

Exploring Hyperparameter Usage and Tuning in Machine Learning Research

Sebastian Simon*, Nikolay Kolyada†, Christopher Akiki‡, Martin Potthast‡, Benno Stein†, Norbert Siegmund‡

*Leipzig University

†Bauhaus-University Weimar

‡Leipzig University, ScaDS.AI Dresden/Leipzig

Abstract—The success of machine learning (ML) models depends on careful experimentation and optimization of their hyperparameters. Tuning can affect the reliability and accuracy of a trained model and is the subject of ongoing research. However, little is known on whether and how hyperparameters are used and optimized in research practice. This lack of knowledge not only limits the adoption of best practices for tuning in research, but also affects the reproducibility of published results. Our research systematically analyzes the use and tuning of hyperparameters in ML publications. For this, we analyze 2000 code repositories and their associated research papers from *Papers with Code*. We compare the use and tuning of hyperparameters of three widely used ML libraries: scikit-learn, TensorFlow, and PyTorch. Our results show that the most of the available hyperparameters remain untouched, and those that have been changed use constant values. In particular, there is a significant difference between tuning hyperparameters and the reporting about it in the corresponding research papers. Our results suggest that there is a need for improved research and reporting practices when using ML methods to improve the reproducibility of published results.

Index Terms—Hyperparameter, Hyperparameter Tuning, Configuration Settings

I. INTRODUCTION

Machine learning (ML) is a success story in many fields, such as healthcare, finance, and the automotive industry [1]. An important enabler for this success is the highly experiment-driven development of ML models. Experiments evaluate critical design decisions, such as alternative modeling techniques, different ML configurations, and even different data slices. The goal of experimentation is to explore the search space of a family of ML algorithms and their hyperparameters to obtain an ML model that achieves the desired accuracy, reliability, fairness, and robustness.

Experimental settings and hyperparameters play an important role in finding the best possible ML model and learning setup. While ML models can be highly sensitive to experimental settings, such as random seeds and train-test data slices, optimizing their hyperparameters often affects the accuracy, learning effort, and generalizability to the point where a model with initially poor performance may turn out to be state of the art (SOTA). To quantify the importance of hyperparameter tuning in ML, we counted relevant¹ publications from DBLP

¹To identify relevant papers, we considered all titles with the keyword “hyperparameters” combined with either “importance,” “tuning,” or “optimization.” This was followed by a manual title check to exclude false positives. Papers on hyperparameter tuning whose authors did not indicate this fact in this way were disregarded.

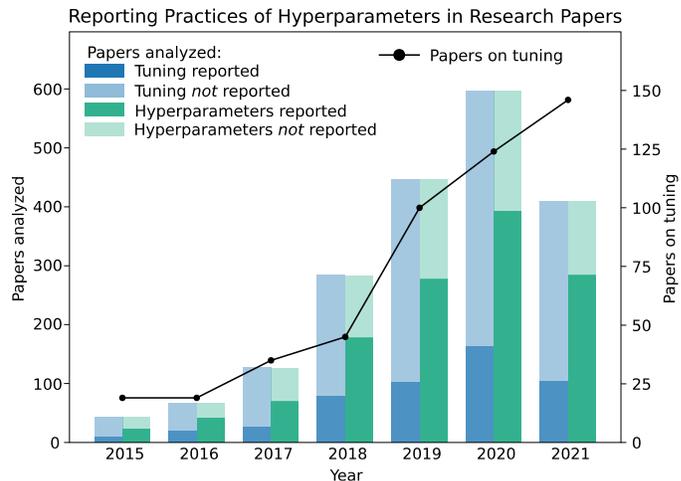


Fig. 1: Number of papers indexed in the *dblp computer science bibliography* between 2015 and 2021 containing the title keywords: hyperparameter importance, hyperparameter tuning, and hyperparameter optimization, plotted against reported hyperparameter (HP) tuning and stated hyperparameter values in research papers.

from 2015 to 2022 and plotted their growth per year in Figure 1. We observe a significant increase in publications with a seven-fold increase in the observation period. We interpret this growth as a strong indicator of the practical importance of this subject.

Surprisingly, despite this extensive research, little is known on whether and how hyperparameters are actually used and optimized in research practice, and what methods are used. This is striking given the importance of tuning and what we could learn by deriving best practices or analyzing divergent community standards. This information alone would already provide new and important insights into the ML community. When combined with information from the publications themselves, even more insight into the behavior of the community can be gained. With this work, we take the first step to filling this gap. By comparing whether hyperparameters are changed in a paper’s associated code and what has been reported about them. We further track reporting differences among ML disciplines and raise awareness of this issue. This is crucial for the reproducibility of ML approaches: not knowing whether or which hyperparameters were optimized in which way makes it

difficult or impossible to reproduce a result [2], [3].

We analyze the state of hyperparameter usage and tuning in ML research in 2000 ML code repositories and associated research papers from *Papers with Code*. For these repositories, we analyze the use of hyperparameters in the APIs of three widely used ML libraries—scikit-learn, TensorFlow, and PyTorch. Specifically, we use static code-analysis techniques, such as control and data flow analysis, to determine which ML methods² are most commonly used and which of them have custom hyperparameter settings. We supplement this information with manually extracted information from a repository’s associated research paper. This allows us to identify community practices and shed light on reporting standards for tuning, which can be a future way for assessing the robustness of an ML model and the reproducibility of the results obtained. To guide the analysis of this data, we explore and answer the following research questions:

- **RQ₁**: Which, how, and to what extent are ML methods configured with respect to their hyperparameter settings?
- **RQ₂**: How are hyperparameter settings reported in the accompanied papers?

We find that, on average, only a few hyperparameters are set, while the majority of available ones remain untouched. Therefore, most hyperparameters retain their default values when an ML method is called. This can mean two things: First, the default values are already carefully chosen or theoretically derived and work in most use cases. Second, researchers do not tune them, even though this could improve their results. The former is likely not unanimously true for all methods, or else it is surprising that research on hyperparameter tuning is growing so rapidly (see Figure 1). However, Figure 1 also shows that the majority of our analyzed papers do not report tuning (about 75% across all years) and only half of the papers state the used hyperparameters. In other words: Why produce so many papers on the subject when the default values are already sufficient? In the second case, the reasons for not tuning an ML method need to be identified. Possible explanations may include lack of an appropriate experimental setup, limited knowledge of the effects of certain hyperparameters, time pressure, or knowledge gaps. In summary, there is a striking difference between research on and research with hyperparameter tuning in the ML community. Thus, our results are an important contribution to the critical discussion of research practices in this area.

In summary, we make the following contributions:

- An overview of the literature and a domain analysis in terms of reporting practices on hyperparameter tuning (Section II).
- A large dataset that includes 2000 papers and their associated code from *Papers with Code* from various ML disciplines, as well as methodology for the (manual) labels indicative of tuning (Section III).

²For simplicity, we refer to the ML classes provided by the ML libraries as “ML methods” in the following.

- A comprehensive analysis of the use of hyperparameters in ML code repositories using static code-analysis techniques (Section IV).

To ensure reproducibility, all data and code are published alongside this paper.³

II. BACKGROUND AND STATE OF THE ART

In this section, we first describe the selection of ML libraries and give an overview of typical parameters of ML algorithms. We then briefly describe approaches for hyperparameter tuning and finally present related work on the importance of hyperparameters.

