performance of various choices of $k$ for their particular data set. In a second step, communicators may then interview potential users about their perceived cognitive costs of various choices of $k$. For example, Altig et al. (2022) argue that $k = 5$ resonates well with participants of an online survey on firm performance.

# 4 Efficient Prediction of Tandem Mass Spectra using the Neural Tangent Kernel

This chapter is based on joint work with Adityanarayanan Radhakrishnan, Julian Avila-Pacheco, Clary B. Clish, Martin Stražar, Ramnik J. Xavier, and Caroline Uhler. It presents a straightforward way to predict so-called mass spectra of a molecule based on its structure with a kernel ridge regression approach, utilizing a kernel that emulates an infinitely wide NN with much simpler training routines.

## 4.1 Introduction

Liquid chromatography – tandem mass spectrometry (LC-MS/MS), is a well-established and accurate technique to detect the presence of molecules in a sample, often drawn from blood, stool, or urine. Put simply, liquid chromatography isolates molecules found in the given sample. In a second step, tandem mass spectrometry fragments each isolated molecule into smaller substructures and their abundance is measured. This process results in mass spectra, which are typically unique to each molecule.

Specifically, in metabolomics, the study of small molecules that delineate life, one goal is to find and identify compounds in specific tissues, sites, or organs to link symptoms to specific compounds. For example, using LC-MS/MS, Wang et al. (2013) identified 2-aminoadipic acid as a biomarker for diabetes risk, and Roje et al. (2024) recently linked gut bacterial metabolism of environmental compounds to bladder cancer.

**Problem outline** Inferring the compound[1] from a mass spectrum, also known as annotation, is challenging because the spectrum records only the mass of fragments,

[1]The terms 'compound' and 'molecule' are used interchangeably in this work.

not the substructures themselves. Although experts can annotate mass spectra with some success, the process remains time-consuming and cumbersome. In an ideal world, databases containing all recorded mass spectra and their corresponding compounds would exist, allowing for simple lookups. While such databases of around $10^4$ mass spectra exist (Goldman et al., 2024), the space of compounds in metabolomics is estimated to be around $10^{60}$ (Kirkpatrick and Ellis, 2004). Other estimates suggest that we can currently annotate fewer than 2% of existing compounds (da Silva et al., 2015). Thus, the question remains: *How can we accurately, swiftly, and cost-efficiently find the molecule that corresponds to a given spectrum?*

If a database lookup is unsuccessful, the accepted method to prove the presence of any molecules in a sample is to synthesize the expected molecules and measure their mass spectra. This approach allows us to tackle the problem from the opposite direction. These 'pure' mass spectra, derived from the known, synthesized compounds, can then be matched against the mass spectra measured of our biological sample. Clearly, this process is time-consuming, costly, and may involve many iterations. Therefore, tools that predict a compound from a given spectrum ('spectrum-to-structure') or predict a spectrum from a given compound ('structure-to-spectrum') are needed to increase speed and success rate of this process. Perfect spectrum-to-structure and structure-to-spectrum methods would significantly reduce the number of molecules that need to be synthesized.

**Related Literature** Existing spectrum-to-structure methods attempt to infer the molecule from a given mass spectrum based on bottom-up reconstruction of the fragmentation process (Dührkop et al., 2019) or with machine learning approaches such as Goldman et al. (2023b) who used a transformer architecture. However, literature has mostly focused on the second mentioned approach, predicting a mass spectrum based on a given molecule (in the form of a feature vector). Wei et al. (2019), Murphy et al. (2023), and Goldman et al. (2024) use feed-forward NNs, a graph NN, and multiple graph NNs, respectively. Another branch of literature predicts spectra based on the physical properties of molecules, such as their structure, bonds between molecules, or the presence of certain molecules (Wang et al., 2021; Ridder et al., 2014; Wolf et al., 2010). Goldman et al. (2023a) use a hybrid approach, predicting fragments via physical properties and their respective intensities with a transformer-based NN. Existing approaches also differ in how they model the output space, i.e., the mass spectra. For example, Wei et al. (2019) use a simple binning scheme, while Goldman et al. (2023a) argue that predicting the

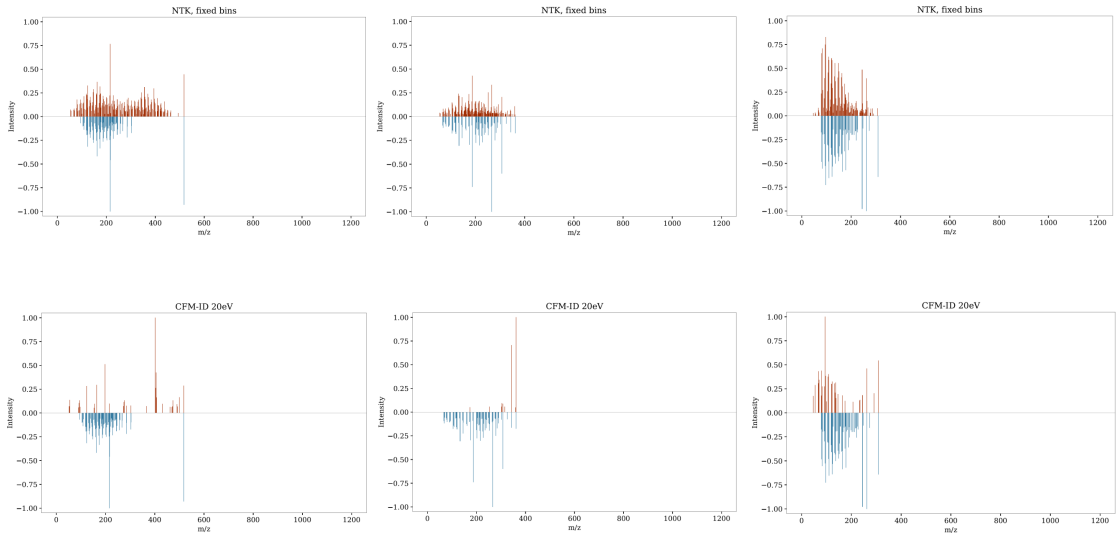



**Figure 4.1: Exemplary mirror plots** for our simple kernel method with the NTK and CFM-ID with 20eV. The blue mirrored spectrum is the true observation, the red-shaded spectrum is the respective prediction.

presence of chemical fragments is more accurate than binned estimations. The latter also found no significant impact on database retrieval performance between the two schemes.

**Contributions of this Work** Our approach follows structure-to-spectrum methods, utilizing kernel ridge regression, a flexible machine learning model that is fast in both training and prediction, unlike existing NN approaches. Utilizing the NTK (Jacot et al., 2018), we emulate the predictive behavior of an infinitely wide NN sidestepping the cumbersome training process, requiring only a few tunable hyperparameters. In contrast to theoretical fragmentation, our approach is purely data-driven. This way, we are able to predict fragments that were observed but are theoretically challenging to derive. Figure 4.1 visualizes predictions and true mirrored spectra for three compounds (columns) for our simple-bin NTK model (top row) and CFM-ID (bottom row) (Wang et al., 2021). More details on mass spectra can be found in Section 4.2.2. We refer to this type of plot as 'mirror plot' hereafter. Comparing the two models, we can see that our model predicts the general structure of the mass spectra more accurately, whereas the CFM-ID predictions are sparse and fail to predict even high intensity fragments. We investigate

multiple output modeling strategies using both uniformly-spaced and data-driven bins, which are not tied to a particular resolution and capture the most abundant fragments in the data set. Our experiments demonstrate that our models significantly outperform existing methods in the annotation of unknown spectra. To simulate the adaptability of our model to under-represented compound classes in data sets, we set up a scenario where we fine-tune or re-train the kernel models on a specific class of compounds. This is only viable in practice if training is fast, and we find that with only a few hundred additional samples, we achieve a significant advantage over the original model. Note that while in our data set all mass spectra are produced via LC-MS/MS, it is generally independent of the MS/MS method.

**Structure of the Chapter** The chapter is structured as follows: Section 4.2 introduces the theoretical concepts needed to understand our modeling approach and provides an intuitive introduction to tandem mass spectrometry. Section 4.3 presents our methodology and available data, including both the modeling side and the preprocessing steps undertaken to prepare the data. Section 4.4 explains our experimental setup and showcases the results. Finally, we discuss and conclude in Section 4.5.

## 4.2 Theoretical Background

In this section, we introduce the necessary concepts for this work, both formally and intuitively. We refer to more detailed literature for interested readers where applicable.

### 4.2.1 Kernel Methods

We now introduce kernel ridge regression, the predictive model used in our experiments, in an intuitive manner. Note that the use of kernel functions is not limited to regression. For an in-depth introduction to kernels and their applications, see Schölkopf and Smola (2001). For a Gaussian process perspective, we refer to Rasmussen and Williams (2005).

**Kernel Functions** We introduce the general concept of kernel functions, particularly positive definite kernel functions:

**Definition 4.2.1** (positive definite kernels)**.** *A positive definite (p.d.) kernel on a general set $\mathcal{X}$ is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that is symmetric:*

$$\forall(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{X}^2, \quad k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x}),$$

*and which satisfies, for all $N \in \mathbb{N}, (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) \in \mathcal{X}^N$ and $(a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$:*

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0.$$

We assume $\mathcal{X} \subseteq \mathbb{R}^d$ in the following. For vectors, the simplest useful kernel is the linear kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$, where the symmetry condition is obviously fulfilled. It is also positive definite, as $\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j \langle \boldsymbol{x}, \boldsymbol{x}' \rangle = || \sum_{i=1}^{N} a_i \boldsymbol{x}_i ||^2 \geq 0$.
More interesting and useful kernels can be written as $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle$ with $\Phi : \mathcal{X} \to \mathcal{H}$ and $\mathcal{H}$ denoting a Hilbert space of functions.[2]
For example, if $\boldsymbol{x} \in \mathbb{R}^2$ (and $\boldsymbol{x}_i$ denoting the $i$th coordinate of $\boldsymbol{x}$ in this example), a non-linear feature mapping $\Phi(\boldsymbol{x}) = (\boldsymbol{x}_1^2, \sqrt{2}\boldsymbol{x}_1\boldsymbol{x}_2, \boldsymbol{x}_2^2)$. This leads to the corresponding kernel function $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}_1^2 \boldsymbol{x}_1'^2 + 2\boldsymbol{x}_1\boldsymbol{x}_2\boldsymbol{x}_1'\boldsymbol{x}_2' + \boldsymbol{x}_2^2 \boldsymbol{x}_2'^2 = (\boldsymbol{x}_1\boldsymbol{x}_1' + \boldsymbol{x}_2\boldsymbol{x}_2')^2$, called the polynomial kernel. This mapping is useful for example, if the data in its original space is arranged in concentric circles and is therefore not linearly separable. With this feature mapping applied, we can fit a linear model with perfect discrimination. Aronszajn (1950) showed that any kernel function of the form $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle$ is a valid kernel for any choice of $\Phi$. Hence, we can map any point $\boldsymbol{x}$ to a function in $\mathcal{H}$, which we can consider an infinite dimensional feature mapping. We then compute the scalar product in $\mathcal{H}$. Put differently, a kernel function computes the 'similarity' of a data point $\boldsymbol{x}$ to all other data points in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$. The similarity between all pairs of input data can be summarized in a (symmetric and positive definite) kernel matrix $\boldsymbol{K} \in \mathbb{R}^{N \times N}$.
If we only care about this similarity between data points, it is therefore more convenient to merely imply a feature map $\Phi$ by defining a kernel function $k$ directly. This way, we only implicitly compute the inner product of two feature maps but explicitly calculate the similarity between $\boldsymbol{x}$ and $\boldsymbol{x}'$, which is sometimes called the 'kernel trick'.

[2] The codomain, i.e., the space $\mathcal{H}$ in which $\Phi$ maps to is called 'reproducing kernel Hilbert space', where the name stems from its 'reproducing' property: for a $\Phi := \sum_i^N \alpha_i k(\cdot, \boldsymbol{x}_i)$ with $\alpha_i \in \mathbb{R}$ and $\boldsymbol{x} \in \mathcal{X}$, we can verify that $\langle k(\cdot, \boldsymbol{x}), \Phi \rangle = \Phi(\boldsymbol{x})$. $k$ is then also called the reproducing kernel.

**Example 4.2.2** (prominent positive definite kernels)**.** *A prominent example of kernel functions is the Gaussian kernel:*

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp -\frac{||\boldsymbol{x} - \boldsymbol{x}'||_2^2}{\sigma^2},$$

*with parameter* $\sigma \in \mathbb{R}^+$.
*The polynomial kernel introduced above for the general case can be written as:*

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + c)^p,$$

*with* $p \in \mathbb{N}$ *determining the degree of the polynomial and parameter* $c \in \mathbb{R}$. *Note that parametrization can vary slightly depending on the notational style.*

In this work, we investigate the Laplace kernel besides the NTK.[3] It is also referred to as Ornstein-Uhlenbeck kernel, where compared to the Gaussian kernel we do not square the euclidean distance:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp -\frac{||\boldsymbol{x} - \boldsymbol{x}'||_2}{\sigma^2}. \tag{4.1}$$

We define the parameter $\gamma = \sigma^2$ and refer to it from hereon as bandwidth. Intuitively, the larger $\gamma$ is, the smoother $k$ will be and vice versa.

**Neural Tangent Kernel (NTK)** It is well-established that there is a correspondence between Gaussian processes (and therefore kernel methods) and NNs with infinitely wide hidden layers (see e.g., Neal, 1996; Williams, 1996; Rahimi and Recht, 2007), where the last layer is trained while all hidden layers remain fixed. Lee et al. (2018) provide a comprehensive overview of this correspondence and introduce the term neural network Gaussian process (NNGP). Jacot et al. (2018) recently introduced the NTK, a kernel function that resembles the learning dynamics of infinitely wide NNs with an arbitrary number of hidden layers. They show that, under certain initialization schemes, the solution provided by infinitely wide NNs is equivalent to that obtained by solving kernel ridge regression with the NTK. For an NN $f$ with parameter vector $\theta$, the NTK is defined as:

$$k_\theta(\boldsymbol{x}, \boldsymbol{x}') = \langle \nabla_\theta f(\boldsymbol{x}; \theta), \nabla_\theta f(\boldsymbol{x}'; \theta) \rangle.$$

[3] Note that in the literature, the same kernel with the L1-norm is sometimes also referred to as a Laplace kernel.

Intuitively, the NTK remains constant during training as NNs are linear in their parameters. For a rigorous derivation and proof, see Jacot et al. (2018), who examine the dynamics of training, or Liu et al. (2020), who provide a detailed analysis of the Hessian matrix of $f$. The analytical form of the NTK depends on the specific network architecture and activation function. In this work, we use the NTK for a feed-forward NN with ReLU activation of depth two. Following Bietti and Mairal (2019)[4], the NTK for this architecture, with $u \coloneqq \frac{\langle \boldsymbol{x}, \boldsymbol{x}' \rangle}{||\boldsymbol{x}|| \cdot ||\boldsymbol{x}'||}$, is defined as follows:

$$k(\boldsymbol{x}, \boldsymbol{x}') = ||\boldsymbol{x}|| \cdot ||\boldsymbol{x}'|| \left(u \cdot \kappa_0(u) + \kappa_1(u)\right), \tag{4.2}$$

where $\kappa_0(u) = \frac{1}{\pi}(\pi - \arccos(u))$ and $\kappa_1(u) = \frac{1}{\pi}(u(\pi - \arccos(u)) + \sqrt{1-u^2})$. Subsequent works introduced the NTK also for other NN architectures, including CNNs as well as recurrent NNs (Yang, 2020) and transformers (Hron et al., 2020). In practice, there exist libraries (e.g., Novak et al., 2020) that compute the NTK given an NN architecture. We can now either choose an NN architecture and continue development or search over the space of NTKs, much like neural architecture search (Elsken et al., 2019) but without hyperparameters such as initialization scheme or learning rate that do not apply to kernel ridge regression. Even though there may still be the need to tune hyperparameters, there are far fewer to tune and training a NTK is much faster compared to NNs. In summary, the NTK allows us to emulate an infinitely wide NN using a kernel function. Thus, we can avoid much of the cumbersome process of training such an NN but instead solve kernel ridge regression, which is introduced below.

**Kernel Ridge Regression** Our goal is to utilize and benefit from kernels in a regression setting: let us consider a univariate regression setting with $\mathcal{X} \subseteq \mathbb{R}^d$ as above, $\boldsymbol{y} \in \mathbb{R}^n$ and $\mathcal{D}_n = \{\boldsymbol{x}_i, y_i\}_{i=1}^n \in (\mathcal{X} \times \mathbb{R})^n$ a training set of size $n$. Our goal is to minimize the regularized squared error loss:

$$\arg\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i}^{n} (y_i - f(\boldsymbol{x}_i))^2 + \lambda ||f||_{\mathcal{H}}^2. \tag{4.3}$$

By the representer theorem (Kimeldorf and Wahba, 1970) which in summary states that the minimum norm solution lies in the span of all $k(\boldsymbol{x}, \cdot)$: $\hat{f}(\boldsymbol{x}) = \sum_i^n \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i)$ with

[4] See Chen and Xu (2021) for arbitrary depth.

$\boldsymbol{\alpha} \in \mathbb{R}^n$, we can write the last equation as

$$\arg\min_{\boldsymbol{\alpha}\in\mathbb{R}^n}(\boldsymbol{K\alpha} - \boldsymbol{y})^T(\boldsymbol{K\alpha} - \boldsymbol{y}) + \lambda\boldsymbol{\alpha}^T\boldsymbol{K\alpha}. \tag{4.4}$$

Since this is a convex problem, we can take the derivative with respect to $\boldsymbol{\alpha}$ and set it to zero. The optimal solution is then[5] $\boldsymbol{\alpha}^* = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$. Thus, we have our predictive function

$$\hat{f}(\boldsymbol{x}) = \boldsymbol{K_{xX}}(\boldsymbol{K_{XX}} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}, \tag{4.5}$$

where $\boldsymbol{K_{xX}} \in \mathbb{R}^{1\times n}$ denotes the kernel matrix of the test point $\boldsymbol{x}$ to all training samples $\boldsymbol{X}$ and similarly for $\boldsymbol{K_{XX}}$.

### 4.2.2 Liquid Chromatography - Tandem Mass Spectrometry

In order to understand the application and problem domain properly, we introduce MS/MS and specifically LC-MS/MS on an intuitive level. The interested reader is referred to Siuzdak (2024). LC-MS/MS is used to measure molecules, including metabolites, lipids and proteins in biological samples. Here, we focus on small molecule metabolites and lipids. Metabolites are small molecules, usually not heavier than 1200 g/mol that occur in a human's metabolism either as an intermediate or end product. The samples are usually collected in stool, urine or blood. The branch of life sciences that is concerned with such metabolites in any form is called metabolomics. The basic steps of LC-MS/MS are as follows:

1. *Inject* In metabolomics, a biological sample can be a drop of blood, which contains a complex mixture of thousands of molecules. To study individual compounds or prepare reference standards, which we address here, a purified compound in a solvent is injected. The sample is prepared by following an extraction protocol, which includes mixing with solvents (e.g., methanol) to extract molecules of a desired polarity and remove undesired molecules (e.g., proteins). The prepared sample is injected in a LC-MS/MS system.
2. *Separate* Compounds are separated in time by binding to a surface with defined chemical properties (stationary phase) with liquid chromatography. Over a defined time period (e.g., 10 minutes), the ratios between two solvents (e.g., water and

[5] The second derivative with respect to $\boldsymbol{\alpha}$ is $\boldsymbol{K} + \lambda\boldsymbol{I}$ which is positive definite.

acetonitrile) is varied to allow defined subsets of compounds to pass to the mass spectrometer.

3. *Ionize* The sample is ionized via electrospray ionization ('soft ionization'). A voltage is applied to a capillary through which the sample is passed, causing the liquids to disintegrate. This results in a gas of charged molecule-ions. One can control the amount of fragmentation by adapting the voltage.
4. *Accelerate* Since the fragments are charged, it is possible to accelerate them through an electric field (e.g., instantiated by a magnet).
5. *Detect* We measure the degree of deflection for each incoming fragment. Depending on the mass, a fragment is less deflected (higher weight) or more deflected (lower weight). The position of the incoming fragments depends on their mass and their charge, quantified as the *mass to charge ratio* (m/z). In our experiments, we always assume the charge to be equal to one, such that m/z effectively quantifies the mass measured in g/mol. In the following, we use the terms 'm/z' and 'mass' interchangeably. Note that in our case, the detected m/z value includes the adduct mass.
6. *Analyze* Compare results to reference measurements, which can be used to identify and quantify known compounds.

The result can then be visualized in a mass spectrum, where we plot the mass on the x-axis and the measured intensity on the y-axis. Most of the time, the largest recorded m/z value corresponds to the original compound as it is the heaviest 'fragment' of the compound and is also called precursor. Figure 4.2 shows a MS/MS spectrum for 'adipic acid' with the chemical formula $O_4C_6H_{10}$. As we can see, we find the last peak at an m/z value of around 146, which is the mass of adipic acid.

If we did not know the compound corresponding to this spectrum, we could predict a compound based on the present peaks. For example, the peak at an m/z value of around 55 likely corresponds to the fragment $C_4H_8$. As the stability of bonds between atoms determines how the analyzed molecule will fragment, it is possible to infer the molecules structure by analyzing the spectrum. A seasoned practitioner could then infer the compound corresponding to the spectrum. One may be tempted to interpret a spectrum as a distribution of continuous values. This, in fact, is a wrong interpretation. Take the following example: the molecules carbon dioxide ($CO_2$) and nitrous oxide ($N_20$)

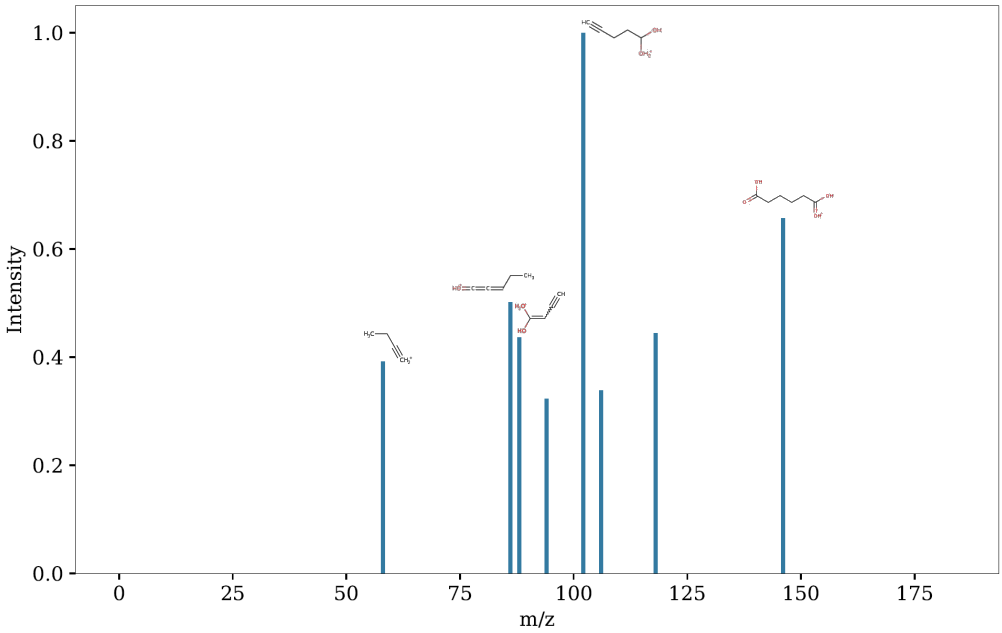



**Figure 4.2: Exemplary MS/MS spectrum** for a compound called 'adipic acid', $O_4C_6H_{10}$. CFM-ID identified some fragments, which are shown above the respective bar.

have a mass of 44.01 g/mol and 44.013 g/mol, respectively. Even though they would appear very close to each other on a mass spectrum, they are two very distinct molecules and cannot be compared with each other. Hence, values on the x-axis cannot be seen as continuous but rather as distinct classes, similar to a picture of a skyline, where each 'peak' corresponds to a unique skyscraper.

## 4.3 Methodological Setup

In this section, we introduce the data used in our experiments, describe the preprocessing steps undertaken, and outline the modeling approaches for both the input and output spaces, as well as the models themselves.

### 4.3.1 Training Data

In this work, we use MS/MS spectra collected from internal and a number of public sources. The complete training data set includes 28 832 spectra for 27 475 unique molecules (after preprocessing; we refer to the next section).

### Internal Data Sets

The internal standards were collected at the Broad Institute, Metabolomics platform using two LC-MS/MS methods. Data were acquired using LC-MS/MS systems comprised of Nexera X2 U-HPLC systems (Shimadzu Scientific Instruments) coupled to Q Exactive/Exactive Plus Orbitrap mass spectrometers (Thermo Fisher Scientific). The methods are optimized for extraction of polar metabolites and lipids from human biological samples such as plasma or stool and are described in Lloyd-Price et al. (2019). Spectra collected at multiple collision energies are merged by summing raw intensity values and normalized relative to the maximum value.

**Polar metabolites** A total of 1008 unique polar metabolites were collected using a 150 × 2-mm Atlantis HILIC column (Waters) in positive mode, yielding 1136 unique spectra, across multiple precursor ions where available.

**Lipids** A total of 228 unique lipids were collected using a 100 × 2.1-mm ACQUITY BEH C8 column (1.7 $\mu$m; Waters) in positive mode, yielding 336 unique spectra, across multiple precursor ions where available.

### Public Data Sets

**Human Metabolome Database (HMDB)** LC-MS/MS spectra were downloaded from Human Metabolome Database (HMDB; Wishart et al., 2022), which includes spectra from heterogeneous sources and formats. Where multiple spectra were available for compounds, intensities were first normalized relative to individual maximum values and subsequently merged into a single spectrum. This yielded 3443 spectra for 3194 unique human metabolites in positive mode.

**Global Natural Product Social Molecular Networking (GNPS)** LC-MS/MS spectra were downloaded from Global Natural Product Social Molecular Networking (GNPS; Wang et al., 2016) repository and normalized to their maximum value. This yielded 1691 spectra for 990 unique compounds in positive mode.

**MS2DeepScore** Finally, we also collect curated spectra from a recent publication (Huber et al., 2021) which includes 127 285 total spectra for 25 551 unique compounds.

### 4.3.2 Preprocessing

To ensure high data quality, we use an intricate preprocessing routine, which we present in the following.

1. *SMILES validity* SMILES (Weininger, 1988) is a string representation of a molecule, containing all relevant information about it, including its atoms and structure. Therefore, it is crucial that this SMILES string is correctly specified. We retain a spectrum if all the following conditions are met:
   - The given SMILES is syntactically valid and contains only the following elements: C, H, N, O, P, S, Na, B, I, Cl, Se, Br, F, Si, K.
   - Meta information is complete.
2. *Fragment database* During preprocessing, we attempt to match any signal in a spectrum to a valid fragment. To do so accurately and swiftly, we create a subset of the BUDDY database (Xing et al., 2023) by filtering out subformulae not part of the given SMILES and masses larger than the compound's mass.
3. *Peak processing* For each peak in each spectrum, we apply the following steps:
   a) The m/z values are rounded to six decimal places.
   b) We subtract the adduct mass from the m/z values to increase flexibility as we are now independent of the used adduct.
   c) If a peak is below 0.3% of the maximum peak intensity of the spectrum, we assume this to be peak noise and therefore ignore it.
   d) If a peak occurs at an m/z value larger than the compound's mass plus 24 g/mol (in order to account for measurement inaccuracies), we ignore it.
   e) If we can identify the peak as the 'parent peak' (m/z value of the peak is within 5ppm of the precursor mass), there is no need to look up the fragment. If we cannot, we perform a lookup in our database subset. If we find a fragment within 25ppm of the considered peak, we retain the peak in the spectrum; otherwise, we ignore it.
4. *Conditions on processed spectrum* If any of the following conditions are met, the spectrum is discarded:

- The spectrum contains fewer than three identified peaks.
- The maximum intensity of the remaining peaks is less than 5% of the maximum intensity value before preprocessing.

5. *Reverse lookup & duplicates* Once all noisy (according to our heuristic) peaks are removed and the spectrum is not discarded, we perform a reverse lookup step. For example, if a compound has a mass of 400 m/z and we observe a peak at m/z value 72.11 (which could be Butanal, $C_4H_8O$), we should theoretically also observe a peak at $400 - 72.11 = 327.89$. If this peak is not present in the data and a lookup in the database is successful, we artificially add it to the spectrum with the same intensity as the observed peak. With MS/MS, it can sometimes be hard to detect small compounds. Thus, this is step especially useful to add small fragments, where only the larger counterpart was detected. To ensure that there are not duplicate peaks (whether or not they resulted from the reverse lookup), we remove any duplicates, retaining the maximum intensity.

Out of 133 308 total available spectra, we keep 28 832. Additionally, we find that some compounds occur multiple times in the data set. As a simple heuristic, for each of those spectra, we retain those that lost the fewest peaks during preprocessing. This intuitively signifies the quality of the raw data: the more peaks retained, the higher the signal. The final data set size will depend on the modeling of the output space and is discussed below.

### 4.3.3 Modeling Approaches

Our goal is to predict a MS/MS spectrum given a compound represented as a SMILES string. In this section, we introduce our modeling approach and aspects thereof.

#### Output Space Modeling

As spectra are theoretically vectors with infinite dimensionality, it is crucial to model the output space with care: while we want the output space to be as realistic as possible, a large output space increases model complexity. Let $\boldsymbol{Y} \in [0, 1]^{n \times b}$ denote the matrix containing all $n$ MS/MS spectra contained in the data set and $\boldsymbol{y} \in [0, 1]^b$ a single MS/MS spectrum, where $b \in \mathbb{N}$ specifies the output space dimension.

**Intensity Scaling** We first scale the peaks within each spectrum $\boldsymbol{y}$, where we choose to follow Wei et al. (2019) and scale the peaks by taking the square root element-wise before normalizing such that $\max \boldsymbol{y} = 1$:

$$\boldsymbol{y} = \frac{\sqrt{\boldsymbol{y}}}{\max \sqrt{\boldsymbol{y}}}. \tag{4.6}$$

We investigate two options for binning the scaled spectra:

**Fixed Binning** This approach is straightforward and commonly used in MS/MS prediction (Wei et al., 2019; Goldman et al., 2024, among others). We bin a spectrum in a fixed number of bins $b$. While simple, it may not capture the reality of MS/MS spectra: a detected fragment at an m/z value $h$ may be something entirely different than a second fragment detected at $h + \epsilon$, yet, they may end up in the same bin. However, our experiments have shown, that it seems to be sufficient to choose a relatively low bin number of $b = 2200$. The final data set contains 28 832 spectra, 27 475 of which are unique molecules.

**Flexible Binning** One way to prevent different fragments falling in the same bin is to output intensities for all prevalent fragments in the data set. As this would be infeasible, we can restrict ourselves to the most frequent fragments. By controlling the minimum number of occurrences, we also control for the number of 'bins' $b$. A downside of this approach, however, is that important fragments that do not occur as often (for example, heavier fragments) may not be modeled at all. As flexible binning removes peaks, we keep spectra only if they exhibit at least three peaks and if the maximum intensity is at least 0.1 (while unlikely, it is possible that large peaks are discarded during the binning process). For our experiments, we choose a minimum number of occurrence of 250, which results in $b = 3030$ bins. With the altered modeling, the final data set contains 23 255 spectra, all of which unique molecules.

### Predictive Model

To predict spectra from a given structure, we design kernel methods because of their flexibility and speed in training and inference. Kernel regression as presented in Section 4.2.1 are most commonly used in univariate prediction settings. Multivariate extensions of kernel regression exist, however, they are computationally intractable for our case as

the kernel matrix would be of size $(n \cdot b \times n \cdot b) \approx 6.3 \times 10^7 \times 6.3 \times 10^7$ depending on the specific choices of $n$ and $b$. See for example Álvarez et al. (2012) for an overview on such models. Hence, we need to find the balance between practicality and theoretical soundness. We suggest the following options:

**Single Model** Here, we use a single kernel regression model to predict the entire spectrum $\boldsymbol{y}$ for each compound. This approach results in a very simple, easy to understand and fast model. A downside of this approach is that we weigh each dimension within $\boldsymbol{y}_i$ with the same weight. Recall Equation 4.5:

$$\hat{f}(\boldsymbol{x}) = \underbrace{\boldsymbol{K}_{\boldsymbol{x}\boldsymbol{X}}(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \lambda \boldsymbol{I})^{-1}}_{=\boldsymbol{\alpha}} \boldsymbol{Y} = \sum_i \alpha_i \boldsymbol{y}_i^{(b)}, \tag{4.7}$$

where we denote each spectrum in the training data set as $\boldsymbol{y}_i$ and highlight the multivariate nature of $\boldsymbol{y}_i$ with a superscript $(b)$. Although the solution $\boldsymbol{a}$ to the linear system is not modeled independently for each output dimension, dependence between bins is modeled implicitly as the dependence of occurring peaks in each $\boldsymbol{y}_i^{(b)}$ is given by the data. Computationally, this modeling approach is very compelling as we only need to invert the training kernel matrix $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \lambda \boldsymbol{I}$ once, resulting in a training time of a few seconds on a single GPU.

**Multi Model** For flexible bins, we also introduce a multi-model approach where each bin is treated separately by training independent, univariate models on the respective bins, as described in Section 4.2.1. Note that if we did not change the input to these models, this would be equivalent to the single model approach. For each compound, we only use models for those bins, for which the m/z value corresponds to a subset of a compound's chemical formula. Put differently, we 'mask' the training data set for each bin-model. For example, if the given compound has the chemical formula $O_4C_6H_{10}$, we do not include this data point for bins which correspond to a fragment containing nitrogen $(N)$ as this is impossible. Thus, the number of inputs for each model (corresponding to each bin) varies. Theoretically, the same approach is applicable for fixed bins, however, this masking approach is not sensible as we cannot map each bin to a single compound. Compared to the single model approach, we need to invert a (different) kernel matrix $d$ times, which makes this approach comparatively slow.

**Model Input** We use (unscaled) Morgan fingerprints (Morgan, 1965) with 1024 bits with the precursor m/z value appended, which has been standardized for all binning and model variants. These fingerprints are a way of quantifying a molecule by binary encoding specific structures such as rings or types of bonds between atoms in a given molecule. The fingerprints are calculated based on the SMILES of a given molecule. It is possible, that two different (but similar) compounds result in the same Morgan fingerprint, however, we chose to follow current literature (Wei et al., 2019; Goldman et al., 2023a, 2024) to maintain comparability. Let $\boldsymbol{X} \in \mathbb{R}^{n \times 1025}$ denote the matrix containing all $n$ feature vectors contained in the data set and $\boldsymbol{x} \in \mathbb{R}^{1025}$ a single feature vector. Without the appended m/z value, we have 15 439 unique fingerprints and 24 712 with the respective m/z value appended.

**Kernel Hyperparameter Choices** We choose hyperparameter values for all of our kernel methods based on default values frequently used in the literature (Belkin et al., 2018b; Radhakrishnan et al., 2023, 2024) and minor tests on a hold-out data set. As kernel models are not lower-bounded by zero but spectra are bounded between zero and one, we further introduce a threshold hyperparameter for the respective model outputs, to get rid of small, presumably noisy and possibly negative predicted peaks. We fix this threshold to $2 \times 10^{-2}$ for all experiments and models. The parameters for all models are summarized in Table 4.1.

**Table 4.1: Kernel hyperparameters.** Summary of all hyperparameter choices.

| Model | Kernel | Hyperparameter | Value |
|---|---|---|---|
| All | All | Threshold | $2 \times 10^{-2}$ |
| Single | Laplace | Bandwidth | 10 |
| | Laplace | Regularization | 1 |
| | NTK | Depth | 2 |
| | NTK | Bias | True |
| | NTK | Regularization | 25 |
| Multi | Laplace | Bandwidth | 10 |
| | Laplace | Regularization | 1 |
| | NTK | Depth | 2 |
| | NTK | Bias | True |
| | NTK | Regularization | 60 |

## 4.4 Experiments

In this section, we present our experimental setup as well as the respective results.

### 4.4.1 Experimental Setup

In order to showcase the performance and versatility of our kernel-based approach to MS/MS prediction, we carry out a series of different experiments which highlight their value for different application scenarios.

**MS/MS Prediction** The goal in this multivariate regression task is to predict an MS/MS spectrum based on a input vector that quantifies a molecule. The prediction is evaluated using the metrics listed in Table 4.2. Our problem encompasses elements from both classification and regression tasks, hence we must consider different metrics capturing those different aspects. For example, a model with nearly perfect squared error could still predict many small, noisy peaks where no peak occurs. Conversely, a TNR of one could be achieved by predicting no intensity. We include the squared error as it is minimized by the kernel models. Cosine similarity measures the angle between two vectors, or equivalently, the un-centered correlation between their elements. It ranges from zero (uncorrelated, vectors are perpendicular) to one (perfectly correlated, vectors align perfectly). Both metrics are useful as they measure different aspects of regression and are used in the annotation experiments (see next paragraph). We also report the true positive rate (TPR), true negative rate (TNR), false positive rate (FPR), and false negative rate (FNR), commonly used for evaluating classification tasks. A peak is considered 'present' if its value exceeds zero (or a chosen threshold).

Our models are evaluated using 4-fold cross-validation, utilizing approximately 75% of the data set for training. To compare our models on a test set, we use the fold that was not trained on the particular sample. The test set comprises 8826 samples.

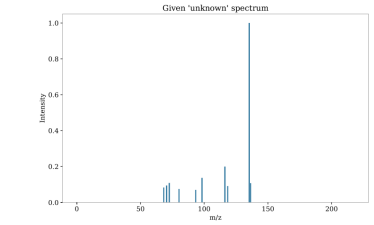
Comparing our results directly to those reported in other works is not possible, as many use a different data set and setup. Therefore, we benchmark our models against NEIMS (Wei et al., 2019), ICEBERG (Goldman et al., 2024), and CFM-ID (Wang et al., 2021) on our data. Predictions for ICEBERG at 10eV, 20eV, and 40eV were kindly provided by the authors. Since these predictions were provided to us, we used the same binning scheme for evaluation, i.e., an upper m/z limit of 1500 and 15 000 bins. For training and evaluating NEIMS, we use the same setup, preprocessing, and hyperparameters as

**Table 4.2: Prediction evaluation metrics.** We use those metrics to evaluate our forecasts. If not explicitly mentioned otherwise, we take the mean over all test samples. The definitions for the four classification metrics are slightly adapted to fit our case. $||\cdot||_0$ denotes the L0-norm, which counts the number of non-zero elements in a vector.

| Metric | Perspective | Formula |
|---|---|---|
| Squared Error (SE) | Regression | $\sum_{i=1}^{b}(\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i)^2$ |
| Cosine Similarity | Regression | $\frac{\langle \hat{\boldsymbol{y}}, \boldsymbol{y} \rangle}{\|\|\hat{\boldsymbol{y}}\|\|_2 \|\|\boldsymbol{y}\|\|_2}$ |
| true positive rate (TPR) | Classification | $\frac{\sum_{i=1}^{b} \mathbf{1}\{\hat{\boldsymbol{y}}_i > 0\} \cdot \mathbf{1}\{\boldsymbol{y}_i > 0\}}{\|\|\boldsymbol{y}\|\|_0}$ |
| true negative rate (TNR) | Classification | $\frac{\sum_{i=1}^{b} \mathbf{1}\{\hat{\boldsymbol{y}}_i = 0\} \cdot \mathbf{1}\{\boldsymbol{y}_i = 0\}}{\sum_{i=1}^{b} \mathbf{1}\{\boldsymbol{y}_i = 0\}}$ |
| false positive rate (FPR) | Classification | $\frac{\sum_{i=1}^{b} \mathbf{1}\{\hat{\boldsymbol{y}}_i > 0\} \cdot \mathbf{1}\{\boldsymbol{y}_i = 0\}}{\sum_{i=1}^{b} \mathbf{1}\{\boldsymbol{y}_i = 0\}}$ |
| false negative rate (FNR) | Classification | $\frac{\sum_{i=1}^{b} \mathbf{1}\{\hat{\boldsymbol{y}}_i = 0\} \cdot \mathbf{1}\{\boldsymbol{y}_i > 0\}}{\|\|\boldsymbol{y}\|\|_0}$ |

described by the authors (5000 bins), but additionally evaluate a model trained on spectra with 2200 bins. CFM-ID predictions are binned into 2200 bins to maintain comparability.

