How Green is AutoML for Tabular Data?

Felix Neutatz
BIFOLD & TU Berlin
Germany
f.neutatz@tu-berlin.de

Marius Lindauer
L3S & Leibniz Universität Hannover
Germany
lindauer@tnt.uni-hannover.de

Ziawasch Abedjan
BIFOLD & TU Berlin
Germany
abedjan@tu-berlin.de

ABSTRACT

AutoML has risen to one of the most commonly used tools for day-to-day data science pipeline development and several popular packages exist. While AutoML systems support data scientists during the tedious process of pipeline generation, it can lead to high computation costs that result from extensive search or pre-training. In light of concerns with regard to the environment and the need for Green IT, we holistically analyze the computational cost of pipelines generated through various AutoML systems by combining the cost of system development, execution, and the downstream inference cost. Our findings show the benefits and disadvantages of implementation designs and their potential for Green AutoML.

KEYWORDS

Green AutoML, Green Data Processing

1 GREEN DATA SCIENCE

With the development of data-intensive approaches and systems, we face a heavy computation challenge, often leading to heavy energy consumption and a critical CO_2 footprint of AI systems [52, 69]. In particular, with the rise of machine learning (ML), these problems have been aggravated because of computationally expensive hyperparameter optimization (HPO) and model training to achieve optimal predictive quality. While there has been prior work that considered the computation of databases and data centers and their impact on energy consumption [54], for the area of data science pipelines and ML systems, energy consumption has not been compared across different systems. Yet, many existing data-intensive solutions [4, 5, 9, 30, 32, 48, 57, 59, 70, 73] rely on the additional ML steps. Nevertheless, automated machine learning (AutoML) and its HPO have been shown to relieve data scientists from tedious and error-prone tasks and have become a powerful component of ML projects [15, 19, 36, 40, 72]. From our own practical experience with researchers and also developers at companies, application experts sometimes spend weeks or in the worst case even months figuring out how to design their ML pipelines s.t. they perform well even on small datasets. Instead of spending weeks or months to find the optimal pre-processing steps, models, and their corresponding hyperparameters, it is possible to leverage an AutoML system of choice and run it for a specified compute budget. This way, the AutoML system automatically searches the space of ML pipelines to identify one or an ensemble of promising pipelines. This automation relieves the data scientists of manual hyperparameter optimization and frees up time to focus on the data (data-centric ML [45]) - collecting additional data instances and features, cleaning the data, and adjusting the objective functions. One drawback of such automated systems is the computationally expensive search process and the associated energy consumption. Tornede et al. [62] proposed *Green* AutoML as a paradigm aimed at enhancing the overall environmental sustainability of the AutoML process. As AutoML is increasingly included in data processing systems and applications [57, 58], it is natural for the database community to explore its footprint with regard to resource consumption. We investigate empirically how *green* current AutoML systems are and how we can reduce their energy consumption. In this paper, we consider the consumed energy as a proxy for CO2 emissions because CO2 emissions per kWh differ significantly across countries and institutions because countries have different energy source distributions and institutions might offset their CO2 emissions [65]. We limit the scope of our evaluation to tabular data because it is the most studied data modality by AutoML systems [15, 19].

Tornede et al. identify three different stages where AutoML is subject to or influences energy consumption: developing and configuring an AutoML system, executing an AutoML system, and predicting with the resulting ML pipeline.

Development: Developing AutoML systems consists of two phases: implementing the components of the system and setting the parameters for these components. The energy consumption during implementation does not only depend on the developing time but also on energy consumed during unit and integration testing. Configuring an AutoML system and its many parameters, such as the search space, the search strategy, and the validation strategy requires extensive trial and error iterations. Some AutoML systems [19, 20, 26, 55] leverage meta-learning that requires often a large amount of meta-data, e.g., AutoML training runs. As a result, AutoML system development and configuration remain resource-intensive. Yet, the premise of AutoML proponents is that over time the energy consumption of advanced AutoML systems amortizes in comparison to more simple, inefficient search strategies, such as grid or random search [2, 64].