A. ML Libraries

In this study, we target ML methods and their hyperparameter settings in the APIs of three major and widely used ML libraries, namely scikit-learn, TensorFlow, and PyTorch. We chose these ML libraries for two reasons: First, we opt for open-source libraries, such that we can employ static code analysis to identify hyperparameter usage patterns. Second, we aim to select prominent and widely-used libraries to have a substantial and externally valid data source available for an analysis. According to Kaggle’s recent survey⁴ on the state of data science and machine learning reveals that scikit-learn, TensorFlow, and PyTorch are among the most popular ML libraries used by data scientists.

Scikit-learn is a high-level library and provides many off-the-shelf and ready to use ML methods for supervised and unsupervised problems, such as linear regression, random forests (RF), and support vector machines (SVM) [4]. In addition to classifier and regressor methods, scikit-learn also incorporates many other algorithms that can be used, for example, for data processing, feature extraction, and even hyperparameter tuning. Scikit-learn has been developed since 2007 and is almost exclusively written in Python [5]. By contrast, TensorFlow and PyTorch are competitors in the field of deep learning (DL) and neural networks. While TensorFlow was originally developed by Google and became an open-source library in 2015 [5], PyTorch was developed by Facebook and open-sourced on GitHub in 2017 [6]. TensorFlow and PyTorch are low-level libraries that allow for building ML models by using a sets of building blocks. Both libraries, enable the implementation of DL architectures and algorithms.

B. Types of Parameters

In the field of ML, different types of parameters are used. Those can be divided into experimental settings, model parameters, and hyperparameters. Experimental settings, such as seeds for random values or learning-validation split ratios of the dataset, cover a wide range of parameters to steer the execution of experiments and make them reproducible. Model parameters and hyperparameters are configuration options of an ML method (or algorithm) and the resulting ML model. Specifically, a model parameter, such as the weight of neuron

³Supplementary Website: <https://zenodo.org/record/7745740>

⁴<https://www.kaggle.com/kaggle-survey-2022>

connections in a neural network, is an internal value of the ML model that is not set manually by an ML engineer, but is learned or estimated from data during training [7]. A hyperparameter, on the other hand, represents a configuration option that can be manually specified by an ML engineer to steer the learning process [8], thereby determining the values of model parameters that a model learns during the learning phase [9]. Typical hyperparameters are, for instance, the regularization parameter C in SVM, the learning rate in optimization algorithms, or the number of neurons and hidden layers in neural networks.

C. Hyperparameter Tuning

Different ML methods have different kinds of hyperparameters. The selection of these methods and their configuration settings strongly depend on the modality, domain and available training data [10], [11]. When developing ML models, ML engineers typically begin by selecting an ML algorithm applicable for the problem and data at hand, and then manually specify its hyperparameter values. The simplest way to configure hyperparameters is to keep their default values, which are usually provided by the ML library. However, ML engineers often tune hyperparameters to find the best set of hyperparameter values that result in an ML model with a desired accuracy. To this end, ML engineers define a search space (i.e., the set of hyperparameters and what value ranges to consider) and select the search heuristic used to find the best hyperparameter values in the search space [12].

Many approaches have been proposed to optimize hyperparameters, such as manual tuning [13], grid and random search [14], Bayesian optimization [15]–[18], meta-learning [19]–[21], and bandit-based method [22]. When applied properly, they often result in a significant performance gain for the resulting ML model [23]. However, hyperparameter tuning is generally a computationally expensive task [21], which is why ML engineers may often keep default values or only tune specific hyperparameters.

D. Hyperparameter Importance

The importance of hyperparameters and their tuning has been addressed in several studies. For example, Mantovani et al. [24] developed a meta learning-based recommendation system to predict whether optimization techniques lead to a performance gain of ML models compared to the performance of ML models obtained when hyperparameters are left at their default values. The authors compared the performance of ML models induced by SVM on 143 datasets against the default values provided by ML libraries, showing that they can accurately predict when optimization techniques should be used instead of default values. Lavesson and Davidsson [25] systematically compared four ML methods to investigate algorithm configurations on classifier performance. Interestingly, they found that it is more important to tune the hyperparameters of an ML method than to choose a specific ML method. Moreover, they support the assertion that some ML methods are more robust than others regarding their hyperparameter settings. Weerts et al. [12] propose a methodology based on non-inferiority test and tuning risk

to determine which hyperparameters are important to tune. Since their approach relies on the notion of default values, they also define reasonable default values. They applied their approach on 59 different datasets from OpenML, focusing on RF and SVM. Interestingly, they found that leaving certain hyperparameters at their computed default values is non-inferior to tuning them. Moreover, in some cases, the calculated default values even outperformed the tuned hyperparameter values. Probst et al. [26] studied the tunability of hyperparameters represented by the performance gain obtained when tuning the hyperparameters. To this end, they investigated six common ML methods (i.e., elastic net, decision tree, k-nearest neighbors, SVM, RF, gradient boosting) on 38 OpenML datasets to assess the tunability of their hyperparameters. Their results yield default values for hyperparameters and enable users to determine the importance of tuning a hyperparameter.

By contrast to these studies, researchers have also addressed the importance of hyperparameters. Specifically, they aimed to identify hyperparameters that are important to optimize. To this end, many techniques have been developed to assess the importance of hyperparameter, such as forward selection [27], ablation analysis [28], [29], and functional analysis of variance (ANOVA) [30]–[32].

And yet, while all of the aforementioned studies contribute valuable techniques to gain insights about the importance of hyperparameters, we do not know how they are actually used and tuned in ML research. For this reason, we complement the above studies in the following aspects: first, we analyze the usage of hyperparameters of three popular ML libraries in 2000 code repositories to identify how hyperparameters are actually used. Second, we analyze the research papers of the code repositories with respect to tuning. Finally, we superimpose the information extracted from the code repositories and their papers to shed light on experimentation and reporting practices with respect to hyperparameter usage and tuning.

III. METHODOLOGY

To systematically answer our research questions (see Section I), we empirically studied hyperparameter usage and tuning in ML research. We collected a dataset of 2000 randomly selected code repositories with their respective research papers from *Papers with Code*. We analyzed the repositories using static code-analysis to identify which ML methods are mostly used and how their hyperparameters are set. We also analyzed the code repositories accompanied research papers to shed light on hyperparameter tuning in ML research. Our empirical study follows the methodology shown in Figure 2 as described next.

A. Project Selection

For our code repository analysis, we needed suitable code repositories that are accompanied with research papers and incorporate the ML libraries of interest. To this end, we targeted the *Papers with Code* corpus⁵, since it aims at creating an open source resource for researchers and practitioners to facilitate

⁵<https://paperswithcode.com/>

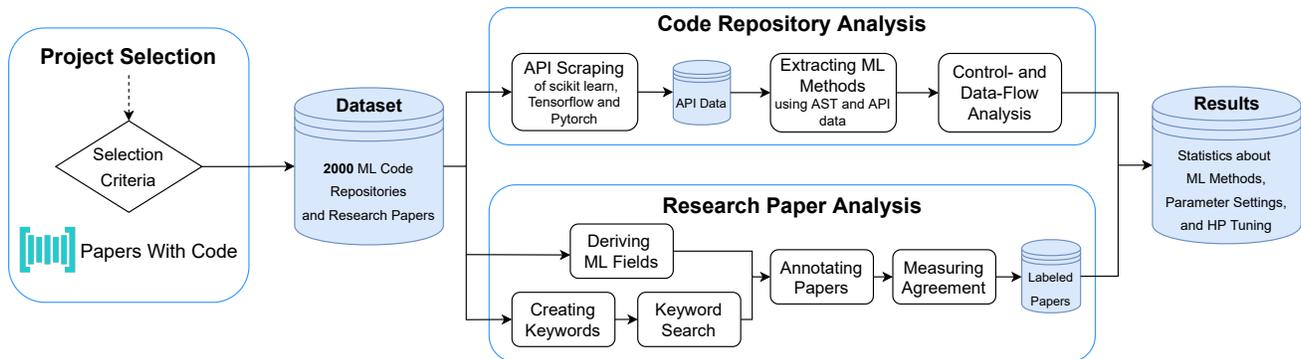


Fig. 2: Methodology of analyzing ML code repositories and associated research papers from the *Papers with Code* corpus.