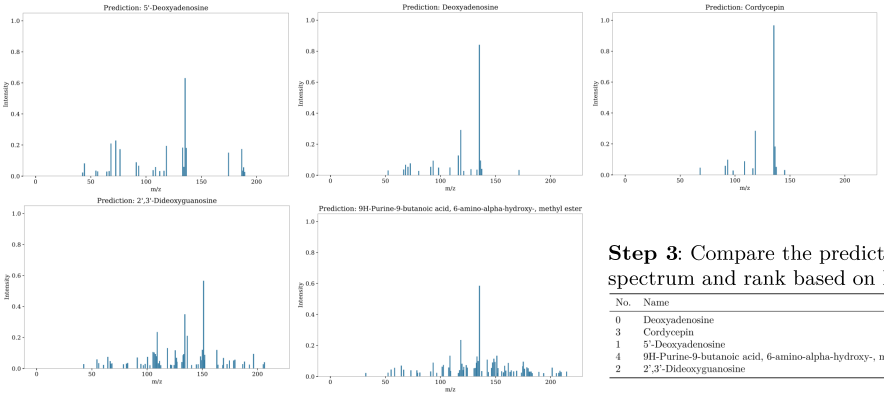
**Annotation** The goal of the annotation task is to identify the compounds corresponding to given but unknown MS/MS spectra $\boldsymbol{S} \in [0,1]^{a \times b}$, where $a \in \mathbb{N}$ represents the number of spectra stored in rows, and $b$ represents the number of bins as described in Section 4.3.3. To test performance on this task, we need spectra for which we know the corresponding compounds and candidates. For our annotation data set, we have $\sum_{i=0}^{b} c_i = 16\,548$ and $b = 3422$, averaging about five candidates per 'unknown' spectrum. Figure 4.3 depicts the process and visualizes the steps undertaken for the spectrum shown in the top left corner. We proceed in three steps: First, we identify all possible $c \in \mathbb{N}$ candidates for each given spectrum $\boldsymbol{S}_i$, as shown in the table on the top right of Figure 4.3. A candidate is any compound with the same mass as the true compound, inferred from the spectrum. In general, some candidates may be similar in structure compared to the true corresponding molecule. Second, for each given spectrum $\boldsymbol{S}_i$, we predict the MS/MS spectrum for each of its candidates $\hat{\boldsymbol{S}}^{(i)} \in [0,1]^{c \times b}$. In Figure 4.3, these predictions are shown in the bottom five spectra. Third, we measure the similarity between a given spectrum $\boldsymbol{S}_i$ and each corresponding predicted spectrum (one for each candidate) and rank $r \in \mathbb{R}$ the predicted




**Step 1**: Find all candidates to the given spectrum (left) $\boldsymbol{S}_i$
For our experiments, we know the compound the spectrum corresponds to

| No. | Name | SMILES | Truth | Mol. Weight |
|---|---|---|---|---|
| 0 | Deoxyadenosine | Nc1ncnc2c1ncn2C1CC(O)C(CO)O1 | True | 251.1018 |
| 1 | 5'-Deoxyadenosine | CC1OC(n2cnc3c(N)ncnc32)C(O)C1O | False | 251.1018 |
| 2 | 2',3'-Dideoxyguanosine | Nc1nc(=O)c2ncn(C3CCC(CO)O3)c2[nH]1 | False | 251.1018 |
| 3 | Cordycepin | Nc1ncnc2c1ncn2C1OC(CO)CC1O | False | 251.1018 |
| 4 | 9H-Purine-9-butanoic acid, 6-amino-alpha-hydroxy-, methyl ester | COC(=O)C(O)CCn1cnc2c(N)ncnc21 | False | 251.1018 |

**Step 2**: Predict the spectrum for each of the candidates (below) $\hat{\boldsymbol{S}}^{(i)}$




**Step 3**: Compare the predicted spectra to the given spectrum and rank based on L2 or cosine score $\tilde{\boldsymbol{S}}^{(i)}$

| No. | Name | Truth | Score | Cos. Score |
|---|---|---|---|---|
| 0 | Deoxyadenosine | True | -0.313871 | 0.958287 |
| 3 | Cordycepin | False | -0.351713 | 0.943723 |
| 1 | 5'-Deoxyadenosine | False | -0.631083 | 0.801839 |
| 4 | 9H-Purine-9-butanoic acid, 6-amino-alpha-hydroxy-, methyl ester | False | -0.731294 | 0.722701 |
| 2 | 2',3'-Dideoxyguanosine | False | -1.342248 | 0.044175 |

**Figure 4.3: Demonstration of the annotation experiment** for a compound called 'Deoxyadenosine'. In practice, the given spectrum is unknown.

spectra accordingly, as shown in the bottom right corner in Figure 4.3. Similarity is measured using cosine similarity and L2-distance (square root of the squared error). While ongoing research explores measuring the distance between different MS/MS spectra (e.g., Huber et al., 2021; Li et al., 2021), cosine similarity remains a straightforward and established measure (Bittremieux et al., 2022; Marissen et al., 2023). We use L2-distance for comparison. Denote $\tilde{\boldsymbol{S}}^{(i)}$ as the matrix that holds predictions $\hat{\boldsymbol{S}}^{(i)}$ with rows sorted according to the ranking. The resulting ranking for each spectrum in $\boldsymbol{S}_i$ represents the model's ability to match a spectrum to its corresponding compound. If the predicted spectrum that ranks first is the true spectrum, our model predicts spectra well enough such that it is closest to the true spectrum. Performance is measured by Top-$k$ accuracy, where we calculate the fraction of true spectra within the top $k$ ranked spectra:

$$\text{Top-}k \text{ Accuracy} = \frac{\sum_{i=1}^{a} \mathbf{1}\{\boldsymbol{S}_i \in \tilde{\boldsymbol{S}}^{(i)}_{[:k]}\}}{a},$$

using the subscript $[:k]$ to select the first $k$ rows.

The annotation experiment is based on models trained for MS/MS prediction. Therefore, we use the same models and benchmarks as described above. Additionally, we randomly rank candidates for each spectrum to establish a lower bound.

**Transfer Learning** It is common for practitioners to have only a few compounds (possibly newly synthesized) available to train a model for their specific use case. For example, we may only be interested in a certain class of compounds whose spectra have unique features. Hence, it is crucial that a model can adapt to a specific set of compounds. We can either use a model as is and hope it can predict any set of compounds well, we can re-train the model on novel training data, or apply transfer learning on our model to adapt to the specific compounds at hand. Since kernel models are fast to train, they can be re-trained easily, especially compared to NNs. Therefore, we investigate the transfer learning and re-training capabilities of our models by recreating such a scenario: We train a fixed-bin, single-kernel model on a training set that includes samples from all but one available compound class.[6] Next, we test the model's performance for prediction and annotation on an unseen class of compounds, which is exclusive to the test set. In subsequent steps, we iteratively add samples of this class of compounds to the training set to either completely retrain the model or apply transfer learning and test at each step. This approach allows us to observe how incremental additions of data impact the model's performance, providing insights into the model's adaptability. The transfer learning approaches are based on Radhakrishnan et al. (2023), where a kernel model is post-processed in various ways. Let $\boldsymbol{X}_s \in \mathbb{R}^{n_s \times d}$ be the source data set with corresponding observations $\boldsymbol{y}_s \in \mathbb{R}^{n_s}$, and similarly, let $\boldsymbol{X}_t \in \mathbb{R}^{n_t \times d}$ be the target data set with corresponding observations $\boldsymbol{y}_t \in \mathbb{R}^{n_t}$, where $n_t << n_s$. A model $\hat{f}_s$ was trained on the source data set, and we wish to adapt it to the target data set. We apply four methods, described intuitively below. The first three methods follow Definitions 1–3 of Radhakrishnan et al. (2023).

1. *Projection* We use the prediction of $\hat{f}_s$ as input for $\hat{f}_t$. Our prediction is then $\hat{\boldsymbol{y}}_t = \hat{f}_t(\hat{f}_s(\boldsymbol{X}_t))$.

2. *Translation* We treat the residuals of $\hat{f}_s$, i.e., $\boldsymbol{r}_t \coloneqq \hat{f}_s(\boldsymbol{X}_t) - \boldsymbol{y}_t$, as observations. The prediction will then be $\hat{\boldsymbol{y}}_t = \hat{f}_s(\boldsymbol{X}_t) + \hat{f}_t(\boldsymbol{X}_t)$.

[6] For the sake of overview, we chose our simplest model. The overall transfer learning approach does not vary for different modeling approaches.

3. *Projection & Translation* This is a combination of the previous two approaches, where we use the prediction of $\hat{f}_s$ together with the targets regressors as input, i.e., $\tilde{\boldsymbol{X}}_t \coloneqq \begin{bmatrix} \hat{f}_s(\boldsymbol{X}_t) & \boldsymbol{X}_t \end{bmatrix}$, and regress it on $r_t$ as defined above. Our prediction is then $\hat{\boldsymbol{y}}_t = \hat{f}_s(\boldsymbol{X}_t) + \hat{f}_t(\tilde{\boldsymbol{X}}_t)$.

4. *Re-training* Train a model $\hat{f}_t$ on the whole data set, i.e., $\tilde{\boldsymbol{X}}_t \coloneqq \begin{bmatrix} \boldsymbol{X}_s & \boldsymbol{X}_t \end{bmatrix}$. In other words, we do not apply transfer learning but train a single model on all available data. Kernel methods allow this due to their simple training procedure.

In our experiment, we use available information from the HMDB-class (Wishart et al., 2022) within the data set, listed in Table 4.3. For the transfer learning experiment,

**Table 4.3: HMDB-Class distribution** in our data set. The class 'Unknown' summarizes all fragments for which we do not have information about the respective class, while 'Other' summarizes all other, less frequently occurring classes.

| HMDB class | Count |
|---|---|
| Benzene and substituted derivatives | 1905 |
| Carboxylic acids and derivatives | 1458 |
| Flavonoids | 1205 |
| Organooxygen compounds | 993 |
| Steroids and steroid derivatives | 990 |
| Prenol lipids | 863 |
| Glycerophospholipids | 731 |
| Fatty Acyls | 609 |
| Indoles and derivatives | 316 |
| Other | 6372 |
| Unknown | 13 390 |

we train a source model on all samples except for the class 'Benzene and substituted derivatives'. We then apply transfer learning or re-train the model on randomly drawn samples from that class and test it on a fixed set of 300 randomly drawn samples (also from that class). Figure 4.4 visualizes the data points used for training, transfer learning, and testing using t-SNE (van der Maaten and Hinton, 2008). Gray points represent the training data, green points represent the test data, and red points represent the transfer-learning data. To reduce overhead and maintain clarity, we use the same hyperparameters for the target model as for the source model. This consistency ensures that any observed differences in performance are due to the transfer learning process rather than changes

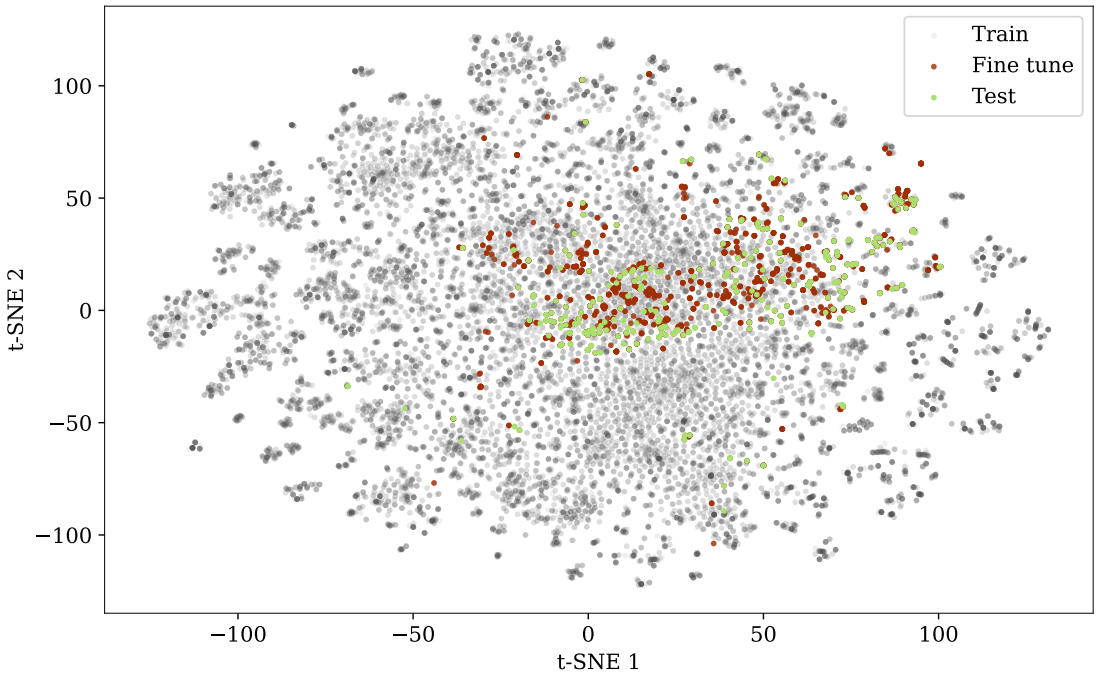



**Figure 4.4: t-SNE of fingerprints** where we indicate data points in gray that used for training the source model. Green symbolizes the test set and the red dots are used for transfer learning. Both red and green data points make up the class of 'Benzene and substituted derivatives'.

in hyperparameters. Additionally, maintaining the same hyperparameters simplifies the experimental setup and reduces potential sources of variability.

### 4.4.2 Results

In this section, we present the results on the three experimental setups outlined in Section 4.4.1.

#### MS/MS Prediction

The regression results on the employed performance metrics for our MS/MS prediction models are shown in Table 4.4.

First, we analyze the kernel models, and then we compare them to the benchmark models. Note that we cannot compare results from models with different output sizes because the problem is different. Hence, we interpret those models together with the same output space modeling. Regarding SE for models with 2200 bins, both kernel

models perform at a similar level. While the NTK achieves higher cosine similarity and TNR, Laplace leads with a higher TPR (and therefore lower FNR). Compared to NEIMS and CFM-ID, they also achieve a similar SE but outperform the baselines in cosine similarity and TPR. However, both benchmarks have very high TNR, suggesting that kernels overpredict the presence of peaks, while NEIMS and CFM-ID (along with ICEBERG) seem to predict relatively sparse spectra. Regarding the flex-binning kernel models, we do not see a big difference in SE for both model types and kernel functions. As for the fixed binning models, the Laplace kernel is trailing slightly behind the NTK in cosine similarity. While the multi models seem to predict more peaks due to the higher TPR and lower TNR compared to the single models, neither SE nor cosine similarity seem to be affected by this fact. These results indicate that while all of our models may produce more false positives, they are also more sensitive in detecting true peaks, which could be advantageous in certain applications where missing a peak is more critical than predicting an extra one.

**Table 4.4: MS/MS prediction results.** For CFM-ID and ICEBERG, we selected the respective best performing version. We add the number of bins in thousand for each model behind the binning scheme. The respective best results for each metric are shown in bold face. Cosine Similarity is abbreviated 'CS' and is shown in percent, as are TPR, TNR, FPR and FNR. Fixed binning is abbreviated 'Fix.'.

| Binning | Model | Kernel | SE | CS | TPR | TNR | FPR | FNR |
|---|---|---|---|---|---|---|---|---|
| Fix. (2.2) | Single | NTK | 0.0022 | **54.85** | 74.67 | 95.76 | 4.24 | 25.33 |
| Fix. (2.2) | Single | Laplace | 0.0023 | 52.01 | **81.78** | 94.79 | 5.21 | **18.22** |
| Flex (3.03) | Single | NTK | 0.0009 | 49.74 | 57.18 | 97.41 | 2.59 | 42.82 |
| Flex (3.03) | Single | Laplace | 0.0010 | 46.37 | 56.98 | 97.56 | 2.44 | 43.02 |
| Flex (3.03) | Multi | NTK | 0.0009 | 49.50 | 71.16 | 93.38 | 6.62 | 28.84 |
| Flex (3.03) | Multi | Laplace | 0.0009 | 48.29 | 72.54 | 93.66 | 6.34 | 27.46 |
| Fix. (2.2) | NEIMS | - | 0.0020 | 32.81 | 26.38 | 99.36 | 0.64 | 73.62 |
| Fix. (2.2) | CFM-ID | - | 0.0030 | 37.59 | 19.35 | **99.56** | **0.44** | 80.65 |
| Fix. (15) | ICEBERG | - | 0.0004 | 42.98 | 57.58 | 99.50 | 0.50 | 42.42 |

We suspect that our models predict too many peaks due to the nature of kernel models being a weighted sum over training observations (refer to Equation 4.5). Thus, if the data exhibits presumable noisy peaks, it is difficult to remove noise in their predictions entirely.[7] A higher threshold value could potentially reduce the issue. To verify, we

[7] Note that it can be hard in practice to distinguish a noisy peak from signal.

visualized the $\boldsymbol{\alpha}$-values (refer to Equation 4.7) for a simple Laplace kernel with fixed binning for 30 randomly drawn test samples in Figure 4.5. A negative value is shown in a red shade, a positive value is shown in blue, and zero-values are shown as white. The resulting matrix is sometimes referred to as 'smoother matrix' in statistical literature (Hastie et al., 2009). Each test case is shown in a row, training observations with known and frequently corresponding HMDB-class (roughly 6800 compounds) are shown as columns. Training observations are grouped by their respective HMDB-class and within the group sorted by the first PCA-dimension based on their feature vector. To maintain overview, we choose to discard samples for classes 'Unknown' and 'Other'. We refer to Figure C.4 in the appendix for the same visualization including all training samples. Clearly, most values for each case are non-zero which confirms our hypothesis. A second reason for 'noisy' peaks may be that the kernel models are optimized on the SE. This

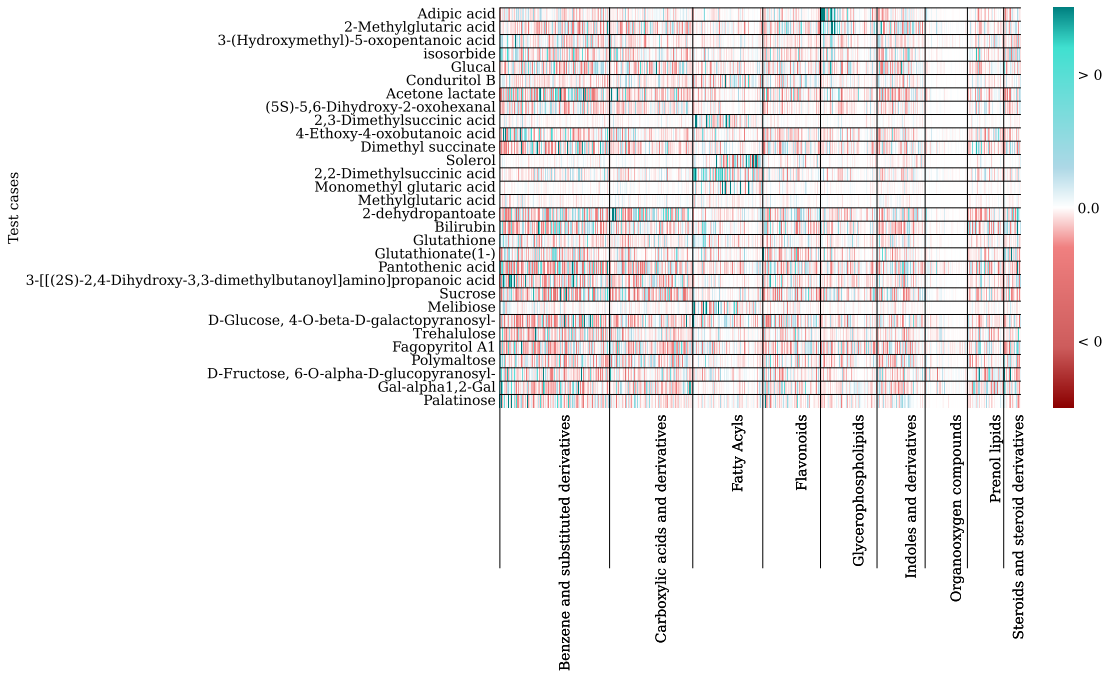



**Figure 4.5: Smoother matrix (partial) for test predictions** of the simple Laplace with fixed binning. Each Row, separated by a black line, represents a test point, each column a training point, grouped by the respective HMDB-classes. We only show known and frequently observed classes, discarding class 'Unknown' and 'Other'. See Figure C.4 in the appendix for the same matrix with all training compounds. Columns within each class are sorted by their first PCA dimension of their Morgan fingerprint. Negative weights are shown in red, positive weights are shown in blue and zero-weights are visualized as white.

loss incentivizes a model's (probability) prediction to be accurate and calibrated (in classification tasks, the SE is also referred to as Brier score; see e.g., Gneiting and Raftery, 2007). Thus, if a model predicts values close to zero for a particular m/z value, we can interpret this as the model's (low) expected probability of intensity being present. In other words, the model expects a peak with some chance if a 'noisy' peak occurs.

We can further use Figure 4.5 to interpret the predictions. For example, none of the test compounds are steroids and steroid derivatives, prenol lipids, glycerophospholipids nor fatty acyls. Indeed, this is reflected by the weight values as many columns are zero for these classes and additionally flavonoids. This suggests that the kernel model has learned meaningful relations between observed spectra (columns) and unseen test cases (rows). Moreover, we can detect similarities in rows for some compounds. For example, the rows for 'acetone lactate', '(5S)-5,6-dihydroxy-2-oxohexanal' and '4-ethoxy-4-oxobutanoic acid' exhibit a lot of similarities. Considering Figure C.5 in the appendix visualizing chemical structures of some test compounds, we can see that they are indeed similar in their structure. This is also the case for 'monomethyl glutaric acid' and 'methylglutaric acid'. On the other hand, 'Palatinose' is structured very differently to the previously mentioned compounds and their corresponding $\boldsymbol{\alpha}$-values reflect this. This highlights that we can intuitively and visually verify what the kernel model has learned in an intuitive way. We further inspect our predictions visually, where Figure 4.6 visualizes a randomly chosen test sample prediction for the simple kernels with fixed bins and the mirrored true spectrum. The particular molecule is called 'Quetiapine'. Both kernels output similar spectra in their overall structure. They pick up the most prominent peaks, but also exhibit noisy peaks which resembles the interpretation of Table 4.4. For this case, it seems that the NTK is able to predict intensities more accurately compared to the Laplace kernel. Further, while both kernel functions predict the precursor correctly, the Laplace kernel predicts a small peak to the right of the precursor. However, it fails to find the correct largest m/z value as there are a few, smaller peaks exceeding the compounds m/z value which the Laplace kernel is not. In practice this is not a problem as we always know a compounds m/z value. Thus we can simply discard peaks at m/z values larger than the compounds m/z value. To compare, Figure 4.7 shows the the mirror plot of CFM-ID and ICEBERG with 20eV. For this particular sample, the ICEBERG prediction confirms the results shown in Table 4.4 as it is sparse, with relatively few of the high-intensity peaks of the true spectrum being predicted. Whether the sparsity is due to the model being

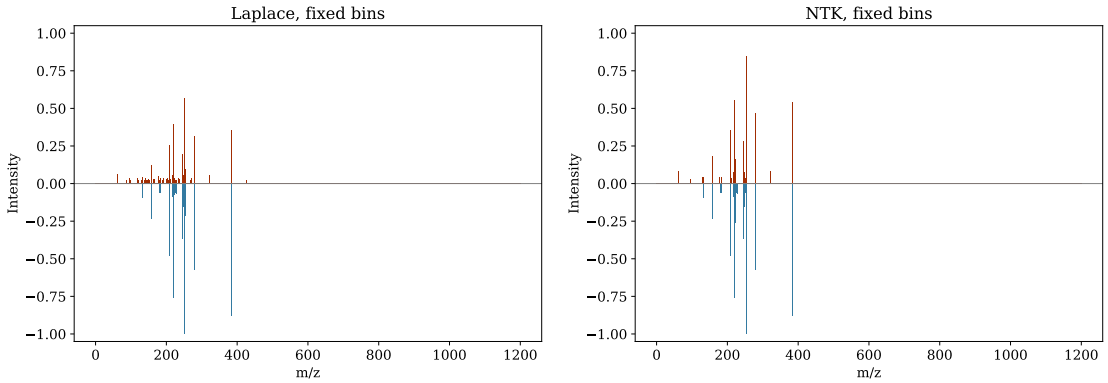
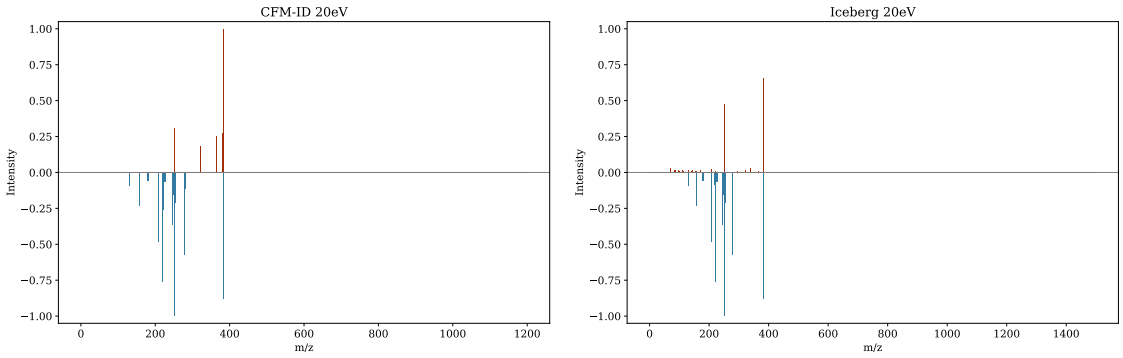



**Figure 4.6: Mirror plot for simple, fixed-bin models** where the Laplace prediction is shown on the left and the NTK prediction is visualized on the right. The blue mirrored spectrum is the true observation, the red-shaded spectrum is the respective prediction.

based on chemical properties of the compound or not is unclear. CFM-ID's prediction is less sparse but fails to predict most peaks correctly. For the sake of overview, we include mirror plots for all other models in Appendix C.1.

### Annotation

The results for the annotation task are presented in Table 4.5. While we could not compare models with different output space modeling in the previous section, we can do so for the annotation task. In this task, it is sufficient to predict a given spectrum well enough that the predicted spectrum of the true compound is closest to the given true spectrum. All kernel models outperform our benchmarks by at least four percentage points (pp), depending on the considered distance metric. NEIMS and ICEBERG perform at a similar rate (even though NEIMS is a much simpler architecture), while CFM-ID performs only marginally better than random guessing with the L2 distance and about 8 to 9 pp better with cosine similarity. The choice of kernel function, similar to the prediction results above, does not significantly influence ranking accuracy. However, for both L2 distance and cosine similarity, the NTK slightly outperforms Laplace in all setups. The best-performing model is the multi-model NTK with flexible bins. Given the additional complexity of the multi-model approach and the negligible increase in performance compared to the single-model approaches (both fixed and flexible bins), the practitioner's preference will determine which model is more appropriate for the




**(a)** CFM-ID **(b)** ICEBERG

**Figure 4.7: Mirror plots for CFM-ID & ICEBERG prediction** with 20eV. The blue mirrored spectrum is the true observation, the red-shaded spectrum is the respective prediction. Note that ICEBERG is using 15000 bins up to an m/z value of 1500, while CFM-ID is binned to 2200 bins up to 1200 m/z.

annotation task. These results support the findings of Goldman et al. (2023a), indicating that predicting uniformly-sized bins result in similar performance compared to data-driven bins, i.e., present fragments. Moreover, cosine similarity appears to be a better metric than L2 distance, as it ranks more accurately in most cases. This makes sense, because intensity readouts are not consistent across MS/MS instruments. Cosine similarity is therefore more robust to outliers.

Considering both prediction and annotation, it is evident that neither prediction metric alone can reliably predict good annotation performance. Even though the single model NTK with fixed binning achieves the highest cosine similarity, it does not peak in annotation performance. Similarly, the single models with flex binning achieve a relatively low TPR, yet perform just as well in the annotation step compared to the other kernel methods. Considering the benchmark models, while they all exhibit a very high TNR, this does not yield in good annotation performance, especially for CFM-ID. Thus, annotation does not seem to require the most accurate predictions as long as they are 'similar enough'. To summarize, the kernel methods demonstrate a good trade-off between predictive performance and the ability to annotate well, outperforming benchmarks in all key metrics.

**Table 4.5: MS/MS annotation results.** For CFM-ID and ICEBERG, we selected the same models as above.

| Binning | Model | Kernel | Top-1 Acc. [%] | |
|---|---|---|---|---|
| | | | L2 | Cosine |
| Fixed | Single | NTK | 88.98 | 90.01 |
| Fixed | Single | Laplace | 87.64 | 89.36 |
| Flex | Single | NTK | 88.75 | 90.41 |
| Flex | Single | Laplace | 87.96 | 89.80 |
| Flex | Multi | NTK | 88.60 | **90.50** |
| Flex | Multi | Laplace | **89.04** | 90.21 |
| - | Random | - | $\sim$ 49-50 | |
| - | NEIMS | - | 83.58 | 81.18 |
| - | CFM-ID | - | 51.55 | 58.26 |
| - | ICEBERG | - | 82.14 | 80.95 |

### Transfer Learning

Figure 4.8 visualizes the annotation performance on 300 test samples from the HMDB-class 'Benzene and substituted derivatives', as shown in Figure 4.4. We run the experiment several times, iteratively adding more data points (visualized red in Figure 4.4) to the transfer learning model and record the performance for all methods. Performance at '0' shows the performance of the source model on the test points, i.e., no transfer learning was applied. The dotted lines highlight this performance for the NTK (blue) and Laplace kernel (orange), respectively. Without transfer learning, the annotation accuracy for both distance metrics is about 75%, clearly better than random guessing. When applying transfer learning, particularly the Projection and Projection & Translation methods drop significantly in performance up to about 200 samples for both kernel functions. The re-trained kernels remain at approximately the same level until about 200 samples. Thereafter, all methods steadily improve in performance. For both distance metrics, the NTK Projection & Translation and Laplace Projection methods require the most samples to regain the source model's performance. Interestingly, the cosine distance seems to be more forgiving, as the two methods reach the original performance with about 500 samples, while for L2, the models need 800 samples. Unsurprisingly, the Re-training methods are relatively stable from the beginning and show increased performance starting at 500 samples. Projection & Translation and Re-training seem to be the superior

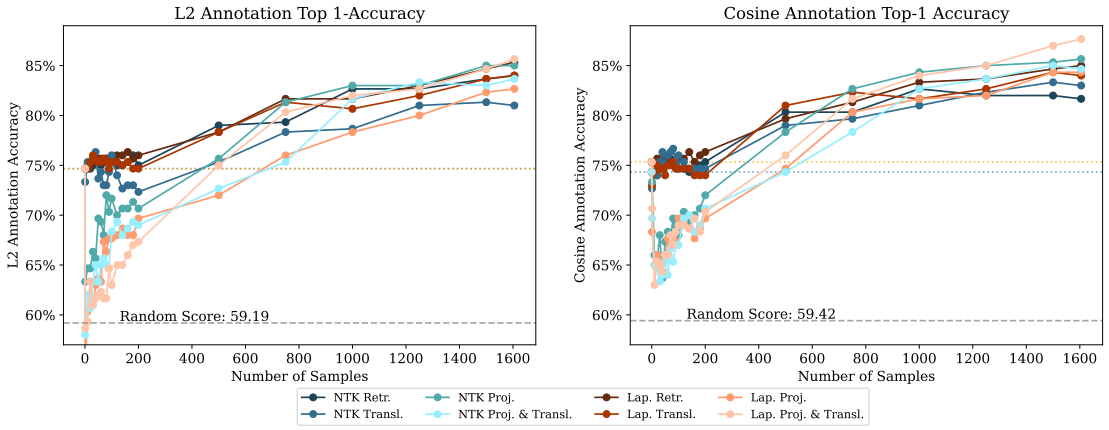



**Figure 4.8: Transfer learning: Annotation performance** of different methods methods and kernels. Laplace is abbreviated as 'Lap.', Projection as 'Proj.', Translation as 'Transl.' and Re-training as 'Retr.'. The gray dashed lines show the respective random score. The dotted blue and orange lines show the performance of no transfer learning applied for the NTK and Laplace kernel, respectively.

methods for the Laplace kernel, while Projection results are better for the NTK. The final annotation accuracies reach over 85% for the Laplace Projection & Translation method. However, all methods improve with transfer learning and re-training, suggesting the usefulness of each method. Depending on the method, only a few hundred samples are needed to increase performance.

Figure 4.9 reports on the average number of predicted peaks for each transfer learning run. The test set's average number of predicted peaks is 31.79, visualized as the dashed line. We can see that for the first few runs, both kernel functions for Projection and Projection & Translation predict almost no peaks. This can be attributed to the fact that only a few samples are used for transfer learning, showcasing that this is not enough data for the models to learn anything meaningful. Apart from this effect in the early stages, all methods and kernels heavily over-predict the presence of peaks. While the number decreases in later runs, it is still roughly twice as high as in the observed data. Figures C.9, C.10, C.11, and C.12[8] show mirror plots for the same compound at different stages of transfer learning and re-training. All models change their predictions quite significantly for a better prediction. Comparing methods for this particular compound, Projection does not model the intensities as accurately as the other methods. Furthermore, all models find the correct positions of prominent peaks in the true spectrum but also exhibit

[8]Due to space constraints, the figures are presented in Appendix C.1.

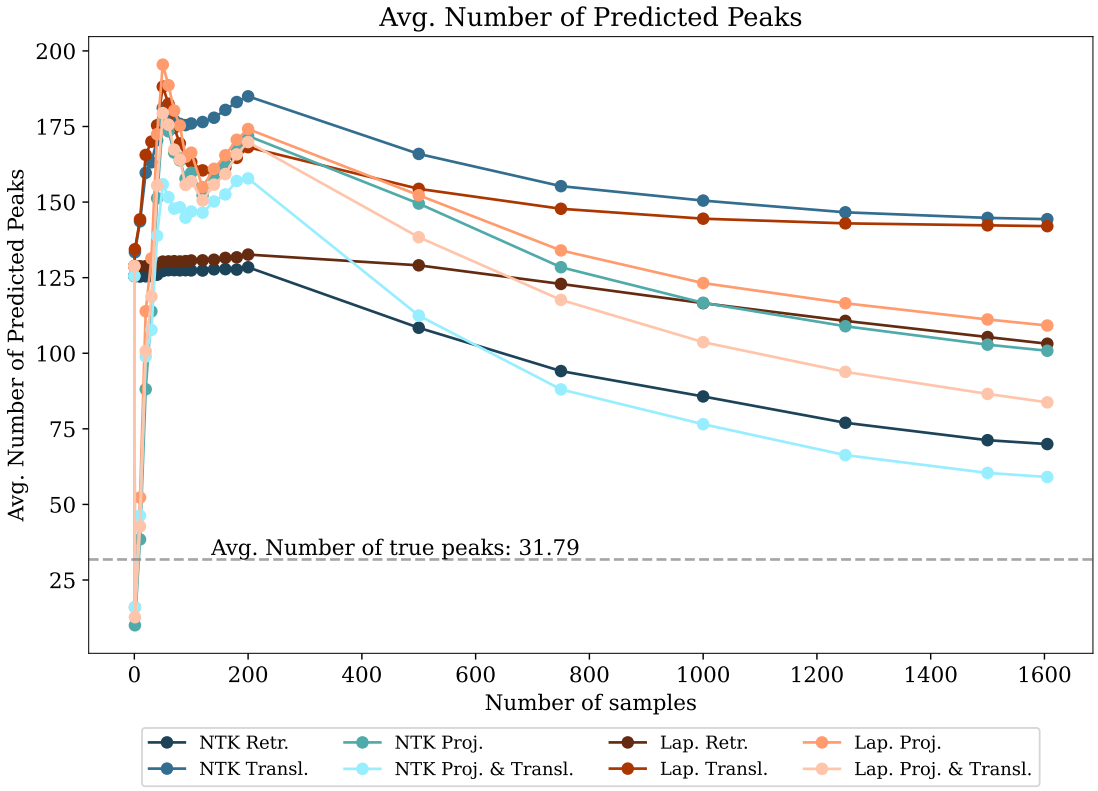



**Figure 4.9: Transfer learning: Average umber of predicted peaks** in each transfer learning run. Laplace is abbreviated as 'Lap.', Projection as 'Proj.', Translation as 'Transl.' and Re-training as 'Retr.'.

noisy peaks. Combining these results, over-predicting the presence of peaks does not seem to negatively impact annotation performance. The remaining metrics for each transfer learning run can be found in Figures C.6, C.7, and C.8 in Appendix C.1. To summarize, all transfer learning methods improve performance compared to the source model when testing on a (at least somewhat) distinct group of compounds, previously unseen for sample sizes larger than 400, depending on the transfer learning method applied. Re-training seems to be the most stable method, as performance fluctuates slightly for a small number of new samples. Projection & Translation excels for a larger number of samples. For all methods, both kernel methods perform at a similar level. This increases the flexibility of our overall modeling approach, as we outperform current literature's models, train much faster, and can adapt our models to the specific task at hand.

## 4.5 Discussion

In this chapter, we demonstrated the feasibility of predicting MS/MS spectra using kernel ridge regression. In humans, there are many more metabolites than we are currently able to measure (da Silva et al., 2015), primarily due to a lack of reference standards. Running chemical standards is the most reliable approach, but synthesis can be expensive, costing up to thousands of dollars for complex molecules. With improved predictive methods, this step can become much more accessible, allowing us to look for predicted spectra instantly with negligible cost.

**Improved Performance** The NTK, which emulates the learning dynamics of an infinitely wide NN, slightly outperformed the Laplace kernel in our experiments. Additionally, all tested kernel models and modeling approaches surpassed existing literature on the annotation task. The similar performance of the NTK and Laplace kernel supports recent theoretical work indicating that they share the same RKHS (Chen and Xu, 2021). Our approach is particularly useful given its simplicity. Compared to NNs, which existing literature predominantly focuses on (Wei et al., 2019; Goldman et al., 2023a, 2024, among others), the development process for kernel ridge regression, with only a few hyperparameters, is much simpler and faster.