Execution: Most research on AutoML systems focuses on making the execution stage more efficient, i.e., finding an ML pipeline or ensemble that achieves high predictive performance as fast as possible. They highlight various strategies to speed up the execution of AutoML, such as warm starting [20], multi-fidelity optimization [18], few-shot learning [26], and ensembling [15, 20].

Inference: The energy consumed during prediction depends on the chosen ML pipelines. For instance, an ensemble of models typically requires more energy than a single model. ML models can also be arbitrarily complex, which affects the required energy for prediction. Further, ML pipelines can also have significant preprocessing steps that require additional energy for prediction. One way of reducing the energy consumed during inference is to consider CO2 emissions as a constraint during search for the ML pipelines. For instance, we can incorporate this constraint in the objective function [47]. This way, AutoML will focus on discovering ML pipelines within a specified consumption budget. So, if the model is used often, we can significantly reduce the CO2 footprint by reducing the CO2 emission caused by prediction. So

far, research attempted to optimize the energy consumption of individual stages, disregarding the potential negative impact on the other stages.

Holistic Consideration: Energy savings in one stage might lead to additional energy consumption in another stage. For instance, meta-learning strategies, such as warm starting or few-shot learning, require heavy resources in the development stage while allowing the AutoML system to significantly speed up the execution stage and save energy there. However, meta-learning, as by Feurer et al. [20] on warm starting Bayesian optimization (BO), does not necessarily have an impact on the energy consumption during inference. Hollmann et al. [26] meta-learn a transformer that learns on many synthetically generated datasets to apply supervised classification. As the transformer model is big and requires forward propagating of the training data during inference, the inference requires significant energy. So, the impact on resource consumption for inference depends on the meta-learning strategy. Another strategy to save energy in the execution stage is ensembling [15, 20] because we do not need to find a single best model, but instead aggregating the predictions of many good models is often sufficient and can thus help to speed up AutoML. The disadvantage of ensembling large models is the increasing energy consumption during inference: The more (complex) models an ensemble combines, the larger the inference energy consumption. The choice of when to follow which strategy to trade-off energy consumption and model performance often depends on the application context. Different tasks might require a different strategy for reducing energy consumption across stages. For instance, running a fraud detection model on millions of bank transactions might require a focus on inference energy consumption. Predicting whether a patient has a specific kind of cancer might happen far less often, and thus, the focus could be on execution efficiency. Depending on the application and the perspectives of all stakeholders, one would invest more energy first to avoid costly inference later or vice versa.

1.1 Contributions

Motivated by the aforementioned observations, we conduct an in-depth and holistic analysis of current state-of-the-art AutoML systems – as representative systems of different AutoML approaches – to understand the energy consumption trade-offs in the different stages of AutoML. We focus on tabular data with numeric and categorical attributes because this is the most well-known AutoML use case [15, 19, 51]. We make the following contributions:

(1) We report extensive experiments with state-of-the-art AutoML systems on commonly used benchmark datasets and evaluate the trade-off between performance and energy consumption.
(2) From our experimental results, we derive concrete suggestions on when to use which strategy and show how adjusting AutoML system parameters can reduce energy consumption during inference. We also provide all implementations and the evaluation framework in our repository [46].

Our study allows us to reach the following conclusions:

Observation O1: AutoML systems [15, 20] that leverage ensembling selection [6], require at least one order of magnitude more energy compared to using one model during inference.

Observation O2: For a use case with rather small predictions - fewer than 26k in our experiments -, few-shot AutoML systems, such as TabPFN, are most energy efficient because it does not search for hyperparameters. If an AutoML system is supposed to be executed very often beyond simple exploration - in our experiments more than 885 times -, it is advised to optimize the AutoML parameters in the development stage. We show that we can optimize the AutoML system parameters, e.g. for CAML [49], and achieve higher predictive performance using less energy compared to all state-of-the-art systems [15, 20]. This finding shows that further work on Green AutoML should not neglect the emissions of development.