discovery and comparison in the field of ML. It is one of the largest platforms that collects and provides ML papers, code, and data. In addition to papers, the corpus also provides a list of repositories that link to the paper, including the official or third-party implementations of the publication, all of which constitutes an ideal data source for our study.

a) Code Repository Selection.: In October 2021, the *Papers with Code* corpus indexed about 63,517 papers. For these papers, we processed the list of repositories and downloaded about 86,053 HEAD revision of repositories hosted on GitHub as well additional metadata on them. Due to the enormous number of code repositories, we selected a representative sample set of 2000 code repositories. Since we aimed to systematically gather a dataset of code repositories, we established criteria for the inclusion of code repositories as follows. A code repository must: (1) be accessible on GitHub; (2) incorporate at least one of the ML libraries of interest; (3) be written syntactically correct in Python 3; (4) be associated with a research paper.

The first criterion limits the scope of code repositories to those accessible on GitHub. Since the *Papers with Code* corpus contains papers and the corresponding code repositories published between 2009 and 2021, it may happen that some code repositories have been removed. The second criterion ensures that the ML code repositories use at least one of the target ML libraries. To identify code repositories that incorporate at least one of the ML libraries, we checked the import statements in each Python file to see if one of the ML libraries is imported. The third criterion had a technical background. Since the Python AST ⁶ library (Version 3.10.4) that we used in combination with a control- and data-flow analysis to analyze the code repositories is not compatible with Python 2.x, we limit our selection to code repositories that are syntactically correct written in Python 3. Finally, a code repository must be associated with a research paper, as we aim to analyze the corresponding research paper to compare the tuning activities with the actual code.

We randomly selected code repositories from the *Papers with Code* corpus and checked them against our guidelines

for inclusion. We stopped the selection after we found 2000 repositories—alongside their corresponding papers—that met all of our criteria.

B. Code Repository Analysis

The analysis of code repositories consists of two steps: scraping the APIs of scikit-learn, TensorFlow, and PyTorch to extract the current state of API calls and their configuration settings, and extracting ML methods and their hyperparameter settings based on the scraped API data in combination with a control- and data-flow analysis.

a) API Scraping.: To reliably locate and extract ML methods within the source code, we first needed a current overview about all existing ML methods and their configuration settings that are offered by the API of an ML library. To this end, we scraped the API of each targeted library, thereby extracting each possible call and its configuration settings, including all its classes and methods. Thus, we obtained an up-to-date overview of existing methods provided by scikit-learn, TensorFlow, and PyTorch. In Table I, we show the number of scraped API calls scikit-learn, TensorFlow, and PyTorch. The scraped API data builds the basis for locating and extracting ML methods and their hyperparameter settings within the source code.

TABLE I: Number of scraped API calls (i.e., all classes) and corresponding parameters for specific versions of scikit learn, Tensorflow, and Pytorch.

ML library	Version	API Calls	Params
Scikit Learn	1.1.1	262	1866
Tensorflow	2.9.1	2273	11657
Pytorch	1.12	384	1288

b) Locating and Extracting ML methods.: To automatically locate and extract ML methods and their configuration settings within source-code files, we transferred the code to the Python abstract syntax tree (AST) ⁷ and combined with the the scraped API data. Specifically, we found ML methods by checking the AST of the corresponding source-code file against

⁶<https://docs.python.org/3/library/ast.html>

⁷<https://docs.python.org/3/library/ast.html>

the scraped API data. Once we found an AST object with the same name as a ML method in the API data, we extracted the corresponding ML method with its configuration settings from the documentation . Using the AST objects, we extracted the configuration settings for each ML method actually used in the source-code of each repository. Our AST analysis relies on pattern matching that bears the risk of finding other AST objects with the same name as an API call (e.g., call to user defined methods), but not representing an ML method of one of our three ML libraries. For this reason, we additionally checked all imports in the corresponding Python file to ensure that the AST object actually references an ML library of interest.

c) *Control- and Data-Flow Analysis*: When implementing ML models, it is quite common to define variables that store specific configuration values. Usually, those variables are defined before the ML method is initialized and later passed to the ML method when the method is being called. In these cases, locating and extracting the concrete configuration values is limited, since we do not know the actual value of the variables that are passed to the ML method. To this end, we additionally performed a control- and data-flow analysis in order to identify all possible values of a variable that is passed to a hyperparameter of an ML method.

The control- and data flow analysis relies on the third-party library Scalpel [33]. Scalpel is an open source Python library that provides essential program analysis functions, such as control-flow graph construction, static single assignment representation, constant propagation, and alias analysis⁸. Using Scalpel, we first created all possible control-flow graphs for Python source code files and then applied static single assignment and constant propagation to them to identify the actual values of variables. Specifically, static single assignment module created an intermediate representation of variables by renaming variable assignments with different names. This intermediate representation of variables was then used by the constant propagation module to record all values from an assignment for a single variable. This way, we could track the flow of variables through the source code and were able to determine all possible values for a variable.

C. Research Paper Analysis

To identify whether hyperparameters are tuned and how it is reported in the research papers, we conducted a domain and a systematic research paper analysis.

1) *Domain Analysis*: Technically, the domain analysis consists of three steps. First, we derived a set of diverse fields from the state of the art categories provided by *Papers with Code*⁹ and the category taxonomy of arXiv¹⁰. Second, we manually assigned a field to each research paper. Finally, we measured the inter-annotator agreement to assess the annotation process and the resulting annotations.

⁸<https://github.com/SMAT-Lab/Scalpel>

⁹<https://paperswithcode.com/sota>

¹⁰https://arxiv.org/category_taxonomy

a) *Deriving ML Fields.*: Almost each research paper in the *Papers with Code* corpus contains metadata, such as code statistics, GitHub metadata, and paper metadata. The paper metadata includes ML tasks, which refer to the state of the art categories of *Papers with Code*, and arXiv categories, representing different ML fields. Since there is an overlap between the ML fields from both sources, we first merged all categories, then two authors discussed the resulting categories. If two authors could not find an agreement for an ML field, a third author was consulted to discuss the ML field. A final agreement was reached when at least two authors agreed on the ML field. Finally, we end up with 17 different ML field that cover all arXiv and state of the art categories of *Papers with Code*: Computer Vision, Natural Language Processing, Audio, Robotic, Mathematics, Physics, Machine Learning, Games, Physics, Information Retrieval, Software Engineering, Finance, Biology, Security, Electrical Engineering, Social and Information Networks, Databases, Miscellaneous. To have a clear distinction between each ML field, we explain in detail what areas an ML field covers at our supplementary website¹¹.

b) *Annotating Paper.*: The main author went through all papers and manually assigned one ML field to each paper based on ML task and arXiv categories that are assigned to the papers. For papers that did not have ML tasks or arXiv categories assigned, we read the title and abstract to assign an ML field to a paper.

c) *Inter-Annotator Agreement.*: To check the annotation process and the agreement among annotator, a second author labeled a set of 100 randomly selected papers. For this dataset, we calculated Cohen's kappa coefficient [34], a statistic to measure the agreement between a pair of annotators. Given Landis and Kochs scale [35], Cohen's kappa statistic was 0.67, indicating moderate agreement.