**Improved Handling and Interpretability** Training an NN almost always requires extensive and complex hyperparameter tuning, leading to a lot of back-and-forth during development. Moreover, inherent interpretability allows us to assess whether the model learned useful patterns and understand how predictions were formed. The smoother matrix in Figure 4.5, recently advocated by Coulombe et al. (2024) as an interpretation tool, serves as a good example. For example, we can deduce that a particular sample class is under-represented in the training data if a 'smoother vector' exhibits little sign of similarities. Kernel regression, with its close relation to Gaussian processes, can also be used to output an uncertainty estimate for intensities. Future work can focus on an adapted modeling approach that accounts for this. While our setup results in many predicted peaks, some of them possibly noisy, we argue that estimating spectra from data alone can be of advantage (Xing et al., 2023). For example, it includes the presence of radical fragment ions (Xing and Huan, 2022).

**Transfer Learning and Re-Training** Finally, our approach provides the flexibility and speed to adapt to any available data, as demonstrated in Section 4.4.2. Re-training is important because of the poor representation of many structural classes in open databases. Therefore, open and straightforward methods will play a key role here. The importance of focusing on specific classes is shown in recent generative modeling approaches, optimized for, e.g., psychoactive molecules (Skinnider et al., 2021) or human primary metabolites (Qiang et al., 2024), where the generated compounds are simulated through CFM-ID and searched in human samples. Improved structure-to-spectrum predictions are key to an affordable approach. With our method, we argue that we provide such an affordable, interpretable, and straightforward modeling approach that, in the future, can potentially be combined with generative models.

# 5 Learning to Forecast: The Probabilistic Time Series Forecasting Challenge

This chapter is based on a joint *publication* with Johannes Bracher, Fabian Krüger and Sebastian Lerch, in *The American Statistician* (Bracher et al., 2024). It deals with the development and insights of a novel lecture, in which students weekly issue and submit probabilistic forecasts in a competitive and real-time environment. Permission to reuse the article, as well as the figures, in this work is granted by the copyright holder.

## 5.1 Introduction

Forecasting is a core task of statistics. Accordingly, the development, implementation, and evaluation of prediction models play a central role in statistics courses within degree programs such as economics, mathematics, and computer science. Much of statistical forecasting takes place in a time series context, where the goal is to predict future observations of a univariate or multivariate time series. Reflecting the breadth and depth of time series analysis, there is a wealth of excellent textbooks, teaching materials, and university courses on the topic. For the sake of coherence and simplicity, however, even applied courses typically refer to idealized settings that do not match the requirements of real-world time series forecasting.

In particular, practical forecasting by definition refers to quantities not yet known at the time of prediction. Academic research and teaching, on the other hand, typically use historical data examples, and the quantities to be predicted are not actually unknown to the forecaster. The real-time setting can be mimicked, e.g., by considering a rolling forecast origin, sometimes also referred to as 'rolling window' or 'rolling sample'. In this approach, the data are repeatedly split into training and test sets (using the $R$ newest observations up to time $t$ to predict the outcome at $t + 1$, observations up to $t + 1$ for the outcome at $t + 2$, and so forth). This, however, is only an imperfect

imitation of the real-time setting. Firstly, modeling decisions are inevitably taken based on knowledge of the entire time series, thus including information which would not have been available in real time. Secondly, the procedure smooths over many imperfections of real-time data, which may be subject to initial errors, delays, and revisions. (While some impressive efforts have been made to compile data sources that reconstruct forecasters' past information – see e.g., Croushore (2006) for economic examples – the availability of such sources remains an exception.) Both aspects contribute to hindsight bias, i.e. after the fact the observed events may seem more predictable than they actually were. This tendency is further reinforced by the absence of time constraints in retrospective forecasting.

Practical forecasters need a diverse set of skills in statistics, involving theoretical knowledge (e.g., being able to select and test appropriate methodology), programming skills, as well as the ability to keep track of the forecasting workflow (ideally by using professional techniques like version control). Teaching these skills – and practicing them ourselves – was our main motivation for offering a course on real-time forecasting. As an additional feature, we sought to emphasize *probabilistic* forecasts which acknowledge and quantify uncertainty (Gneiting and Katzfuss, 2014). These considerations led to the 'Probabilistic Time Series Forecasting Challenge' that we describe in this chapter. Its setup was inspired by collaborative forecasting efforts in epidemiology (e.g., Cramer et al., 2022). In the hope of making the course as entertaining as possible, we combined this format with gamification-type aspects (Hamari et al., 2014; Dicheva et al., 2015) that are also present in data science competitions on the Kaggle platform (Bojer and Meldgaard, 2021). All rules and procedures for the challenge were pre-defined in a public preregistration which was available to participants (Bracher et al., 2021a).

This chapter describes our experiences and learnings from running this novel course format at Karlsruhe Institute of Technology (KIT) in Germany between October 2021 and February 2022. The second edition of the course just started at the time of this writing. We hope that the chapter is of interest to statisticians who consider offering a similar project to an audience of students or professionals.

The chapter is organized as follows. Section 5.2 describes the structure of the course, thereby highlighting relevant design parameters. Section 5.3 compares the course format to existing collaborative forecasting projects. Section 5.4 presents empirical results on participants' forecasts, and Section 5.5 provides a concluding discussion. To avoid clutter,

we collect web resources in Table D.1 of the appendix and refer to them as 'URL1' to 'URL15' in the text. Additional empirical analyses can be found in Table D.2 in Appendix D.1. Forecast and outcome data are available on the course's GitLab repository at URL1. Code to replicate the empirical analysis in the present chapter is available at URL2. An interactive visualization of the forecasts and outcomes is available at URL3.

## 5.2 Structure of the Forecasting Challenge

In this section, we describe the context of the course, as well as its collaborative forecasting process.

### 5.2.1 Context

**Background of Participants** Bachelor's and Master's students enrolled at KIT in the degrees of industrial engineering and management, computer science and business mathematics were eligible to apply for the course. In the Bachelor's program, these degrees contain at least two lectures covering basic statistical knowledge and three lectures of mathematical foundations, ensuring that all applicants had sufficient knowledge of the topic. Based on their transcript of records as well as a short letter of motivation, we admitted two Bachelor's, twelve Master's, and four PhD students (with the latter not required to pass an examination). Of these 18 students, 17 participated continuously throughout the challenge.

**Provided Material** In preparation, we recorded several video lectures for the students. Students could hence re-watch the lectures, a substantial benefit as we anticipated a steep learning curve in the first few weeks of the course. Besides information about the course logistics, the lectures contained basics in regression modeling, probabilistic forecasting, time series analysis and real-time forecasting as well as pointers to R (R Core Team, 2022) and Python (van Rossum et al., 2011) resources. While we did not restrict the students' choice of programming language, our support was limited to R and Python. We further provided code for the respective baseline models (all three variables), for advanced benchmark models (for temperature and wind, see below), as well as some data loading scripts and scripts to check the correct format of a submission file.

### 5.2.2 Targets and Benchmarks

**Target Variables** Each Wednesday, participants were required to provide forecasts for the following three target variables (see Section 5.2.3 for details on the submission process):

- accumulated *log-return* of the German stock index (Deutscher Aktienindex) (DAX) of the next five business days (defined as 100 times the difference between the log-transformed value of the DAX at the target time and its last known value at the time of prediction). We used Yahoo Finance (URL4) as an easily accessible data source.
- *temperature* 2 meters above ground, measured in °C, for a weather station close to Karlsruhe (Rheinstetten) for 12:00 UTC on the respective upcoming Thursday, Friday and Saturday as well as 0:00 UTC on the respective upcoming Thursday and Friday.
- *wind speed* 10 meters above ground, measured in km/h, for the same weather station and horizons as for temperature.

Figure 5.1 shows the time series of the three targets and illustrates the chosen lead times. As the goal of the challenge was probabilistic forecasting, we required forecasts for the respective 2.5%, 25%, 50%, 75% and 97.5% quantiles for said targets. Participants were allowed to use any additional data sets available to them.

By choosing these targets, we aimed for a diverse set of forecasting tasks in a classical time series format. The data are relatively easy to access and understand, and constructing plausible statistical forecasts is feasible for university students.

Unfortunately, the Rheinstetten weather station, which had been chosen for its geographic proximity to Karlsruhe, did not report data due to a connectivity problem during the first week of the challenge. We therefore switched to the Berlin-Tempelhof station (URL5) whose observations displayed broadly similar empirical properties.

**Setup for Numerical Weather Processing (NWP)** Nowadays, weather forecasts are typically based on numerical weather processing (NWP) models which represent physical processes via systems of partial differential equations. Weather services usually run an ensemble of NWP model simulations, e.g. based on randomly perturbed initial

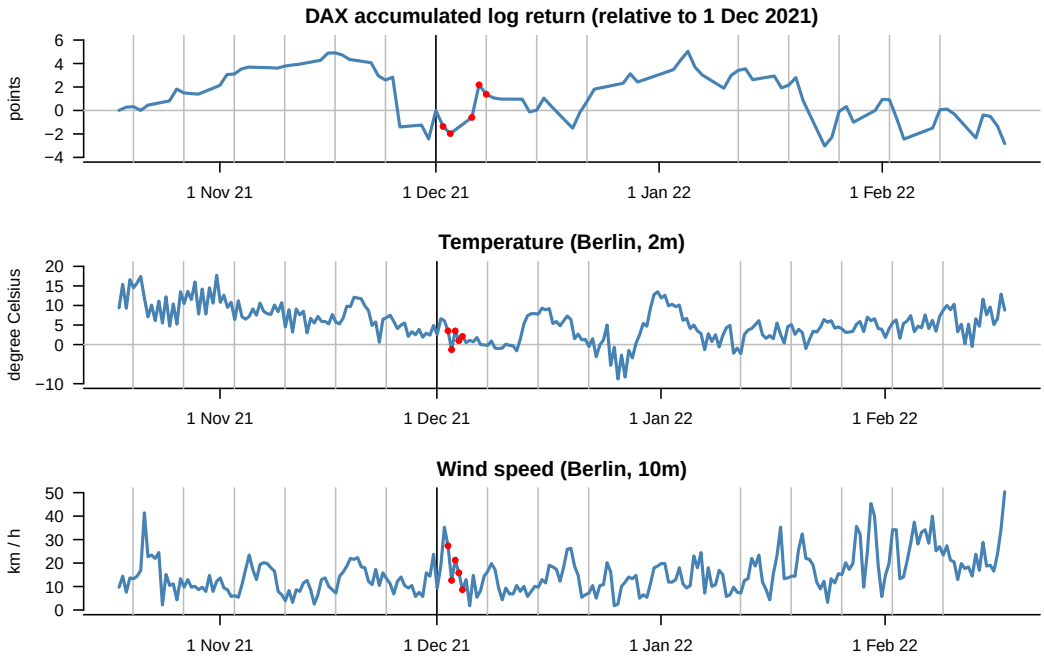



**Figure 5.1: Time series to be predicted**: accumulated log returns of the DAX stock index, temperature 2 meters above the ground and wind speed 10 meters above the ground in Berlin (note: only values at 0:00 and 12:00 UTC shown). Grey vertical lines show time points at which forecasts were made. To illustrate the defined forecast lead times, the black vertical lines show 1 December and the red dots the quantities which needed to be predicted on 1 December.

conditions. To reflect the common practice in weather forecasting for the temperature and wind speed targets in the setting of the challenge, we utilized NWP ensemble forecasts from the operational weather prediction model of the German weather service (Deutscher Wetterdienst) (DWD), the sahedral Nonhydrostatic (ICON) model (Zängl et al., 2015). The ICON model is run at a 20 km resolution over Europe and contains 40 ensemble members. While ICON forecast data is openly available via DWD's Climate Data Center (URL6), downloading and handling these large data sets (e.g., to extract forecasts at specific locations) can be cumbersome and requires specialist knowledge. Using automated downloading and pre-processing scripts run on a remote server, we provided up-to-date ICON ensemble forecasts to all participants. The forecasts were disseminated on a daily basis in the form of text files via a dedicated KIT-internal GitLab repository, and contained predictions of seven weather variables (mean sea level pressure, total cloud cover, direct downward shortwave radiation, temperature at 2 meters and at

850 hPa, wind speed and maximum wind gust at 10 meters) with lead times up to 120 hours for the target location(s).

NWP ensembles are typically biased and lack calibration. They thus require correction via so-called post-processing methods. The distributional regression approaches employed in operational weather forecasting and research range from simple statistical techniques to advanced machine learning methods, see Vannitsem et al. (2021) for a recent overview. One of the most popular approaches is the ensemble model output statistics (EMOS) method proposed by Gneiting et al. (2005) where a parametric distribution is assumed for the target variable. The distribution parameters are then modeled as functions of summary statistics from the NWP ensemble predictions. To enable the implementation of post-processing models for the weather targets, we compiled an additional data set of past ICON ensemble predictions and corresponding observations at the target locations from December 2018 to September 2021 which was provided to the participants, along with implementations of standard EMOS models for temperature and wind based on the crch software package for R (Messner et al., 2016).

**Benchmark Models** For each target variable, we defined a benchmark model in the study preregistration. These benchmark models, which are summarized in Table 5.1, were intended to put participants' forecast performance into perspective (e.g. by calculating skill scores). Our benchmark models were chosen to be simple and easy to understand, while still being empirically plausible. In addition to the benchmark models, we generated EMOS forecasts for the two weather targets. All benchmark forecasts were based on code that we also provided to participants.

**Table 5.1: Summary of benchmark models**.

| Variable | Description of benchmark |
|---|---|
| DAX | Empirical return quantiles, computed from a rolling sample $\mathcal{S}$ of 1 000 days, i.e. $\mathcal{S} = \{r_{t-999,h}, \ldots, r_{t,h}\}$, with $r_{t,h} = 100 \times (\log \mathrm{DAX}_t - \log \mathrm{DAX}_{t-h})$ |
| Wind | Empirical quantiles of raw ensemble forecast |
| Temperature | Empirical quantiles of raw ensemble forecast |

**Ensemble Forecasts** Ensembles (or 'combinations') are a common and empirically successful tool to summarize a set of forecasts of the same target variable (see e.g. Gneiting

and Raftery, 2005; Timmermann, 2006). In our context, ensembles summarize the numerous contributed predictions and illustrate the potential of collaborative forecasting. Specifically, we generated two different forecast ensembles:

1. *Mean ensemble* The $\alpha$ quantile of the ensemble is given by the mean of the respective $\alpha$ quantiles of its members.

2. *Median ensemble* The $\alpha$ quantile of the ensemble is given by the median of the respective $\alpha$ quantiles of its members.

As specified in our preregistration of the course project, we initially planned to consider an inverse score-weighted ensemble in addition. However, we ultimately refrained from adding this layer of complexity as we found the simpler mean and median ensembles well sufficient for the purpose of summarizing collaborative forecast performance in the course.

### 5.2.3 Submission and Evaluation

**Submission** The project took place throughout the whole winter semester 2021/2022, where the first submission had to be handed in by October 27th, 2021. Due to the Rheinstetten weather station's connectivity problems noted above, weather forecasts started only at November 3rd, 2021. The last submission deadline was February 9th, 2022. Excluding a two-week Christmas break, this resulted in 14 weeks of DAX forecasts and 13 weeks of weather variable forecasts. Each Wednesday, forecasts needed to be sent to us via email in a specified .csv-format by 11:59 PM. Table 5.2 shows an illustrative submission file for November 3rd, 2021. The latest available ensemble weather forecasts were those from the NWP model runs initialized on Wednesday at 00 UTC. The first weather targets to be predicted corresponded to observations made on Thursday at 12 UTC, which is 36 hours after the NWP model's initialization time and 12 hours after the submission deadline. The participants' forecast files were then uploaded Thursday morning on our public repository. Students were allowed to skip submissions twice without failing the course.

In order to protect the students' anonymity, each student was asked to pick a character from five fictional universes as their alias: Discworld (URL7, not chosen at all), Brooklyn Nine-Nine (URL8, chosen three times), Friends & Joey (URL9, chosen five times), Game of Thrones (URL10, chosen five times) and Star Wars (URL11, chosen seven times).

**Table 5.2: Example of a correctly specified submission file** for November 3rd, 2021.

| forecast_date | target | horizon | q0.025 | q0.25 | q0.5 | q0.75 | q0.975 |
|---|---|---|---|---|---|---|---|
| 2021-11-03 | DAX | 1 day | -1.8 | -0.3 | 0.1 | 0.6 | 1.7 |
| 2021-11-03 | DAX | 2 day | -3.0 | -0.5 | 0.2 | 0.9 | 2.0 |
| 2021-11-03 | DAX | 5 day | -3.0 | -0.7 | 0.2 | 1.2 | 2.4 |
| 2021-11-03 | DAX | 6 day | -3.6 | -0.9 | 0.3 | 1.2 | 2.7 |
| 2021-11-03 | DAX | 7 day | -3.6 | -0.9 | 0.5 | 1.4 | 3.2 |
| 2021-11-03 | temperature | 36 hour | 6.5 | 8.0 | 8.6 | 9.2 | 10.4 |
| 2021-11-03 | temperature | 48 hour | 6.2 | 7.9 | 8.7 | 9.2 | 10.6 |
| 2021-11-03 | temperature | 60 hour | 7.9 | 9.8 | 10.9 | 11.7 | 13.4 |
| 2021-11-03 | temperature | 72 hour | 4.3 | 6.8 | 7.6 | 8.3 | 9.7 |
| 2021-11-03 | temperature | 84 hour | 8.5 | 10.4 | 11.3 | 12.0 | 14.2 |
| 2021-11-03 | wind | 36 hour | 8.7 | 13.8 | 16.5 | 19.4 | 26.2 |
| 2021-11-03 | wind | 48 hour | 5.8 | 15.5 | 18.9 | 23.1 | 30.8 |
| 2021-11-03 | wind | 60 hour | 9.7 | 14.2 | 16.7 | 19.0 | 23.8 |
| 2021-11-03 | wind | 72 hour | 6.9 | 11.9 | 14.2 | 17.1 | 24.3 |
| 2021-11-03 | wind | 84 hour | 8.9 | 14.4 | 17.7 | 20.8 | 26.3 |

Each Thursday, course participants and one or two instructors met online to discuss the past week of forecasting. This allowed students to learn from each other's experiences and obtain ideas to improve their forecasting approach. Although the call was not mandatory, participation remained high during the whole semester.

**Public Display and Evaluation of Forecasts** An interactive visualization tool (URL3) implemented in R Shiny (RStudio, Inc, 2013) and plotly (Plotly Technologies Inc., 2015) was made available and updated after each round of submissions. This allowed students to browse their own and others' past forecasts and visually compare them against later observed data. The website also included a regularly updated leader board covering the various employed evaluation measures. The app was hosted using the shinyapps.io service, with the underlying code and data available at URL12.

**Performance Measures** Several different statistical measures were used to asses the probabilistic forecasts submitted by the participants. These were specified prior to the start of the course in the study preregistration (Bracher et al., 2021a). The primary

outcome measure was the linear quantile score

$$\mathrm{QS}_\alpha(q_\alpha, y) = 2 \times \{\mathbf{1}(y < q_\alpha) - \alpha\} \times (q_\alpha - y),$$

where $q_\alpha$ is the submitted predictive $\alpha$ quantile with $\alpha \in \{0.025, 0.25, 0.5, 0.75, 0.975\}$ and $y$ is the observed value. The linear quantile score is a proper scoring rule (Gneiting and Raftery, 2007), meaning that it encourages 'honest' forecasting and is immune to hedging. For each forecast target this score was averaged across the five submitted quantile levels, which results in an approximation of the widely-used CRPS (Laio and Tamea, 2007). In addition, the AE and coverage percentages of the central 50% and 95% prediction intervals were assessed (see Gneiting et al. (2023) for a more detailed overview of evaluation methods for quantile forecasts).

While in principle linear quantile scores can simply be averaged across different targets, we refrained from doing so as the level and variability of scores was expected to vary considerably across targets and forecast horizons. For each target type $t$ (i.e., DAX, temperature and wind speed) and forecast horizon $h$ we therefore first computed average scores

$$\bar{S}_{t,h} = \frac{1}{N_t} \sum_{i=1}^{N_t} S_{t,h,i}$$

across the $N_t$ submission weeks, where $N_t = 14$ for DAX and $N_t = 13$ for the weather variables. To improve comparability across targets and horizons, we then translated these into skill scores

$$S^*_{t,h} = 1 - \frac{\bar{S}_{t,h}}{\bar{S}^{\text{bench}}_{t,h}}, \tag{5.1}$$

where $\bar{S}^{\text{bench}}_{t,h}$ is the average score achieved by the respective benchmark model. For each target and horizon, participants were then ranked by the skill score (or, equivalently, the average score). To obtain an overall ranking, we averaged the linear quantile score ranks participants achieved for the three targets and five horizons. In case of equal average ranks, the best achieved rank was used as a tiebreaker, followed by the average rank achieved for temperature and ultimately a coin flip.

We note that from a theoretical perspective, incentives in forecast competitions are complex and depend on participants' preferences (e.g., risk attitudes) and the details of the forecasting setup (e.g., dependent versus independent outcomes across submission rounds). While the recent literature has made intriguing progress (see, e.g., Pfeifer et al.,

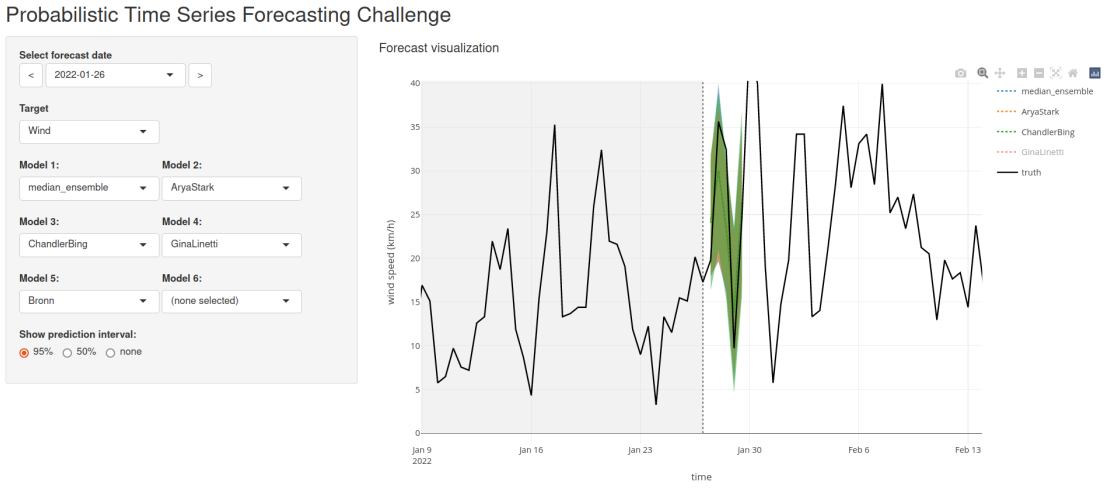



**Figure 5.2: Interactive graphic display** showing wind speed forecasts made by participants and the later observed time series.

2014; Frongillo et al., 2021; Witkowski et al., 2023), it does not yet provide clear guidance for complex practical settings like ours. We thus chose the pragmatic methodology described above.

If participants did not submit forecasts for one or more targets, their score for each target was imputed with 1.01 times the worst average linear quantile score achieved by any participant for that target. This system was chosen to incentives regular participation.

**Examination and Grading** The final examination of participants was *not* based on the formal predictive performance of their submitted forecasts. Instead, they needed to hand in a seminar paper summarizing their prediction approach, technical pipeline and learnings from the challenge. Grades were determined based on this assignment. There were, however, small awards for the best-performing participants.

## 5.3 Comparison to Collaborative Forecasting Projects

We next relate our setup to collaborative forecasting projects across different areas of applied statistics. While these projects have been framed in somewhat different ways (as ‘competitions’, ‘surveys’, and ‘hubs’), a key commonality among them is that several participants make forecasts of the same target variable.

### 5.3.1 Forecasting Competitions

Forecasting competitions have found widespread use in statistics, complementing large-scale benchmarking and model comparison efforts in machine learning (such as the ImageNet Large Scale Visual Recognition Challenge at URL13) and other fields. Forecasting competitions have the potential to improve the theory and practice of forecasting, for example by motivating novel methodological developments. For recent overviews, we refer to Hyndman (2020), Petropoulos et al. (2022, Section 2.12.7) and Makridakis et al. (2022). In addition to forecasting competitions organized in an academic context, there is a plethora of competitions available on platforms such as Kaggle (Bojer and Meldgaard, 2021). Most forecasting competitions focus on deterministic rather than probabilistic forecasts, and few use a real-time format or focus on time series aspects. We discuss two notable exceptions in the following.

**Global Energy Forecasting Competitions** The Global Energy Forecasting Competition (GEFCom), a series of three international competitions received considerable interest in energy forecasting research. The focus of GEFCom2014 (Hong et al., 2016) was on probabilistic forecasts in the form of quantiles in four tracks on load, price, wind, and solar forecasting. The competition used historic data, but was conducted in a pseudo real-time setting, with weekly submissions of forecasts based on incremental data releases.

**Competitions on Subseasonal Weather Forecasting** While subseasonal weather forecasts with forecast horizons of around 2 to 6 weeks are of considerable importance for many application areas such as agriculture, energy or health, NWP models typically lose most of their predictive ability on those time-scales (White et al., 2022). Recent international competitions on subseasonal weather forecasting have thus focused on improving those predictions, in particular through the use of machine learning methods. The Subseasonal Climate Forecast Rodeo (S2S Rodeo) was a real-time forecasting competition organized by the U.S. Bureau of Reclamation and the National Oceanic and Atmospheric Administration that took place over a year from April 2017 to April 2018 (Hwang et al., 2019). Participants had to submit deterministic forecasts of average temperature and accumulated precipitation over the western contiguous U.S. 3–4 and 5–6 weeks ahead.

In a conceptually similar setup, the World Meteorological Organization (WMO) organized the WMO Prize Challenge to Improve Sub-Seasonal to Seasonal Predictions Using Artificial Intelligence (URL14) in 2021 (Vitart et al., 2022). Probabilistic predictions of categorical events (below normal / normal / above normal) had to be submitted for the same target variables for all land grid points on the entire globe. Unlike the S2S Rodeo, the competition was not conducted in a real-time setting. Within both competitions, data sets of historic and current ensemble forecasts from NWP models were provided to the participants.

### 5.3.2 Survey of Professional Forecasters (SPF)

Initiated already in 1968, the SPF is a quarterly forecast survey covering US macroeconomic outcomes like the gross domestic product or the inflation rate (Croushore and Stark, 2019). A European analogue of the survey was established by the European Central Bank in 1999 (Bowles et al., 2007). The SPF's strict real-time schedule and its coverage of probabilistic forecasts are similar to our course. On the other hand, many macroeconomic variables are measured only quarterly and are published with a delay of several weeks. In consequence, obtaining feedback on forecast performance is much harder for SPF participants than for forecasters in fields like meteorology, where outcome data are available at high frequency.

### 5.3.3 COVID-19 Forecast Hubs

The overall format of the challenge as well as parts of the technical infrastructure were inspired by ongoing forecasting activities on the COVID-19 pandemic. Specifically, one member of the organizing teams were involved in the operation of various *Forecast Hub* platforms (Cramer et al., 2022; Bracher et al., 2021b; Sherratt et al., 2022), which collate and combine short-term forecasts of COVID-19 case, hospitalization and death numbers. These platforms operate using a similar quantile-based format with weekly submissions and multiple lead times.

## 5.4 Empirical Results

In this section, we describe empirical properties of students' probabilistic forecasts.

### 5.4.1 Forecasting Approaches Used by Participants

In the following, we present the forecasting methods undertaken by the participants throughout the challenge. Table 5.3 summarizes the implemented approaches, as indicated in the students' final project reports (excluding PhD students, who did not have to submit reports). We group approaches into their broader methodological classes, that is, we do not distinguish between different parameter or architecture choices. The row on 'additional data' indicates whether students considered feature variables constructed from external data sources, which was allowed by the rules of the course.

**Table 5.3: Forecasting approaches implemented by the students** grouped into broad categories for each target variable. The first number denotes how often the method was implemented, while the second number denotes how often the method performed best in the students' final evaluation.

| | Wind | Temperature | DAX |
|---|---|---|---|
| Baseline | 3/2 | 3/2 | 4/3 |
| EMOS | 8/6 | 8/7 | - |
| Quantile Regression | 2/1 | 2/1 | 9/2 |
| QRF | 4/0 | 3/0 | 2/0 |
| Gradient Boosting | 4/2 | 5/2 | 4/2 |
| Neural Network NN | 3/1 | 4/1 | 4/3 |
| AR(I)MA | 1/0 | 1/0 | 4/1 |
| GARCH | 0/0 | 0/0 | 5/3 |
| Other | 2/2 | 2/2 | 3/2 |
| Ensemble of Models | 2/2 | 1/1 | 3/3 |
| Additional Data | 0/0 | 0/0 | 4/2 |

Students treated the wind and temperature targets similarly. For those targets, EMOS is by far the most implemented and best final model. Interestingly, for two students the baseline performed best. We suspect that this is due to programming errors, as EMOS did indeed outperform the baseline (see Section 5.4.3 for details). Other methods, such as quantile regression and QRFs were tried a few times, but apparently did not perform as well. Model classes that do not fit the weather forecasting task, specifically AR(I)MA and GARCH were rarely used. No participant tried to find additional data sets, which seems plausible given the rich information set condensed by the ensemble forecasts. Instead of considering additional data, students tried to gain performance by adapting the model (class). Some students did not only utilize the ensemble forecasts of the respective target variable, but also included other variables (e.g., cloud cover) in their predictions.

For the DAX, where no feature data was provided, students' approaches were more heterogeneous than for wind and temperature. No model stands out as a clear favorite. While most students implemented quantile regression methods, few chose them as their final model. Besides the baseline (possibly adapted), neural networks, GARCH and ensembles of models performed best for the students. Four students used additional data sets.

Regarding their challenges during the semester, students mostly named technical problems with software packages and implementation errors in their own code. Given that students were required to develop their own code, this was to be expected as the workload remained high during the semester. This is a key difference to most other challenges (e.g., on Kaggle) where the real-time aspect is missing. Lastly, some of the implementation choices and method descriptions in the reports suggest conceptual misunderstandings on the part of participants. Similar to occasional coding errors, this finding is not surprising given the difficulty and volume of the course content.

### 5.4.2 Evaluation Sample

Our forecast evaluation sample comprises 14 weeks for the DAX (starting with the submissions on October 27, 2021) and 13 weeks for the weather targets (starting with the submissions on November 3, 2021). For all targets, the final submission round was on February 9, 2022, and submissions were paused during a Christmas break. The later submission start for the weather targets was due to the switch to the Berlin weather station described in Section 5.2.2. This setup yields a total of 199 forecast/observation pairs (DAX: 14 weeks times 5 horizons, minus one banking holiday on Christmas; two weather targets: 13 weeks times five horizons).

### 5.4.3 Forecast Performance

Table 5.4 presents the coverage rates of the benchmark (see Table 5.1 for a summary) and ensemble forecasts. Coverage is defined as the share of test-sample observations that are within the prediction interval. For example, the coverage rate for the 50% interval should be 50%. For brevity, we focus on the mean ensemble (see end of Section 5.2.2). The results for the median ensemble are qualitatively similar. Since each individual coverage rate shown in Table 5.4 is computed from a rather small sample of 13 or 14 weeks, we limit ourselves to a broad summary interpretation. The DAX benchmark generally has

satisfactory coverage close to its nominal level. This result is unsurprising, given that the benchmark is an estimate of the unconditional return distribution. While the latter distribution is not sharp (in particular, it uses no conditioning information), it is known to satisfy various notions of forecast calibration, including interval coverage that we consider here (see e.g. Gneiting and Katzfuss, 2014). For temperature, the benchmark is overconfident at the shortest horizon (with coverage rates falling short of their nominal levels), but achieves satisfactory coverage rates otherwise. For wind, the benchmark seems overconfident at all horizons. For the ensemble forecasts, Table 5.4 shows no clear signs of miscalibration, with coverage rates being reasonably close to their nominal levels for all targets and horizons.

**Table 5.4: Coverage rates of prediction intervals** for each respective horizon, in percent. 'Ensemble' denotes mean ensemble, 'Temp' denotes temperature.

| | | Coverage (50% level) [%] | | Coverage (95% level) [%] | |
|---|---|---|---|---|---|
| Target | Horizon | Benchmark | Ensemble | Benchmark | Ensemble |
| DAX | 1 day | 71.4 | 71.4 | 100.0 | 100.0 |
| DAX | 2 day | 50.0 | 42.9 | 92.9 | 92.9 |
| DAX | 5 day | 50.0 | 50.0 | 92.9 | 85.7 |
| DAX | 6 day | 35.7 | 35.7 | 100.0 | 85.7 |
| DAX | 7 day | 53.8 | 46.2 | 100.0 | 100.0 |
| Temp | 36 hour | 15.4 | 30.8 | 61.5 | 92.3 |
| Temp | 48 hour | 46.2 | 76.9 | 92.3 | 100.0 |
| Temp | 60 hour | 53.8 | 76.9 | 100.0 | 100.0 |
| Temp | 72 hour | 30.8 | 53.8 | 76.9 | 92.3 |
| Temp | 84 hour | 69.2 | 61.5 | 92.3 | 92.3 |
| Wind | 36 hour | 7.7 | 61.5 | 53.8 | 100.0 |
| Wind | 48 hour | 23.1 | 46.2 | 46.2 | 92.3 |
| Wind | 60 hour | 30.8 | 61.5 | 61.5 | 84.6 |
| Wind | 72 hour | 30.8 | 38.5 | 53.8 | 92.3 |
| Wind | 84 hour | 53.8 | 30.8 | 61.5 | 92.3 |

Figure 5.3 summarizes performance as measured by the quantile scoring function, based on which we compute skill scores relative to the benchmark forecasts (see Equation 5.1). A positive skill score indicates that a forecast outperforms the benchmark. For DAX, a majority of forecasters performs worse than the benchmark. Furthermore, the mean ensemble performs similar to the benchmark. For wind, we observe the opposite result: Many individual forecasters, as well as the mean ensemble, outperform the benchmark.

The results for temperature are between these two polar cases: Whereas many individual forecasters (and the ensemble) outperform the benchmark of horizons of 36, 48 and 72 hours, the benchmark performs very well at the other two horizons of 60 and 84 hours. For both weather targets, the mean ensemble performs similar to the EMOS model. For all three targets, the mean ensemble performs similar to the best individual forecasters, and is remarkably robust to the inclusion of forecasters who perform below average. Similar 'wisdom of the crowds' type results on forecast combination have been documented for various application areas and types of forecasts (see Petropoulos et al., 2022, Section 2.6.4, for a brief review).

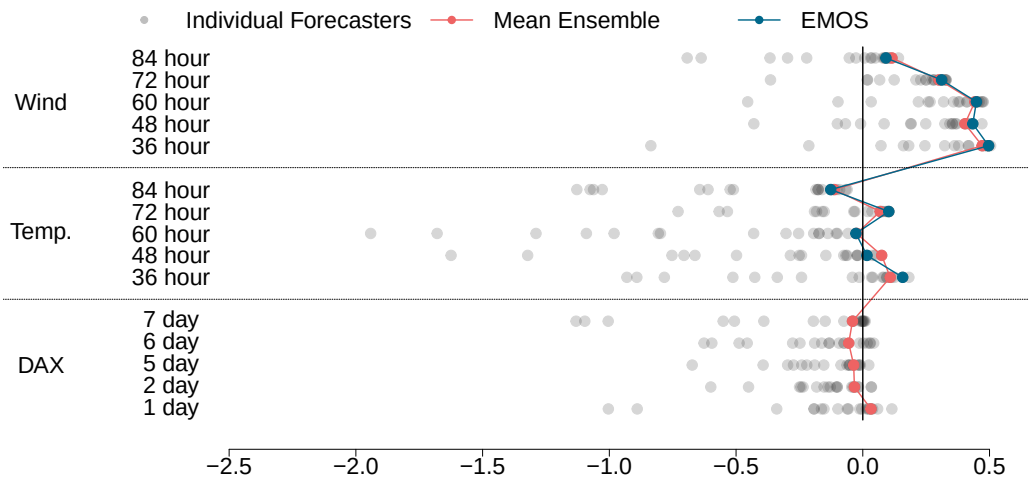



**Figure 5.3: Graphical summary of forecast performance.** Dots are skill scores relative to benchmark model. Larger skill scores are better and a skill of 0 corresponds to the performance of the baseline.

The results in Table 5.4 and Figure 5.3 are closely in line with each other, in the following sense: for targets and horizons at which the benchmark lacks calibration (i.e., wind at all and temperature at some horizons), we observe that ensemble forecasts improve upon the benchmark by restoring calibration. Conversely, the benchmark seems hard to beat for the remaining targets and horizons. This observation emphasizes the role of calibration for making good predictions. In principle, it is also possible to improve upon the benchmark by making sharper predictions, subject to remaining calibrated (see Gneiting and Katzfuss, 2014; Pohle, 2020; Krüger and Ziegel, 2021). Such an

improvement could occur by using a larger information set than the benchmark, e.g., by making appropriate use of additional regressor variables. In order to investigate this possible channel, Table D.2 in Appendix D.1 considers the length of the benchmark's and ensemble's prediction intervals. For the weather targets, the ensemble's intervals are generally wider than the benchmark's intervals. For the DAX, the ensemble's intervals are slightly shorter, but the ensemble does not outperform the benchmark. These results suggest that the sharpness channel just described plays a limited role in our setting, at least for the mean ensemble.

### 5.4.4 Exploratory Analysis of Learning Effects

The sequential (time series) forecasting process is characteristic of our course, and distinguishes it from 'static' prediction contests where participants are assigned a fixed training and test sample at the same time. In principle, this real-time process makes it possible to learn about the success or failure of alternative forecasting methods as time proceeds. Real-time feedback is in addition to historical data available at the beginning of the course that allow to validate the performance of alternative forecasting methods. Ex ante, we found it hard to judge the practical relevance of real-time feedback, especially in the light of the rather short sample period of 14 weeks and possible heterogeneity across forecast targets and horizons.

In order to explore the relevance of real-time learning, we compute the share of participants who outperform the benchmark in each week of the course. We label this analysis, as well as the the one in Section 5.4.5, as 'exploratory' since it was not specified in our preregistration plan but was devised during the semester. For a given week and forecast target (suppressed for ease of notation), we compute the following share:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\left\{\frac{1}{5}\sum_{h=1}^{5}S_h^{\text{bench}} > \frac{1}{5}\sum_{h=1}^{5}S_{h,i}\right\},$$

where $n$ denotes the number of participants for that week. In words, the indicator function refers to the event that forecaster $i$ attains a smaller (i.e., better) score than the benchmark, on average across the five forecast horizons. Under the assumption that real-time feedback enables learning over time, we would expect the share in question to increase over time, in that forecasters learn how to beat the benchmark. Figure 5.4 indicates that this phenomenon of an increasing share is not observed for either of the

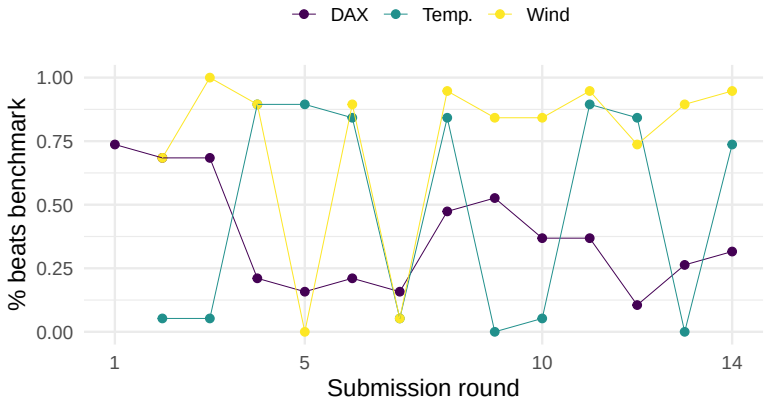



**Figure 5.4: Share of forecasters who beats the benchmark**, for each target and submission round.

three targets. In light of these results, we conjecture that real-time learning effects are limited in our setup, and may be dominated by other effects, such as participants' use of more complex methods as time proceeds.