Observation O3: Fine-grained support of ML application constraints [49], such as inference time, allows the user to reduce energy consumption while choosing the ML pipeline that achieves high predictive performance.

Observation O4: In accordance with findings by Li [38], the selection of an appropriate parallelization strategy depends on the user's preferences and the AutoML system (the workload). If energy consumption is the priority then running it on one core is the best decision for CAML and running it on multiple cores for AutoGluon. If the priority is time efficiency, the user should use all available cores because this approach yields the best predictive performance and requires only sublinear energy increases.

In Section 5, we discuss the generalizability of our findings.

2 BACKGROUND

We describe the AutoML problem and the main components of current state-of-the-art AutoML systems to understand how their design decisions affect their energy consumption. We discuss for each AutoML stage the possibilities and pitfalls with regard to improving energy efficiency. Finally, we explain how to measure the "greenness", i.e., the energy efficiency, of AutoML systems.

2.1 AutoML Problem

The general AutoML problem comprises pipeline construction and HPO [20, 51, 61]. The problem is to identify the ML pipeline $a \in A$ and its corresponding hyperparameters $\boldsymbol{\lambda} \in \Lambda$ that leads to a trained model $f^{(a_{\boldsymbol{\lambda}}; D_{train})}(\cdot)$ on the training dataset D_{train}, that achieves the lowest loss $\mathcal{L}_{val}$ across all labeled instances (x_i, y_i) of the validation set D_{val}. Formally: $\arg\min_{a \in A, \boldsymbol{\lambda} \in \Lambda} \sum_{(x_i, y_i) \in D_{val}} \mathcal{L}_{val}(y_i, f^{(a_{\boldsymbol{\lambda}}; D_{train})}(x_i))$.

In addition to reporting the loss/predictive performance (balanced accuracy) on the test set, we report the consumed energy for both executing the AutoML system using training and validation data and predicting the test data.

2.2 AutoML Systems

Figure 1 describes the general AutoML process. First, the AutoML system's users have to define the ML configuration space, such as which classifier, preprocessor, and corresponding hyperparameters should be optimized (or use default settings). During the search initialization, the system has to evaluate an initial set of configurations. Then, the search can leverage this initial set and learn which configurations should be evaluated next. After a specified search time has elapsed, the AutoML system can leverage the models trained during the search in an ensemble and finally return the predictions to the users. To provide a broad analysis of Green AutoML, we use representative, state-of-the-art packages for the different AutoML approaches. These AutoML approaches are ensembling [15], Bayesian optimization [19, 20], cost-effective optimization [67], few-shot AutoML [26], and constraint-aware AutoML [49].

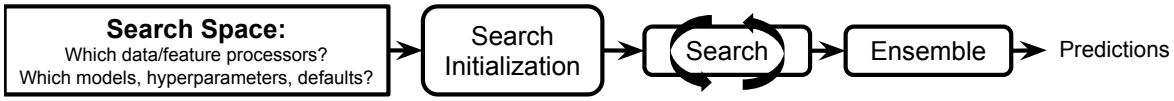

Figure 1: The AutoML Process - Search Space Setup, Initialization, Search, and Ensembling

AutoGluon (0.6.2) [15]: The system leverages ensembling to its fullest potential. First, it trains a number of predefined models. Then, it leverages the predictions of these models in another layer of the same models. This way all models have access to all information from the other models of the lower layers. Finally, AutoGluon leverages the ensembling proposed by Caruana [6] to weigh the predictions of the models of the final stacking layer. *We chose AutoGluon for its top performance in the AutoML benchmark [21] and its effective use of ensembling.*