2) *Paper Analysis*: The paper analysis consists of four steps. First, we created a vocabulary of keywords related to hyperparameter tuning. Second, we applied these keywords to our research papers to identify sections that describe the usage and tuning of hyperparameters. We then read each section found with our keywords and annotated each paper to answer RQ₂. Finally, we measured the inter-annotator agreement to assess the annotation process and the resulting annotations.

a) *Creating Keywords.*: Due to the large number of papers, we did not read the entire papers. Instead, we focused on a keyword search to identify sections describing implementation and experimentation details, as these sections typically reveal whether and how ML methods are tuned. To find these sections, we applied a keyword search to all papers. As the set of keywords is crucial to find relevant sections, we first built our vocabulary by initially reading 50 papers and extracting terms related to hyperparameter tuning. The final keywords in our vocabulary are the following: parameter, tune, fit, train, search, sweep, optimize, tuning, optimizing, optimization, implementation, experiment.

¹¹Supplementary Website: <https://zenodo.org/record/7745740>

b) *Keyword Search.*: We created a search query for each keyword in our vocabulary and applied it to the papers, including the papers used to create the vocabulary. This way, we ensure that we apply the same search queries to each paper. Applying the search queries to our papers resulted into sections that possibly describe the usage and tuning of hyperparameters.

c) *Annotating Papers.*: Next, we carefully read each section found with our keywords and annotated each paper guided by RQ₂. To answer RQ₂, we divided RQ₂ into the following sub-questions, which we answered for each paper accordingly:

- Q₁: Is hyperparameter tuning reported?
- Q₂: Are the final hyperparameter values reported?
- Q₃: Which technique is used to tune hyperparameters?

Due to the nature of Q_{1,2}, we answered these questions by annotating the papers with *yes* or *no*. Specifically, when we found evidence that hyperparameter tuning was performed (e.g., the tuning technique and the tuned hyperparameters were clearly described in a paper), we answered Q₁ with *yes*. Conversely, if we did not find any reference of hyperparameter tuning in a paper, we answered this question with *no*. Similarly, we answered Q₂ with *yes* when we found that final hyperparameter values (i.e., the respective hyperparameters and their final values) were reported, otherwise we annotated this question with *no*. During our analysis, we also extracted each hyperparameter tuning approach and framework that were mentioned in these sections to answer Q₃.

d) *Inter-Annotator Agreement.*: As two authors conducted the analysis of research papers, we finally measured the inter-annotator agreement (IAA) to assess the annotation process and ensure the correctness of the resulting annotations. To this end, each author randomly selected a sample set of 100 paper annotated by the other author and annotated the papers again. This helped us to cross-validate the annotations and measure the annotator agreement. For this dataset, we calculated Cohen’s kappa statistic [34] for Q_{1,2}. Given Landis and Kochs scale [35], Cohen’s kappa statistic was 0.60 for Q₁, indicating moderate agreement, and 0.70 for Q₂, indicating a substantial agreement.

IV. RESULTS

In this section, we present the results of analyzing the code repositories along with their respective research papers. To answer our research questions, we first analyzed the code repositories to determine the use of hyperparameters using static code analysis. We then systematically reviewed related research papers and combined our findings with the data from the code repositories to shed light on hyperparameter tuning and reporting practices in ML research.

A. RQ₁: Which, how, and to what extent are ML methods configured with respect to their hyperparameter settings?

Finding 1: *The most commonly used methods are neural-network building blocks provided by PyTorch and TensorFlow. By contrast, few methods from scikit-learn are used that cover ML and experimental methods.*

TABLE II: Top 10 most used methods in the analyzed corpus. Under *Parameter Settings*, column *Count* indicates the number of available of parameters to configure the method, while column *Avg.* shows the average number of parameters used to configure the methods.

	ML Library Usage			Parameter Settings			
	Method	Count	Category	Count	Avg.	Avg. %	Most adjusted
scikit-learn	StandardScaler	192	preprocessing	3	0.12	(4.0)	default
	PCA	136	decomposition	9	1.23	(13.7)	n_components
	KMeans	134	cluster	9	2.28	(25.3)	n_clusters
	LogisticRegression	124	linear_model	15	2.40	(16.0)	C
	TSNE	98	manifold	16	2.74	(16.9)	n_components
	KFold	98	model_selection	3	2.47	(91.3)	n_splits
	LinearRegression	85	linear_model	5	0.36	(7.2)	default
	LabelEncoder	71	preprocessing	0	0.00	-	default
	MinMaxScaler	67	preprocessing	3	0.42	(14.0)	default
	SVC	65	svm	15	1.48	(9.9)	kernel
TensorFlow	Variable	2007	tensorflow	12	1.98	(16.5)	initial_value
	Session	1572	compat	3	0.58	(19.3)	default
	Dense	1554	keras	11	2.72	(24.7)	units
	Saver	1002	compat	15	0.68	(4.5)	default
	AdamOptimizer	908	compat	6	1.41	(23.5)	learning_rate
	DEFINE_string	836	compat	6	3.00	(50.0)	name, default, help
	ConfigProto	763	compat	17	1.21	(7.1)	allow_soft_placement
	Dropout	693	keras	4	1.03	(25.8)	rate
	DEFINE_integer	654	compat	8	3.00	(37.5)	name, default, help
	TensorShape	612	tensorflow	1	1.00	(100)	dims
PyTorch	Conv2d	15072	neural networks	11	4.95	(45.0)	in_channels
	Linear	14360	neural networks	5	2.16	(43.2)	in_features
	Sequential	11247	neural networks	1	0.93	(93.0)	*args
	ReLU	9097	neural networks	1	0.61	(61.0)	inplace
	BatchNorm2d	6507	neural networks	7	1.34	(19.1)	num_features
	Parameter	4812	neural networks	2	1.17	(58.5)	data
	DataLoader	4511	utils	15	4.09	(27.3)	dataset
	ModuleList	4169	neural networks	1	0.50	(50.0)	default
	Dropout	3694	neural networks	2	0.95	(47.5)	p
	Adam	2234	optim	7	1.57	(22.4)	default

The number of the ML libraries per code repository varies. We obtained 1258 code repositories that import only one ML library, 653 that import two, and 89 that import all three libraries. Among all code repositories, PyTorch is the most frequently used ML library, being imported in 65 % of them. Scikit-learn comes next, being imported in 43 % of the code repositories, followed by TensorFlow, which is imported in 34 % of the code repositories. In Table II, we show the ten most frequently used methods of each library in these code repositories, how often each method has been used, and information about their configuration (i.e., the number of available hyperparameters and how many have been set, and the most frequently set parameter). The most commonly used methods in our corpus belong to PyTorch, followed by TensorFlow. These correspond to the building blocks used to construct neural networks, such as type and number of layers and activation functions. Also, these parameter may technically not tune an ML model, but rather specify its architecture and complexity. Hence, the large number is to be expected for these frameworks. By contrast, we found few scikit-learn methods in the code repositories, which include ML methods, such as LogisticRegression, but a larger variety, such as experimental methods for preprocessing (e.g., StandardScaler).

Although this first insight may not be surprising and not

TABLE III: Top 5 most commonly used ML methods of the analyzed repositories. *Call Stats* states the total number of calls and the calls without any given parameter. Symbol '-' indicates that a method cannot be called without a given parameter. *Param Stats* provides the number of available hyperparameters and the average usage across all calls. Column *Avg.** subtracts mandatory parameters.