### 5.4.5 Exploratory Analysis of Ranking Methodology

Our overall ranking of forecast performance intended to provide simple-to-interpret feedback to participants, in addition to more granular information on quantile scores for the different target variables and forecast horizons. The ranking also contributes to the course's gamification aspect. Since there are several plausible methods for constructing such a ranking, Section D.2.2 in the Appendix investigates various methods and assess their sensitivity. From a practical perspective, the empirical results are very similar across the methods considered.

## 5.5 Discussion

We conclude by discussing student reactions as well as possible adaptations of the course format.

### 5.5.1 Student Reactions

**Student Feedback** In the following, we shortly summarize students' textual comments to a mid-semester course evaluation survey. A majority of students liked the free nature,

practical relevance, and steep learning curve of the course, and appreciated the support of the team. Conversely, some students did not like the steep learning curve at the beginning of the course, and suggested that its requirements were not communicated clearly enough. Presumably, our stated prerequisites of a 'good working knowledge in statistics' and 'proficiency in a programming language' should have been more specific indeed. In order to ease participants' workload at the beginning of the course, instructors may consider making materials (such as video lectures) available early so that participants can study them well before the first round of forecast submissions.

**Choice of Forecasting Methods** Many approaches used by participants seem overly complex in hindsight, likely indicating a tendency towards overfitting and a lack of rigorous comparison of the devised methods to the provided benchmarks. By contrast, simple forecasting methods (careful extensions of the benchmarks) performed best. Apart from avoiding overfitting, simpler approaches also make it easier to spot and fix implementation errors. Potential reasons for participants' preference for more complex models likely include prior exposure to machine learning techniques, usually in idealized and/or big data settings where those tend to work well. Furthermore, curiosity and gaming aspects made it natural to move away from the simple benchmarks. Finally, the hesitancy to use simple models might be due to a fear of not using 'state of the art' approaches and not having done 'enough' to attain a good grade. From the perspective of an individual forecaster who wants to optimize their performance (in terms of the quantile score, say), the use of simple models would have been advisable in retrospect. From a broader perspective, the implications are more subtle. Firstly, for an individual participant, experimenting with complex approaches may yield higher learning gains, and be more entertaining, than selecting simple and conservative methods from the onset. Secondly, from the perspective of a meta-forecaster who seeks to optimize ensemble performance, a diverse set of (possibly uncalibrated) forecasts may well be preferable to a homogeneous set of calibrated forecasts. Indeed, Figure 5.3 indicates that the mean ensemble outperforms the vast majority of individual participants, and compares favorably to the benchmarks in the case of the weather variables. These results are much in line with 'wisdom of the crowd' type effects discussed earlier.

### 5.5.2 Possible Adaptations of the Course Format

**Substantive Context of Target Variables** The choice of target variables is highly flexible and can be adapted depending on participants' interests and backgrounds. While the course described here focused on meteorology and finance, we expect that variables from epidemiology (e.g., case numbers of a disease), energy (e.g., demand in megawatt hours), business (e.g., counts of product sales) and other fields are equally suitable. See Section 5.2.2 for further discussion and practical considerations regarding the choice of target variables.

**Schedule of Submissions** The frequency of forecast submissions is an important design parameter in practice. We chose weekly submissions to reflect the real-time idea of the course, with each week's forecasts responding to the most recent input data and evaluation feedback from previous weeks. While this motivation seems coherent, weekly submissions were quite challenging to some students, and we conjecture that the weekly format emphasizes forecasting workflow (i.e., well-organized code and data, procedures for plausibility checks) over purely statistical aspects. Whether or not this emphasis is desirable is open to debate; it certainly seems reflective of some practical forecasting settings (see, e.g., Taylor and Letham, 2018). Alternative conceivable submission formats include monthly submissions with more forecasts horizons for each submission round (e.g. 20 rather than five horizons). While this would reduce the workload for the participants, such a setting might be not suitable for all of the target variables (e.g., due to ensemble weather forecasts being available for horizons up to a few days only), and forecast evaluation would become more challenging due to the required aggregation over many forecast horizons. In addition, the practical predictability will be limited for longer forecast horizons and forecasts might merely reflect unconditional distributions. As an alternative to submitting specific probabilistic forecasts, participants could also be required to submit code that maps an input vector $X$ of features to a set of quantile forecasts. This code would need to be submitted only once (or perhaps two to three times, if revisions are allowed), which would focus the workload on a few occasions. However, this approach would not only brush over practically relevant aspects (e.g., some features becoming unavailable at short notice), but might also be discouraging if there is no chance to correct suboptimal forecast model code.

**Duration of the Course** Our course spanned 14 weeks. While extending the course duration seems unproblematic, shortening the course seems trickier given students' initial investment in setting up a practical forecasting workflow, and the requirement of sufficient outcome data for forecast evaluation.

**Level of Difficulty** Our course offering involved Bachelor's, Master's and PhD students with a good quantitative background. To some degree, the difficulty of the course can likely be reduced by providing participants with more specific material (e.g., example code for alternative forecasting models) and more specific guidance (e.g., providing check lists and practical forecasting tips, recommending specific models). Conversely, the difficulty could be increased by providing less specific material and guidance. Overall, the diverse set of skills required for practical yet rigorous forecasting may imply a steep learning curve even for very qualified students. We recommend that instructors communicate this aspect in order to set students' expectations and avoid frustration.

**Parameters Affecting Teachers' Workload** The real-time format of the course implies that teachers may need to react to unforeseen developments at short notice. Key examples include missing or erroneous real-time data, or software issues related to the process of receiving, visualizing and evaluating forecasts. In order to limit the workload related to real-time data, we recommend to focus on an applied setting close to the teachers' expertise, and for which stable data sources are available. In terms of software maintenance, some core functionalities (e.g., providing basic feedback on forecast performance) also seem achievable by simple means (e.g., sending an email newsletter to course participants).

## Code and Data Availability

The repository containing students' forecasts and all codes used to run the challenge is available at URL1. Codes to reproduce the tables and figures from this chapter are available at URL2. This repository also contains a simplified version of the interactive visualization dashboard created for the challenge, which can be used as a starting point for similar efforts.

## Acknowledgments

We thank our students for investing considerable time and effort into this novel course format. Furthermore, we acknowledge a student award generously provided by the International Institute of Forecasters (URL15). Sebastian Lerch gratefully acknowledges support by the Vector Stiftung through the Young Investigator Group 'Artificial Intelligence for Probabilistic Weather Forecasting.' Johannes Bracher acknowledges support by the Helmholtz Foundation via the SIMCARD Data Science Pilot Project. Finally, Marco Wurth and Benedikt Schulz provided excellent assistance with the weather data.

# Appendices

# A Appendix to Chapter 2: Signing the Supermask: Keep, Hide, Invert

## A.1 Derivation of ELUS

In the following paragraphs, we revise the initialization methods of He et al. (2015) and Glorot and Bengio (2010). This is needed because first, we introduce masking and a the ELU activation function. Both concepts alter the assumptions of the cited work. Here, we derive a novel adaption of the common He initialization (He et al., 2015).
In this section, we use the following notation:

- Activation function $\sigma(x)$
- Layer $l$ has $n_l$ neurons. This means that a layer takes $n_{l-1}$ inputs per neuron. In the literature, $n_{l-1}$ is sometimes called *fan-in* and $n_l$ is sometimes referred to as *fan-out*.
- $\boldsymbol{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$, $\boldsymbol{b}_l \in \mathbb{R}^{n_l \times 1}$
- Weighted layer input $\boldsymbol{z}_l = \boldsymbol{W}_l \boldsymbol{o}_{l-1} + \boldsymbol{b}_l$, $\boldsymbol{z}_l \in \mathbb{R}^{n_l \times 1}$
- Layer output $\boldsymbol{o}_l = \sigma(\boldsymbol{z}_l)$, $\boldsymbol{o}_l \in \mathbb{R}^{n_l \times 1}$
- $\mathcal{L}$ denotes the network's loss function
- $\nabla \boldsymbol{o}_l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{o}_l}$
- $\nabla \boldsymbol{z}_l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_l} = \sigma'(\boldsymbol{z}_l) \nabla \boldsymbol{o}_{l+1}$
- In order to maintain clarity, we will denote the transposed weight matrix of layer $l$ $\boldsymbol{W}_l^T$ as $\hat{\boldsymbol{W}}_l$ in the derivation of the backward-pass.
- $\bar{\boldsymbol{W}}_l = \boldsymbol{W}_l \odot \bar{\boldsymbol{M}}_l$. This leads to the suboptimal but necessary choice to denote the transpose of $\bar{\boldsymbol{W}}_l$ as $\tilde{\boldsymbol{W}}_l$ (instead of $\bar{\boldsymbol{W}}_l^T$) to maintain clarity.

We will further assume, that the following holds:

(1) We use the ELU activation function, introduced by Clevert et al. (2016).

(2) Variances of input features $\boldsymbol{x}$ are the same $\text{Var}[x]$.

(3) For each layer $l$, the respective weight matrix $\boldsymbol{W}_l$ is drawn from an independent and symmetric distribution with $\mathbb{E}[\boldsymbol{W}_l] = 0$ and $\text{Var}[\boldsymbol{W}_l]$. It follows that $\mathbb{E}[\boldsymbol{z}_l] = 0$.

(4) For each layer $l$, the respective mask matrix $\boldsymbol{M}_l$ is drawn from an independent and symmetric distribution with $\mathbb{E}[\boldsymbol{M}_l] = 0$. To make the derivations more tangible, we assume $p_0^l = P(\bar{\boldsymbol{M}}_{ij}^l = 0) = 0.5$ and $p_1^l = P(\bar{\boldsymbol{M}}_{ij}^l = 1) = P(\bar{\boldsymbol{M}}_{ij}^l = -1) = 0.25$ $\forall i \in 1, \dots, n_l, j \in 1, \dots, n_{l-1}$ at initialization. In general, the variance is $\text{Var}[\bar{\boldsymbol{M}}^l] = 1 - p_0^l$.

(5) For each layer $l$, all elements in $\boldsymbol{z}_l$ and $\boldsymbol{o}_l$ share the same variance $\text{Var}[\boldsymbol{z}_l]$ and $\text{Var}[\boldsymbol{o}_l]$, respectively.

Since we use ELU activation functions in our experiments, the question arises whether we can still use the initialization of He et al. (2015). Clevert et al. (2016) do so and achieve good results. Let us therefore calculate the important values that might alter the assumptions necessary.

Like Glorot and Bengio (2010) and He et al. (2015) differ in the choice of activation function, we can particularly investigate the parts of the derivation where the expected value of the activation function is used. In particular, this is $\mathbb{E}[(\boldsymbol{o}_{l-1})^2]$ in the forward-pass and $\mathbb{E}[(\nabla \boldsymbol{o}_l)^2]$ in the backward-pass.

We can calculate $\mathbb{E}[(\boldsymbol{o}_{l-1})^2]$ as follows:

$$\mathbb{E}[(\boldsymbol{o}_{l-1})^2] = \mathbb{E}[(\alpha e^{\boldsymbol{z}_{l-1}} - \alpha)^2] + \mathbb{E}[\boldsymbol{z}_{l-1}^2].$$

From here on, we drop index $l$ for the sake of clarity. For the calculation of $\mathbb{E}[(\alpha e^{\boldsymbol{z}_{l-1}} - \alpha)^2$, we additionally assume that $\boldsymbol{z}_l \sim \mathcal{N}(0, 1)$, although this is a simplification and is therefore

only an approximation. With this, we can calculate the first term like so

$$\begin{aligned}\mathbb{E}[(\alpha e^{\boldsymbol{z}} - \alpha)^2] &= \int_{-\infty}^{0} (\alpha e^{\boldsymbol{z}} - \alpha)^2 \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\boldsymbol{z}^2}\, d\boldsymbol{z} \\ &= \frac{\alpha^2}{\sqrt{2\pi}} \int_{-\infty}^{0} (e^{2\boldsymbol{z}} - 2e^{\boldsymbol{z}} + 1) \cdot e^{-\frac{1}{2}\boldsymbol{z}^2}\, d\boldsymbol{z} \\ &= \underbrace{\frac{\alpha^2}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{1}{2}\boldsymbol{z}^2 + 2\boldsymbol{z}}\, d\boldsymbol{z}}_{=:(a)} - \underbrace{\frac{2\alpha^2}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{1}{2}\boldsymbol{z}^2 + \boldsymbol{z}}\, d\boldsymbol{z}}_{=:(b)} + \underbrace{\frac{\alpha^2}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{1}{2}\boldsymbol{z}^2}\, d\boldsymbol{z}}_{=\frac{1}{2}\alpha^2 \mathrm{Var}[\boldsymbol{z}] = \frac{1}{2}\alpha^2}.\end{aligned}$$

We can observe that the last term is half of the area under the standard normal probability density function. Note that it holds that $-ax^2 + bx = -a(x - \frac{b}{2a})^2 + \frac{b^2}{4a}$. With this, we can re-write $(a)$ and $(b)$ as follows

$$(a) = \frac{\alpha^2}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{1}{2}(\boldsymbol{z}-2)^2 + 2}\, d\boldsymbol{z} = \frac{\alpha^2 e^2}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{1}{2}(\boldsymbol{z}-2)^2}\, d\boldsymbol{z}, \tag{A.1}$$

$$(b) = \frac{2\alpha^2}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{1}{2}(\boldsymbol{z}-1)^2 + \frac{1}{2}}\, d\boldsymbol{z} = \frac{2\alpha^2 e^{\frac{1}{2}}}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{1}{2}(\boldsymbol{z}-1)^2}\, d\boldsymbol{z}. \tag{A.2}$$

We can see that $(a)$ shows the probability density function of $\mathcal{N}(2, 1)$ and $(b)$ shows the probability density function of $\mathcal{N}(1, 1)$, which can be easily evaluated. With $(a) = \alpha^2 e^2 \cdot \Phi(\frac{z-2}{1}) = 0.168102\alpha^2$ and $(b) = 2\alpha^2 e^{\frac{1}{2}} \cdot \Phi(\frac{z-1}{1}) = 0.523157\alpha^2$ we have

$$\mathbb{E}[(\alpha e^{\boldsymbol{z}} - \alpha)^2] = 0.168102\alpha^2 - 0.523157\alpha^2 + 0.5\alpha^2 = 0.144945\alpha^2 \coloneqq k.$$

If we take the case $x > 0$ into account, i.e. $\mathbb{E}[\boldsymbol{z}^2] = \frac{1}{2}\mathrm{Var}[z]$, we have

$$\mathbb{E}[(\boldsymbol{o}_{l-1})^2] = (\frac{1}{2} + k)\mathrm{Var}[\boldsymbol{z}_{l-1}]. \tag{A.3}$$

Let us now move on to calculating $\mathbb{E}[(\nabla \boldsymbol{o}_l)^2]$. Note that the derivative of ELU is simply 1 if $\boldsymbol{z} > 0$ (which is the same as for ReLU) and $\alpha e^{\boldsymbol{z}}$ if $\boldsymbol{z} \leq 0$. Let us look at the squared expectation of the second case:

$$\mathbb{E}[(\sigma'(\boldsymbol{z}))^2] = \mathbb{E}[\alpha^2 e^{2\boldsymbol{z}}].$$

If we assume again, that $\boldsymbol{z}_l \sim \mathcal{N}(0,1)$, we immediately see that we already calculated this expected value in Equation A.1. Therefore, we have

$$\mathbb{E}[(\sigma'(\boldsymbol{z}))^2] = 0.168102\alpha^2 := h.$$

For the total expected value, we then have

$$\mathbb{E}[(\nabla \boldsymbol{o}_l)^2] = \frac{1}{2} + h. \tag{A.4}$$

To summarize, in comparison to He et al. (2015), we only have to scale Equations A.3 and A.4 with the constants $h$ and $k$, respectively. The results above are equal to Equation 15 in He et al. (2015), where they assume a PReLU activation (PReLU as an activation function is also introduced in He et al. (2015)). However, they do not supply a derivation. Therefore we cannot guarantee equality with confidence.
Hence, we are careful with the pronoun 'our' (initialization) and we will therefore simply refer to the initialization method that follows from the derivations above as '**ELU**'.
Let us now use these results and calculate $\text{Var}[\boldsymbol{W}_l]$.

**Forward-Pass** We start with the following equation

$$\text{Var}[\boldsymbol{z}_l^j] = \sum_{k=0}^{n_{l-1}} \text{Var}[\bar{\boldsymbol{W}}_l{}^{jk} \boldsymbol{o}_{l-1}^k].$$

Because $\mathbb{E}[\bar{\boldsymbol{W}}_l] = 0$ we can re-write this as

$$\text{Var}[\boldsymbol{z}_l] = n_{l-1}\text{Var}[\bar{\boldsymbol{W}}_l]\mathbb{E}[(\boldsymbol{o}_{l-1})^2].$$

With Equation A.3 it follows that

$$\text{Var}[\boldsymbol{z}_l] = (\frac{1}{2} + k)n_{l-1}\text{Var}[\bar{\boldsymbol{W}}_l]\text{Var}[\boldsymbol{z}_{l-1}] = \text{Var}[\boldsymbol{x}] \prod_{i=1}^{l} (\frac{1}{2} + k)n_{i-1}\text{Var}[\bar{\boldsymbol{W}}_i]. \tag{A.5}$$

If we want to achieve constant variance throughout the network, we can set

$$(\frac{1}{2}+k)n_{l-1}\text{Var}[\boldsymbol{W}_l]\text{Var}[\bar{\boldsymbol{M}}_l] \overset{!}{=} 1$$
$$\iff \text{Var}[\boldsymbol{W}_l] = \frac{1}{(\frac{1}{2}+k)n_{l-1}(1-p_0^l)}.$$

With $\text{Var}[\bar{\boldsymbol{M}}_l] = p_1^l \cdot (-1)^2 + p_0^l \cdot 0^2 + p_1^l \cdot 1^2 = 0.5$ as stated above and $\alpha = 1$ we have

$$\text{Var}[\boldsymbol{W}_l] = 1.5505 \cdot \frac{2}{n_{l-1}}. \tag{A.6}$$

Note that for normal training, we would not have to account for the variance of the mask, which would result in $\text{Var}[\boldsymbol{W}_l] = 1.5505 \cdot \frac{1}{n_{l-1}}$.

**Backward-Pass** We know that

$$\text{Var}[\nabla \boldsymbol{o}_l^i] = \sum_{j=0}^{n_l} \text{Var}[\tilde{\boldsymbol{W}}_l{}^{ij} \underbrace{\sigma'(\boldsymbol{z}_l^j)\nabla \boldsymbol{o}_{l+1}^j}_{=\nabla \boldsymbol{z}_l^j}]. \tag{A.7}$$

Recall that ELU derivative is

$$\sigma'(\boldsymbol{z}_l) = \begin{cases} 1, & \boldsymbol{z}_l \geq 0 \\ \alpha e^{\boldsymbol{z}_l}, & \boldsymbol{z}_l < 0. \end{cases}$$

We know from Equation A.4 that $\mathbb{E}[(\sigma'(\boldsymbol{z}_l))^2] = \frac{1}{2} + h$ and $\mathbb{E}[\nabla \boldsymbol{o}_{l+1}] = 0$. It follows, that $\text{Var}[\nabla \boldsymbol{o}_{l+1}] = \mathbb{E}[(\nabla \boldsymbol{o}_{l+1})^2]$, resulting in

$$\text{Var}[\nabla \boldsymbol{z}_l] = (\frac{1}{2}+h)\text{Var}[\nabla \boldsymbol{o}_{l+1}].$$

With the obtained results we can now re-write Equation A.7 as

$$\text{Var}[\nabla \boldsymbol{o}_l] = (\frac{1}{2}+h)n_l\text{Var}[\tilde{\boldsymbol{W}}_l]\text{Var}[\nabla \boldsymbol{o}_{l+1}].$$

Herewith we are able to calculate the variance throughout all layers as

$$\text{Var}[\nabla \boldsymbol{o}_1] = \text{Var}[\nabla \boldsymbol{o}_{L+1}] \prod_{l=1}^{L} (\frac{1}{2}+h)n_l\text{Var}[\tilde{\boldsymbol{W}}_l]. \tag{A.8}$$

Like in the forward-pass, an easy way to keep the variance constant is to set

$$\begin{aligned} &(\frac{1}{2}+h)n_l \text{Var}[\tilde{\boldsymbol{W}}_l] \stackrel{!}{=} 1 \\ \iff &\text{Var}[\boldsymbol{W}_l] = \frac{1}{(\frac{1}{2}+h)n_l(1-p_0^l)} = 1.49678 \cdot \frac{2}{n_l}. \end{aligned} \tag{A.9}$$

As above, if we use this initialization for training a normal NN, we would use $\text{Var}[\boldsymbol{W}_l] = 1.49678 \odot \frac{1}{n_l}$.

We will follow the intuition of He et al. (2015) by leaving the choice of Equation A.9 or A.6 to the user, as their initialization scheme has proven to outperform Glorot and Bengio (2010) in practice and takes the CNN-architecture into account. However, since Equations A.9 and A.6 are relatively close to each other, we suggest to use the following variance for the sake of simplicity

$$\text{Var}[\boldsymbol{W}_l] = \frac{1.5}{n_l(1-p_0^l)}.$$

## A.2 Models and Hyperparameters

**Table A.1: Fully-connected NN & CNN architectures** used in the experiments of this work. To be able to compare the results to the existing literature, the architectures are equal to Frankle and Carbin (2019); Zhou et al. (2019) in addition to the Conv8 model used in Ramanujan et al. (2020). Each CNN performs various convolutions (*pool* denotes max-pooling) with stated number of filters, followed by fully connected (FC) layers with specified output neurons. For the CNNs, we always use a filter size of $3 \times 3$. Total parameter count for each architecture is reported in the last row.

| | FCN | Conv2 | Conv4 | Conv6 | Conv8 |
|---|---|---|---|---|---|
| | | | | | 64, 64, pool |
| | | | | 64, 64, pool | 128, 128, pool |
| Conv | | | 64, 64, pool | 128, 128, pool | 256, 256, pool |
| Layers | | 64, 64, pool | 128, 128, pool | 256, 256, pool | 512, 512, pool |
| FC Layers | 300,100,10 | 256, 256, 10 | 256, 256, 10 | 256, 256, 10 | 256, 256, 10 |
| Parameter Count | 266.200 | 4.300.992 | 2.425.024 | 2.261.184 | 5.275.840 |

**Table A.2: FCN/CNN Baseline: Hyperparameter choices** training the *baseline* models of FCN and Conv2 - Conv8. For simplicity, we chose parameters that work well on all models

| | FCN | Conv2 | Conv4 | Conv6 | Conv8 |
|---|---|---|---|---|---|
| Learning Rate | 0.008 | 0.008 | 0.008 | 0.01 | 0.01 |
| Decay Rate / Step | .96/10 | .96/5 | .96/10 | .96/10 | .96/10 |
| Red. LR on Plat. (Patience) | - | - | - | - | - |
| Weight Decay | 7e-4 | 7e-4 | 7e-4 | 7e-4 | 7e-4 |
| Momentum | .9 | .9 | .9 | .9 | .9 |
| Iterations | 50 | 50 | 50 | 50 | 50 |
| Optimizer | SGD | SGD | SGD | SGD | SGD |

**Table A.3: ResNet Baseline: Hyperparameter choices** for training the *baseline* ResNet models.

| | ResNet20 | ResNet56 | ResNet110 |
|---|---|---|---|
| Learning Rate | 0.2 | 0.15 | 0.15 |
| Decay Rate / Step | - | - | - |
| Red. LR on Plat. (Patience) | .2 (15) | .2 (15) | .2 (15) |
| Weight Decay | 1e-4 | 1e-4 | 1e-4 |
| Momentum | .9 | .9 | .9 |
| Iterations | 80 | 90 | 90 |
| Optimizer | SGD | SGD | SGD |

**Table A.4: FCN/CNN signed Supermask: Hyperparameter choices** for training the *signed Supermask* models of FCN and Conv2 - Conv8. For simplicity, we chose parameters that work well on all models, except for Conv2 which suffered overfitting and we therefore reduced the learning rate.

| | FCN | Conv2 | Conv4 | Conv6 | Conv8 | ResNet20 |
|---|---|---|---|---|---|---|
| Learning Rate | 0.05 | 0.02 | 0.05 | 0.05 | 0.05 | 0.2 |
| Decay Rate / Step | .96/10 | .96/5 | .96/10 | .96/10 | .96/10 | - |
| Red. LR on Plat. (Patience) | - | - | - | - | - | .2 (15) |
| Weight Decay | 5e-4 | 5e-4 | 5e-4 | 5e-4 | 5e-4 | 1e-4 |
| Momentum | .9 | .9 | .9 | .9 | .9 | .9 |
| Iterations | 100 | 100 | 100 | 100 | 100 | 110 |
| Optimizer | SGD | SGD | SGD | SGD | SGD | SGD |
| Initial Pruning Rate | 6.3% | 29.5% | 19.3% | 11.9% | 12.9% | 75% |

**Table A.5: ResNet signed Supermask: Hyperparameter choices**.

| | ResNet20 | ResNet56 | ResNet110 |
|---|---|---|---|
| Learning Rate | 0.2 | 0.15 | 0.15 |
| Decay Rate / Step | - | - | - |
| Red. LR on Plat. (Patience) | .2 (15) | .2 (15) | .2 (15) |
| Weight Decay | 1e-4 | 1e-4 | 1e-5 |
| Momentum | .9 | .9 | .9 |
| Iterations | 110 | 120 | 120 |
| Optimizer | SGD | SGD | SGD |
| Initial Pruning Rate | 75% | 85% | 85% |

## A.3 Further Experimental Results

Figure A.1 seeks to draw a rough comparison between a FCN trained with a (ELUS) binary Supermask and a (ELUS) signed Supermask. Note that a comparison to Supermasks of Zhou et al. (2019) is not directly feasible as there are other important factors that differ, namely the ELUS initialization, the ELU activation function and the same configuration as used for the ELUS signed Supermask. However, comparing a binary with a signed Supermask can uncover important differences. The upper matrix shown in Figure A.1 depicts the sum of 50 trained binary Supermasks, i.e., during the training process the weights could either be pruned or 'let through'. The same is shown on the bottom for the signed Supermask. A darker shade represents a larger value. In order for any negative and positive masking not to cancel each other out, the absolute masks were summed.

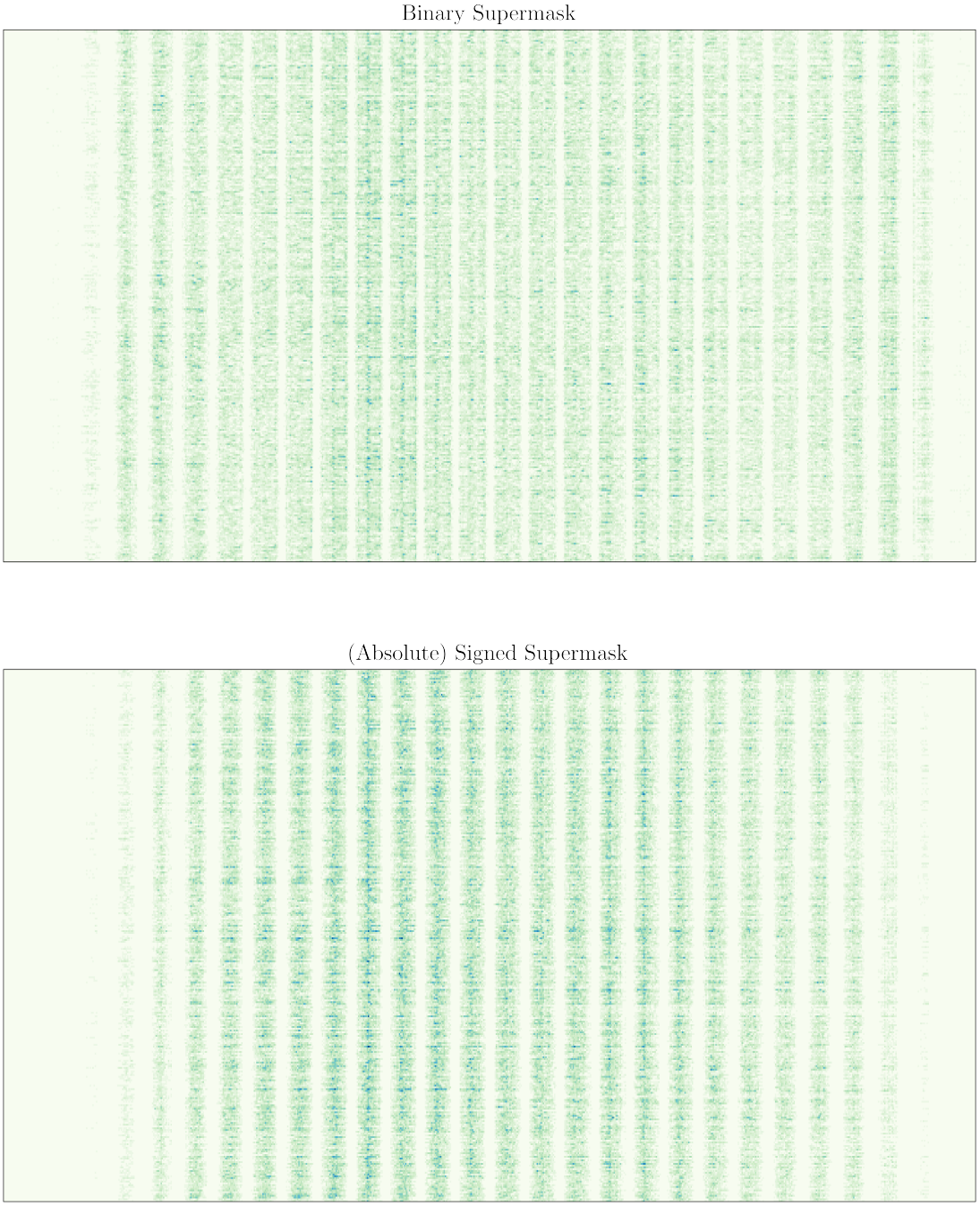



**Figure A.1: FCN (signed) Supermask: Visual mask comparison of binary and signed Supermask.**

Put simply, if a single mask (out of the 50 runs) prunes a weights every time except once, the respective value in the matrices would be 1 and so on. Both versions have an average pruning rate of 97.2% in the first layer. The shown matrices still exhibit a pruning rate of 54.11% in case of the signed Supermask and 43.63% for the binary Supermask, i.e., 54% (43%) of the weights were always pruned. This number is impressive by itself, if we consider that normal NNs could never reach a similar number. The overall average pruning rates are 3.77% and 4.04% for the signed and binary Supermasks, respectively. While the signed Supermask reached an average performance of 97.48%, the binary Supermask reached an average test accuracy of 97.03%. In other words, the signed Supermask left the same number of weights in the first layer, while the distribution differed more in the other layers but reached a higher performance with roughly 0.3pp less weights. The focus of this examination is to compare the properties of the respective first masks.

At first glance, the binary Supermask looks more erratic, while the signed Supermask inhibits a more ordered structure. The impression is not deceiving: out of the 784 columns (i.e., input neurons), 284 in case of the signed Supermask are completely masked, whereas the binary Supermasks only masks 144 (we strongly assume, that the fact that one number is almost double the other is a coincidence). That is, those columns were masked in every considered mask, which shows the robustness of Supermasks in general. Even if we allow 100 outliers (that is, a column-wise maximum sum of 100) per column, binary Supermasks only count 260 of those columns (364 for signed Supermasks). We can observe this fact in two parts: the signed Supermask is more sparse at the edges and the 0-columns in the center are wider and less noisy. This leads to the conclusion that, although on average having the same pruning rate, signed Supermasks are more consistent (otherwise we would count fewer 0-columns) and more understanding of the given data, as it focuses on specific inputs and mostly disregards the image background, which (in the flattened vector) is positioned at the sides. This strongly suggests that not the pruning rate alone is of importance, but also the distribution of pruning per layer.

To summarize, although a comparison is not possible without compromise, we can see that the binary Supermask, although exhibiting the same average pruning rate is far less consistent compared to the signed Supermask.

Figure A.2 visualizes average test accuracy for each epoch during training with the respective 5% and 95% quantiles for FCN and all Conv models for He, Xavier and

**Table A.6: CNN signed Supermask: Average test accuracy, test loss and required training time per epoch** for each model architecture and weight initialization method. The 5% and 95% quantiles are given as well.

| Architecture | Init | Test Acc. [%] | Test Loss | Time/Epoch [s] |
|---|---|---|---|---|
| Conv2 - *Baseline* | He | 68.79 [68.35, 69.19] | 0.9083 [0.8986, 0.9166] | 2.87 [2.86, 2.88] |
| | ELU | 68.60 [68.23, 68.99] | 0.9189 [0.9047, 0.9331] | 2.88 [2.85, 2.97] |
| | Xavier | 67.93 [67.39, 68.41] | 0.9535 [0.9281, 0.9716] | 2.86 [2.84, 2.88] |
| Conv2 - *Si. Su.* | He | 66.58 [65.92, 67.15] | 0.9596 [0.9452, 0.9741] | 3.16 [3.14, 3.18] |
| | ELUS | 68.37 [67.74, 69.04] | 0.9784 [0.9524, 1.0090] | 3.16 [3.14, 3.18] |
| | Xavier | 54.24 [53.36, 55.39] | 1.2862 [1.2608, 1.3067] | 3.20 [3.15, 3.31] |
| Conv4 - *Baseline* | He | 75.03 [74.71, 75.42] | 0.7266 [0.7200, 0.7347] | 3.99 [3.97, 4.01] |
| | ELU | 75.00 [74.65, 75.29] | 0.7281 [0.7213, 0.7363] | 3.98 [3.97, 4.00] |
| | Xavier | 74.20 [73.68, 74.75] | 0.7542 [0.7390, 0.7714] | 3.99 [3.96, 4.01] |
| Conv4 - *Si. Su.* | He | 75.92 [74.93, 76.78] | 0.8084 [0.7695, 0.8438] | 4.22 [4.19, 4.26] |
| | ELUS | 77.40 [76.73, 78.25] | 0.8309 [0.7863, 0.8721] | 4.22 [4.20, 4.25] |
| | Xavier | 73.99 [73.47, 74.64] | 0.7544 [0.7390, 0.7680] | 2.51 [4.21, 4.34] |
| Conv6 - *Baseline* | He | 75.54 [74.89, 76.03] | 0.7159 [0.7001, 0.7324] | 4.98 [4.92, 5.32] |
| | ELU | 75.91 [75.43, 76.36] | 0.7009 [0.6914, 0.7159] | 4.84 [4.82, 4.85] |
| | Xavier | 75.54 [74.98, 76.13] | 0.7151 [0.7007, 0.7300] | 4.94 [4.92, 4.96] |
| Conv6 - *Si. Su.* | He | 78.42 [77.57, 79.27] | 0.6544 [0.6306, 0.6791] | 5.07 [5.05, 5.10] |
| | ELUS | 79.17 [78.05, 80.49] | 0.6685 [0.6277, 0.7151] | 5.14 [5.09, 5.21] |
| | Xavier | 78.47 [77.52, 79.23] | 0.6724 [0.6397, 0.7252] | 5.12 [5.09, 5.20] |
| Conv8 - *Baseline* | He | 72.87 [72.22, 73.53] | 0.7857 [0.7675, 0.8026] | 6.38 [6.27 , 6.89] |
| | ELU | 72.24 [71.39, 72.92] | 0.7979 [0.7814, 0.8208] | 6.27 [6.26 , 6.29] |
| | Xavier | 72.81 [72.12, 73.39] | 0.7922 [0.7734, 0.8095] | 6.31 [6.27 , 6.39] |
| Conv8 - *Si. Su.* | He | 78.92 [78.17, 79.77] | 0.6194 [0.5992, 0.6430] | 6.67 [6.61 , 6.74] |
| | ELUS | 80.91 [79.93, 81.71] | 0.6057 [0.5769, 0.6511] | 6.65 [6.60 , 6.77] |
| | Xavier | 78.51 [76.28, 79.92] | 0.6360 [0.5957, 0.7147] | 6.68 [6.64 , 6.73] |

ELU(S) initialization. We can observe the larger variance in test accuracy for the signed Supermask models. The ELUS signed Supermask models visually coincides with the baseline or lies above it, whereas He and Xavier cannot reach the same performance. Furthermore, for the deeper models, ELUS needs a few epochs to 'warm up'.

Figure A.6 shows the equality of masks for all layers in the FCN and Conv architectures. Only looking at the equality of the signed Supermasks, we can see that the picture is divided into FCN and CNN behavior. For the FCN the first layer is the largest and attains the highest pruning rate (refer to Figure A.5), whereas the hidden and output layer are not pruned as much. All initializations produce masks that are similar across all runs in about 50% of all elements. For the CNNs, we have almost no similarity in the first layers which can be partially attributed to the nature of CNNs and the concept of

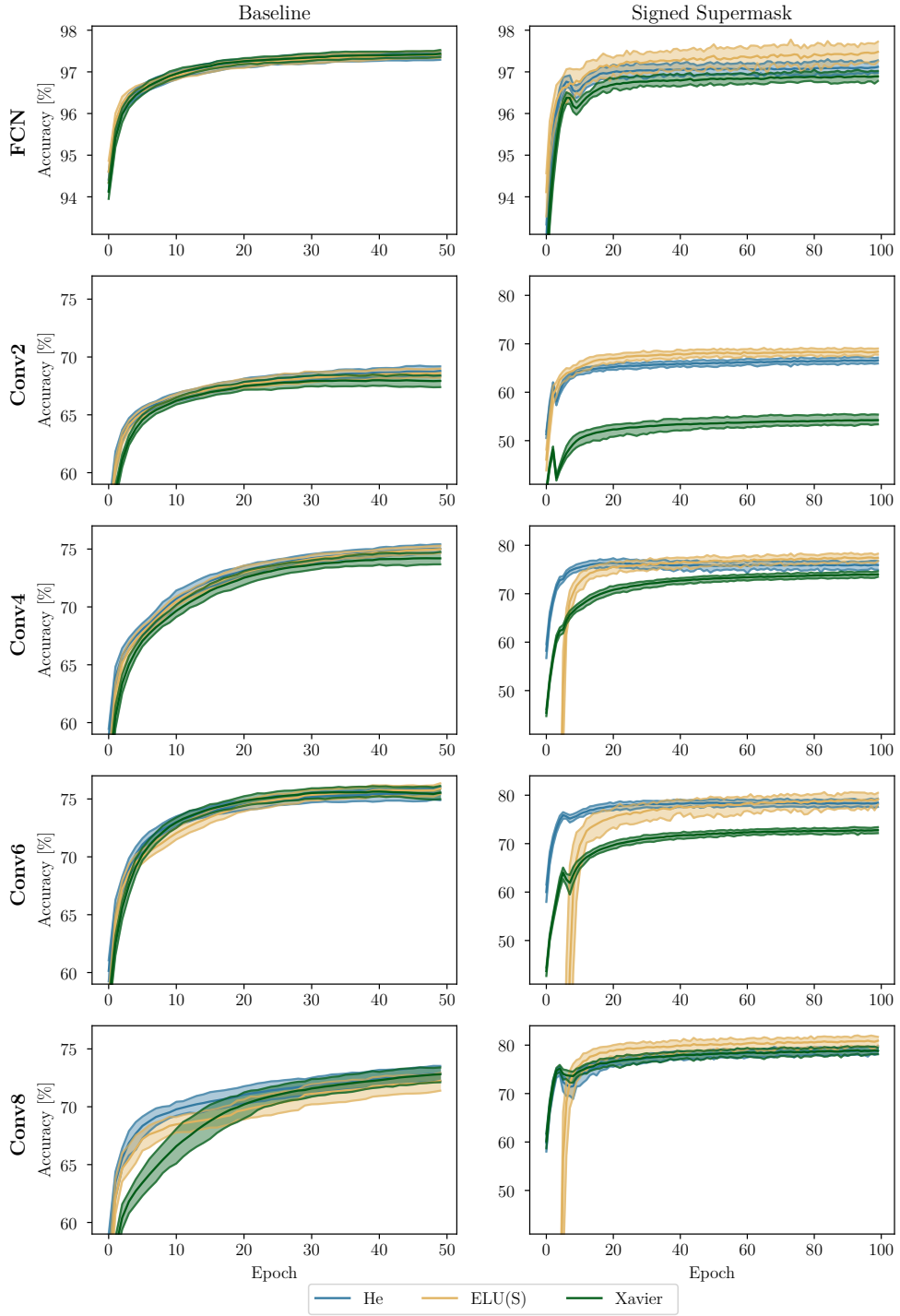



**Figure A.2: Signed Supermask: Average test accuracy** over 50 runs of all baseline (left) and signed Supermask (right) models and architectures. We also report the 5% and 95% quantiles.