AutoSklearn (ASKL) 1 & 2 (0.14.7) [19, 20]: The system leverages Bayesian optimization (BO) to find ML pipelines, consisting of data/feature preprocessors and the model, that optimize validation accuracy. By default, BO is initialized with random search. They further propose a warm starting strategy to improve on random search. The warm starting is based on an offline search for predictive ML pipelines on 140 repository datasets each for 24h. For a new dataset, they then use the most similar dataset based on selected metadata features and use the best-performing ML pipelines for the corresponding dataset as initial evaluation candidates for the new dataset. To improve the predictive performance, they leverage the ensembling approach proposed by Caruana [6] to weigh the predictions of the top 50 ML pipelines evaluated during search. *We choose ASKL 1 & 2 because they are the classic examples for BO.*

FLAML (1.2.4) [67]: The system was developed to yield single low-cost models, e.g., small random forest models. Therefore, they enforce the prior of cost during search. They start by evaluating low-cost models, e.g. a random forest with 5 trees with at most 10 leaves each, and they evaluate these models on small training sets. Then, they incrementally, increase the model complexity. Once increasing model complexity does not yield more accuracy gains, they increase the training set size and repeat the process. *We choose FLAML because it searches for a single low-cost model in contrast to most systems that prioritize predictive performance.*

TabPFN (0.1.9) [26]: The system is a few-shot AutoML system. It requires only a few labeled instances to make predictions for the remaining instances of a new dataset. TabPFN does neither require model training nor HPO during execution for a new dataset. As a few-shot model, transformer model learned on synthetic datasets how to predict new instances given a number of labeled examples. *We choose TabPFN because it is a recent system that allows for zero-shot AutoML without explicit training.*

TPOT (0.11.7) [51]: The system leverages genetic programming to find ML pipelines. It starts with random ML pipelines and iteratively evolves them using NSGA-II [8]. *We choose TPOT because it is the most well-known AutoML system that leverages genetic programming.*

CAML [49]: Similar to ASKL, CAML uses BO. It is designed to allow for dynamic adjustment of its own AutoML parameters dependent on the provided dataset and user-provided constraints, such as inference/training time and fairness. It leverages successive halving to prune ML pipelines that violate constraints as early as possible. Finally, to avoid overfitting of BO, it leverages validation split resampling. It can be used in static or dynamic mode. The dynamic mode leverages meta-training to optimize the search space based on the user-provided constraints. In this work, we leverage a static version of CAML, which does not optimize the search space through meta-training to exclude meta-training costs from our calculation. *We choose CAML because it enables the compilation of various constraints, such as inference time, as first-class citizens.*

2.3 Energy efficiency of different AutoML strategies

Table 1 describes how the aforementioned AutoML systems implement the steps illustrated in Figure 1. We describe how different designs of the AutoML steps positively or negatively affect energy consumption on a conceptual level.

(1) Search Space: The AutoML system users have to define the space of all ML pipelines - *the search space*. This space determines the data/feature preprocessors, models, and corresponding hyperparameters. ASKL [20] supports the search space of 15 models, 13 feature preprocessors, and 4 data preprocessors. CAML supports the same space without the feature preprocessors. The developers of AutoGluon [15] identified promising pipeline candidates upfront. FLAML [67] supports lightweight models. TabPFN [26] does not have any search space because it trains a transformer model. The costs for training the transformer fall into the development stage. Avoiding search - as does TabPFN - saves the most energy during execution. As AutoGluon has predefined candidate pipelines, it drastically reduces the search space. This way, it avoids out-of-time exceptions. ASKL's search space might produce pipelines s.t. even the very first pipeline cannot be executed in the specified time.

(2) Search Initialization: The most naive way to initialize the search is to choose the initial ML pipelines randomly. For instance, CAML first evaluates 10 random ML pipelines. To improve on random initialization, FLAML focuses on low-complexity models on a very small training set, ASKL [20] applies meta-learning, and AutoGluon [15] starts with manually picked default pipelines. The least energy-efficient option is random initialization because it potentially wastes computation on poorly performing pipelines. The warm starting approach through meta-learning or manual selection of initial pipelines is more efficient. However, again the computation is moved to the development stage. For ASKL, Feurer et al. [20] searched for predictive ML pipelines for 140 datasets each for 24h. For AutoGluon, Erickson et al. [15] do not specify how they identified the hand-picked ML pipelines – it will not be negligible.