ML Library		Call Stats		Param Stats		
Method / Constructor		Total	Without	Count	Avg.	Avg.*
scikit-learn	KMeans	134	-	9	2.28	1.28
	LogisticRegression	124	30	15	2.40	2.40
	LinearRegression	85	62	5	0.36	0.36
	SVC	65	15	15	1.48	1.48
	RandomForestClassifier	58	12	18	2.34	2.34
TensorFlow	AdamOptimizer	909	41	6	1.41	1.41
	Adam	265	29	14	1.29	1.29
	GradientDescentOptimizer	136	-	3	1.01	1.01
	MomentumOptimizer	83	-	5	2.28	0.28
	RMSPropOptimizer	78	-	7	2.08	1.08
PyTorch	Adam	2234	-	7	1.57	0.57
	SGD	1057	-	7	2.33	0.33
	RMSprop	150	-	7	2.37	1.37
	AdamW	62	-	7	1.74	0.74
	Adagrad	55	-	6	1.29	0.29

relevant to the research question at first sight, it enables us to distinguish parameter settings used for building an ML model versus parameter settings used for tuning an ML model. Since we are interested in tuning, we focus in the following on ML methods from the optimization classes of TensorFlow and PyTorch, as well as classifier, regressor, and clustering methods from scikit-learn as these methods are also the main target for auto tuning and hyperparameter optimization frameworks.

Finding 2: *Only a few hyperparameters of ML methods are set, the majority remain untouched. Consequently, most hyperparameters retain their default values.*

Already Table II indicates in column *Avg. %* that only a low percentage of available hyperparameters are actually used for tuning. However, since the model-building methods may distort the picture, we list the five most commonly used ML methods with respect to tuning parameters in Table III. The number of hyperparameters (column *Count*) ranges from 3 (GradientDescentOptimizer) to 18 (RandomForestClassifier). Most of them have at least one mandatory parameter (e.g., number of clusters in KMeans) such that we must account for this in our analysis. So, column *Avg.** has an adjusted average parameter usage where we removed the mandatory parameters. We clearly see that for most methods, we have at most only one additional parameter set.

Clearly, some parameters may be relevant only in special cases; however, even the optimizer methods of TensorFlow and PyTorch are barely used for tuning. Here, we have parameters, such as *decay* and *weight_decay* for regularization (e.g., for Adagrad), which is important for regularization. These

parameters are mostly untouched. Similar tuning options exist for the other methods and are mostly not set. For TensorFlow, the first parameter of the 5 most used methods is always the learning rate. Although there exists a default value, the learning rate gets nearly always set, but the remaining regularization parameters remain unchanged, similar to PyTorch. Using the default values for most parameters is surprising, as they are generally not ideal choices [7], [36], [37]. Naturally, there are some cases where sticking to default values is reasonable, e.g., when the computational effort required to train and tune and ML model makes hyperparameter tuning infeasible, or when ML libraries already provide sensible default parameter that might be good enough in standard settings to achieve a sufficient performance [38]. Nevertheless, these default values may not provide the best possible results, which could be achieved with hyperparameter tuning.

Finding 3: *Hyperparameters are set by a large fraction with a constant value, ranging from 42 % up to 69 % depending on the framework. It is unclear how these values have been obtained by the developer. In the newer frameworks, more hyperparameter values originate from a variable context (e.g., method calls or stored program variables) that enable an active tuning of these parameters.*

Hyperparameters can be passed in different forms to an ML method. To derive a better picture how parameters are set, we extracted the AST-type information from the parsed ML code. This resulted in 8 type AST-type categories, covering classical data types, such as string, numeric, and boolean values, but also more complex types, such as function calls, operations, and passed variables. In Table IV, we show the categories of types that are assigned to the hyperparameters of the scikit-learn, TensorFlow, and PyTorch ML methods.

We see a large variation of passing values to an ML method and differences between scikit-learn and the two libraries TensorFlow and PyTorch. In scikit-learn, 69 % parameters are given as constant values whereas for TensorFlow and PyTorch, the number drops to 58 % and 42 %. This is surprising as constant values cannot be used by experimentation frameworks, such as MLflow to automatically adapt and document these values. Also, it means that they cannot be easily change via configuration files. Furthermore, it remains unclear how these values have been determined. For the remaining cases, we see that method calls are only marginally used. This means that we do not see parameter tuning frameworks or functions probing for different distributions of values in our corpus. So, again, this hints that parameter tuning is rarely used or only realized via external settings of variables.

To underpin this observation, we depict in each cell of the rightmost column of Table V a list of the most used parameter types that corresponds to the list of parameter names in column *Top 3*. These parameter names represent the most changed parameters of the respective ML method. Furthermore, column *Count* depicts how often a parameter has been given a value. Columns *Variable* and *Constant* that their relative percentage of

TABLE IV: Distribution of Python AST-types passed as hyperparameters to ML methods. Constant means that a constant value has been assigned to a variable which was then passed to an ML method. For unknown cases, the data-flow analysis did not terminate or produced an error.

	Type	scikit-learn	TensorFlow	PyTorch
Constant	Numeric	33.9 %	29.3 %	21.8 %
	String	16.7 %	0.7 %	0.0 %
	Boolean	6.8 %	1.7 %	3.3 %
	None type	2.6 %	0.1 %	0.1 %
	Mapping	1.7 %	0.0 %	0.0 %
	Constant	7.3 %	26.3 %	16.8 %
	Total:	69.0 %	58.1 %	42.0 %
Variable	Variable	23.1 %	36.8 %	40.6 %
	Call	3.9 %	4.1 %	6.9 %
	Operation	3.2 %	1.0 %	1.0 %
	Total:	30.2 %	41.9 %	48.5 %
	Unknown	0.8 %	0.0 %	9.5 %

parameter values passed as either a fixed value or a changeable value that can be set within a (tuning) function, operation, or external experimentation framework. In total, 951 (35.5 %) of parameter values map to a constant value whereas 1753 (65.5 %) can be variably set. This indicates still a large fraction of a constant value, but shows that, in contrast to all other methods, more parameters are set with customization in mind for the most commonly used methods.

Finding 4: *The most important hyperparameters stated by related work have been most often used in our corpus.*

Related work on hyperparameter importance and tuning [12], [39], [40] report several hyperparameters that should be adjusted. We found the same parameters in our corpus. For instance, the learning rate to train neural networks [41] is the most commonly set hyperparameter across all ML optimizer methods of TensorFlow and PyTorch in our data. Similarly, the most commonly set hyperparameter for scikit-learn methods are also considered important, such as `n_clusters` for KMeans [39], the learning rate of the GradientBoostingClassifier [40], and the regularization strength `C` for LogisticRegression, LinearSVC, or SVC [12], as shown in Table V.

Answer RQ₁: *ML methods provide many tuning parameters, but only a fraction is actually set. Consequently, most hyperparameters commonly retain their default values when an ML method is being called. Moreover, if hyperparameters are set, the majority of them are constant values without the possibility for tracking and automatically tuning them with external frameworks.*

B. RQ₂: *How are hyperparameter settings reported in the accompanied papers?*

Finding 5: *Despite the fact that hyperparameter tuning is crucial for the resulting ML model, hyperparameter tuning is explicitly reported only in a few research papers. In about two thirds of our corpus, authors reported the final hyperparameter values, indicating a good, but not perfect state of reproducibility for the published papers.*

Out of 2000 papers, we identified 514 (26 % of all papers), in which hyperparameter tuning was explicitly reported. This is a surprising finding, considering the importance of tuning and the abilities to substantially outperform related, but untuned models. We discuss the consequences of this result in Section V.