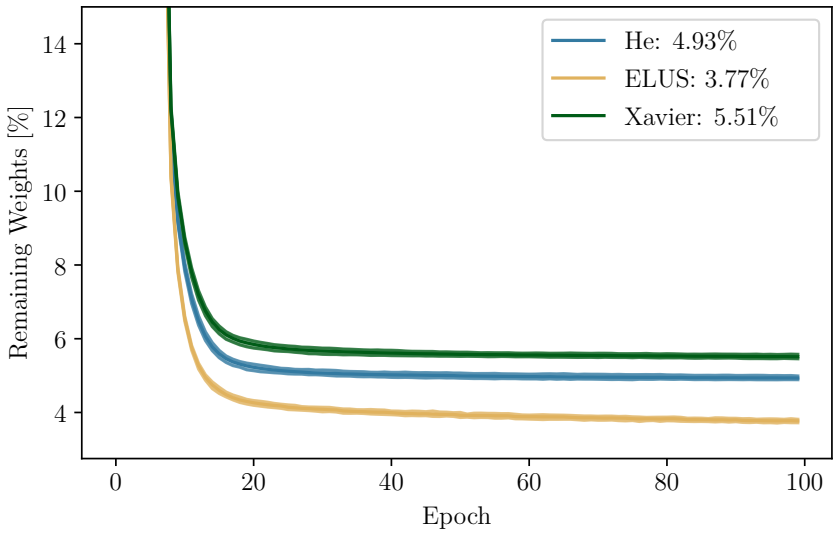



**Figure A.3: FCN signed Supermask: Average ratio of remaining weights.** We also report the 5% and 95% quantile, although the interval is barely visible indicating a robust metric. Weight count drops early in the learning phase and plateaus thereafter.

weight-sharing. Here, similarity gradually increases as the layer size and pruning rate are increased, peaking at the respective largest layer.

Taking the absolute signed Supermasks into account, we do not see any meaningful differences in those layers that gain high similarity in the signed Supermask case. However, for the output layers across almost all models that were initialized with He and Xavier we see high similarity. If contrasted with the pruning rate again, its notable that those layers are not pruned much. This can be seen most clearly in the FCNs: He and Xavier only prune the last layer very little, resulting in an absolute signed Supermask that contains many 1's. The same holds for the CNN output layers and some CNN input layers.

These findings suggest that the 0-elements are the main driver of mask similarity, i.e. the same weights are being pruned in every run.

Note that this metric is more takes a more heuristic-approach to analyze the problem. Comparing matrices or tensors is a difficult task. Introducing more precise metrics would be far beyond the scope of this paper. Figure A.7 shows two random draws of masks for the FCN and all Conv models. For each architecture, we notice unique patterns which are present in both draws. For example, the Conv8 masks show a grid-like pattern which has an area with almost completely pruned weights in the upper part of the mask. The

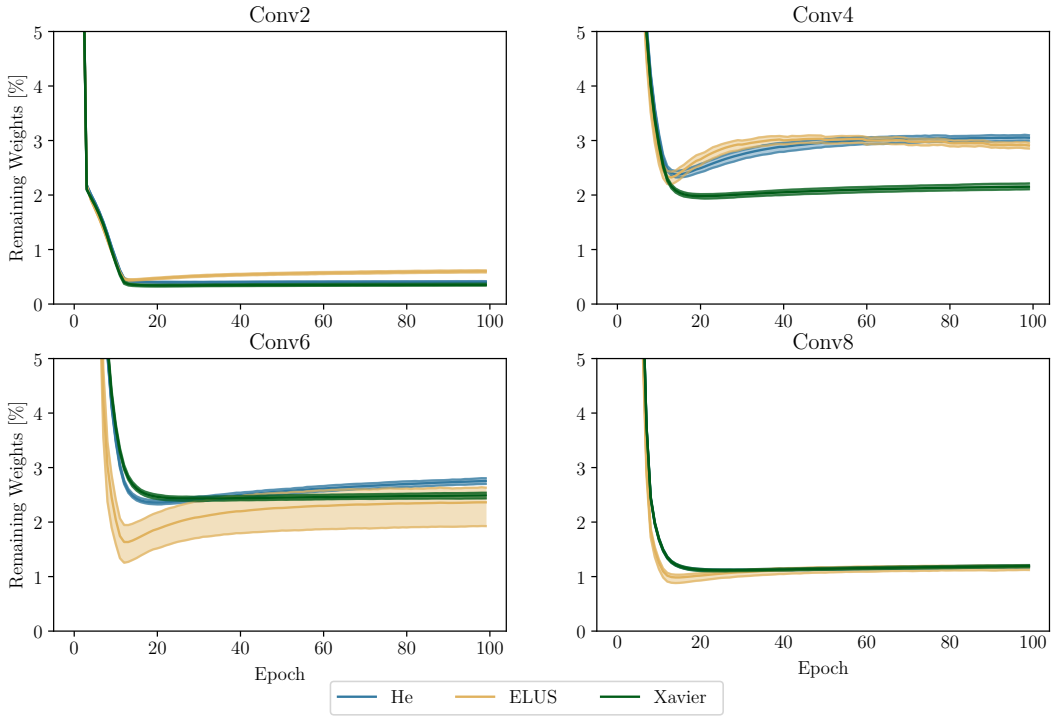



**Figure A.4: CNN signed Supermask: Average ratio of remaining weights** over 50 runs of the CNN signed Supermask models initialized with He, Xavier and ELUS and architectures during training. We also report the 5% and 95% quantiles. Besides for the Conv6 model, all confidence intervals are very narrow. Furthermore, all architectures drop the weights heavily during the first few epochs and plateau thereafter.

structure is similar for the two Conv6 masks. For Conv4, the masks show more pruning activity on the left part of the mask and a more row-wise structure. Both Conv2 samples are very sparse. The lower third of both masks show very similar patterns, while a wide horizontal band on top is almost completely pruned again.

Interpretation of the masks, e.g. like in Figure 2.1 is very difficult as the layers here are hidden and depend a lot on the previous and following layers. Still, for each architecture the layers are remarkably similar which cannot be said for standard NNs.

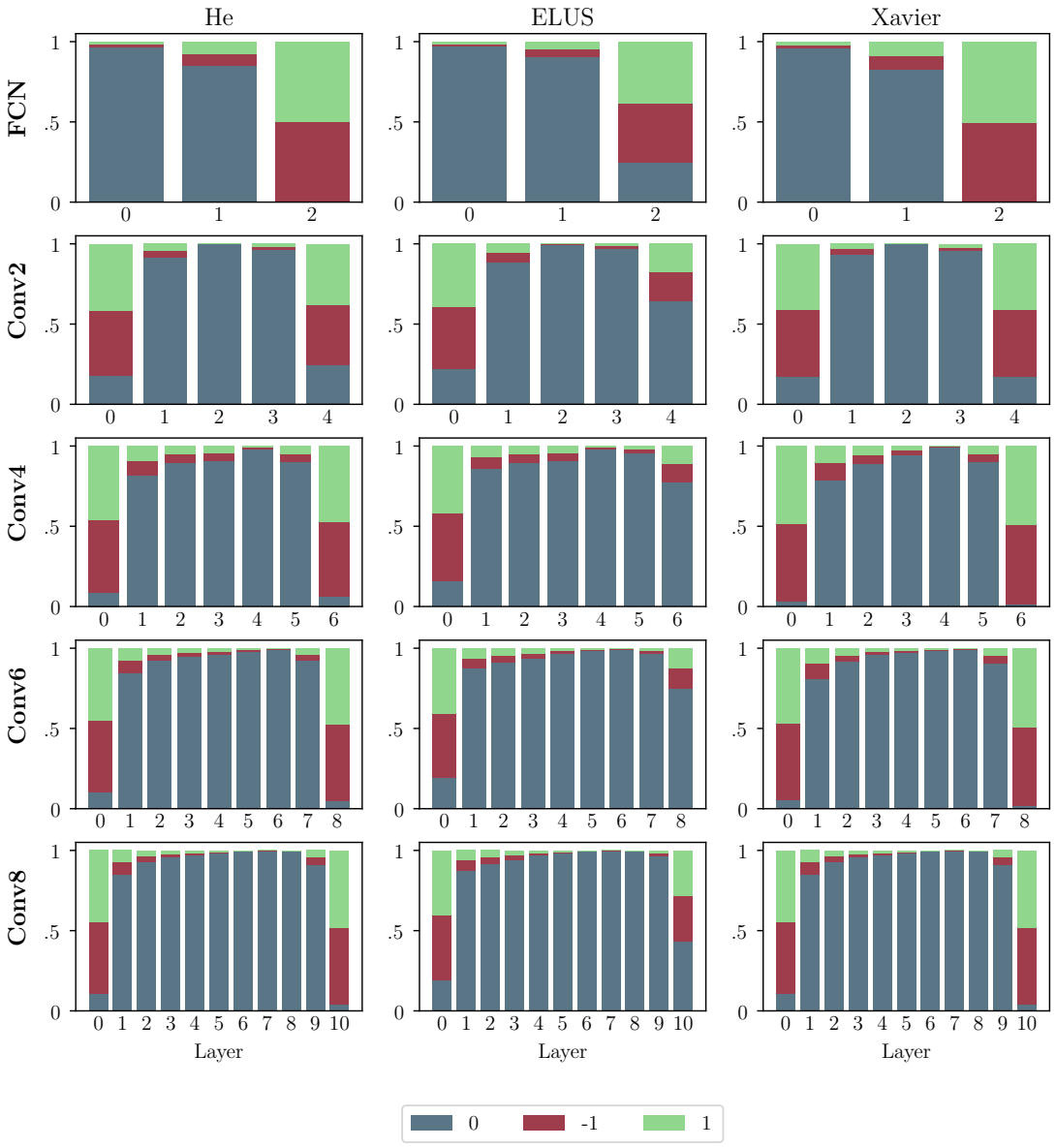



**Figure A.5: Signed Supermask: Average mask distribution** over 50 runs of all models and weight initializations. It holds for all configurations that the first layer undergoes only little pruning in contrast to the large hidden layers, where almost all weights are pruned. The behavior in the last layer differs for ELUS: there, the pruning rate is significantly higher in the last layer as compared to He and Xavier. The sparsity correlates with the size of the layer: the larger a layer, the more it is pruned.

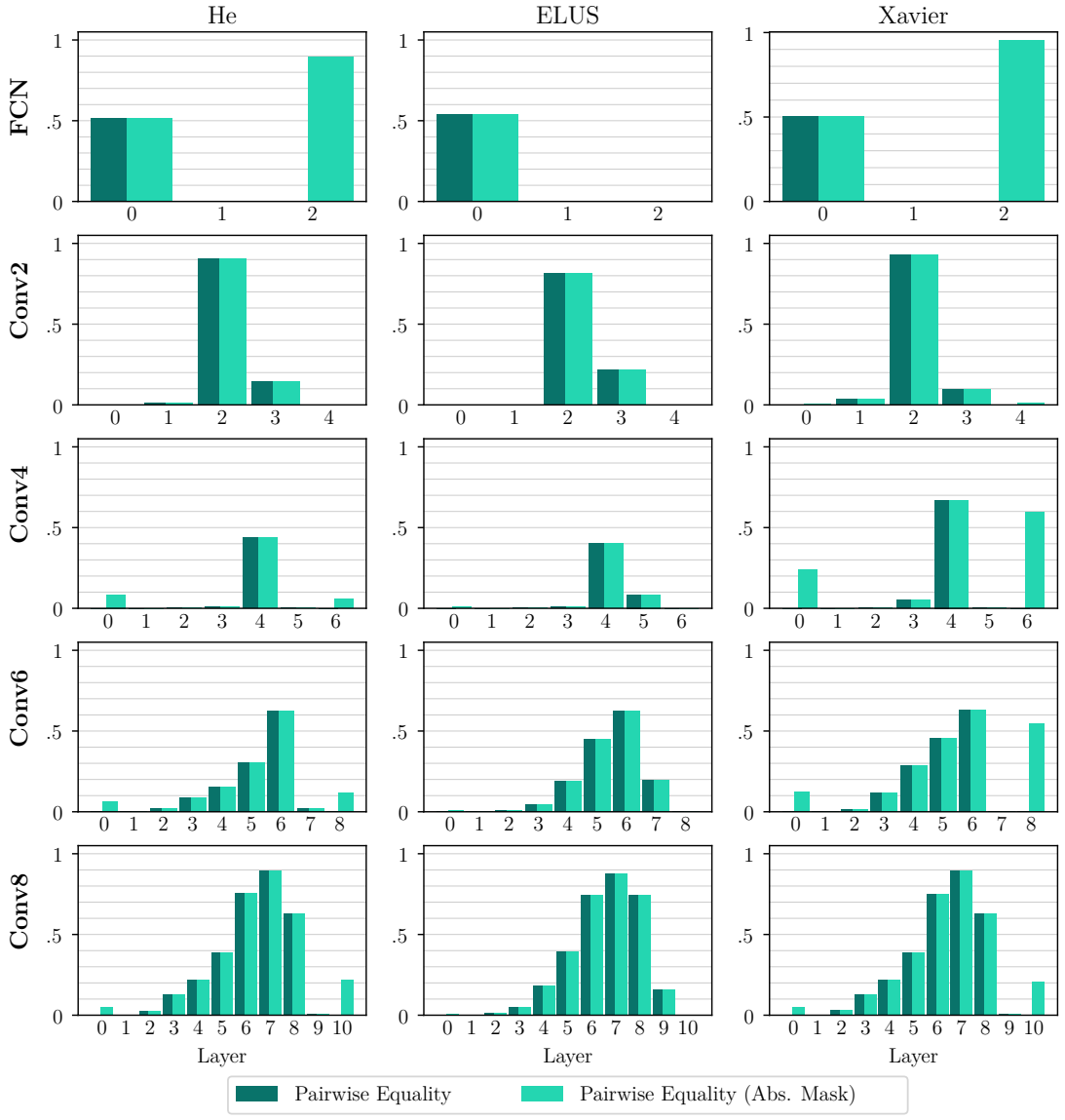



**Figure A.6: Signed Supermask: Equality of masks and absolute (i.e. binary) masks** for each architecture and weight initialization over the 50 conducted runs.

**Table A.7: ResNet signed Supermask**: Additional training time (abbreviated 'TT' below) and compression rate compared to the respective baselines. Signed Supermask require 3 to 14.5% more training time, which is made up for by the higher compression rate. Unsurprisingly, pruning rate correlates highly with the compression rate.

| | Add. TT/Epoch [%] | Compression Rate [%] |
|---|---|---|
| ResNet20 | 3.42 | 57.65 |
| ResNet20x1.5 | 8.53 | 76.08 |
| ResNet20x2.0 | 13.69 | 84.58 |
| ResNet20x2.5 | 14.42 | 89.13 |
| ResNet20x3.0 | 12.57 | 91.85 |

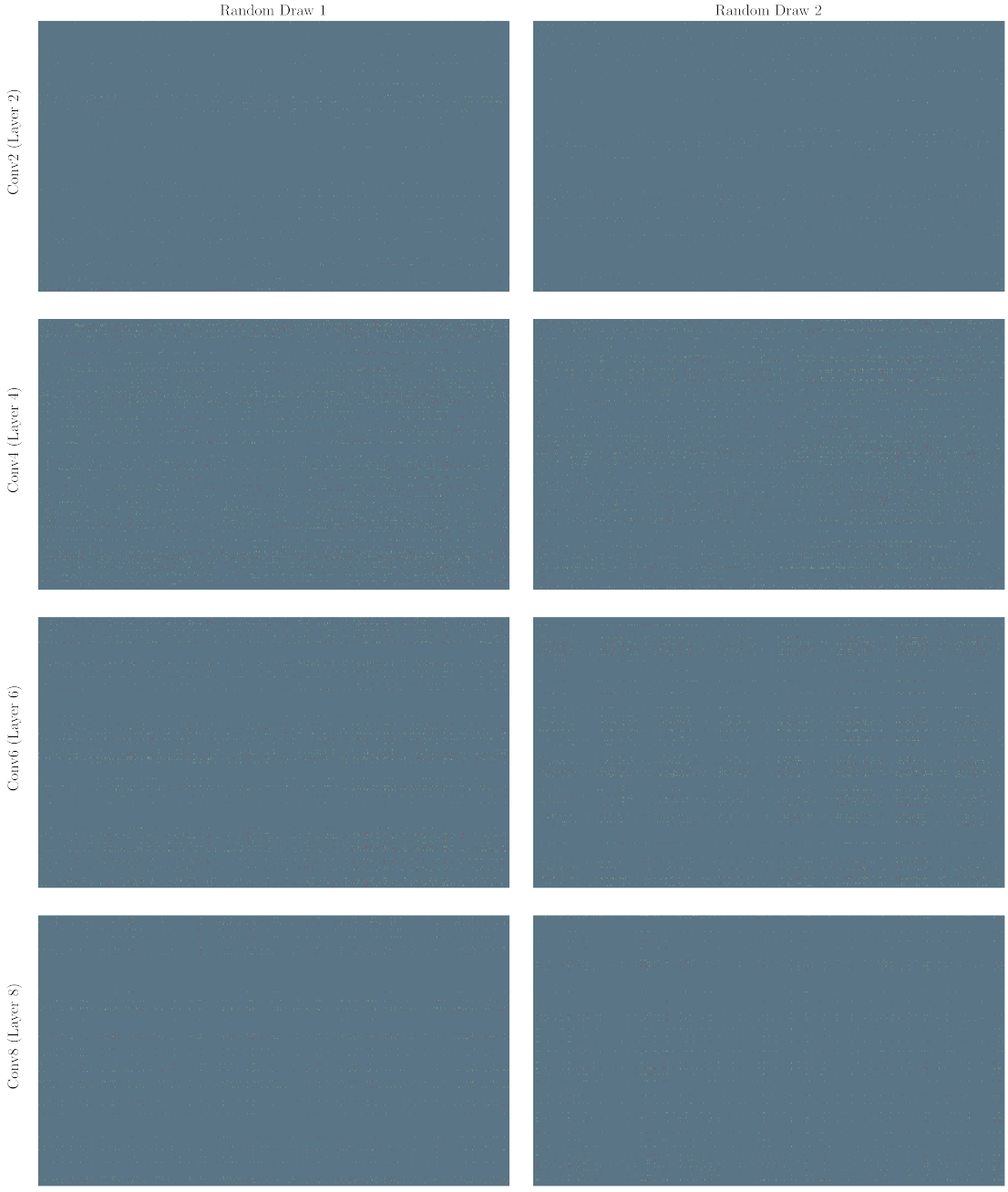



**Figure A.7: CNN signed Supermask:** We show two randomly drawn trained signed Supermasks for each architecture. The chosen layers are the respective first fully connected layer in the respective architecture (which are of different shapes for each architecture). A green dot represents '1', a red dot '-1'.

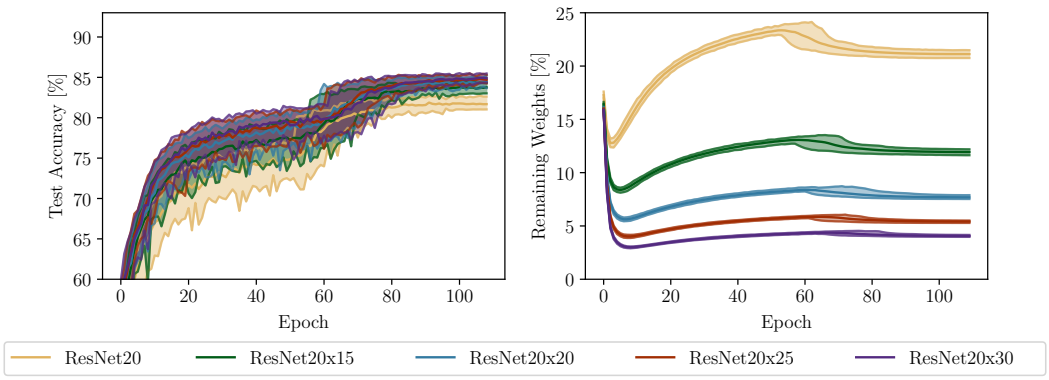



**Figure A.8: ResNet signed Supermask: Mean test accuracy and ratio of remaining weights** in addition to the respective 5% and 95% quantiles for ResNet20 and its wider siblings.


**Figure A.9: ResNet signed Supermask: Average mask distribution** (over the 50 conducted runs) of the investigated ResNet20 models. The wider the layer, the higher the pruning rate. Layers 1, 8 and 15 are 1x1 shortcut layers. Interestingly, those layers are relatively sparse compared to other hidden layers. As with the CNNs, the first and output layer are least sparse.

**Table A.8: Signed Supermask: Comparison with a broader literature periphery** with regard to the total number of remaining weights on CIFAR-10. We show those results that are also compared with other obtained results in the respective papers. Please note that, as already stated in the introduction, a direct comparison of all those methods is not possible for multiple reasons: first, complexity between the final result varies. As an example, pruned networks allow trained weights whereas a BNN only allows for binary weights and activations. Second, many works experiment with different networks and the usage of additional layers such as batch normalization or dropout. If you start with a very overparameterized network, probability theory gives the network much higher chances to include a better subnetwork than a very small one. On top of that, batch normalization or dropout can have a big impact on the predictive performance, which might hide the true performance of the proposed method.
The methods we compare our signed Supermasks to, are: Supermasks Zhou et al. (2019), edge-popup Ramanujan et al. (2020), IteRand Chijiwa et al. (2021), MPT-1/32 Diffenderfer and Kailkhura (2021), FORCE de Jorge et al. (2021), SET Mocanu et al. (2018), RigL Evci et al. (2020), GraNet Liu et al. (2021), TWN Li et al. (2016), TTQ Zhu et al. (2017), SCA Deng and Zhang (2020), LR-Net Shayer et al. (2018), BinaryConnect Courbariaux et al. (2015), BNN Hubara et al. (2016) and BBG Shen et al. (2020). Considered models are Conv6 and Conv8, VGG-19, ResNet-20, ResNet-18 and Wide ResNet 22-2. VGG-Small summarizes all small VGG-like architectures, i.e. they can slightly differ. We refer to the respective work for details. The handling of the weights can be categorized as either a binary mask ('Masked (B)'), pruned weights where the weight values change during training, ternarized and binarized weights as well as ternary masks ('Masked (T)') as is the case for our signed Supermasks.

| Method | Model | Weights | Acc. [%] | Init. Weights | Rem. Weights |
|---|---|---|---|---|---|
| Supermask | Conv6 | Masked (B) | 76.5 | 2.3 M | 0.25 - 2.1 M |
| edge-popup | Conv8 | Masked (B) | 86 | 5.3 M | 2.6 M |
| IteRand | Conv6 x2 | Masked (B) | 90 | 4.6 M | 2.3 M |
| MPT-1/32 | Conv8 | Masked (B) | 91.48 | 5.3 M | 0.23 M |
| MPT-1/32 + BN | ResNet-18 | Masked (B) | 94.8 | 11.2 M | 2.2 M |
| FORCE | VGG-19 | Masked (B) | 90 | 20 M | 0.1 M |
| SET | VGG-Small | Pruned | 88 | 8.7 M | 0.47 M |
| RigL | Wide ResNet 22-2 | Pruned | 94 | 26.8 M | 13.4 M |
| GraNet | VGG-19 | Pruned | 93.1 | 20 M | 1 M |
| TWN | VGG-Small | Ternarized | 92.56 | 5.1 M | 5.1 M |
| TTQ | ResNet-20 | Ternarized | 91.13 | 0.27 M | 0.27 M |
| SCA | VGG-Small | Ternarized | 93.41 | 4.9 M | 4.9 M |
| LR-Net | VGG-Small | Ternarized | 93.26 | 5.1 M | 5.1 M |
| BinaryConnect | VGG-Small | Binarized | 91.7 | 4.6 M | 4.6 M |
| BNN | VGG-Small | Binarized | 88.6 | 4.6 M | 4.6 M |
| BBG | ResNet-20 | Binarized | 85.3 | 0.27 M | 0.27 M |
| Sig. Supermask | Conv8 | Masked (T) | 80.91 | 5.3 M | 0.061 M |
| Sig. Supermask + BN | ResNet-20 | Masked (T) | 83.38 | 0.27 M | 0.038 M |

## A.4 Influence of Weight Distribution

In this section we analyze the influence of the weight distribution on overall performance. Therefore, we train additional models with weights drawn from a uniform distributions with the standard thresholds of He (He et al., 2015). As a baseline, we utilize the signed Supermask models with weights as signed constants presented in Section 2.3.
For both variants, we arbitrarily choose He initialization as we expect similar behavior for the other initialization methods. Hence in this section, we call the former 'He Uniform', the latter 'He SC' acting as the baseline. Note that these experiments are not exhaustive, as we only examine the two mentioned distributions. The purpose of this section is to give an intuition on the behavior of different weight distributions. We first analyze the FCN architecture, followed by a summary of the CNN architectures.

**FCN** Figure A.10 displays the average test accuracy and average ratio of remaining weights for both models with the respective 5%- and 95%-quantiles. We can note that the variance of He Uniform is a lot higher than the variance of He SC. Moreover, He Uniform performs on average slightly worse than He SC. The final mean result for He SC is 97.12%, whereas He Uniform achieves 96.77%. A Welch's t-test for significance yields that He SC significantly ($p < 0.01$) outperforms He Uniform. The higher variance

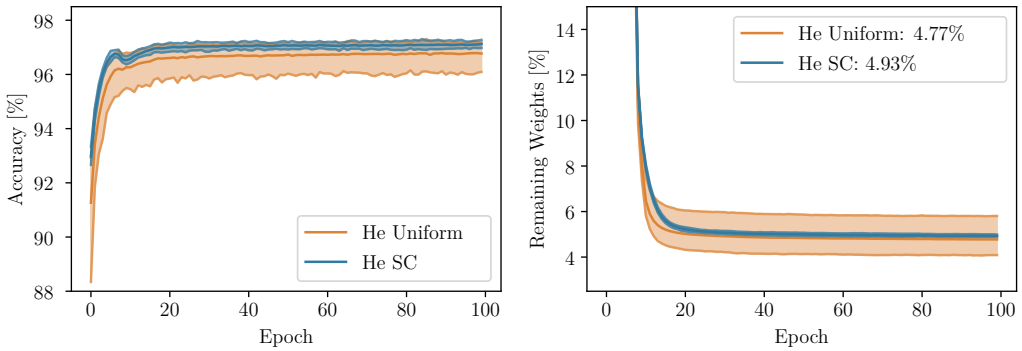



**Figure A.10: FCN signed Supermask: Average test accuracy and average remaining weights** with their respective 5%- and 95%-quantile of a FCN with weights drawn from a uniform distribution and as signed constants ('SC'). We can see that the uniform distribution yields in a much higher variance of both, test accuracy and ratio of remaining weights. Furthermore, drawing weights from a uniform distribution does not improve performance nor sparsity.

of He Uniform is also visible in the ratio of remaining weights. On average, He SC and He Uniform achieve almost the same level of sparsity, however the 5%- and 95%-quantile are almost 2pp apart for He Uniform.
We postulate that a uniform distribution does not have additional value compared to signed constants for a dense NN in the context of signed Supermask. Apparently, initializing weights with small values deteriorates performance. Recall that neither He SC nor He Uniform are scaled for the use of a signed Supermask. We further speculate that the higher variance is a cause of random initialization such that training cannot nullify the impact of randomness and ill-initialized weights. This leads to the conclusion that the value of a weight is important if not initialized well, as He Uniform holds more weight values compared to He SC and performs worse.
To summarize, using signed constants instead of a uniform distribution provides a much higher level of robustness for the FCN. Our results suggest that for signed Supermask and dense NNs, a signed constant initialization surpasses the performance of a uniform distribution.

**CNN** We now investigate the influence of weight distribution on our CNN architectures. Figure A.11 visualizes the average test accuracy with the respective 5% and 95% quantiles during training for each model. We first note that He Uniform is not behaving consistently. While with Conv2 and Conv8 the variance of He Uniform is much higher compared to He SC, it is roughly equal for Conv4 and Conv6. The average test accuracy of He Uniform is of the same magnitude as He SC, and we can only register a significantly ($p < 0.01$) better performance for Conv2 He SC compared to He Uniform. This is not sufficient to conclude on one distributions significance for the general case.
The results indicate that the additional weight values delivered by a uniform distribution do not improve the learning capability of a signed Supermask model. This is in line with the findings of the FCN architecture. It seems sufficient to initialize the weight matrix with a well-chosen constant. We further speculate that due to the higher parameter count of the investigated CNN architectures compared to the FCN, the optimizer is to some extent able to neglect the influence of poorly initialized weights.
Let us consider the average ratio of remaining weights over the course of training for each CNN model. Figure A.12 visualizes this metric for each network in addition to the

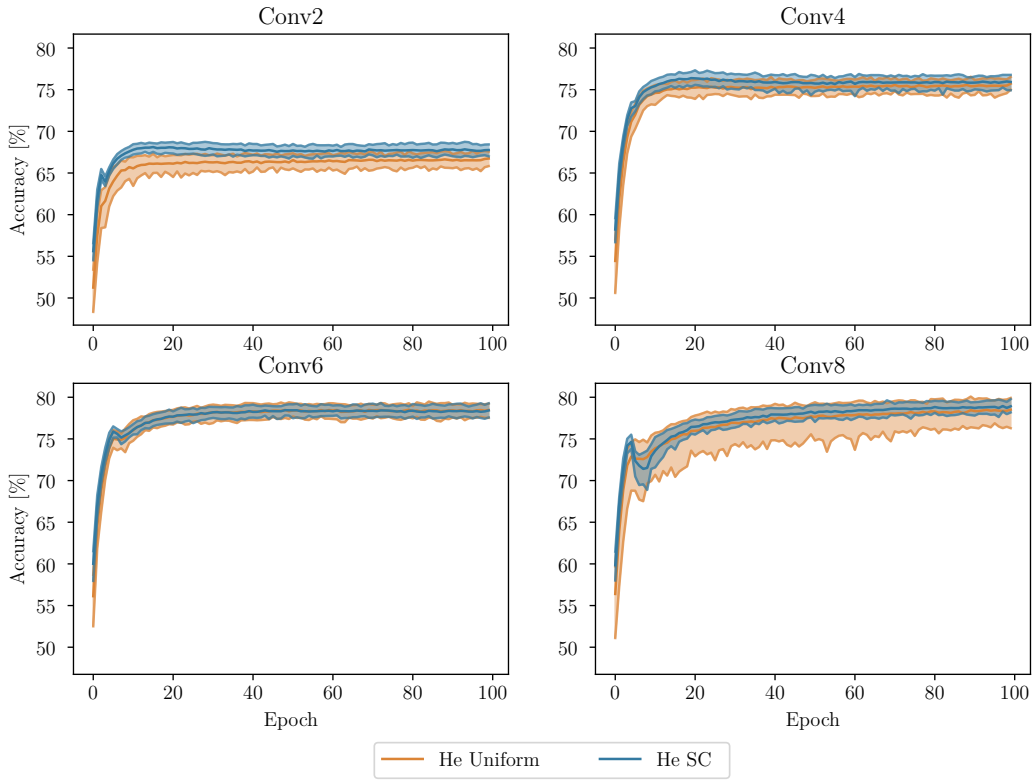



**Figure A.11: CNN signed Supermask: Average test accuracy** for He Uniform and He SC during training. We also report the respective 5%- and 95%-quantile. We find that the variance of He Uniform is at least as large as the variance of He SC, but especially in models Conv2 and Conv8 He Uniform has a much larger variance. He SC outperforms He Uniform in all architectures.

5% and 95% quantiles. Unlike observed in the case of the FCN, we cannot report a high variance for He Uniform within CNN models. In fact, the weight distribution does not seem to have an impact on the level of sparsity, as the ratios move almost equally over the course of training for each model.

Considering both the performance and level of sparsity, we do not find that the choice between uniform and signed constant distribution impacts the end result on average. However, since the test accuracy of He Uniform varies a lot more compared to He SC, the findings suggest to prefer signed constants over a uniform distribution, as signed constants bring positive side-effects such as efficient storing and more robust behavior.

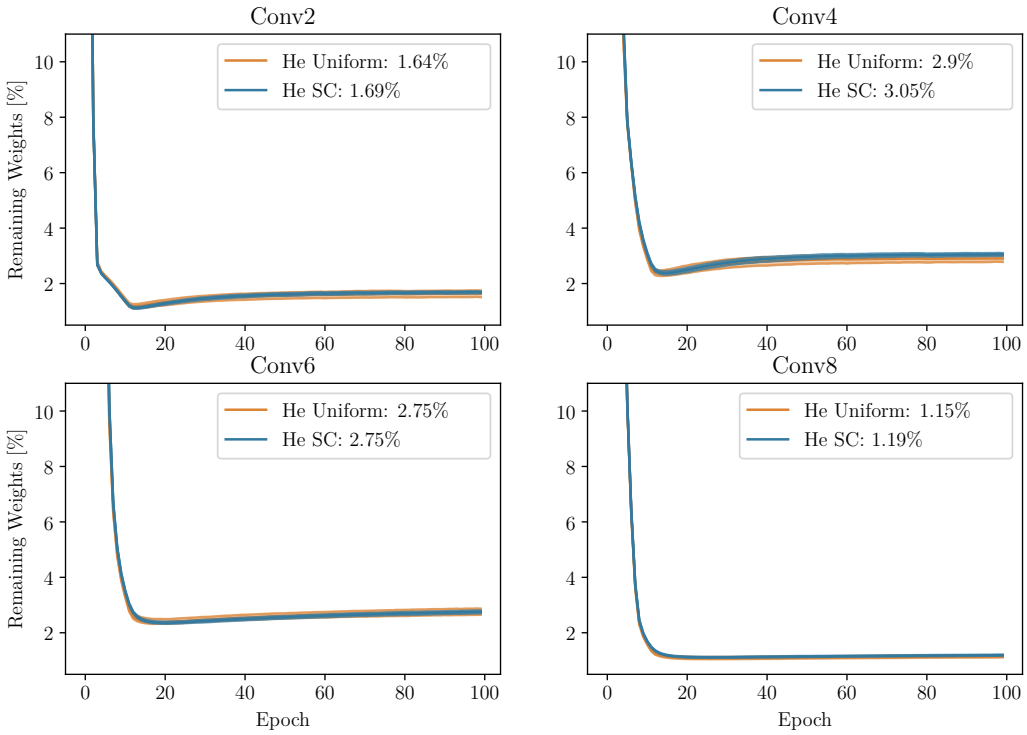



**Figure A.12: CNN signed Supermask: Average ratio of remaining weights** for He Uniform and He SC during training in addition to the respective 5%- and 95%-quantile. We find that the variance of He Uniform is at least as large as the variance of He SC but especially in models Conv2 and Conv8 He Uniform has a much larger variance. He SC outperforms He Uniform in all architectures.

To summarize both FCN and CNN behavior, it can be stated that a uniform distribution is not advantageous over a signed constant approach. We further want to re-emphasize the simplicity, yet good performance of signed constants. They have shown a very robust behavior with regards to the variance in test accuracy while performing equal or better compared to a uniform distribution. It further simplifies the storage of a network as we only have to store each weight's sign, zero and a single weight value for each layer.

## A.5 Influence of Mask Initialization

We now pursue the question to what extent mask initialization plays a role in performance. Therefore, we use the ELUS models of Section 2.3 as a baseline with Xavier uniform mask initialization (abbreviated in this section as 'ELUS/Xavier'). Furthermore, we train

the same models with ELUS uniform mask initialization (abbreviated in this section as 'ELUS/ELUS'). We compare the results separately for the FCN and CNN architectures. As in the last section, we note that these experiments are not exhaustive but give an intuition on the relationship between mask and weight initialization.

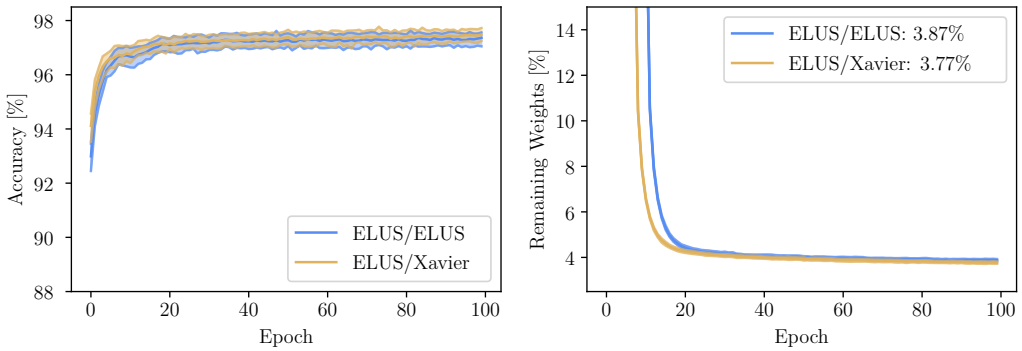



**Figure A.13: FCN signed Supermask: Average test accuracy and average remaining weights** with their respective 5%- and 95%-quantile of a ELUS/ELUS and ELUS/Xavier FCN. We can see that both models behave very similar with regards to performance and variance. ELUS/ELUS drops its weights a few epochs later but the general behavior is equal.