(3) Search: explores the space of ML pipelines. The most common approach is Bayesian Optimization [20], e.g. used in ASKL and CAML, which leverages a surrogate model to learn the most promising hyperparameter settings for high predictive performance. An alternative is genetic programming as leveraged by TPOT. Modern BO-based AutoML systems further accelerate the validation process through multi-fidelity [28, 39]. An inherent risk of search is the overfit to the validation set. There are multiple ways to avoid overfitting caused by search, e.g., avoiding search in the first place like TabPFN. The second approach is to reduce the parameter space as done by AutoGluon and FLAML.

Table 1: The search space of each AutoML system and the applied strategy in each execution stage

System	Search Space	Search Init.	Search	Ensembling
ASKL	data/feature p. & models	warm starting	BO (random forest)	Caruana
AutoGluon	predefined pipelines	manual	predefined pipelines	Caruana & bagging & stacking
CAML	data p. & models	random	BO & successive halving	-
TabPFN	-	-	-	unweighted ensemble
FLAML	models	low complexity models	cost-based	-
TPOT	data/feature p. & models	random	genetic programming	-

AutoGluon explicitly only relies on ML pipelines evaluated during the search initialization. FLAML uses a prior to navigate the search from low-complex models toward high-complex models, implicitly leading to much simpler models that do not use the entire parameter space. Finally, one can reshuffle the validation set in each evaluation [37]. In summary, avoiding search is the most energy-efficient approach. Yet, this would generally come at the cost of more effort during the development phase.

(4) Ensembling: To further improve model performance, we can employ ensembling of several pipelines. In fact, ensembling can also be used as an alternative for parameter search. Even if the best-found ML pipeline is overfitting, an ensemble of ML pipelines can alleviate this problem [11]. There are multiple ensembling approaches. ASKL and AutoGluon leverage both the ensembling approach proposed by Caruana et al. [6]. Additionally, AutoGluon uses bagging and stacking to leverage the information of each model of the lower stacking layers in all models of the higher stacking layers. While ensembling improves generalization, it also requires more energy for inference. The more models in the ensemble, the more energy is required for prediction.

2.4 Measuring Environmental Impact

For Green AutoML, the CO_2 emissions are the most relevant measure. However, CO_2 emissions depend on the electricity mix of each institution and/or country. For instance, a company might offset the electricity for its entire cluster or even rely on 100% self-produced renewable energy. We can measure the impact of AutoML systems through different proxies: runtime, CPU/GPU hours, floating point operations, energy consumption, or CO_2 emissions [62]. All of these measures are hardware dependent and have different difficulty levels with regard to taking the measurements. Therefore, we measure the environmental impact of AutoML systems by their consumed energy - kWh. As we do not have physical access to the machines, we have to approximate the energy consumption and thus use the library CodeCarbon [16]. For instance, it leverages Intels Running Average Power Limit (RAPL) interface that provides energy readings for CPUs and DRAM [10, 31]. It also accesses Nvidia drivers to track GPU energy consumption. This way, approximating the energy for AutoML execution and inference is straightforward. However, it is not trivial to estimate the cost of the development stage.

2.5 Approximating AutoML Development Cost

Developing AutoML systems includes the tuning of its parameters, such as the validation strategy and the search space. Usually, AutoML system users identify those by testing different parameter settings on hundreds of datasets with various search budgets, which leads to huge computational costs - also known as graduate student descent. In contrast to the execution and inference stage, the analysis of the development stage can only be approximated as it depends on the user and other unpredictable factors. In view of tremendous computational costs that are associated with studying the development phase of different AutoML systems, we limit ourselves to CAML as a representative AutoML system because it strictly enforces the search time budget and allows non-invasive fine-grained adjustments of all ML hyperparameters. The most naive approach to tuning CAML's hyperparameters would be to take a large number of datasets, and leverage BO to find the best AutoML system parameters for a given search time.