Furthermore, values of hyperparameters have reported in 1281 papers (64 % of all papers). The immediate question is how these values have been obtained. Do they originate from expert knowledge, pre-studies, tuning activities, or just first guesses? Unfortunately, this question cannot be answered, but when comparing these numbers to parameter usages, especially whether variable settings have been used (cf. Table IV), we see that most often only constant values are applied. Only barely are tuning activities recognizable such that tuning must either be done outside the provided repository or is absent. So, although it is a good sign that values are reported at all, it hinders the reproducibility of these results when it is unclear how these parameters have been obtained as such a tuning may be different for different data slices or contexts.

Finding 6: *Regardless the ML field, most research papers do not explicitly report hyperparameter tuning, but computer vision and software engineering seem to have especially low hyperparameter-tuning practices.*

To get a more objective picture about how and whether different ML fields report hyperparameter tuning, we counted the number of papers for each ML field that actually reported hyperparameter tuning. We show our results in Table VI.

Our findings reveal that most of the papers in our corpus belong to computer vision, natural language processing, and machine learning, accounting for approximately 81 % of all papers. We observe very similar reporting practices for all ML fields. Higher and lower values for some fields may be an artifact of a low sample in our corpus, such that we refrain from a possible misleading interpretation. At most, we can state that computer vision has a very low tuning activity compared to all other fields, which indicates an interesting result. Moreover, the 12 software-engineering papers in our dataset have a very low fraction of reported tuning. This, at least, indicates that our community should consider this activity more seriously. Possible explanations for the low tuning activities in the different fields may include the lack of an appropriate experimental setup, limited knowledge of the effects of certain hyperparameters, time pressure, or knowledge gaps. Moreover, the computational effort, which is often required to train and tune large ML models, may also be a reason for low tuning activities, especially in the field of natural language processing and computer vision.

TABLE V: The five most commonly used ML methods with the top three most commonly set hyperparameter. Column *Most Used Type* represents the types that are most frequently assigned to the hyperparameters, column *Count* depicts how often the corresponding parameter has been set, column *Constant* and *Variable* represent the percentage of constant or changeable parameter values passed as the corresponding parameter.

ML Library Usage		Hyperparameter					
Method	Calls	Top 3	Count	Most Used Type	Constant in %	Variable in %	
scikit-learn	LogisticRegression	33	[C, solver, random_state]	[15, 13, 12]	[Variable, String, Numeric]	[20, 100, 67]	[80, 0, 33]
	SVC	28	[gamma, kernel, C]	[9, 6, 4]	[Numeric, String, Call]	[67, 67, 25]	[33, 33, 75]
	KMeans	22	[n_clusters, random_state, init]	[22, 13, 7]	[Variable, Numeric, String]	[23, 54, 71]	[77, 46, 29]
	GradientBoostingClassifier	20	[n_estimators, learning_rate, random_state]	[19, 17, 15]	[Numeric, Numeric, Numeric]	[74, 94, 71]	[26, 6, 28]
LinearSVC	17	[C, dual, class_weight]	[12, 8, 7]	[Variable, Boolean, Call]	[25, 100, 86]	[75, 0, 14]	
Tensorflow	AdamOptimizer	414	[learning_rate, beta1, beta2]	[405, 223, 19]	[Variable, Numeric, Numeric]	[5, 94, 42]	[95, 6, 58]
	Adam	91	[learning_rate, epsilon, clipvalue]	[81, 26, 22]	[Numeric, Numeric, Numeric]	[36, 65, 41]	[64, 35, 59]
	GradientDescentOptimizer	42	[learning_rate, use_locking]	[42, 2]	[Variable, Variable]	[69, 50]	[31, 50]
	AdagradOptimizer	30	[learning_rate, initial_acc_val, use_locking]	[30, 7, 1]	[Variable, Numeric, Variable]	[3, 71, 0]	[97, 29, 100]
	MomentumOptimizer	29	[learning_rate, momentum, use_nesterov]	[29, 29, 12]	[Variable, Numeric, Boolean]	[14, 62, 83]	[86, 38, 17]
Pytorch	Adam	646	[lr, betas, weight_decay]	[601, 188, 181]	[Variable, Sequence, Variable]	[22, 96, 23]	[78, 4, 77]
	SGD	233	[lr, momentum, weight_decay]	[219, 151, 101]	[Variable, Variable, Variable]	[14, 37, 20]	[86, 63, 80]
	RMSprop	47	[lr, momentum, alpha]	[44, 25, 24]	[Variable, Variable, Variable]	[0, 16, 25]	[100, 84, 75]
	AdamW	18	[lr, weight_decay, eps]	[18, 13, 11]	[Call, Numeric, Numeric]	[6, 77, 73]	[94, 23, 27]
	Adagrad	13	[lr, weight_decay, lr_decay]	[11, 7, 3]	[Numeric, Numeric, Numeric]	[9, 0, 33]	[91, 100, 67]

TABLE VI: Number of research papers per ML field.

ML Field	Count	Hyperparameter Tuning	
		Reported	Not reported
Computer Vision	797	123 (15 %)	674 (85 %)
Machine Learning	479	187 (39 %)	292 (61 %)
Natural Language Processing	349	114 (33 %)	235 (67 %)
Physics	63	20 (32 %)	43 (68 %)
Audio	46	8 (17 %)	38 (83 %)
Robotic	40	5 (12 %)	35 (88 %)
Information Retrieval	38	18 (47 %)	20 (53 %)
Security	31	5 (16 %)	26 (84 %)
Math	29	2 (7 %)	27 (93 %)
Miscellaneous	25	5 (20 %)	20 (80 %)
Biology	24	9 (38 %)	15 (62 %)
Games	23	5 (22 %)	18 (78 %)
Electrical Engineering	21	5 (24 %)	16 (76 %)
Social and Information Networks	13	3 (23 %)	10 (77 %)
Software Engineering	12	2 (17 %)	10 (83 %)
Databases	6	3 (50 %)	3 (50 %)
Finance	4	1 (25 %)	3 (75 %)

Finding 7: *Most papers do not state a concrete tuning technique and the remaining papers use rather conservative techniques for tuning, such as random probing, grid search, and manual tuning, despite the variety of techniques available for hyperparameter tuning.*

By analyzing the papers that explicitly describe hyperparameter tuning, we found that 281 papers (ca. 55 %) did not mention a concrete hyperparameter tuning technique. Among the papers that mentioned specific tuning techniques, the most popular techniques were grid search, manual tuning, random search, and Bayesian optimization, which were mentioned in 133, 53, 20, and 20 papers, respectively. Since grid search, manual tuning, and random search cannot be considered an advanced or systematic approach, we are astonished by the low state of practice, especially compared when seen a growing number

of papers focusing on tuning. Furthermore, we found several highly specialized tuning methods, such as ant lion optimization, Gaussian bandit optimization, or particle swarm optimization which were only mentioned in a very few papers. We also found a few highly specialized hyperparameter tuning libraries, such as Optuna, Hyperopt, and GPyOpt, indicating that a few papers employed different libraries for hyperparameter tuning. We will provide a list of all tuning techniques and frameworks found with their occurrences at our supplementary website ¹².