**FCN** Figure A.13 visualizes the average test accuracy and average remaining weight ratio with the respective 5% and 95% quantiles. We can note that there is almost no difference in variance and average test accuracy for ELUS/ELUS and ELUS/Xavier. The former achieves an average accuracy of 97.36%, whereas the latter marginally but significantly ($p < 0.01$) exceeds ELUS/ELUS with an average accuracy of 97.48%. As stated in Section 2.2.1, we presume that mask initialization is not as impactful, as long as weight and mask values are of the same magnitude. The FCN results indicate to support this thesis.

We find a slight deviation in training behavior in the ratio of remaining weights during training. ELUS/ELUS reduces its weight count slightly later in training as ELUS/Xavier does. However, the results are again very alike regarding mean and variance.

We can conclude that for the FCN, the difference in mask initialization did not influence training behavior and robustness meaningfully, yet ELUS/Xavier outperformed ELUS/ELUS slightly but significantly by 0.12pp. Moreover, the average ratio of remaining weights differs only by 0.1pp. Thus, we argue that both weight/mask

initialization combinations might be useful, depending on the task at hand.

**CNN** Our focus shifts towards the CNN architectures. Figure A.14 displays the average test accuracy during training with the respective 5% and 95% quantiles for each architecture. We can note a very robust training behavior for both mask initializations, as variance is equal across all models. Observe that only Conv6 ELUS/ELUS performs significantly ($p < 0.01$) better than ELUS/Xavier. We further find that for Conv4, Conv6 and Conv8 ELUS/ELUS begins to optimize effectively a few epochs later than ELUS/Xavier. Yet, this does not influence the final result. We assume this is caused by the optimizer and straight-through estimation, as the gradient might not be approximated properly in the very early learning phase.

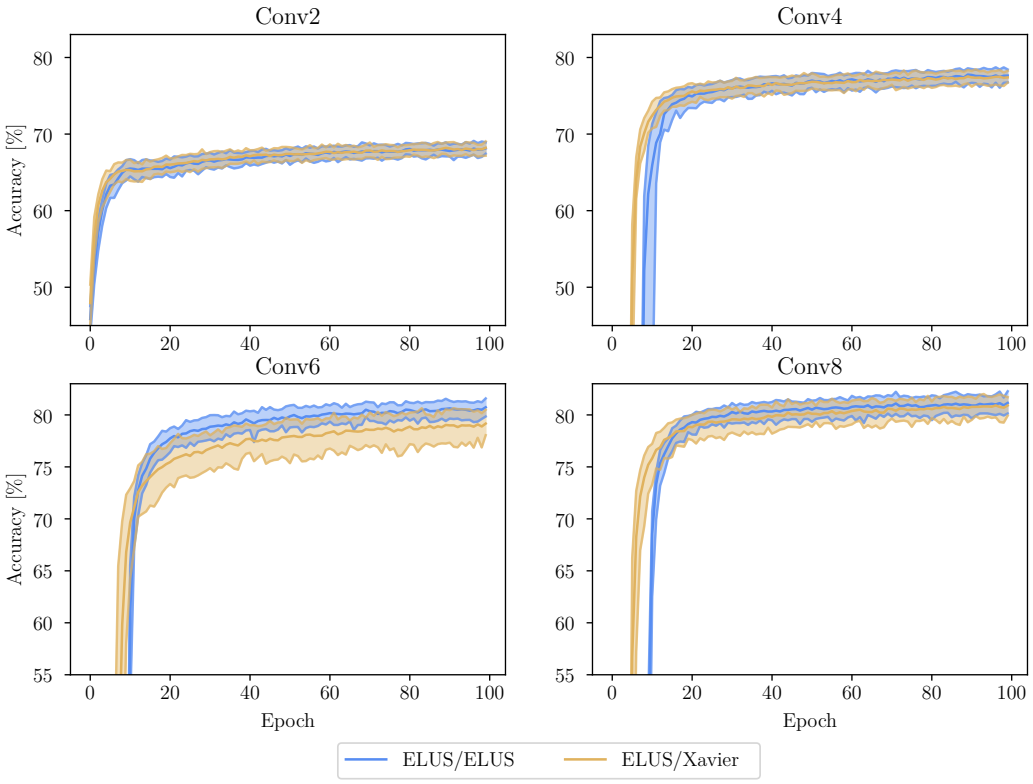



**Figure A.14: CNN signed Supermask: Average test accuracy** for ELUS/ELUS and ELUS/Xavier during training. We also report the respective 5%- and 95%-quantile. We find that for all architectures, both combinations behave and perform very similar. However, ELUS/ELUS visibly exceeds ELUS/Xavier for Conv6.

Figure A.15 visualizes the ratio of remaining weights for each CNN architecture with the respective 5% and 95% quantiles. First, we note that each model has a distinct behavior. Whereas Conv2 ascends its weights rapidly, it then almost linearly reduces the weights further until it plateaus. The other architectures exhibit a similar behavior towards the end of training, but without the linear section. This is in line with the findings in Section 2.3. Additionally, we observe that the choice of weight/mask initialization has little influence on the respective curves for each architecture. As already seen with the FCN model, ELUS/ELUS drops its weights a few epochs later than ELUS/Xavier. Moreover, ELUS/Xavier utilizes 0.02pp to 0.4pp weights less than ELUS/ELUS, depending on the network architecture.

Both patterns are consistent for FCN and CNN models. We hypothesize, that the later rise of accuracy and later drop of weight count for ELUS/ELUS is a cause of the gradient estimation. The shift between the two initializations is consistent throughout all architectures. It is furthermore consistent that ELUS/ELUS results in a marginally larger ratio of remaining weights.

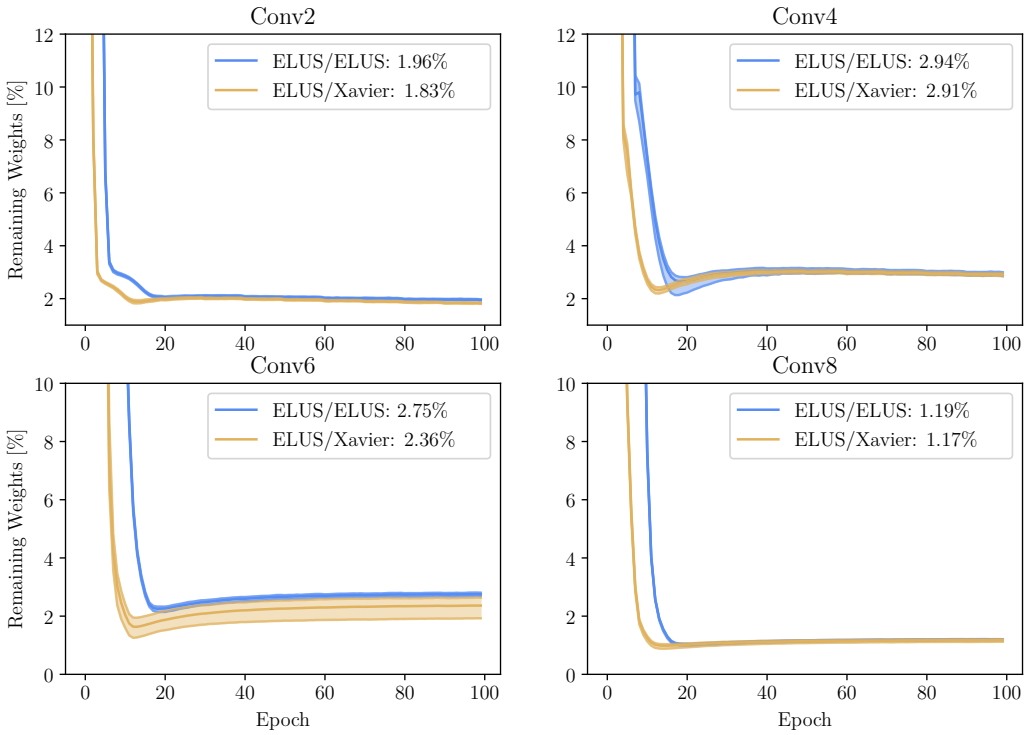



**Figure A.15: CNN signed Supermask: Average ratio of remaining weights** for ELUS/ELUS and ELUS/Xavier during training in addition to the respective 5%- and 95%-quantile. For each architecture, it is visible that the training behavior of both combinations is very similar. Moreover, we can note the different shapes of ascend of the four architectures. ELUS/ELUS drops its weights a few epochs later and achieves a marginally higher ratio at the end of training. Both initialization combinations behave very robust.

## A.6 How Far Can We Go?

Two important questions still need to be answered. Can we control the pruning rate? And if so, how small can the subnetworks get before the performance is impacted negatively? As we saw in Table 2.2, Conv2 achieved the lowest pruning rate of all CNN models. However, as described in the beginning of this section, we lowered the learning rate of Conv2 to tackle its tendencies to overfit. Hence, we can investigate if all CNN models show a similar behavior when altering the learning parameters in the same way. Table A.9 depicts two altered parameter settings. Why the learning rate somewhat influences pruning is for future work to investigate. In the following, we will call the networks initialized with ELUS and trained with those hyperparameters SiNN (**S**imple-m**i**nded

| | SiNN 1 | SiNN 2 |
|---|---|---|
| Learning Rate | 0.01 | 0.008 |
| Weight Decay | 5e-4 | 3e-4 |

**Table A.9:** Simple-minded NNs: Altered hyperparameter choices to minimize weight count during training.

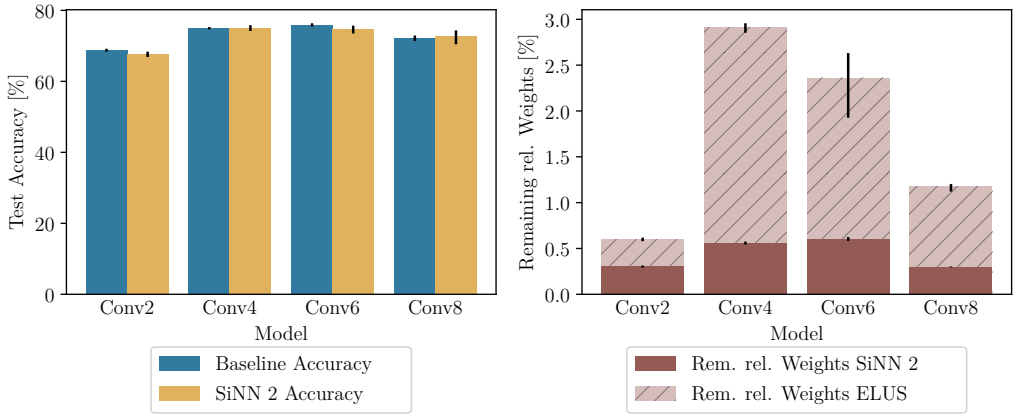



**Figure A.16: CNN SiNN 2: Test accuracy (left) for baselines and SiNN 2 models as well as average ratio of remaining weights on the right side for SiNN 2 and ELUS models.** 5% and 95% quantiles are reported in the form of error bars for both metrics.

**N**eural **N**etworks) models. As Figure A.17 reports, we see that both models prune more than 99% of the original weights, SiNN 1 reaching even pruning rates above 99.5%. Considering the predictive performance we can note, that the performance of SiNN 1 trails significantly behind SiNN 2 for each model. Looking at the right side of the plot, the difference in performance can intuitively be explained by the fewer weights that are left active: for this level of sparsity, a certain amount of weights is clearly needed to yield in acceptable performance. Combining this with the information seen in Figure A.19, we can hypothesize that the loss in performance of SiNN 1 compared to SiNN 1 comes from pruning the first layer even more. This further supports the thesis that the first layer in CNNs is crucial for the network's performance, as already stated for the ELUS models. By comparing the masks of SiNN1 and SiNN 2 in greater detail, future work could find e.g., specific weights that influence the performance more than other weights and subsequently draw conclusions on network architectures in general.

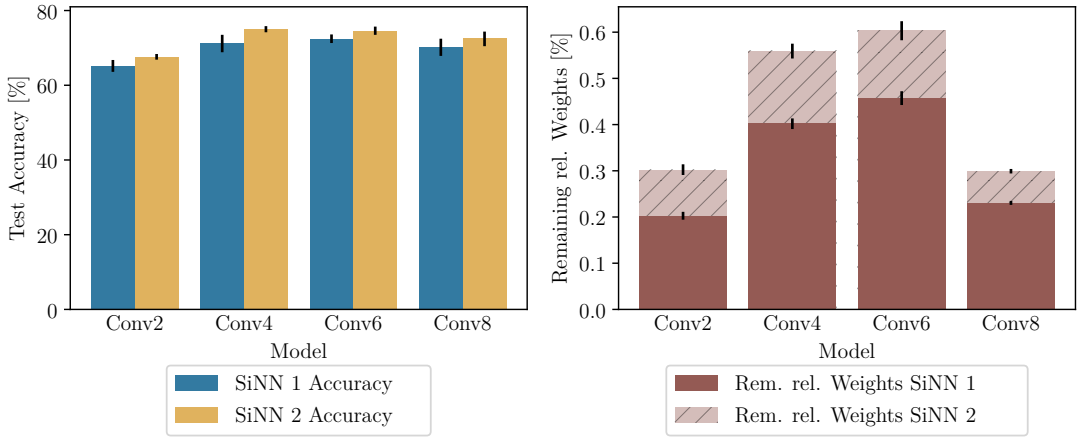



**Figure A.17: CNN SiNN signed Supermasks**: Average SiNN 1 & 2 accuracy (left) and remaining weights (right) over 50 runs of all CNN architectures with error bars indicating the 5% and 95% quantile. SiNN 1 is trumped by SiNN 1 in each architecture in terms of test accuracy.

Let us compare the SiNN models to the baseline models. Figure A.16 depicts the test accuracy of the SiNN 2 and baseline models as well as remaining weights of SiNN 2 and ELUS models on the right side. As SiNN 2 achieves higher accuracy than SiNN 1 with only few more weights, we do not further consider SiNN 1 at this point. As the compression rate after training highly correlates with the ratio of remaining weights, we do not show this metric for the sake of brevity. It can be observed that the SiNN 2 models achieve a similar test accuracy with a maximum of about 0.6% of the original weights. This is a further 50% reduction of numbers of weights compared to the ELUS models. Additionally, it is interesting to see, that the remaining absolute weight count of each CNN model is around 14.000 weights. This suggests that this is the absolute minimum of weights that are needed to map the data set accurately, independently of the model architecture. The performance difference for SiNN 2 between the different architectures then stems from a deeper arrangement of weights, supporting the hypothesis that deeper architectures (with fewer weights) are beneficial. These results reveal three findings: First, we are able to indirectly control the pruning rate of signed Supermask models by adapting the learning parameters. Second, the SiNN 2 models perform similar to their respective baselines. The uncovered subnetworks however, utilize only 0.3% to 0.6% of the original weights. Third, although performance suffers compared to the ELUS models, the signed Supermasks still perform well, even when pushed to the extreme.

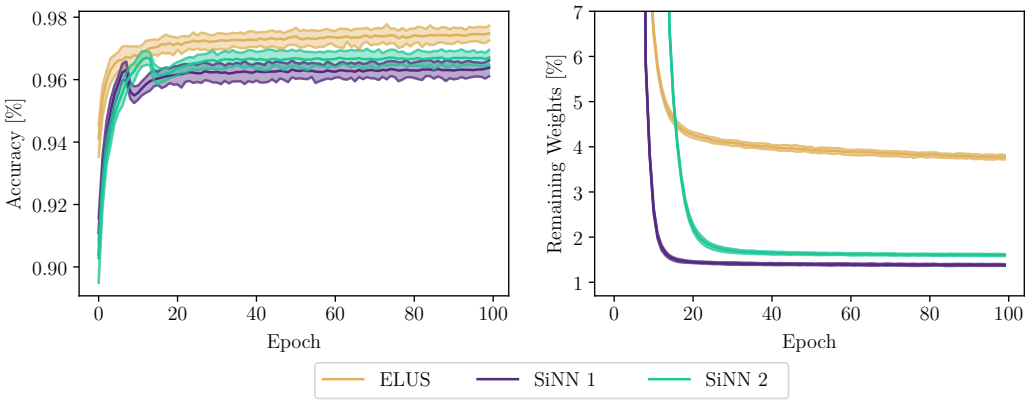



**Figure A.18: FCN SiNN signed Supermask: Average SiNN 1 & 2 accuracy (left) and relative remaining weights (right)** over 50 runs for the FCN model. As a comparison, ELUS is visualized as well. Furthermore, we report the respective 5% and 95% quantiles.

Ultimately, this demonstrate not only the performance quality of the models, but also how few parameters are actually needed in general to solve a given task well. Figure A.18 shows the FCN test accurcacy for SiNN 1 & 2 as well as the remaining weights during training. It can be noted, that SiNN 1 does not reach the test accuracy of neither SiNN 2 nor ELUS, but it also utilizes the fewest weights, around 1.4%. We argue however, that the trade-off of SiNN 2 is better, as the 'cost' of performance is only a marginally lower pruning rate.

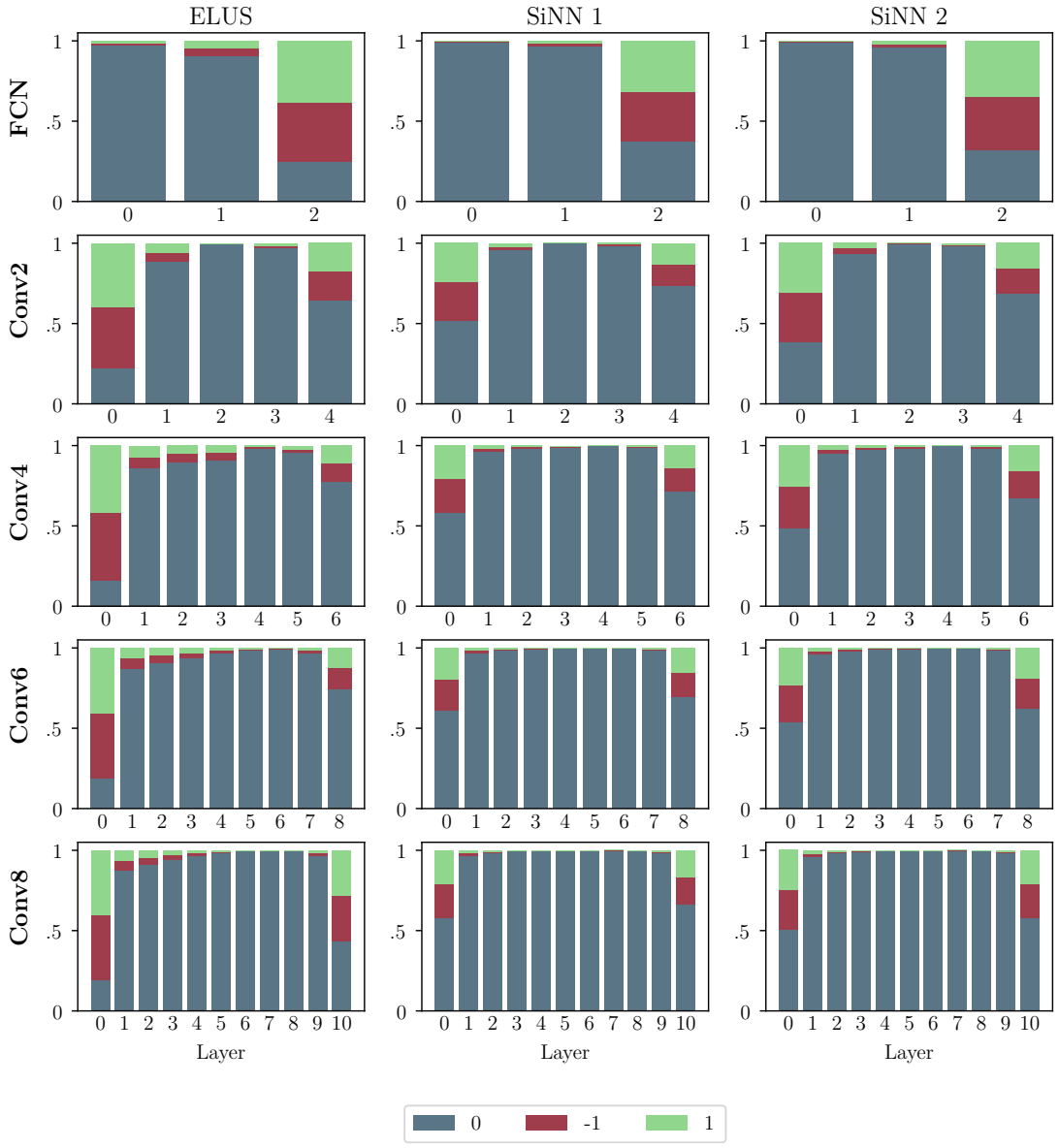



**Figure A.19: SiNN signed Supermask**: Average SiNN 1 & 2 mask distribution over 50 runs of all CNN architectures and ELUS for comparison. For the FCN, the overall picture does not change: SiNN 1 & 2 prune the first two layers even more, while the last layer only experiences little more sparsity. The same is partially true for the CNN architectures: while the last last layer remains on an equal pruning level compared to ELUS, the amount remaining weights in the first layer is considerably smaller. On top, all hidden layers are pruned very heavily. Apart from more aggressive pruning in the first layer for SiNN 1, a difference between SiNN 1 and SiNN 2 cannot be detected visually.

# B Appendix to Chapter 3: Simplifying Random Forests' Probabilistic Forecasts

## B.1 Details on SOEP Data

This section describes the SOEP data set used in Section 3.1 in more detail. We process the original data set with the following steps:

1. The variable `id`, being a unique categorical identifier for each participant, is excluded from the model.
2. If the regressor `sector` is missing, it is imputed with the value 'unknown'.
3. The qualitative regressors `sector` and `employment` are converted into sets of dummy variables (one-hot encoding).
4. The columns `gesund_org` and `lebensz_org` are excluded due to their subjective nature.
5. The sum of variables `einkommenj1` and `einkommenj2` is selected as the target variable and referred to as 'income'.
6. Entries with missing values in any of the utilized variables are removed from the data set.
7. Data from 2015 to 2018 is used for training, while data from 2019 is reserved for testing.

Table B.1 lists all the variables utilized in the data set along their respective data type and a short description. The both RFs, the standard and Top$k$ version, are trained using our standard hyperparameter choices, as detailed in Table B.5. Table B.2 reports results

on forecasting performance for various choices of $k$. The SE and CRPS performance measures reported in the table are introduced in Section 3.2.1.

**Table B.1: SOEP variables.** Description of variables in the SOEP data set, including our encoding choices after our preprocessing.

| Variable Name | Encoding | Explanation |
|---|---|---|
| id (dropped) | Categorical | Number identifying each person in the data set |
| survey_year | Integer | Year in which survey was taken |
| female | Boolean | = 1 person is female, = 0 else |
| age | Integer | Age in years |
| n_persons | Integer | Number of persons living in household |
| n_children | Integer | Number of children living in household |
| years_educ | Float | Years of education |
| employed | Categorical | Employment status of person<br>(6 distinct entries, description and ID given) |
| sector | Categorical | Sector in which person is employed<br>(75 distinct entries, description and ID given) |
| income | Float | Annual income in Euros |

**Table B.2: SOEP performance.** Forecasting performance on the SOEP data set of both, point and probabilistic predictions, measured with SE and CRPS, respectively. Analogous to Tables 3.2 and 3.4, we also report the relative losses.

| $k$ | SE | CRPS | Rel. SE | Rel. CRPS |
|---|---|---|---|---|
| full | $1.55 \times 10^8$ | 4822.18 | - | - |
| 3 | $1.57 \times 10^8$ | 5432.48 | 1.01 | 1.13 |
| 5 | $1.50 \times 10^8$ | 5090.59 | 0.97 | 1.06 |
| 10 | $1.47 \times 10^8$ | 4874.65 | 0.95 | 1.01 |
| 20 | $1.48 \times 10^8$ | 4801.58 | 0.96 | 1.00 |
| 50 | $1.51 \times 10^8$ | 4786.33 | 0.97 | 0.99 |

## B.2 Details on Empirical Experiments

Here we present further details on the empirical experiments of Section 3.3. Table B.3 lists the data sets used in the experiments, following the analysis of Grinsztajn et al. (2022). The data sets cover a wide spectrum of size, number of covariates and domains. They can be easily downloaded using the URLs listed in the table. In the case of `delays_zurich_transport`, we subsampled the data set to 20% of its original size (which is shown in the table) due to computational reasons, resulting in roughly 1.1

million observations. Table B.4 describes the grid of hyperparameter values we considered for the analysis in Section 3.3.3, and Table B.5 presents the best choices selected via cross-validation. For a given data set, we use a grid search with 5-fold cross-validation. For computational reasons, we use simplified procedures for the three largest data sets (`medical_charges`, `nyc-taxi-green-dec-2016` and `delays_zurich_transport`), where we use a single holdout set which consists of 25% of the training data. Furthermore, we subsample the `nyc-taxi-green-dec-2016` and `delays_zurich_transport` data sets to 30% and 15% of their training set size. Table B.5 shows that hyperparameter choices are often the same across both loss functions (CRPS and SE), whereas differences between 'Full' and 'Top3' are more pronounced. Hence users of simplified RFs (such as Top3) should consider tuning hyperparameters to optimize the performance of these simplified RFs directly, instead of optimizing the performance of full RFs.

**Table B.3: Data sets tested.** Following Grinsztajn et al. (2022), this table lists all tested data sets, their respective sizes (nr. of observations and regressors), name of the target variable, and URL.

| Name of Data set | Number of Observations | Number of Regressors | Target Variable | URL |
|---|---|---|---|---|
| cpu_act | 8192 | 21 | usr | https://www.openml.org/d/44132 |
| pol | 15 000 | 26 | foo | https://www.openml.org/d/44133 |
| elevators | 16 599 | 16 | Goal | https://www.openml.org/d/44134 |
| wine_quality | 6497 | 11 | quality | https://www.openml.org/d/44136 |
| Ailerons | 13 750 | 33 | goal | https://www.openml.org/d/44137 |
| houses | 20 640 | 8 | medianhousevalue | https://www.openml.org/d/44138 |
| house_16H | 22 784 | 16 | price | https://www.openml.org/d/44139 |
| diamonds | 53 940 | 6 | price | https://www.openml.org/d/44140 |
| Brazilian_houses | 10 692 | 8 | totalBRL | https://www.openml.org/d/44141 |
| Bike_Sharing_Demand | 17 379 | 6 | count | https://www.openml.org/d/44142 |
| nyc-taxi-green-dec-2016 | 581 835 | 9 | tipamount | https://www.openml.org/d/44143 |
| house_sales | 21 613 | 15 | price | https://www.openml.org/d/44144 |
| sulfur | 10 081 | 6 | y1 | https://www.openml.org/d/44145 |
| medical_charges | 163 065 | 3 | AverageTotalPayments | https://www.openml.org/d/44146 |
| MiamiHousing2016 | 13 932 | 13 | SALEPRC | https://www.openml.org/d/44147 |
| superconduct | 21 263 | 79 | criticaltemp | https://www.openml.org/d/44148 |
| yprop_4_1 | 8885 | 42 | oz252 | https://www.openml.org/d/45032 |
| delays_zurich_transport | 5 465 575 $\times 0.2$ | 8 | delay | https://www.openml.org/d/45034 |

**Table B.4: Hyperparameter search grid**. The listed values are possible values for each hyperparameter for the hyperparameter tuning. Explanations of the hyperparameters can be found in the caption of Table B.5. This results in 44 different hyperparameter combinations. Hyperparameters indicated with an asterisk were not used for `medical_charges` to reduce computational overhead.

| Hyperparameter | Possible Values |
|---|---|
| min_samples_leaf | [1, 2, 4, 6, 8, 10, 15, 20, 30*, 40*, 50] |
| max_features | [0.333, 'sqrt', 0.5, 1.0] |

**Table B.5: Hyperparameters selected via cross-validation**. The table presents the optimal hyperparameters according to 5-fold cross-validation (with exceptions for the three largest data sets, see text for details). The table lists the best choices for both hyperparameters (`max_features` and `min_samples_leaf`), two loss functions (CRPS and SE) and depending on whether we consider the full RF or its simplified Top3 variant. The first row represents our standard hyperparameter choice (considered in Sections 3.3.1 and 3.3.2) which is the same for each data set. `min_samples_leaf` is the minimum number of samples a leaf must contain and `max_features` (also called `mtry` in some software packages) denotes the number of features considered in each split, where 'sqrt' denotes the floored square root of the number of total features and real numbers correspond to the floored fraction of total features. Possible values for each hyperparameter are listed in Table B.4. The number of trees (bootstrap iterations) is set to 1000 for all data sets, the depth remains unrestricted and the minimum number of samples required to be considered for another split is fixed to 5.

| | `max_features` | | | | `min_samples_leaf` | | | |
|---|---|---|---|---|---|---|---|---|
| Tuned on: | Full | | Top3 | | Full | | Top3 | |
| Data set | CRPS | SE | CRPS | SE | CRPS | SE | CRPS | SE |
| standard | sqrt | sqrt | sqrt | sqrt | 1 | 1 | 1 | 1 |
| cpu_act | 0.5 | 0.5 | 0.333 | 0.333 | 1 | 1 | 4 | 4 |
| pol | 0.5 | 0.5 | 0.333 | 0.333 | 1 | 1 | 20 | 20 |
| elevators | 1.0 | 1.00 | 0.5 | 0.5 | 2 | 1 | 4 | 4 |
| wine_quality | 0.333 | 0.33 | 0.333 | 0.333 | 1 | 1 | 8 | 8 |
| Ailerons | 1.0 | 1.00 | 0.5 | 0.5 | 2 | 2 | 20 | 20 |
| houses | 1.0 | 1.00 | 0.5 | 0.5 | 2 | 2 | 4 | 4 |
| house_16H | 0.5 | 0.5 | sqrt | sqrt | 1 | 1 | 8 | 8 |
| diamonds | sqrt | sqrt | 0.333 | 0.333 | 10 | 8 | 50 | 50 |
| Brazilian_houses | 1.0 | 1.00 | 1.0 | 0.5 | 1 | 1 | 1 | 1 |
| Bike_Sharing_Demand | 0.5 | 1.00 | sqrt | sqrt | 6 | 10 | 15 | 15 |
| nyc-taxi-green-dec-2016 | 1.0 | 1.00 | 1.0 | 1.0 | 4 | 4 | 8 | 8 |
| house_sales | 0.5 | 0.50 | 0.5 | 0.5 | 1 | 1 | 6 | 10 |
| sulfur | 1.0 | sqrt | sqrt | 0.333 | 1 | 1 | 2 | 2 |
| medical_charges | 1.0 | 1.00 | 0.333 | 0.333 | 50 | 50 | 15 | 15 |
| MiamiHousing2016 | 0.5 | 0.33 | sqrt | sqrt | 1 | 1 | 8 | 2 |
| superconduct | 0.333 | 0.33 | 0.333 | sqrt | 2 | 1 | 6 | 2 |
| yprop_4_1 | 0.333 | sqrt | sqrt | sqrt | 6 | 2 | 50 | 40 |
| delays_zurich_transport | 0.333 | 0.33 | 0.5 | 0.5 | 50 | 50 | 4 | 8 |

## B.3 Details on Section 3.4

Here we derive the results for $\mathbb{E}[\text{SE}]$ and $\mathbb{E}[\text{CRPS}]$ stated in Equations 3.13 and 3.14.

### B.3.1 SE

We start with the derivation for the squared error, which is given by

$$\mathbb{E}[\text{SE}] = \int \int \mathbb{E}[\text{SE}(\omega, \mathbf{u})] \; d\text{F}_{\omega}(\omega) dF_{\mathbf{u}}(\mathbf{u}).$$

First, we rewrite Equation 3.11:

$$\begin{aligned}
\mathbb{E}[\text{SE}(\omega, \mathbf{u})] &= \sum_{i=1}^{n} \omega_i^* (u_i - \sum_{j=1}^{n} \omega_j u_j)^2 \\
&= \sum_{i=1}^{n} \omega_i^* u_i^2 - 2 \sum_{i=1}^{n} \omega_i^* u_i \sum_{j=1}^{n} \omega_j u_j \\
&+ (\sum_{j=1}^{n} \omega_j u_j)^2 \underbrace{\sum_{i=1}^{n} \omega_i^*}_{=1}.
\end{aligned}$$

We next change the order of integration and calculate the expectation with respect to the support points, i.e., we consider the expected value with respect to $\mathbf{u}$:

$$\begin{aligned}
\mathbb{E}_{\mathbf{u}}[\mathbb{E}[\text{SE}(\omega, \mathbf{u})]] &= \int \mathbb{E}[\text{SE}(\omega, \mathbf{u})] \; dF_{\mathbf{u}}(\mathbf{u}) \\
&= \mathbb{E}_{\mathbf{u}}[\sum_{i=1}^{n} \omega_i^* u_i^2] - 2 \; \mathbb{E}_{\mathbf{u}}[\sum_{i=1}^{n} \omega_i^* u_i \sum_{j=1}^{n} \omega_j u_j] + \mathbb{E}_{\mathbf{u}}[(\sum_{i=1}^{n} \omega_i u_i)^2] \\
&= 1 - 2 \sum_{i=1}^{n} \omega_i^* \omega_i + \sum_{i=1}^{n} \omega_i^2,
\end{aligned}$$

where we have used the assumption that the elements of $\mathbf{u}$ are independently standard normal. Next, recall that

$$\mathbb{E}[\omega_i] = \begin{cases} \theta/k & \text{if } i \in \mathcal{I} \\ (1-\theta)/(n-k) & \text{if } i \notin \mathcal{I}, \end{cases}$$
$$\mathrm{Var}[\omega_i] = \begin{cases} \theta^2 \ \mathrm{Var}[Z_1] & \text{if } i \in \mathcal{I} \\ (1-\theta)^2 \ \mathrm{Var}[Z_2] & \text{if } i \notin \mathcal{I}. \end{cases}$$

Furthermore, from the variance of a Dirichlet distributed random variable, we obtain

$$\mathrm{Var}[Z_j] = \begin{cases} \frac{k-1}{k^2(kd_1+1)} & \text{if } j = 1 \\ \frac{n-k-1}{(n-k)^2((n-k)d_2+1} & \text{if } j = 2. \end{cases}$$

We are now ready to calculate the expectation with respect to $\omega$:

$$\begin{aligned} \mathbb{E}_\omega[\mathbb{E}_{\mathbf{u}}[\mathbb{E}\ \mathrm{SE}(\omega, \mathbf{u})]] &= \int\int \mathbb{E}\ \mathrm{SE}(\omega, \mathbf{u})\ dF_{\mathbf{u}}(\mathbf{u}) d\mathrm{F}_\omega(\omega) \\ &= 1 - 2\left( \sum_{i \in I} \overbrace{\omega_i^*}^{=\frac{\theta^*}{k}} \underbrace{\mathbb{E}_\omega[\omega_i]}_{=\frac{\theta}{k}} + \sum_{i \notin I} \overbrace{\omega_i^*}^{=\frac{1-\theta^*}{n-k}} \underbrace{\mathbb{E}_\omega[\omega_i]}_{=\frac{1-\theta}{n-k}} \right) \\ &\quad + (\sum_{i \in I} \mathbb{E}_\omega[\omega_i^2] + \sum_{i \notin I} \mathbb{E}_\omega[\omega_i]^2) \\ &= 1 - 2\left\{ \frac{\theta^*\theta}{k} + \frac{(1-\theta^*)(1-\theta)}{n-k} \right\} \\ &\quad + \frac{\theta^2}{k} + \frac{(1-\theta)^2}{n-k} + \frac{\theta^2(k-1)}{k(kd_1+1)} + \frac{(1-\theta)^2(n-k-1)}{(n-k)((n-k)d_2+1)} \end{aligned}$$

□

### B.3.2 CRPS

We seek to evaluate the following integral:

$$\mathbb{E}[\mathrm{CRPS}] = \int\int \mathbb{E}[\mathrm{CRPS}(\omega, \mathbf{u})]\ d\mathrm{F}_\omega(\omega) dF_{\mathbf{u}}(\mathbf{u}),$$

where $\mathbb{E}[\mathrm{CRPS}(\omega, \mathbf{u})]$ is given in Equation 3.12. Note that a random variable $W \coloneqq |W_1 - W_2|$ with $W_1, W_2 \sim \mathcal{N}(0,1)$ follows a folded normal distribution with mean

$\mu_Y = 2/\sqrt{\pi}$. Using this fact and Equation 3.12, the expected value with respect to $\mathbf{u}$ is given by

$$\mathbb{E}_{\mathbf{u}}[\mathbb{E}[\mathrm{CRPS}(\omega, \mathbf{u})]] = \frac{2}{\sqrt{\pi}} \sum_{i=1}^{n} \sum_{j \neq i} \omega_i \left( \omega_j^* - \frac{\omega_j}{2} \right). \tag{B.1}$$

To simplify notation, we define $c \coloneqq 2/\sqrt{\pi}$. In order to compute the expected value of the expression in Equation B.1 with respect to $\omega$, we differentiate between four cases:

**Case (1)** $i, j \in \mathcal{I}$**.** It holds that $\mathrm{Cov}(\omega_i, \omega_j) = -\frac{\theta^2}{k^2(kd_1+1)}$. We hence obtain

$$\sum_{i \in \mathcal{I}} \sum_{j \notin \mathcal{I}} \mathbb{E}_\omega[\omega_i]\omega_j^* - \frac{\mathbb{E}_\omega[\omega_i\omega_j]}{2} = c(k-1)\theta \left( \frac{\theta^*}{k} - \frac{\theta d_1}{2(kd_1+1)} \right).$$

**Case (2)** $i, j \notin \mathcal{I}$**.** Here we have $\mathrm{Cov}(\omega_i, \omega_j) = -\frac{(1-\theta)^2}{(n-k)^2((n-k)d_2+1)}$, and hence

$$c \sum_{i \notin \mathcal{I}} \sum_{j \notin \mathcal{I}} \left( \mathbb{E}_\omega[\omega_i]\omega_j^* - \frac{\mathbb{E}_\omega[\omega_i\omega_j]}{2} \right) = c(n-k-1)(1-\theta) \left( \frac{(1-\theta^*)}{n-k} - \frac{(1-\theta)d_2}{2((n-k)d_2+1)} \right).$$

**Case (3)** $i \in \mathcal{I},\ j \notin \mathcal{I}$**.** With $\mathrm{Cov}(\omega_i, \omega_j) = 0$, we obtain

$$\begin{aligned} c \sum_{i \in \mathcal{I}} \sum_{j \notin \mathcal{I}} \left( \mathbb{E}_\omega[\omega_i]\omega_j^* - \frac{\mathbb{E}_\omega[\omega_i\omega_j]}{2} \right) &= ck(n-k) \left( \frac{\theta(1-\theta^*)}{k(n-k)} - \frac{\theta(1-\theta)}{2k(n-k)} \right) \\ &= c \left( \frac{\theta}{2} - \theta\theta^* + \frac{\theta^2}{2} \right). \end{aligned}$$

**Case (4)** $i \notin \mathcal{I},\ j \in \mathcal{I}$**.** It again holds that $\mathrm{Cov}(\omega_i, \omega_j) = 0$, and hence

$$c \sum_{i \notin \mathcal{I}} \sum_{j \in \mathcal{I}} \left( \mathbb{E}_\omega[\omega_i]\omega_j^* - \frac{\mathbb{E}_\omega[\omega_i\omega_j]}{2} \right) = c \left( \theta^* - \theta\theta^* - \frac{\theta}{2} + \frac{\theta^2}{2} \right).$$