To run BO, we need an objective function. The most simple objective would be to compare a given set of AutoML parameters with the default parameters, e.g. full search space and 0.33 hold validation. Similar to algorithm configuration [13, 42], we can use the following equation to make the results comparable across datasets: $\max_{\omega \in \Omega} \sum_{d \in D} (Acc(\omega, d) - Acc(\omega_{default}, d)) / \max(Acc(\omega, d), Acc(\omega_{default}, d))$, where for each dataset d of all datasets D, we sum up the relative difference of the predictive accuracies using the AutoML parameters ω and the default parameters $\omega_{default}$. This way, we enable a fair aggregation across all datasets. Since AutoML is nondeterministic, we choose to run the AutoML system two times as part of the tuning process to reduce the variance without introducing excessive computation overhead.

However, naive BO optimization for this meta-objective requires a lot of computation. Following ideas from instance-specific algorithm configuration [29], we can reduce the cost by relying on the most representative datasets instead of using all datasets. As illustrated in Figure 2, to gather the top-k most representative datasets, we cluster the datasets based on metadata features, such as the number of features, instances, and classes. For each K-Means centroid, we pick the closest dataset. It turns out that for poor-performing AutoML parameters, evaluating a few datasets is sufficient to detect that the parameters are not performing well [27]. To leverage this insight, we use median pruning.

The disadvantage of this approach is that the resulting AutoML parameters are search-time dependent. For instance, as our experiments in Section 3.7 show, for a search time of 30s, an AutoML parameter setting of a small search space consisting of a few classifiers performs better, while for a longer search time a larger search space is better. For each user-specified search time, one would need to run the optimization process. To limit the number of optimization cases, we consider common AutoML use cases, where the search time is either in the order of minutes for ad-hoc development or very large, such as 1 hour.

Also note that the result of this approach is hardware-dependent. Therefore, it is more tailored towards cloud environments where one has the same or similar hardware setup.

3 EXPERIMENTS

Our experiments aim to provide a comprehensive analysis of how much energy existing AutoML systems consume during development, execution, and inference. Our hypothesis is that as most

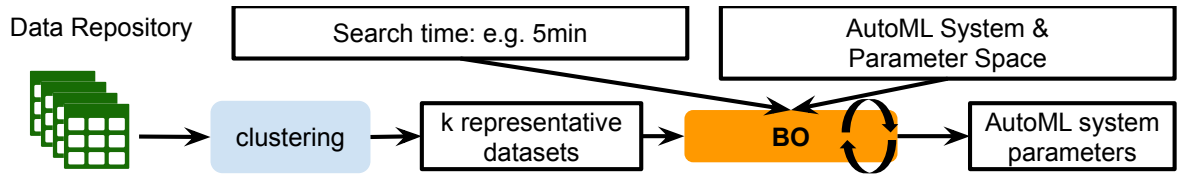

Figure 2: Optimizing AutoML for one search budget.

Table 2: OpenML Test datasets.