Answer RQ₂: *Despite the importance and impact of hyperparameters on the resulting ML model, we found a stark discrepancy between applying hyperparameter tuning and reporting it appropriately in the corresponding research papers. Overall, tuning seems to be not a common practice and it often remains unclear how parameter values have been obtained, hampering reproducibility of results.*

V. DISCUSSION

In this section, we discuss our findings on hyperparameter usage and tuning, as well as reporting practices in ML research.

Hyperparameter Usage of ML methods. Our results indicate that the configuration settings of ML methods do not receive the attention they actually need, since we found that only a few of the available hyperparameters are set, whereas the majority of the available hyperparameters remain untouched. Table VII depicts this result in more detail.¹³ Columns *Actually Set* across all frameworks report only low fractions of parameters with changed values. Interestingly, even there, we some parameters that the default values are still used.

¹²Supplementary Website: <https://zenodo.org/record/7745740>

¹³Columns *Und.* represent the percentage of parameter values for which we cannot definitively state whether a default or a custom value has been used (e.g., because variable values are set outside the reachable data-flow analysis).

TABLE VII: Statistics on hyperparameter usage in code repositories where the associated research paper reported hyperparameter tuning sorted by year, including the available number of hyperparameters (total), the number of actually set hyperparameters, and the fraction between default, customized, and undecidable (Und.) values.

Paper Stats.		scikit-learn				TensorFlow				PyTorch			
Year	Count	Total	Actually Set	Default vs. Custom	Und.	Total	Actually Set	Default vs. Custom	Und.	Total	Actually Set	Default vs. Custom	Und.
2011	1	90	6 (6.7 %)	0%  100%	0 %	-	-	-	-	-	-	-	-
2013	1	-	-	-	-	14	1 (7.1 %)	0%  100%	0 %	-	-	-	-
2014	7	-	-	-	-	91	21 (23.1 %)	0%  100%	0 %	84	24 (28.6 %)	0%  58%	42 %
2015	10	-	-	-	-	6	1 (16.7 %)	0%  100%	0 %	90	25 (27.8 %)	12%  80%	8 %
2016	20	12	2 (16.7 %)	100%  100%	0 %	132	12 (9.1 %)	0%  50%	50 %	21	7 (33.3 %)	0%  86%	14 %
2017	27	25	14 (60.0 %)	0%  57%	43 %	252	45 (17.9 %)	2%  44%	54 %	250	56 (22.4 %)	2%  57%	41 %
2018	79	599	189 (31.6 %)	26%  43%	31 %	592	178 (30.1 %)	4%  40%	56 %	834	171 (20.5 %)	9%  35%	56 %
2019	103	566	72 (12.7 %)	8%  75%	12 %	1761	533 (30.3 %)	38%  50%	12 %	1179	288 (24.4 %)	2%  44%	54 %
2020	162	725	118 (16.3 %)	22%  69%	9 %	1355	212 (15.6 %)	7%  51%	42 %	2545	744 (29.2 %)	2%  49%	49 %
2021	104	1541	211 (13.7 %)	16%  62%	21 %	460	70 (12.7 %)	16%  4%	44 %	1798	438 (24.4 %)	6%  45%	49 %

A similar phenomenon has been observed in the software engineering field. Software systems often provide hundreds of different configuration options to tailor the behavior of a system to user requirements, such as performance, security, and functionality [42]. However, Xu et al. [43] report in their study that only a small percentage of configuration options are set by users, while the majority of configuration options are not touched at all. Their observation is in line with what we have found for the actual parameters settings of ML methods of different ML libraries. Similarly, we found that most hyperparameters of ML methods are not used at all. Those unused parameters enlarge the configuration space of ML methods and neural networks, possibly unnecessarily increasing the optimization space of parameters.

We see two reasons why most of the available hyperparameters are unused. First, the default values might be already well chosen. In light of this, researchers may believe that some ML methods, such as linear and logistic regression, have been extensively studied, such that their default values should work in most cases. Hence, it seems that researchers tend to trust default values for many hyperparameters without questioning whether the default values are actually good. However, previous work has already shown that default values are generally not an ideal solution [7], [36], [37], as they, for example, harm the reproducibility of ML projects and experiments [44]. Second, researchers do not touch hyperparameters, although their tuning could improve the results. In this case, we need to find reasons why this is the case. Researchers may not understand the importance of hyperparameters, require a substantially larger setup and time for tuning, or just do not know how to set them appropriately. Clarifying this question can be an important future research activity in this field.

Rare Tuning Activities. Despite the importance of hyperparameter tuning on the resulting ML model, only 25 % of our papers report tuning. This result can have profound consequences for ML papers: We would expect that similar papers can easily improve SOTA results just by tweaking the parameters, but without substantial new insights. This can increase the already huge number of ML papers (or a reason thereof) without having an actual novel contribution. Moreover,

we have difficulties to assess the actual capabilities of current methods. Does tuning further improve the results or are we already at our maximum. Even worse, when we do not know whether parameters have been tuned at all (as more than half of our papers do not report these), we cannot be sure whether reported improvements stem from tuning. However, our code analysis hint more in the direction that parameters are actually not tuned, leaving out considerable potential of improvements via experimentation tracking frameworks.

Lack of Experimentation and Reporting Practices. A last surprising result is the absence of experimentation practices and frameworks. Recent years have seen the rise of many successful experiment frameworks, such as Weights and Biases and MLflow, but our data does not align with that. It indicates the industrial practices are more advanced and rigorous compared to scientific code and reporting practices. Here is where we see an easy and impactful action: Using experimentation frameworks can not only improve the results of the ML methods, but also provide parameter tracking, logging, and reproducibility capabilities that most of our papers clearly lack.

VI. THREATS TO VALIDITY

Internal Threats to Validity. Identifying sections in research papers that describe hyperparameter usage and tuning may impose a threat to internal validity. To mitigate this risk of missing potentially relevant sections of a paper, we carefully created a set of keywords related to hyperparameter tuning by reading 50 research papers in our corpus and extracting keywords that have been used to describe hyperparameter tuning. Furthermore, since the metadata of some research papers may be incomplete, we could neither rely on state of the art categories of *Papers with Code*, nor on the arXiv categories to classify the research papers into different ML fields. Hence, we merged the categories of *Papers with Code* and arXiv and iteratively refined and discussed each category. If two authors could not agree, a third author was consulted until an agreement was reached. The annotation process of research papers may be subjective and could introduce researcher bias. To mitigate this threat, two authors performed the domain and paper analysis for which we measured the inter-annotator agreement among

the authors. We measured moderate and substantial agreement for the domain and research paper analysis.

Threats to External Validity. Our external threat arises from the selection of code repositories and concerns the ability to generalize our results. In our study, we targeted the *Papers with Code* corpus, which contains research papers and associated code repositories generated between 2009 and 2021 of different domains. We randomly selected a large sample of 2000 code repositories that met our criteria for inclusion. Although we cannot claim that our results can generalize to every code repository, the large corpus and clear results indicate a clear trend of parameter usage.

VII. CONCLUSION

The success story of ML-enabled software systems is based on a careful experimentation process, where data scientists probe the search space of ML algorithms and their hyperparameters. Numerous papers emphasize the importance of hyperparameter tuning on the accuracy and replicability of the resulting ML model, and even more propose novel solutions to this activity. Simultaneously, there is no systematic study that analyzes whether and how parameters are tuned for research papers. To this end, we analyze 2000 code repositories and their associated research papers from the *Papers with Code* corpus. We analyze whether and how hyperparameters are set in three widely used ML libraries—scikit-learn, TensorFlow, and PyTorch—and if they are appropriately reported in the accompanied research papers. We found a considerable discrepancy between the available and actually changed parameters for across ML methods and frameworks. We found that most parameters are given a constant values, indicating the absence of a tuning technique or experimentation framework. Furthermore, we also found a stark contrast between stating concrete hyperparameter values in a paper and reporting the tuning activity leading to these values. There is a substantial lack of reporting with below half the papers not mentioning hyperparameters at all and only about a quarter stating tuning activities. In conclusion, our results reveal a significant need for a critical discussion on research and reporting practices in ML research, advocating the use of industry-strength experiment frameworks, such as MLflow and Wheights & Biases to properly tune ML models and enable better reproducibility of the results.