Using Equation B.1, the earlier definition $c = 2/\sqrt{\pi}$, and summarizing all four cases, we obtain

$$\begin{aligned}\mathbb{E}[\text{CRPS}] = &\frac{2}{\sqrt{\pi}}\left\{\theta(k-1)\left[\frac{\theta^*}{k} - \frac{d_1\theta}{2(kd_1+1)}\right]\right\} \\ &+ \frac{2}{\sqrt{\pi}}\left\{(1-\theta)(n-k-1)\left[\frac{1-\theta^*}{n-k} - \frac{(1-\theta)d_2}{2((n-k)d_2+1)}\right]\right\} \\ &+ \frac{2}{\sqrt{\pi}}\left\{\theta^* + \theta^2 - 2\theta^*\theta\right\} \qquad \text{(B.2)}\end{aligned}$$

□

To highlight the quadratic nature of the expected values in $\theta$, we rewrite the unconditional expectations:

$$\begin{aligned}\mathbb{E}[\text{SE}] = &\theta^2\left(\frac{1}{k} + \frac{1}{n-k} + \frac{k-1}{k(kd_1+1)} + \frac{n-k-1}{(n-k)((n-k)d_2+1)}\right) \\ &+ 2\theta\left(-\frac{\theta^*}{k} - \frac{\theta^*}{n-k} - \frac{n-k-1}{(n-k)((n-k)d_2+1)}\right) \\ &+ \frac{n-k-1}{(n-k)((n-k)d_2+1)} + \frac{2\theta^*-1}{n-k} + 1,\end{aligned}$$

$$\begin{aligned}\mathbb{E}[\text{CRPS}] = &\frac{1}{\sqrt{\pi}}\,\theta^2\left(2 - \frac{(k-1)d_1}{kd_1+1} - \frac{(n-k-1)d_2}{(n-k)d_2+1}\right) \\ &+ \frac{2}{\sqrt{\pi}}\,\theta\left(\frac{\theta^*(k-1)}{k} - \frac{(n-k-1)(1-\theta^*)}{n-k} + \frac{(n-k-1)d_2}{(n-k)d_2+1} - 2\theta^*\right) \\ &+ \frac{2}{\sqrt{\pi}}\left[(n-k-1)\left(\frac{1-\theta*}{n-k} - \frac{d_2}{2((n-k)d_2+1)}\right) + \theta^*\right]\end{aligned}$$

We can rewrite the quadratic term of the squared error, so that the relation to the CRPS becomes clear: we expand $\frac{1}{k}$ by $kd_1+1$ and $\frac{1}{n-k}$ by $(n-k)d_2+1$. After re-arranging

terms, we have

$$\begin{aligned}
&\frac{1}{k} + \frac{1}{n-k} + \frac{k-1}{k(kd_1+1)} + \frac{n-k-1}{(n-k)((n-k)d_2+1)} \\
&= \frac{kd_1+1+k-1}{k(kd_1+1)} + \frac{(n-k)d_2+1)+n-k-1}{(n-k)((n-k)d_2+1)} \\
&= \frac{d_1+1}{kd_1+1} + \frac{d_2+1}{(n-k)d_2+1} \\
&= 1 - \frac{(k-1)d_1}{kd_1+1} + 1 - \frac{(n-k-1)d_2}{(n-k)d_2+1} \\
&= 2 - \frac{(k-1)d_1}{kd_1+1} - \frac{(n-k-1)d_2}{(n-k)d_2+1}.
\end{aligned}$$

Similar re-arrangements can be applied for the linear and constant part: First, notice that $\frac{\theta^*(k-1)}{k} = \theta^* - \frac{\theta^*}{k}$. Second, we expand the fractions for the linear part in the CRPS:

$$\begin{aligned}
&\frac{\theta^*(k-1)}{k} - \frac{(n-k-1)(1-\theta^*)}{n-k} + \frac{(n-k-1)d_2}{(n-k)d_2+1} - 2\theta^* \\
=& -\frac{\theta^*}{k} - \frac{(n-k-1)(1-\theta^*)((n-k)d_2+1) - (n-k-1)d_2(n-k)}{(n-k)((n-k)d_2+1)} \\
&+ \frac{\theta^*(n-k)((n-k)d_2+1)}{(n-k)((n-k)d_2+1)} \\
=& -\frac{\theta^*}{k} - \frac{(n-k-1)((n-k)d_2+1) - (n-k-1)\theta^*((n-k)d_2+1)}{(n-k)((n-k)d_2+1)} \\
&+ \frac{(n-k-1)d_2(n-k) + \theta^*(n-k)((n-k)d_2+1)}{(n-k)((n-k)d_2+1)} \\
=& -\frac{\theta^*}{k} - \frac{\theta^*((n-k)d_2+1) + (n-k-1)((n-k)d_2+1-(n-k)d_2)}{(n-k)((n-k)d_2+1)} \\
=& -\frac{\theta^*}{k} - \frac{\theta^*((n-k)d_2+1)}{(n-k)((n-k)d_2+1)} - \frac{n-k-1}{(n-k)((n-k)d_2+1)} \\
=& -\frac{\theta^*}{k} - \frac{\theta^*}{(n-k)((n-k)d_2+1)} - \frac{n-k-1}{(n-k)((n-k)d_2+1)}.
\end{aligned}$$

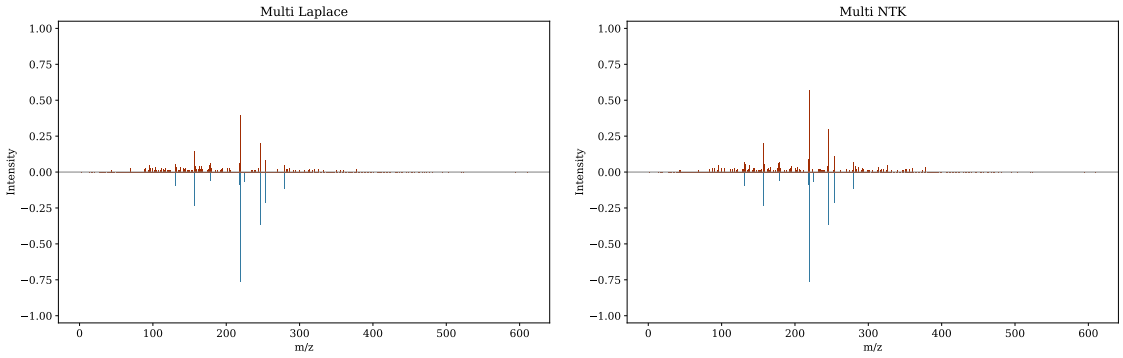
For the constant part, we follow similar steps:

$$
\begin{aligned}
& 2(n-k-1)\left(\frac{1-\theta*}{n-k}-\frac{d_2}{2((n-k)d_2+1)}\right)+\theta^* \\
&= \frac{2(n-k-1)(1-\theta^*)}{n-k}-\frac{(n-k-1)d_2}{(n-k)d_2+1}+2\theta^* \\
&= \frac{2(n-k-1)(1-\theta^*)((n-k)d_2+1)-(n-k)(n-k-1)d_2}{(n-k)((n-k)d_2+1)} \\
&\quad + \frac{2\theta^*(n-k)((n-k)d_2+1)}{(n-k)((n-k)d_2+1)} \\
&= \frac{2(n-k-1)((n-k)d_2+1)-2\theta^*(n-k-1)((n-k)d_2+1)}{(n-k)((n-k)d_2+1)} \\
&\quad + \frac{2\theta^*(n-k)((n-k)d_2+1)-(n-k)(n-k-1)d_2}{(n-k)((n-k)d_2+1)} \\
&= \frac{2(n-k-1)(n-k)d_2+2(n-k-1)-(n-k)(n-k-1)d_2+2\theta^*((n-k)d_2+1)}{(n-k)((n-k)d_2+1)} \\
&= \frac{(n-k-1)(n-k)d_2+2(n-k-1)+2\theta^*((n-k)d_2+1)}{(n-k)((n-k)d_2+1)} \\
&= \frac{(n-k)((n-k)d_2+1)}{(n-k)((n-k)d_2+1)}+\frac{(2\theta^*-1)((n-k)d_2+1)}{(n-k)((n-k)d_2+1)}+\frac{n-k-1}{(n-k)((n-k)d_2+1)} \\
&= 1+\frac{2\theta^*-1}{n-k}+\frac{n-k-1}{(n-k)((n-k)d_2+1)}.
\end{aligned}
$$

Hence, we have shown that $\mathbb{E}[\text{SE}] = \frac{1}{\sqrt{\pi}}\mathbb{E}[\text{CRPS}]$.

# C Appendix to Chapter 4: Efficient Prediction of Tandem Mass Spectra using the Neural Tangent Kernel

## C.1 Further Results




**(a)** Laplace **(b)** NTK

**Figure C.1: Multi Laplace & NTK mirror plots**. The blue mirrored spectrum is the true observation, the red-shaded spectrum is the respective prediction.

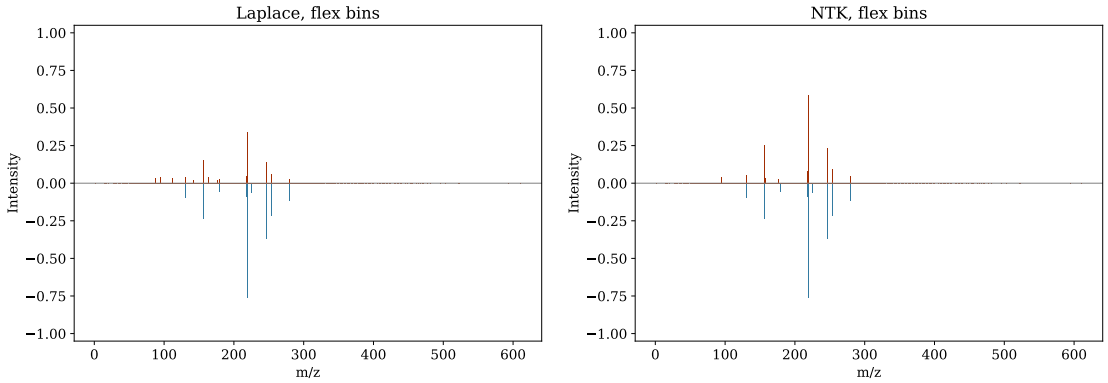



**Figure C.2: Mirror plot for simple, flex-bin Laplace & NTK models**

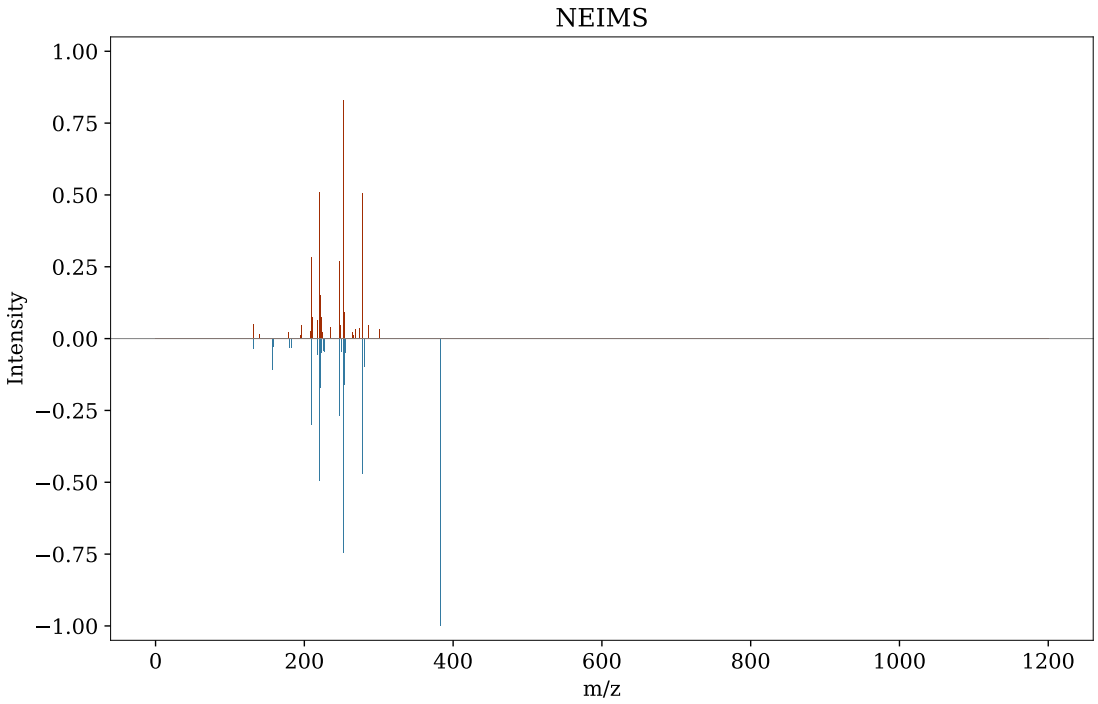



**Figure C.3: Mirror plot for NEIMS** with 2200 bins. The prediction exceeding intensity one may be due to the slightly different scaling scheme.

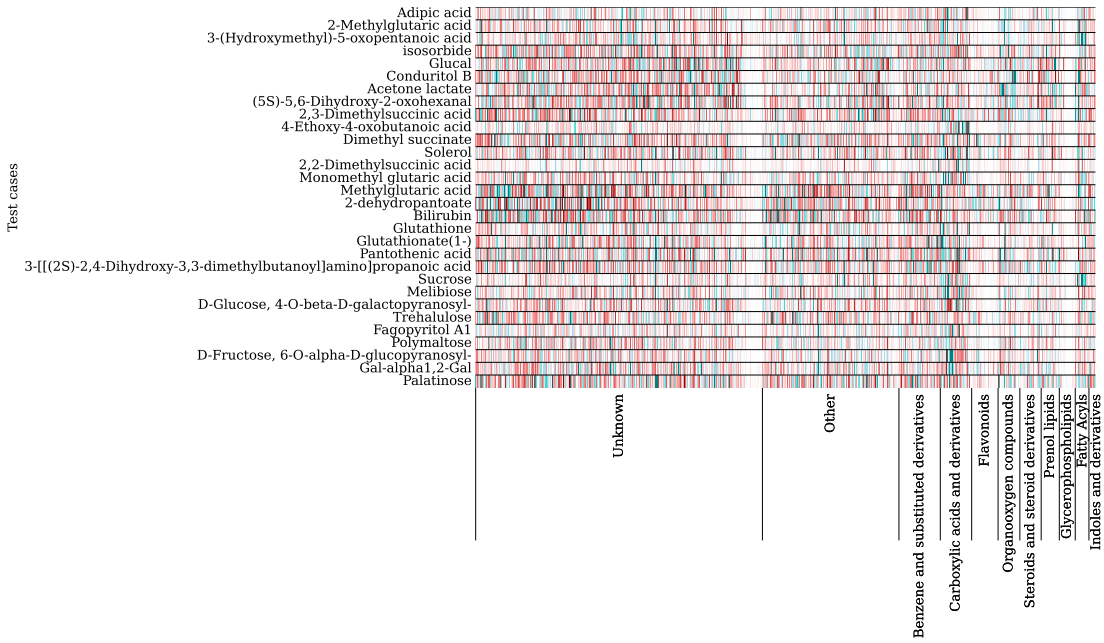



**Figure C.4: Smoother matrix for test predictions** of the simple Laplace with fixed binning. Each Row, separated by a black line, represents a test point, each column a training point, grouped by the respective HMDB-classes. Columns within each class are sorted by their first PCA dimension of their Morgan fingerprint. Negative weights are shown in red, positive weights are shown in blue and zero-weights are visualized as white.

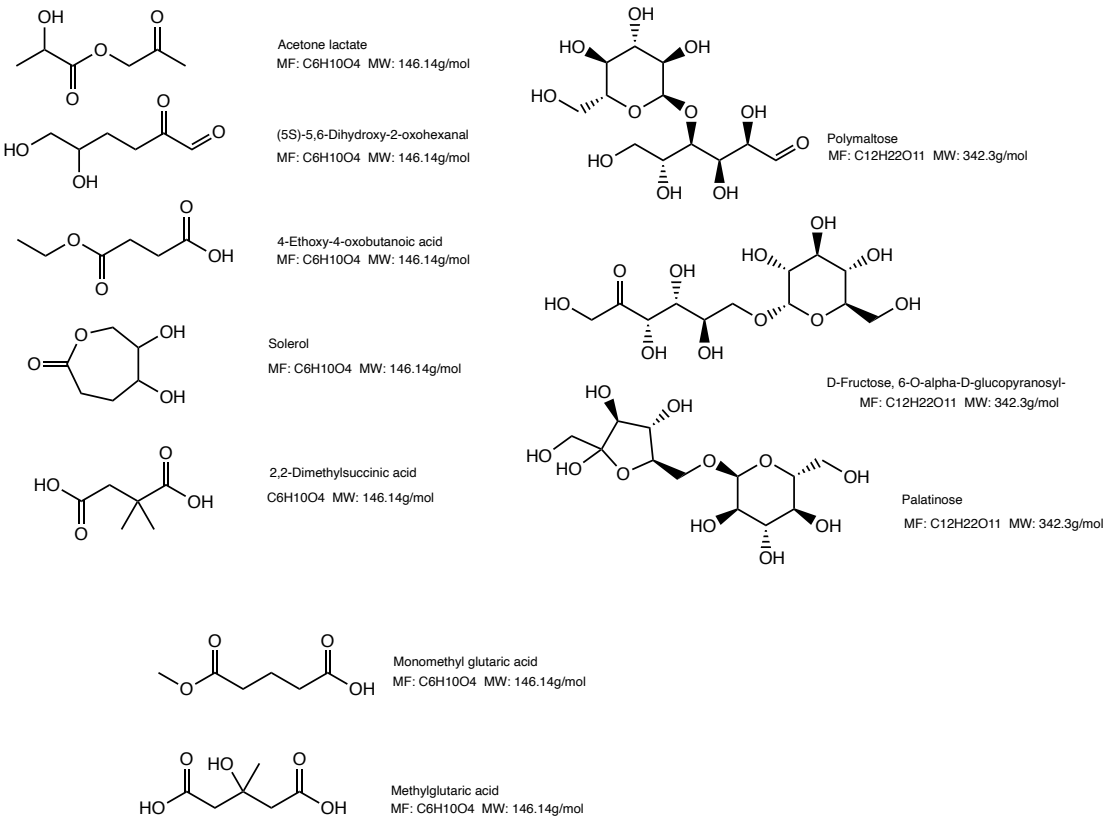



**Figure C.5: Chemical structures of exemplary test compounds in Figures 4.5 & C.4.**

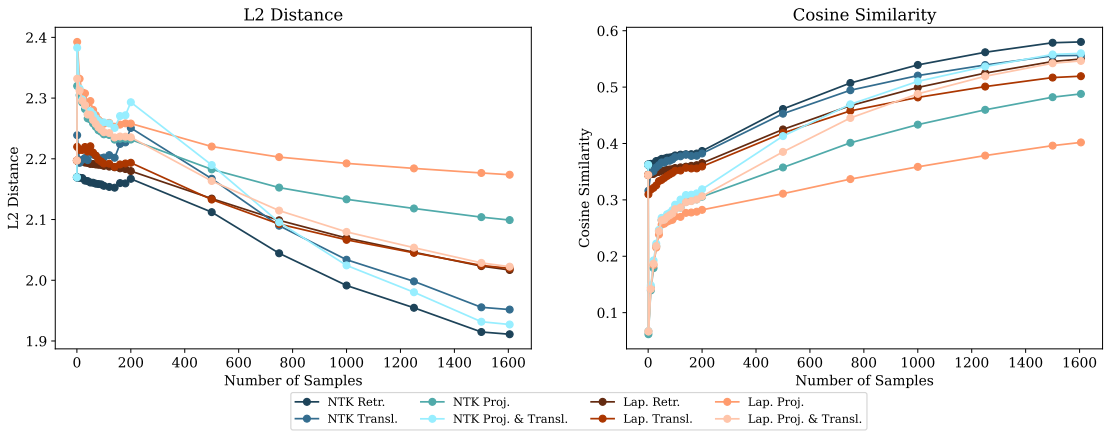



**Figure C.6: Transfer learning: L2 & Cosine Distance**. Laplace is abbreviated as 'Lap.', Projection as 'Proj.', Translation as 'Transl.' and Re-training as 'Retr.'.

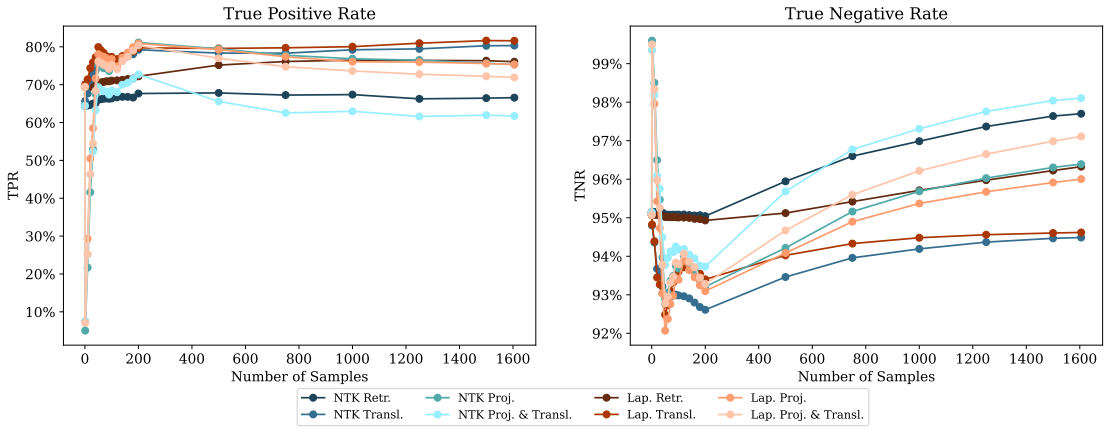



**Figure C.7: Transfer learning: TPR & TNR**. Laplace is abbreviated as 'Lap.', Projection as 'Proj.', Translation as 'Transl.' and Re-training as 'Retr.'.

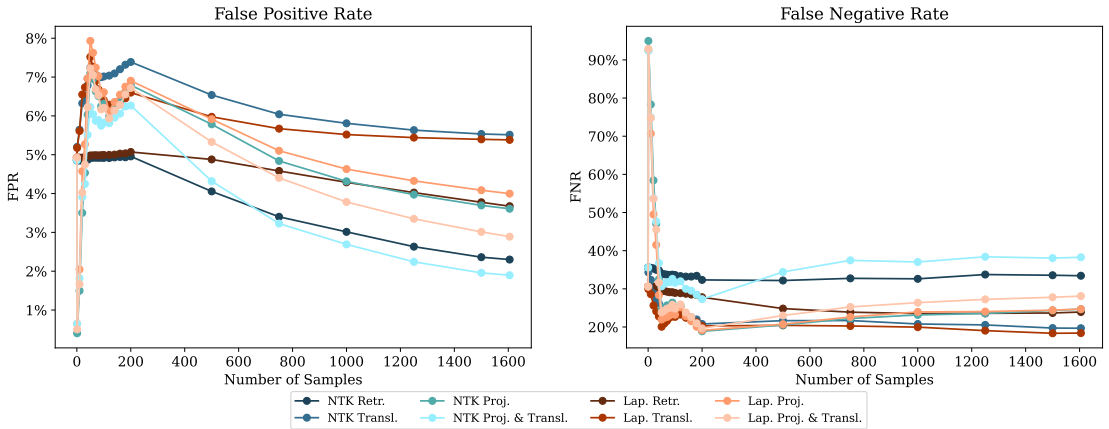



**Figure C.8: Transfer learning: FPR & FNR**. Laplace is abbreviated as 'Lap.', Projection as 'Proj.', Translation as 'Transl.' and Re-training as 'Retr.'.

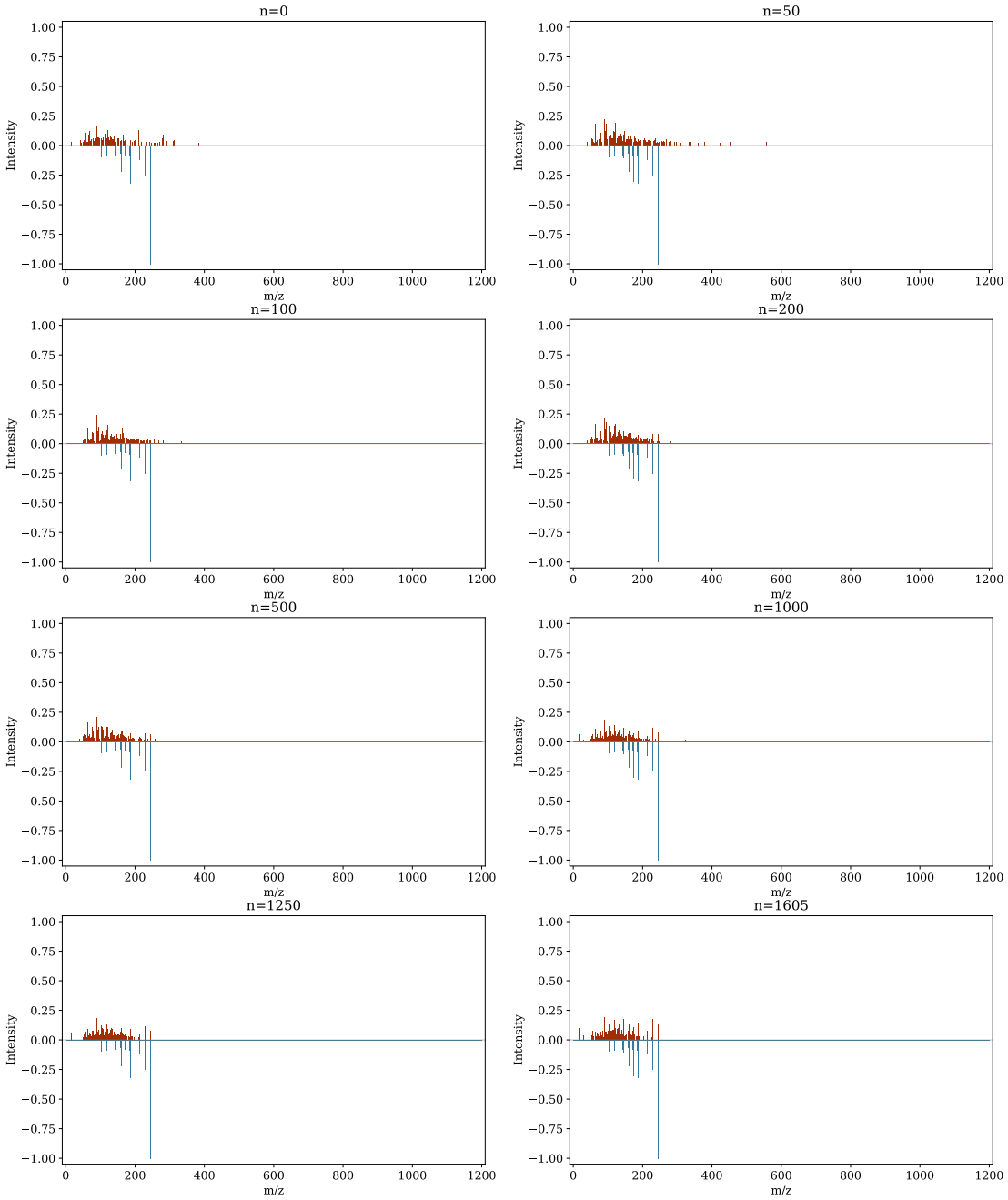



**Figure C.9: Projection evolution** of a randomly drawn test sample for the NTK. We show eight stages of the transfer-learning experiment with no fine-tuning applied, 50, 100, 200, 500, 1000, 1250 and 1605 samples available. True spectra are mirrored and shown in blue, predicted spectra are shown in red.

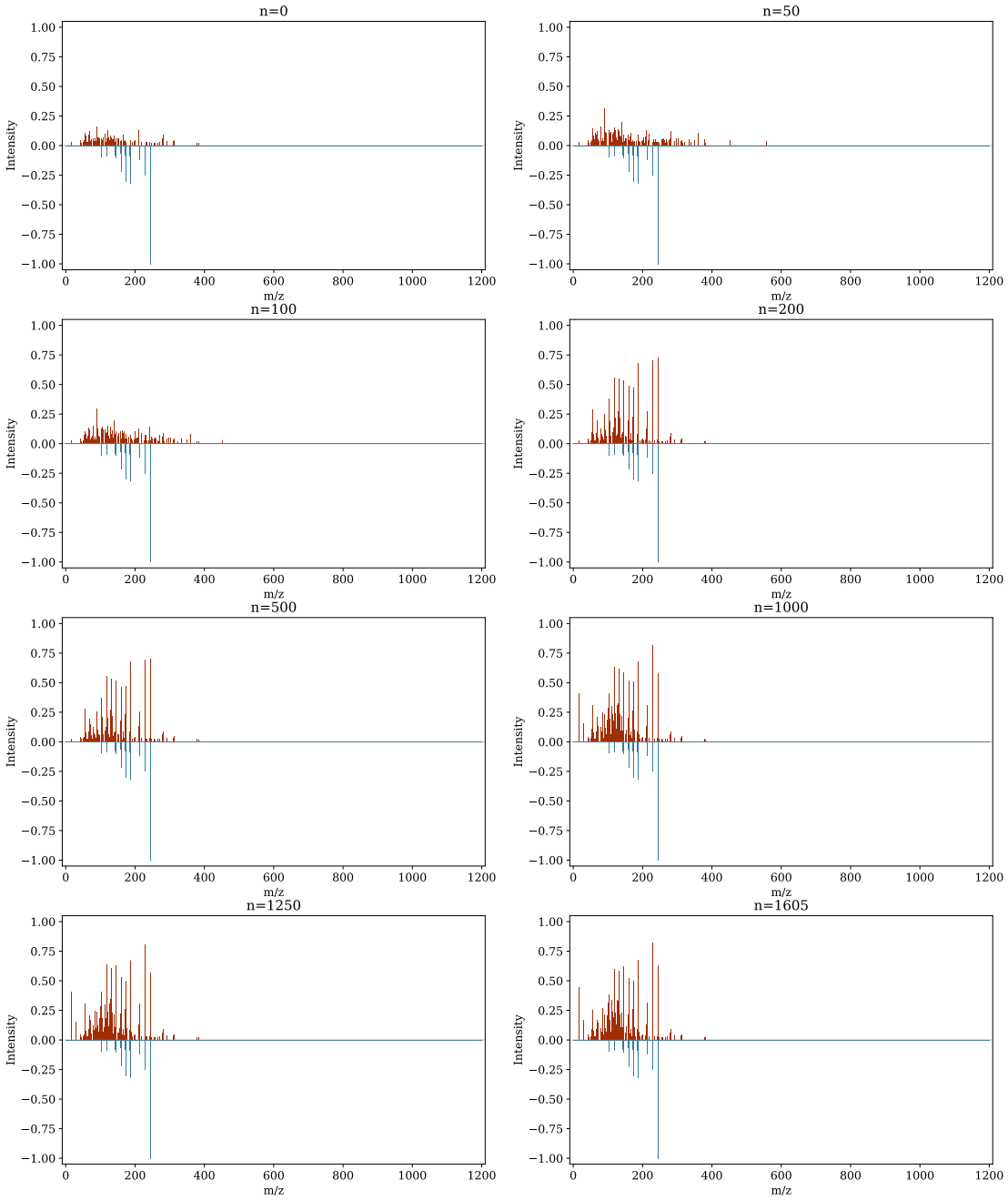



**Figure C.10: Translation evolution** of a randomly drawn test sample for the NTK. We show eight stages of the transfer-learning experiment with no fine-tuning applied, 50, 100, 200, 500, 1000, 1250 and 1605 samples available. True spectra are mirrored and shown in blue, predicted spectra are shown in red.

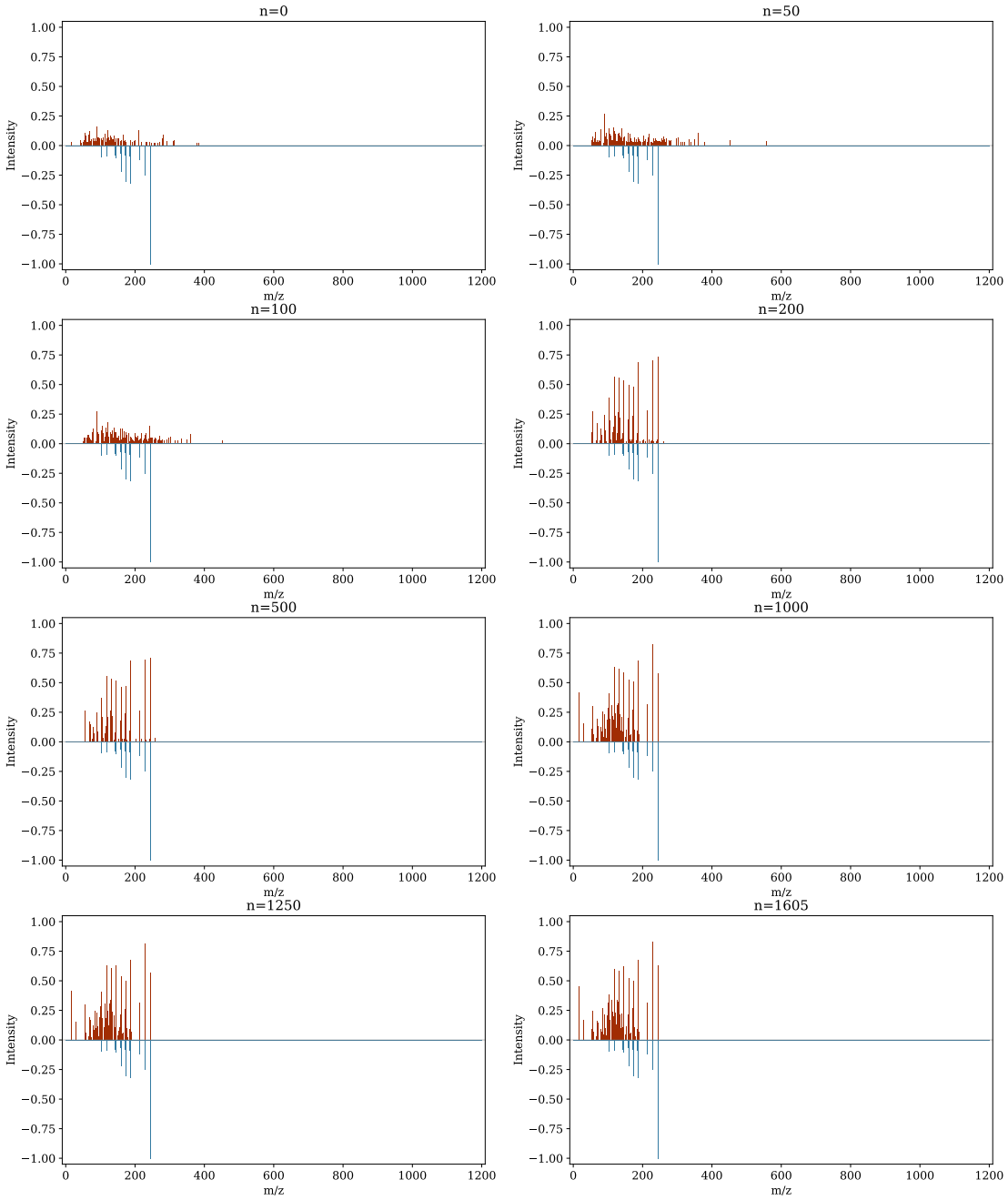



**Figure C.11: Projection & Translation evolution** of a randomly drawn test sample for the NTK. We show eight stages of the transfer-learning experiment with no fine-tuning applied, 50, 100, 200, 500, 1000, 1250 and 1605 samples available. True spectra are mirrored and shown in blue, predicted spectra are shown in red.

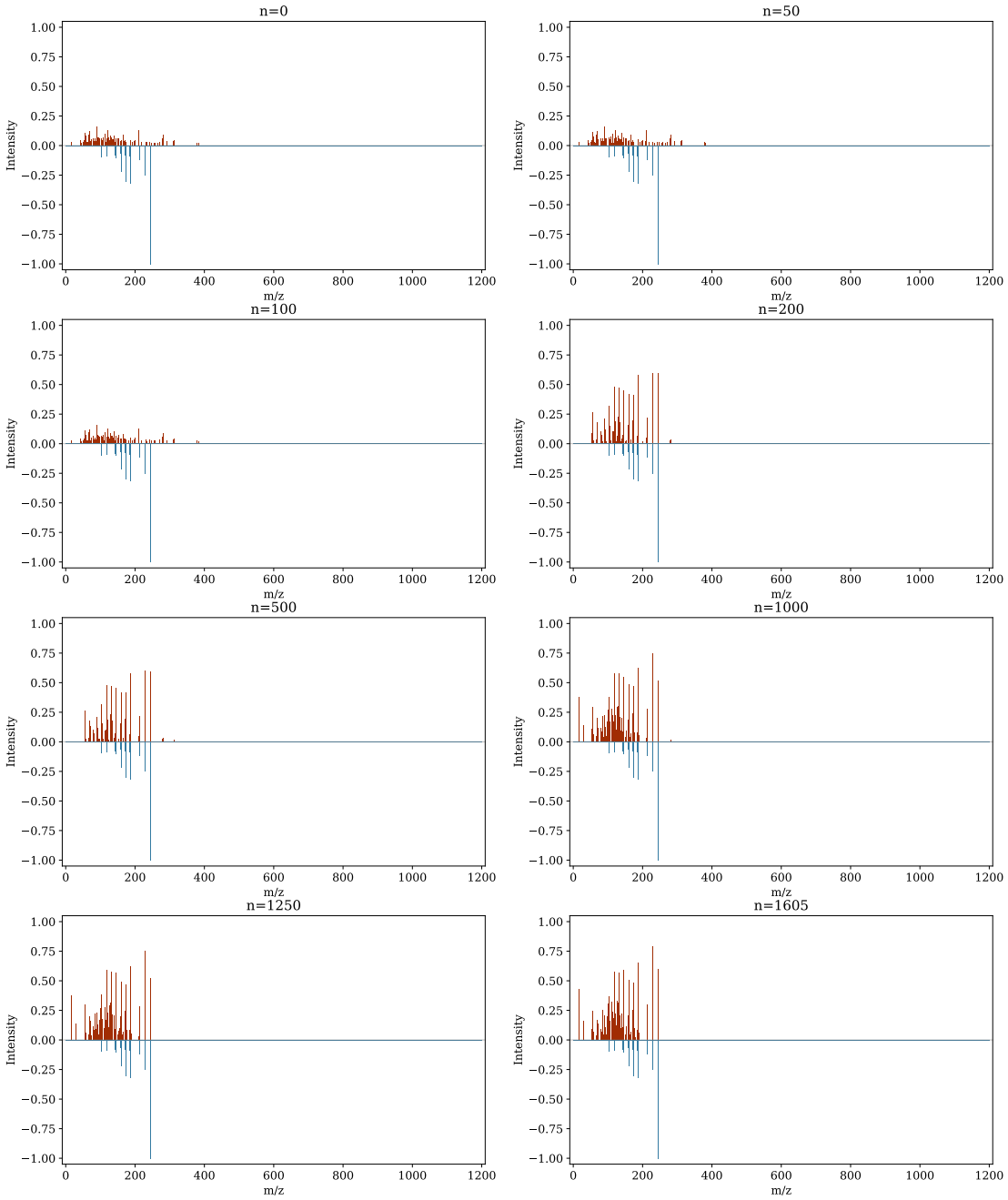



**Figure C.12: Re-training evolution** of a randomly drawn test sample for the NTK. We show eight stages of the transfer-learning experiment with no fine-tuning applied, 50, 100, 200, 500, 1000, 1250 and 1605 samples available. True spectra are mirrored and shown in blue, predicted spectra are shown in red.