Name	DatasetID	# instances	# features	# classes
robert	41165	10000	7200	10
riccardo	41161	20000	4296	2
guillermo	41159	20000	4296	2
dilbert	41163	10000	2000	5
christine	41142	5418	1636	2
cnae-9	1468	1080	856	9
fabert	41164	8237	800	7
Fashion-MNIST	40996	70000	784	10
KDDCup09_appetency	1111	50000	230	2
mfeat-factors	12	2000	216	10
volkert	41166	58310	180	10
APSFailure	41138	76000	170	2
jasmine	41143	2984	144	2
nomao	1486	34465	118	2
albert	41147	425240	78	2
dionis	41167	416188	60	355
jannis	41168	83733	54	4
covertype	1596	581012	54	7
MiniBooNE	41150	130064	50	2
connect-4	40668	67557	42	3
kr-vs-kp	3	3196	36	2
higgs	23512	98050	28	2
helena	41169	65196	27	100
kc1	1067	2109	21	2
numerai28.6	23517	96320	21	2
credit-g	31	1000	20	2
sylvine	41146	5124	20	2
segment	40984	2310	16	7
vehicle	54	846	18	4
bank-marketing	1461	45211	16	2
Australian	40981	690	14	2
adult	1590	48842	14	2
Amazon_employee_access	4135	32769	9	2
shuttle	40685	58000	9	7
airlines	1169	539383	7	2
car	40975	1728	6	4
jungle_chess_2pcs_raw_endgame_complete	41027	44819	6	3
phoneme	1489	5404	5	2
blood-transfusion-service-center	1464	748	4	2

AutoML systems are highly optimized for the execution stage and lack such considerations for the inference stage, their energy consumption impact is unforeseeable in production. We analyze the systems introduced in Section 2 with regard to various parameters that influence the energy consumption, such number of predictions, execution time, parallelization, and hardware environment. To assess inference efficiency, we analyze how AutoML parameters, such as ensemble size and inference time constraints, could impact the energy efficiency. Further, we evaluated the potential benefits of investing energy during the development stage for later stages. For this experiment, we also provide an in-depth analysis of the AutoML parameters that originate from our optimization approach in Section 2.5. For this optimization approach, we also benchmark how different numbers of representative datasets and the number of Bayesian optimization (BO) iterations affect the performance.

3.1 Setup

We evaluate all systems on the 39 datasets (Table 2) as proposed by Gijsbers et al. [22], which is a commonly used AutoML benchmark [19]. Each dataset is split into a training/test set (66/34). Each AutoML system (automatically) chooses its own validation approach and uses a subset set of the training data as the validation set. As a prediction accuracy metric, we leverage balanced accuracy that can handle multi-class and unbalanced classification problems. We report the average performance across datasets by repeatedly sampling one result out of 10 runs with replacement. This approach ensures that we report the uncertainty induced by AutoML systems. We use an Ubuntu 16.04 machine with 28 × Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz cores, 264 GB RAM. For GPU experiments, we leverage a Linux-6.1.58 machine with 8 × Intel(R) Xeon(R) CPU @ 2.00GHz cores, 1 × T4 GPU, 51 GB RAM.

3.2 Energy consumption for Execution and Inference (Single Core)

To measure the energy efficiency of AutoML systems, we follow existing AutoML benchmarks [21, 22] and run the systems with search time as the termination criteria and proxy for the expected predictive performance of the generated pipelines. We run each system for the search times 10s, 30s, 1min, and 5min. Note that these are the search budget times that we provide the AutoML systems to execute. In some cases, the actual search time deviates from the budget as we report in Section 3.10. This setup took already 28 days of computation. Longer search times are contrary to the objectives of Green AutoML [62]. During the execution of each AutoML system, we measure the consumed energy in kWh. Finally, we measure the consumed energy for inference and report the kWh per predicted instance. We choose kWh because it is the most commonly used unit in energy-related fields [33, 43, 62, 69], making the data more directly relevant and understandable to practitioners and researchers alike. Figure 3 shows two charts that depict the corresponding relationship between search time, balanced accuracy, and their impact on the energy consumption during execution and inference, respectively. While the dimensions of the charts capture the relationship between energy consumption and balanced accuracy the corresponding search time of each approach is appended to the measured coordinates in the chart. The connection between runtime to accuracy and energy is established through each point. For each line of the graph, we have four points that resemble search times of 10s, 30s, 1min, and 5min. Each of such points is plotted within the two dimensions of accuracy and energy for each corresponding stage.