VIII. ACKNOWLEDGMENT

The work of the authors has been supported by the Federal Ministry of Education and Research of Germany and by the Sächsische Staatsministerium für Wissenschaft Kultur und Tourismus in the program Center of Excellence for AI-research "Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig", project identification number: ScaDS.AI, and by the BMBF project Agile-AI. Siegmund's work has been funded by the German Research Foundation (SI 2171/2-2).

REFERENCES

- [1] J. Chakraborty, S. Majumder, Z. Yu, and T. Menzies, "Fairway: a way to build fair ml software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 654–665.
- [2] M. Hutson, "Artificial intelligence faces reproducibility crisis," 2018.
- [3] S. Kapoor and A. Narayanan, "Leakage and the reproducibility crisis in ml-based science," *arXiv preprint arXiv:2207.07048*, 2022.
- [4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
- [5] M. N. Gevorkyan, A. V. Demidova, T. S. Demidova, and A. A. Sobolev, "Review and comparative analysis of machine learning libraries for machine learning," *Discrete and Continuous Models and Applied Computational Science*, vol. 27, no. 4, pp. 305–315, 2019.
- [6] Meta, "Pytorch developer ecosystem expands, 1.0 stable release now available," <https://engineering.fb.com/2018/12/07/ai-research/pytorch-developer-ecosystem-expands-1-0-stable-release/>, 2023, accessed: 2023-23-01.
- [7] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.
- [8] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, no. 1, pp. 1–16, 2016.
- [9] S. Shekhar, A. Bansode, and A. Salim, "A comparative study of hyperparameter optimization tools," in *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. IEEE, 2021, pp. 1–6.
- [10] K. Das and R. N. Behera, "A survey on machine learning: concept, algorithms and applications," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, no. 2, pp. 1301–1309, 2017.
- [11] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, no. 3, pp. 1–21, 2021.
- [12] H. J. Weerts, A. C. Mueller, and J. Vanschoren, "Importance of tuning hyperparameters of machine learning algorithms," *arXiv preprint arXiv:2007.07588*, 2020.
- [13] F. Hutter, J. Lücke, and L. Schmidt-Thieme, "Beyond manual tuning of hyperparameters," *KI-Künstliche Intelligenz*, vol. 29, no. 4, pp. 329–337, 2015.
- [14] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research*, vol. 13, no. 2, 2012.
- [15] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.
- [16] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*. Springer, 2011, pp. 507–523.
- [17] I. Dewancker, M. McCourt, and S. Clark, "Bayesian optimization for machine learning: A practical guidebook," *arXiv preprint arXiv:1612.04858*, 2016.
- [18] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.
- [19] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.
- [20] S. Sanders and C. Giraud-Carrier, "Informing the use of hyperparameter optimization through metalearning," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 1051–1056.
- [21] J. N. van Rijn, F. Pfisterer, J. Thomas, A. Muller, B. Bischl, and J. Vanschoren, "Meta learning for defaults: Symbolic defaults," in *Neural Information Processing Workshop on Meta-Learning*, 2018.
- [22] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: Bandit-based configuration evaluation for hyperparameter optimization," in *ICLR (Poster)*, 2017.
- [23] F. Pfisterer, J. N. van Rijn, P. Probst, A. C. Müller, and B. Bischl, "Learning multiple defaults for machine learning algorithms," in *Proceedings*

of the Genetic and Evolutionary Computation Conference Companion, 2021, pp. 241–242.

- [24] R. G. Mantovani, A. L. Rossi, J. Vanschoren, B. Bischl, and A. C. Carvalho, “To tune or not to tune: recommending when to adjust svm hyper-parameters via meta-learning,” in *2015 International Joint Conference on Neural Networks (IJCNN)*. Ieee, 2015, pp. 1–8.
- [25] N. Lavesson and P. Davidsson, “Quantifying the impact of learning algorithm parameter tuning,” in *AAAI*, vol. 6, 2006, pp. 395–400.
- [26] P. Probst, A.-L. Boulesteix, and B. Bischl, “Tunability: importance of hyperparameters of machine learning algorithms,” *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1934–1965, 2019.
- [27] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Identifying key algorithm parameters and instance features using forward selection,” in *International Conference on Learning and Intelligent Optimization*. Springer, 2013, pp. 364–381.
- [28] A. Biedenkapp, M. Lindauer, K. Eggenberger, F. Hutter, C. Fawcett, and H. Hoos, “Efficient parameter importance analysis via ablation with surrogates,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.
- [29] C. Fawcett and H. H. Hoos, “Analysing differences between algorithm configurations through ablation,” *Journal of Heuristics*, vol. 22, no. 4, pp. 431–458, 2016.
- [30] F. Hutter, H. Hoos, and K. Leyton-Brown, “An efficient approach for assessing hyperparameter importance,” in *International conference on machine learning*. PMLR, 2014, pp. 754–762.
- [31] J. N. Van Rijn and F. Hutter, “An empirical study of hyperparameter importance across datasets,” in *AutoML@ PKDD/ECML*, 2017, pp. 91–98.
- [32] —, “Hyperparameter importance across datasets,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2367–2376.
- [33] L. Li, J. Wang, and H. Quan, “Scalpel: The python static analysis framework,” *arXiv preprint arXiv:2202.11840*, 2022.
- [34] J. Cohen, “A coefficient of agreement for nominal scales,” *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.
- [35] J. R. Landis and G. G. Koch, “The measurement of observer agreement for categorical data,” *biometrics*, pp. 159–174, 1977.
- [36] W. Fu, T. Menzies, and X. Shen, “Tuning for software analytics: Is it really necessary?” *Information and Software Technology*, vol. 76, pp. 135–146, 2016.
- [37] A. Bagnall and G. C. Cawley, “On the use of default parameter settings in the empirical evaluation of classification algorithms,” *arXiv preprint arXiv:1703.06777*, 2017.
- [38] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler *et al.*, “Api design for machine learning software: experiences from the scikit-learn project,” *arXiv preprint arXiv:1309.0238*, 2013.
- [39] G. Douzas, F. Bacao, and F. Last, “Improving imbalanced learning through a heuristic oversampling method based on k-means and smote,” *Information Sciences*, vol. 465, pp. 1–20, 2018.
- [40] M. Bahmani, R. E. Shawi, N. Potikyan, and S. Sakr, “To tune or not to tune? an approach for recommending important hyperparameters,” *arXiv preprint arXiv:2108.13066*, 2021.
- [41] D. Passos and P. Mishra, “A tutorial on automatic hyperparameter tuning of deep spectral modelling for regression and classification tasks,” *Chemometrics and Intelligent Laboratory Systems*, p. 104520, 2022.
- [42] T. Xu and Y. Zhou, “Systems approaches to tackling configuration errors: A survey,” *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 1–41, 2015.
- [43] T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadkar, “Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 307–319.
- [44] H. Zhang, L. Cruz, and A. van Deursen, “Code smells for machine learning applications,” *arXiv preprint arXiv:2203.13746*, 2022.