# D Appendix to Chapter 5: Probabilistic Time Series Forecasting Challenge

## D.1 Additional Tables

**Table D.1: Listing of web resources** mentioned in the text. Last accessed on February 24, 2025.

| Reference | URL |
|---|---|
| URL1 | https://gitlab.kit.edu/nils.koster/ptsfc_results |
| URL2 | https://github.com/FK83/ptsfc_replication |
| URL3 | https://jobrac.shinyapps.io/ptsfc_viz/ |
| URL4 | https://de.finance.yahoo.com/ |
| URL5 | https://www.dwd.de/DE/wetter/wetterundklima_vorort/berlin-brandenburg/berlin_tempelhof/_node.html |
| URL6 | https://www.dwd.de/EN/ourservices/nwp_forecast_data/nwp_forecast_data.html |
| URL7 | https://en.wikipedia.org/wiki/List_of_Discworld_characters |
| URL8 | https://en.wikipedia.org/wiki/List_of_Brooklyn_Nine-Nine_characters |
| URL9 | https://en.wikipedia.org/wiki/List_of_Friends_and_Joey_characters |
| URL10 | https://en.wikipedia.org/wiki/List_of_Game_of_Thrones_characters |
| URL11 | https://en.wikipedia.org/wiki/List_of_Star_Wars_characters |
| URL12 | https://gitlab.kit.edu/nils.koster/ptsfc_results/-/tree/main/ptsfc_viz |
| URL13 | https://image-net.org/challenges/LSVRC/ |
| URL14 | https://s2s-ai-challenge.github.io/ |
| URL15 | https://forecasters.org/programs/research-awards/students/ |

**Table D.2:** Length of prediction intervals, on average over the sample period.

| | | Length (50% level) | | Length (95% level) | |
|---|---|---|---|---|---|
| Target | Horizon | Benchmark | Ensemble | Benchmark | Ensemble |
| DAX | 1 day | 1.1 | 1.3 | 5.1 | 4.4 |
| DAX | 2 day | 1.7 | 1.8 | 7.0 | 6.0 |
| DAX | 5 day | 2.0 | 2.2 | 8.7 | 7.5 |
| DAX | 6 day | 2.4 | 2.4 | 9.8 | 8.3 |
| DAX | 7 day | 2.6 | 2.6 | 10.9 | 9.1 |
| Temp. | 36 hour | 1.0 | 1.7 | 2.6 | 5.0 |
| Temp. | 48 hour | 1.1 | 1.7 | 2.8 | 4.9 |
| Temp. | 60 hour | 1.4 | 2.0 | 3.3 | 5.8 |
| Temp. | 72 hour | 1.7 | 2.0 | 4.8 | 5.9 |
| Temp. | 84 hour | 2.3 | 2.4 | 5.1 | 6.9 |
| Wind | 36 hour | 2.7 | 4.5 | 7.2 | 13.2 |
| Wind | 48 hour | 2.8 | 4.9 | 7.8 | 14.4 |
| Wind | 60 hour | 4.0 | 5.3 | 9.2 | 15.1 |
| Wind | 72 hour | 3.4 | 5.1 | 8.8 | 14.8 |
| Wind | 84 hour | 4.2 | 5.4 | 10.9 | 15.8 |

## D.2 Exploratory Analysis of Ranking Methodology

In Sections D.2.1 and D.2.2, we explore the informativeness and robustness of the ranking along two dimensions.

### D.2.1 Stability

Here we analyze the stability of the ranking. In particular, if the ranking merely reflected small or unsystematic differences in forecast performance, or would be driven by a small number of forecast dates, it would not serve its purposes well. In order to assess the ranking's stability, we resample many possible sample paths, each of which represents one hypothetical course of history like the one we observed in reality. We then compare the rankings obtained across many possible histories. More specifically, we proceed as follows: In each Monte Carlo iteration, we

- draw 14 DAX forecast dates (independent draws from 14 actual forecast dates, *with* replacement). Compute DAX score rankings based on the empirical scores for these dates.

**Table D.3:** Summary of forecaster ranks obtained across 10 000 simulated sample periods, generated as described in the text. The table focuses on five best forecasters out of 21 participants.

| Forecaster (Alias) | Share of Wins (in %) | Mean | Rank Quantiles 90% | 99% | Actual rank |
|---|---|---|---|---|---|
| Yoda | 48.5 | 1.8 | 3 | 5 | 1 |
| PhoebeBuffay | 31.8 | 2.3 | 4 | 6 | 2 |
| HotPie | 10.2 | 3.2 | 5 | 7 | 3 |
| Shaggydog | 3.1 | 4.1 | 6 | 8 | 4 |
| KyloRen | 1.2 | 5.1 | 7 | 9 | 5 |

- draw 13 weather forecast dates (independent draws from 13 actual forecast dates, *with* replacement). Compute temperature and wind score rankings based on the empirical scores for these dates. We use the same dates for temperature and wind in order to account for dependence of these variables.
- construct an overall forecast ranking from the rankings in the present iteration.

This simple simulation design can be viewed as investigating the robustness of the forecast ranking across individual weeks of the empirical sample period. Table D.3 summarizes the simulation results for the five best forecasters from the actual (empirical) ranking. The best-performing forecaster in the empirical sample (alias 'Yoda') also performs best across the simulations, in terms of highest share of wins, smallest mean rank, and smallest rank quantiles. Furthermore, one of either 'Yoda' or 'Phoebe Buffay' wins in more than 80% of the simulated time paths. In addition to the results from Table D.3, we find that ten out of 21 forecasters never win across the 10 000 Monte Carlo iterations. We view these overall results as reassuring evidence on the ranking's signal-to-noise ratio.

### D.2.2 Comparison to Alternative Ranking Variants

As discussed in Section 5.2.3 in Chapter 5, the ranking used in the course sums up a forecaster's ranks across 15 sub-categories (three target variables times five horizons). The forecaster's final rank is then determined by the resulting rank sum. This pragmatic approach is motivated by the fact that the scores for the sub-categories are on different scales. For example, one would generally expect higher scores at longer forecast horizons, due to less predictability. Here we compare this original ranking method ('Version 1') to two alternatives:

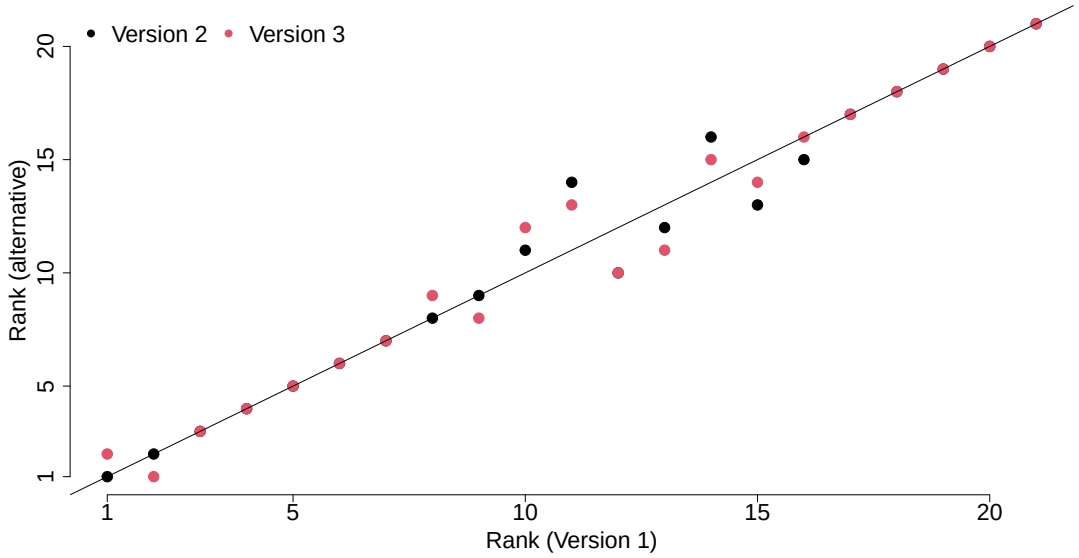



**Figure D.1:** Comparison of forecaster ranks across three different methods. See text for details.

- Version 2: Compute a forecaster's skill score for each of the 15 sub-categories. Determine final rank from the sum of skill scores.
- Version 3: Determine a forecaster's final rank from the sum of raw scores across all forecast cases.

Version 2 is similar in spirit to Version 1, in that it transforms the scores (by computing skill scores) to make them comparable across sub-categories. By contrast, Version 3 does not account for possible scale differences. While this approach may be viewed as a drawback from a pragmatic perspective, its theoretical properties (in terms of being proper) are clearer and more attractive than for Versions 1 and 2.

Figure D.1 summarizes the comparison. Naturally, the three rankings vary to some degree; for example, the second-best forecaster according to Version 1 ranks first according to Version 3 and second according to Version 2. However, these variations seem minor overall, and the correlation between any pair of rankings is at least 98.4%.

# Bibliography


Abbas, A. E. and R. A. Howard (2015): *Foundations of decision analysis*, Pearson.

Alemdar, H., V. Leroy, A. Prost-Boucle, and F. Pétrot (2016): "Ternary neural networks for resource-efficient AI applications," Preprint, arxiv:1609.00222.

Altig, D., J. M. Barrero, N. Bloom, S. J. Davis, B. Meyer, and N. Parker (2022): "Surveying business uncertainty," *Journal of Econometrics*, 231, 282–303.

Álvarez, M. A., L. Rosasco, and N. D. Lawrence (2012): "Kernels for vector-valued functions: A review," *Foundations and Trends in Machine Learning*, 4, 195–266.

Aronszajn, N. (1950): "Theory of reproducing kernels," *Transactions of the American Mathematical Society*, 68, 337.

Beck, E., D. Kozbur, and M. Wolf (2023): "Hedging forecast combinations with an application to the random forest," Preprint, arxiv:2308.15384.

Belkin, M., D. Hsu, S. Ma, and S. Mandal (2018a): "Reconciling modern machine learning practice and the bias-variance trade-off," *Proceedings of the National Academy of Sciences of the United States of America*, 116, 15849–15854.

Belkin, M., S. Ma, and S. Mandal (2018b): "To understand deep learning we need to understand kernel learning," in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, vol. 80, 541–549.

Bengio, Y., N. Léonard, and A. Courville (2013): "Estimating or propagating gradients through stochastic neurons for conditional computation," Preprint, arxiv:1308.3432.

Bi, K., L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian (2023): "Accurate medium-range global weather forecasting with 3D neural networks," *Nature*, 619, 533–538.

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

Bietti, A. and J. Mairal (2019): "On the inductive bias of neural tangent kernels," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Curran Associates Inc., 12893–12904.

Bittremieux, W., R. Schmid, F. Huber, J. J. J. van der Hooft, M. Wang, and P. C. Dorrestein (2022): "Comparison of cosine, modified cosine, and neutral loss based spectrum alignment for discovery of structurally related molecules," *Journal of the American Society for Mass Spectrometry*, 33, 1733–1744.

Bojer, C. S. and J. P. Meldgaard (2021): "Kaggle forecasting competitions: An overlooked learning opportunity," *International Journal of Forecasting*, 37, 587–603.

Bowles, C., R. Friz, V. Genre, G. Kenny, A. Meyler, and T. Rautanen (2007): "The ECB survey of professional forecasters (SPF) – A review after eight years' experience," ECB occasional paper no. 59.

Bracher, J., N. Koster, F. Krüger, and S. Lerch (2024): "Learning to forecast: The probabilistic time series forecasting challenge," *The American Statistician*, 78, 115–127.

Bracher, J., N. Koster, F. Krüger, S. Lerch, and D. Wolffram (2021a): "Preregistration: probabilistic time series forecasting challenge, Karlsruhe Institute of Technology," URL: https://osf.io/npwcv/ (last accessed: February 23, 2025).

Bracher, J., D. Wolffram, J. Deuschel, K. Görgen, J. Ketterer, A. Ullrich, S. Abbott, et al. (2021b): "A pre-registered short-term forecasting study of COVID-19 in Germany and Poland during the second wave," *Nature Communications*, 12, 5173.

Brasoveanu, A., M. Moodie, and R. Agrawal (2020): "Textual evidence for the perfunctoriness of independent medical reviews," *CEUR Workshop Proceedings*, 2657, 1–9.

Braun, A., M. Kohler, S. Langer, and H. Walk (2024): "Convergence rates for shallow neural networks learned by gradient descent," *Bernoulli*, 30, 475–502.

Breiman, L. (2001): "Random forests," *Machine Learning*, 45, 5–32.

Cevid, D., L. Michel, J. Näf, P. Bühlmann, and N. Meinshausen (2022): "Distributional random forests: Heterogeneity adjustment and multivariate distributional regression," *Journal of Machine Learning Research*, 23, 1–79.

Chen, L. and S. Xu (2021): "Deep neural tangent kernel and laplace kernel have the same RKHS," in *International Conference on Learning Representations*, OpenReview.net.

Cheng, H., M. Zhang, and J. Q. Shi (2024): "A survey on deep neural network pruning: taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46, 10558–10578.

Chijiwa, D., S. Yamaguchi, Y. Ida, K. Umakoshi, and T. Inoue (2021): "Pruning randomly initialized neural networks with iterative randomization," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, Curran Associates Inc., 4503–4513.

Clevert, D., T. Unterthiner, and S. Hochreiter (2016): "Fast and accurate deep network learning by exponential linear units (ELUs)," in *International Conference on Learning Representations*, OpenReview.net.

Coulombe, P. G., M. Goebel, and K. Klieber (2024): "Dual interpretation of machine learning forecasts," Preprint, arxiv:2412.13076.

Courbariaux, M., Y. Bengio, and J.-P. David (2015): "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proceedings of the 29th International Conference on Neural Information Processing Systems*, Curran Associates Inc., 3123—3131.

Cramer, E. Y., E. L. Ray, V. K. Lopez, J. Bracher, A. Brennen, A. J. C. Rivadeneira, A. Gerding, et al. (2022): "Evaluation of individual and ensemble probabilistic forecasts of COVID-19 mortality in the United States," *Proceedings of the National Academy of Sciences*, 119, e2113561119.

Croushore, D. (2006): "Forecasting with real-time macroeconomic data," in *Handbook of Economic Forecasting*, Elsevier, vol. 1, 961–982.

Croushore, D. and T. Stark (2019): "Fifty years of the Survey of Professional Forecasters," *Federal Reserve Bank of Philadelphia Economic Insights*, 2019Q4, 1–11.

da Silva, R. R., P. C. Dorrestein, and R. A. Quinn (2015): "Illuminating the dark matter in metabolomics," *Proceedings of the National Academy of Sciences*, 112, 12549–12550.

de Jorge, P., A. Sanyal, H. Behl, P. Torr, G. Rogez, and P. K. Dokania (2021): "Progressive skeletonization: Trimming more fat from a network at initialization," in *International Conference on Learning Representations*, OpenReview.net.

Deng, X. and Z. Zhang (2020): "An embarrassingly simple approach to training ternary weight networks," Preprint, arxiv:2011.00580.

Denning, P. J. and T. G. Lewis (2016): "Exponential laws of computing growth," *Communications of the ACM*, 60, 54–65.

Dicheva, D., C. Dichev, G. Agre, and G. Angelova (2015): "Gamification in education: A systematic mapping study," *Journal of Educational Technology & Society*, 18, 75–88.

Diffenderfer, J. and B. Kailkhura (2021): "Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network," in *International Conference on Learning Representations*, OpenReview.net.

DIW (2022): "SOEP-Übungsdatensatz, Daten der Jahre 2015-2019," Data set, freely available, URL: https://doi.org/10.5684/soep.practice.v36 (last accessed: February 23, 2025).

Dührkop, K., M. Fleischauer, M. Ludwig, A. A. Aksenov, A. V. Melnik, M. Meusel, P. C. Dorrestein, et al. (2019): "SIRIUS 4: A rapid tool for turning tandem mass spectra into metabolite structure information," *Nature Methods*, 16, 299–302.

Elsken, T., J. H. Metzen, and F. Hutter (2019): "Neural architecture search: A survey," *Journal of Machine Learning Research*, 20, 1997–2017.

Emmert-Streib, F., M. Dehmer, and Y. Shi (2016): "Fifty years of graph matching, network alignment and network comparison," *Information Sciences*, 346–347, 180–197.

Evci, U., T. Gale, J. Menick, P. S. Castro, and E. Elsen (2020): "Rigging the lottery: Making all tickets winners," in *Proceedings of the 37th International Conference on Machine Learning*, JMLR.org.

Frankle, J. and M. Carbin (2019): "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, OpenReview.net.

Frankle, J., G. K. Dziugaite, D. M. Roy, and M. Carbin (2019): "Stabilizing the lottery ticket hypothesis," Preprint, arxiv:1903.01611.

Friedman, J. H. (2001): "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, 29, 1189–1232.

Friedman, J. H. and B. E. Popescu (2008): "Predictive learning via rule ensembles," *Annals of Applied Statistics*, 2, 916–954.

Frongillo, R., R. Gomez, A. Thilagar, and B. Waggoner (2021): "Efficient competitions and online learning with strategic forecasters," in *Proceedings of the 22nd ACM Conference on Economics and Computation*, 479–496.

Ghorbani, A., A. Abid, and J. Zou (2019): "Interpretation of neural networks Is fragile," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 3681–3688.

Glorot, X. and Y. Bengio (2010): "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, PMLR, vol. 9, 249–256.

Gneiting, T. and M. Katzfuss (2014): "Probabilistic forecasting," *Annual Review of Statistics and Its Application*, 1, 125–151.

Gneiting, T. and A. E. Raftery (2005): "Weather forecasting with ensemble methods," *Science*, 310, 248–249.

——— (2007): "Strictly proper scoring rules, prediction, and estimation," *Journal of the American Statistical Association*, 102, 359–378.

Gneiting, T., A. E. Raftery, A. H. Westveld III, and T. Goldman (2005): "Calibrated probabilistic forecasting using ensemble model output statistics and minimum CRPS estimation," *Monthly Weather Review*, 133, 1098–1118.

Gneiting, T. and J. Resin (2023): "Regression diagnostics meets forecast evaluation: Conditional calibration, reliability diagrams, and coefficient of determination," *Electronic Journal of Statistics*, 17, 3226–3286.

Gneiting, T., D. Wolffram, J. Resin, K. Kraus, J. Bracher, T. Dimitriadis, V. Hagenmeyer, et al. (2023): "Model diagnostics and forecast evaluation for quantiles," *Annual Review of Statistics and Its Application*, 10.

Goebel, J., M. M. Grabka, S. Liebig, M. Kroh, D. Richter, C. Schröder, and J. Schupp (2019): "The German socio-economic panel (SOEP)," *Jahrbücher für Nationalökonomie und Statistik*, 239, 345–360.

Goldman, S., J. Bradshaw, J. Xin, and C. W. Coley (2023a): "Prefix-tree decoding for predicting mass spectra from molecules," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Curran Associates Inc., 48548–48572.

Goldman, S., J. Li, and C. W. Coley (2024): "Generating molecular fragmentation graphs with autoregressive neural networks," *Analytical Chemistry*, 96, 3419–3428.

Goldman, S., J. Wohlwend, M. Stražar, G. Haroush, R. J. Xavier, and C. W. Coley (2023b): "Annotating metabolite mass spectra with domain-inspired chemical formula transformers," *Nature Machine Intelligence*, 5, 965–979.

Grinsztajn, L., E. Oyallon, and G. Varoquaux (2022): "Why do tree-based models still outperform deep learning on typical tabular data?" in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Curran Associates Inc., 507–520.

Haddouchi, M. and A. Berrado (2019): "A survey of methods and tools used for interpreting random forest," in *1st International Conference on Smart Systems and Data Science (ICSSD)*, 1–6.

Hamari, J., J. Koivisto, and H. Sarsa (2014): "Does gamification work? – A literature review of empirical studies on gamification," in *47th Hawaii International Conference on System Sciences*, 3025–3034.

Han, S., J. Pool, J. Tran, and W. J. Dally (2015): "Learning both weights and connections for efficient neural networks," in *Proceedings of the 29th International Conference on Neural Information Processing Systems*, Curran Associates Inc., 1135—-1143.

Hastie, T., R. Tibshirani, and J. Friedman (2009): *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, Springer, 2 ed.

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

——— (2016): "Deep residual learning for image recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, 770–778.

Hidayetoglu, M., C. Pearson, V. S. Mailthody, E. Ebrahimi, J. Xiong, R. Nagi, and W. M. Hwu (2020): "At-scale sparse deep neural network inference with efficient GPU implementation," *IEEE High Performance Extreme Computing Conference, HPEC*, 1–7.

Hinton, G. (2012): "Neural networks for machine learning. Coursera, video lectures: Lecture 9c," URL: https://www.cs.toronto.edu/~hinton/coursera/lecture9/lec9c.mp4 (last accessed: February 23, 2025).

Hollmann, N., S. Müller, L. Purucker, A. Krishnakumar, M. Körfer, S. B. Hoo, R. T. Schirrmeister, and F. Hutter (2025): "Accurate predictions on small data with a tabular foundation model," *Nature*, 637, 319–326.

Hong, T., P. Pinson, S. Fan, H. Zareipour, A. Troccoli, and R. J. Hyndman (2016): "Probabilistic energy forecasting: Global Energy Forecasting Competition 2014 and beyond," *International Journal of Forecasting*, 32, 896–913.

Hron, J., Y. Bahri, J. Sohl-Dickstein, and R. Novak (2020): "Infinite attention: NNGP and NTK for deep attention networks," in *Proceedings of the 37th International Conference on Machine Learning*, PMLR, vol. 119, 4376–4386.

Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio (2016): "Binarized neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, Curran Associates Inc., 4114—4122.

Huber, F., S. van der Burg, J. J. J. van der Hooft, and L. Ridder (2021): "MS2DeepScore: A novel deep learning similarity measure to compare tandem mass spectra," *Journal of Cheminformatics*, 13.

Hwang, J., P. Orenstein, J. Cohen, K. Pfeiffer, and L. Mackey (2019): "Improving subseasonal forecasting in the western US with machine learning," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2325–2335.

Hyndman, R. J. (2020): "A brief history of forecasting competitions," *International Journal of Forecasting*, 36, 7–14.

Ioffe, S. and C. Szegedy (2015): "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *32nd International Conference on Machine Learning, ICML 2015*, 1, 448–456.

Jacot, A., F. Gabriel, and C. Hongler (2018): "Neural tangent kernel: Convergence and generalization in neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, Curran Associates Inc., 8580––8589.

Janowsky, S. A. (1989): "Pruning versus clipping in neural networks," *Physical Review A*, 39, 6600–6603.

Jordan, A., F. Krüger, and S. Lerch (2019): "Evaluating probabilistic forecasts with scoringRules," *Journal of Statistical Software*, 90, 1–37.

Jumper, J., R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, et al. (2021): "Highly accurate protein structure prediction with AlphaFold," *Nature*, 596, 583–589.

Kimeldorf, G. S. and G. Wahba (1970): "A correspondence between bayesian estimation on stochastic processes and smoothing by splines," *The Annals of Mathematical Statistics*, 41, 495–502.

Kingma, D. P., T. Salimans, and M. Welling (2015): "Variational dropout and the local reparameterization trick," in *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, Curran Associates Inc., 2575—-2583.

Kirkpatrick, P. and C. Ellis (2004): "Chemical space," *Nature*, 432, 823–823.

Kochkov, D., J. Yuval, I. Langmore, P. Norgaard, J. Smith, G. Mooers, M. Klöwer, et al. (2024): "Neural general circulation models for weather and climate," *Nature*, 632, 1060–1066.

Kohler, M. and A. Krzyżak (2021): "Over-parametrized deep neural networks minimizing the empirical risk do not generalize well," *Bernoulli*, 27, 2564–2597.

Koster, N., O. Grothe, and A. Rettinger (2022): "Signing the supermask: Keep, hide, invert," in *International Conference on Learning Representations*, OpenReview.net.

Krizhevsky, A. (2009): "Learning multiple layers of features from tiny images," Technical report, URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (last accessed: February 23, 2025).

Krüger, F., T. E. Clark, and F. Ravazzolo (2017): "Using entropic tilting to combine BVAR forecasts with external nowcasts," *Journal of Business & Economic Statistics*, 35, 470–485.

Krüger, F. and J. F. Ziegel (2021): "Generic conditions for forecast dominance," *Journal of Business & Economic Statistics*, 39, 972–983.

Laio, F. and S. Tamea (2007): "Verification tools for probabilistic forecasts of continuous hydrological variables," *Hydrology and Earth System Sciences*, 11, 1267–1277.

LeCun, Y., C. Cortes, and C. J. Burges (2010): "MNIST handwritten digit database," ATT Labs. URL: http://yann.lecun.com/exdb/mnist (last accessed: February 23, 2025).

LeCun, Y., J. S. Denker, and S. A. Solla (1989): "Optimal brain damage," in *Proceedings of the 3rd International Conference on Neural Information Processing Systems*, Morgan Kaufmann, 598—-605.

Lee, J., J. Sohl-dickstein, J. Pennington, R. Novak, S. Schoenholz, and Y. Bahri (2018): "Deep neural networks as gaussian processes," in *International Conference on Learning Representations*, OpenReview.net.

Li, F., B. Zhang, and B. Liu (2016): "Ternary weight networks," Preprint, arxiv:1605.04711.

Li, H., A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf (2017): "Pruning filters for efficient ConvNets," in *International Conference on Learning Representations*, OpenReview.net.

Li, Y., T. Kind, J. Folz, A. Vaniya, S. S. Mehta, and O. Fiehn (2021): "Spectral entropy outperforms MS/MS dot product similarity for small-molecule compound identification," *Nature Methods*, 18, 1524–1531.

Liaw, A. and M. Wiener (2002): "Classification and regression by randomForest," *R News*, 2, 18–22.

Lin, Y. and Y. Jeon (2006): "Random forests and adaptive nearest neighbors," *Journal of the American Statistical Association*, 101, 578–590.

Liu, C., L. Zhu, and M. Belkin (2020): "On the linearity of large non-linear models: When and why the tangent kernel is constant," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Curran Associates Inc., 15954–15964.

Liu, S., T. Chen, X. Chen, Z. Atashgahi, L. Yin, H. Kou, L. Shen, et al. (2021): "Sparse training via boosting pruning plasticity with neuroregeneration," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, Curran Associates Inc., 9908–9922.

Lloyd-Price, J., C. Arze, A. N. Ananthakrishnan, M. Schirmer, J. Avila-Pacheco, T. W. Poon, E. Andrews, et al. (2019): "Multi-omics of the gut microbial ecosystem in inflammatory bowel diseases," *Nature*, 569, 655–662.

Lundberg, S. M. and S.-I. Lee (2017): "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Curran Associates Inc., 4768–4777.

Mahendran, A. and A. Vedaldi (2015): "Understanding deep image representations by inverting them," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, 5188–5196.

Makridakis, S., C. Fry, F. Petropoulos, and E. Spiliotis (2022): "The future of forecasting competitions: Design attributes and principles," *INFORMS Journal on Data Science*, 1, 96–113.

Marissen, R., M. S. Varunjikar, J. F. J. Laros, J. D. Rasinger, B. A. Neely, and M. Palmblad (2023): "compareMS2 2.0: An improved software for comparing tandem mass spectrometry datasets," *Journal of Proteome Research*, 22, 514–519.

Matheson, J. E. and R. L. Winkler (1976): "Scoring rules for continuous probability distributions," *Management Science*, 22, 1087–1096.

Meinshausen, N. (2006): "Quantile regression forests," *Journal of Machine Learning Research*, 7, 983–999.

——— (2010): "Node harvest," *Annals of Applied Statistics*, 4, 2049–2072.

Messner, J. W., G. J. Mayr, and A. Zeileis (2016): "Heteroscedastic censored and truncated regression with crch," *R Journal*, 8, 173–181.

Mocanu, D. C., E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta (2018): "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Communications*, 9.

Molchanov, P., A. Mallya, S. Tyree, I. Frosio, and J. Kautz (2019): "Importance Estimation for Neural Network Pruning," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, 11256–11264.

Molnar, C. (2022): *Interpretable machine learning: A guide for making black box models explainable*, 2 ed.

Moosavi-Dezfooli, S.-M., A. Fawzi, O. Fawzi, and P. Frossard (2017): "Universal adversarial perturbations," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, 86–94.

Morgan, H. L. (1965): "The generation of a unique machine description for chemical structures – A technique developed at chemical abstracts service," *Journal of Chemical Documentation*, 5, 107–113.

Murphy, M., S. Jegelka, E. Fraenkel, T. Kind, D. Healey, and T. Butler (2023): "Efficiently predicting high resolution mass spectra with graph neural networks," in *Proceedings of the 40th International Conference on Machine Learning*, PMLR, vol. 202, 25549–25562.

Nan, F., J. Wang, and V. Saligrama (2016): "Pruning random forests for prediction on a budget," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, Curran Associates Inc., 2342–2350.

Neal, R. M. (1996): *Priors for infinite networks*, Springer, 29–53.

Nordhaus, W. D. (2007): "Two centuries of productivity growin computing," *The Journal of Economic History*, 67, 128–159.

Novak, R., L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz (2020): "Neural tangents: Fast and easy infinite neural networks in Python," in *International Conference on Learning Representations*, OpenReview.net.

OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, et al. (2023): "GPT-4 technical report," Preprint, arxiv:2303.08774.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. (2011): "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, 12, 2825–2830.

Petropoulos, F., D. Apiletti, V. Assimakopoulos, M. Z. Babai, D. K. Barrow, S. Ben Taieb, C. Bergmeir, et al. (2022): "Forecasting: Theory and practice," *International Journal of Forecasting*, 38, 705–871.

Pfeifer, P. E., Y. Grushka-Cockayne, and K. C. Lichtendahl Jr (2014): "The promise of prediction contests," *The American Statistician*, 68, 264–270.

Plotly Technologies Inc. (2015): *plotly: Collaborative data science*, URL: https://plot.ly (last accessed: February 23, 2025).

Pohle, M.-O. (2020): "The Murphy decomposition and the calibration-resolution principle: A new perspective on forecast evaluation," Preprint, arxiv:2005.01835.

Probst, P., M. N. Wright, and A.-L. Boulesteix (2019): "Hyperparameters and tuning strategies for random forest," *WIREs Data Mining and Knowledge Discovery*, 9, e1301.

Qiang, H., F. Wang, W. Lu, X. Xing, H. Kim, S. A. Merette, L. B. Ayres, et al. (2024): "Language model-guided anticipation and discovery of unknown metabolites," Preprint, biorxiv:2024.11.13.623458.

R Core Team (2022): *R: A language and environment for statistical computing*, R Foundation for Statistical Computing.

Radhakrishnan, A., D. Beaglehole, P. Pandit, and M. Belkin (2024): "Mechanism for feature learning in neural networks and backpropagation-free machine learning models," *Science*, 383, 1461–1467.

Radhakrishnan, A., M. Ruiz Luyten, N. Prasad, and C. Uhler (2023): "Transfer learning with kernel methods," *Nature Communications*, 14, 5570.

Raftery, A. E. (2016): "Use and communication of probabilistic forecasts," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 9, 397–410.

Rahimi, A. and B. Recht (2007): "Random features for large-scale kernel machines," in *Proceedings of the 21st International Conference on Neural Information Processing Systems*, Curran Associates Inc., 1177–1184.

Ramanujan, V., M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari (2020): "What's hidden in a randomly weighted neural network?" in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society.

Rasmussen, C. E. and C. K. I. Williams (2005): *Gaussian processes for machine learning*, MIT Press.

Rasp, S. and S. Lerch (2018): "Neural networks for postprocessing ensemble weather forecasts," *Monthly Weather Review*, 146, 3885–3900.

Ridder, L., J. J. J. van der Hooft, and S. Verhoeven (2014): "Automatic compound annotation from mass spectrometry data using MAGMa," *Mass Spectrometry*, 3.

Roje, B., B. Zhang, E. Mastrorilli, A. Kovačić, L. Sušak, I. Ljubenkov, E. Ćosić, et al. (2024): "Gut microbiota carcinogen metabolism causes distal tissue tumours," *Nature*, 632, 1137–1144.

RStudio, Inc (2013): *shiny: Web application framework for R*, URL: https://shiny.posit.co/ (last accessed: February 23, 2025).

Schmidt-Hieber, J. (2020): "Nonparametric regression using deep neural networks with ReLU activation function," *The Annals of Statistics*, 48, 1875–1897.

Schölkopf, B. and A. J. Smola (2001): *Learning with kernels*, MIT Press.

Shah, A., E. Kadam, H. Shah, S. Shinde, and S. Shingade (2016): "Deep residual networks with exponential linear unit," in *Proceedings of the Third International Symposium on Computer Vision and the Internet*, Association for Computing Machinery, 59—65.

Shayer, O., D. Levi, and E. Fetaya (2018): "Learning discrete weights using the local reparameterization trick," in *International Conference on Learning Representations*, OpenReview.net.

Shen, M., X. Liu, R. Gong, and K. Han (2020): "Balanced binary neural networks with gated residual," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4197–4201.

Sherratt, K., H. Gruson, R. Grah, H. Johnson, R. Niehus, B. Prasse, F. Sandman, et al. (2022): "Predictive performance of multi-model ensemble forecasts of COVID-19 across European nations," Preprint, medrxiv: 2022.06.16.22276024.

Simonyan, K. and A. Zisserman (2015): "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, OpenReview.net.

Siuzdak, G. (2024): *Activity metabolomics and mass spectrometry*, MCC Press.

Skinnider, M. A., F. Wang, D. Pasin, R. Greiner, L. J. Foster, P. W. Dalsgaard, and D. S. Wishart (2021): "A deep generative model enables automated structure elucidation of novel psychoactive substances," *Nature Machine Intelligence*, 3, 973–984.

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

Strubell, E., A. Ganesh, and A. McCallum (2019): "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 3645–3650.

Szegedy, C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus (2014): "Intriguing properties of neural networks," in *International Conference on Learning Representations*, OpenReview.net.

Taieb, S. B., J. W. Taylor, and R. J. Hyndman (2021): "Hierarchical probabilistic forecasting of electricity demand with smart meter data," *Journal of the American Statistical Association*, 116, 27–43.

Taylor, S. J. and B. Letham (2018): "Forecasting at scale," *The American Statistician*, 72, 37–45.

Timmermann, A. (2006): "Forecast combinations," in *Handbook of Economic Forecasting*, Elsevier, vol. 1, 135–196.

van der Maaten, L. and G. Hinton (2008): "Visualizing data using t-SNE," *Journal of Machine Learning Research*, 9, 2579–2605.

van Rossum, G. et al. (2011): *Python programming language*, URL: https://www.python.org (last accessed: February 23, 2025).

Vannitsem, S., J. B. Bremnes, J. Demaeyer, G. R. Evans, J. Flowerdew, S. Hemri, S. Lerch, et al. (2021): "Statistical postprocessing for weather forecasts:

Review, challenges, and avenues in a big data world," *Bulletin of the American Meteorological Society*, 102, E681–E699.

Vitart, F., A. Robertson, A. Spring, F. Pinault, R. Roškar, W. Cao, S. Bech, et al. (2022): "Outcomes of the WMO prize challenge to improve sub-seasonal to seasonal predictions using artificial intelligence," *Bulletin of the American Meteorological Society*.

Wager, S. and S. Athey (2018): "Estimation and inference of heterogeneous treatment effects using random forests," *Journal of the American Statistical Association*, 113, 1228–1242.

Wang, F., J. Liigand, S. Tian, D. Arndt, R. Greiner, and D. S. Wishart (2021): "CFM-ID 4.0: More accurate ESI-MS/MS spectral prediction and compound identification," *Analytical Chemistry*, 93, 11692–11700.

Wang, M., J. J. Carver, V. V. Phelan, L. M. Sanchez, N. Garg, Y. Peng, D. D. Nguyen, et al. (2016): "Sharing and community curation of mass spectrometry data with Global Natural Products Social Molecular Networking," *Nature Biotechnology*, 34, 828–837.

Wang, T. J., D. Ngo, N. Psychogios, A. Dejam, M. G. Larson, R. S. Vasan, A. Ghorbani, et al. (2013): "2-Aminoadipic acid is a biomarker for diabetes risk," *The Journal of clinical investigation*, 123, 4309–4317.

Wei, J. N., D. Belanger, R. P. Adams, and D. Sculley (2019): "Rapid prediction of electron-ionization mass spectrometry using neural networks," *ACS Central Science*, 5, 700–708.

Weininger, D. (1988): "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules," *Journal of Chemical Information and Computer Sciences*, 28, 31–36.

White, C. J., D. I. Domeisen, N. Acharya, E. A. Adefisan, M. L. Anderson, S. Aura, A. A. Balogun, et al. (2022): "Advances in the application and utility of subseasonal-to-seasonal predictions," *Bulletin of the American Meteorological Society*, 103, E1448–E1472.

Williams, C. K. I. (1996): "Computing with infinite networks," in *Proceedings of the 10th International Conference on Neural Information Processing Systems*, MIT Press, 295—301.

Winkler, R. L. (1996): "Scoring rules and the evaluation of probabilities," *TEST*, 5, 1–26.

Wishart, D. S., A. Guo, E. Oler, F. Wang, A. Anjum, H. Peters, R. Dizon, et al. (2022): "HMDB 5.0: The Human Metabolome Database for 2022," *Nucleic Acids Research*, 50, D622–D631.

Witkowski, J., R. Freeman, J. W. Vaughan, D. M. Pennock, and A. Krause (2023): "Incentive-compatible forecasting competitions," *Management Science*, 69, 1354–1374.

Wolf, S., S. Schmidt, M. Müller-Hannemann, and S. Neumann (2010): "In silico fragmentation for computer assisted identification of metabolite mass spectra," *BMC Bioinformatics*, 11, 148.

Wright, M. N. and A. Ziegler (2017): "ranger: A fast implementation of random forests for high dimensional data in c++ and R," *Journal of Statistical Software*, 77, 1–17.

Xing, S. and T. Huan (2022): "Radical fragment ions in collision-induced dissociation-based tandem mass spectrometry," *Analytica Chimica Acta*, 1200, 339613.

Xing, S., S. Shen, B. Xu, X. Li, and T. Huan (2023): "BUDDY: Molecular formula discovery via bottom-up MS/MS interrogation," *Nature Methods*, 20, 881–890.

Yang, G. (2020): "Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation," Preprint, arxiv:1902.04760.

Zängl, G., D. Reinert, P. Rípodas, and M. Baldauf (2015): "The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core," *Quarterly Journal of the Royal Meteorological Society*, 141, 563–579.

Zhao, X., Y. Wu, D. L. Lee, and W. Cui (2019): "IForest: Interpreting random forests via visual analytics," *IEEE Transactions on Visualization and Computer Graphics*, 25, 407–416.

Zhou, H., J. Lan, R. Liu, and J. Yosinski (2019): "Deconstructing lottery tickets: zeros, signs, and the supermask," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Curran Associates Inc., 3597–3607.

Zhu, C., S. Han, H. Mao, and W. J. Dally (2017): "Trained ternary quantization," in *International Conference on Learning Representations*, OpenReview.net.