For CAML, AutoGluon, and FLAML, we benchmark the search times 10s, 30s, 1m, and 5m. The other systems do not support such short search times. Therefore, we benchmark AutoSklearn 1 & 2 starting at 30s and TPOT starting at 1m. TabPFN does not have a search time parameter. Therefore, we only report one measurement for TabPFN. Next, we discuss the results for each approach in detail. **TabPFN** is represented by a single dot on each chart of Figure 3 as it is independent of search time. As shown in the first chart, it requires the least amount of energy for execution compared to the other systems. TabPFN only loads the transformer model into memory and does not require any search. However, it requires orders of magnitude more energy than the competitors during inference, as depicted in the right-hand chart, because it has to feed forward the training data through its transformer network. We note that TabPFN achieves relatively low

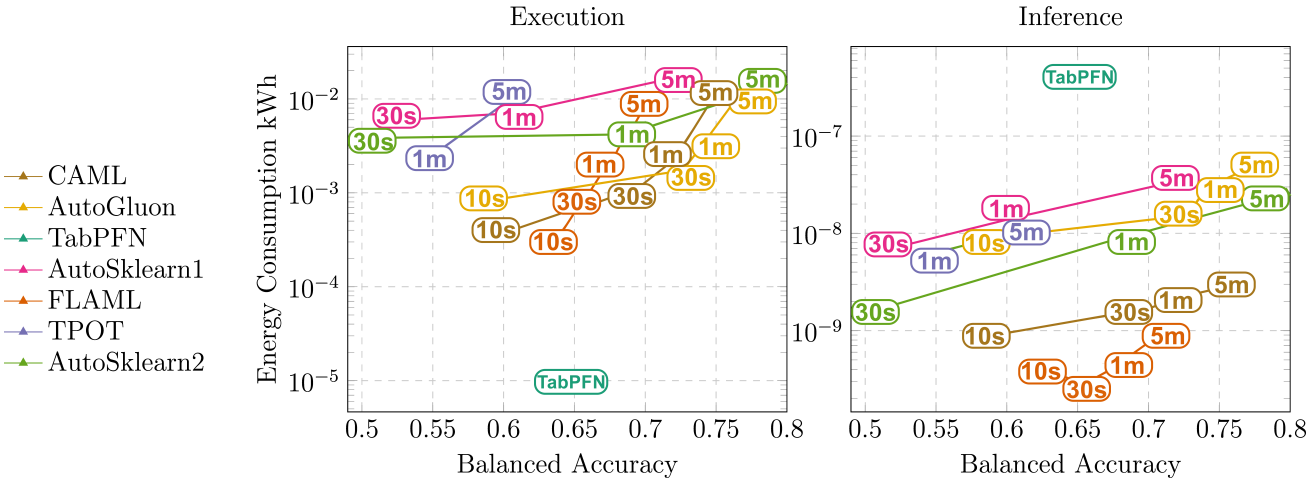

Figure 3: Search time, average balanced accuracy, and their impact on the energy consumption (kWh) during execution and inference for each AutoML system. Each marked point represents one of these execution time budgets.

predictive performance because the currently official implementation of TabPFN only supports up to 10 classes (and was mainly developed for datasets with up to 1k instances). If we exclude datasets with more than 10 classes its performance is still 7% higher below the best-performing systems for more than 30s search time. **AutoGluon** converges to the best predictive performance because ensembling leverages the rich information of all evaluated models in contrast to FLAML and CAML, which only search for the best model. However, stacking ensembling also leads to one order of magnitude higher energy consumption for inference. **CAML**'s execution shows higher energy efficiency for small search times 30s and 60s because it leverages successive halving to quickly achieve high predictive performance - especially for large datasets. As CAML only searches for one ML pipeline instead of an ensemble, it requires one order of magnitude less energy for inference compared to AutoGluon and ASKL. **ASKL** requires the most energy for execution because, in addition to searching for ML pipelines, it runs ensembling that computes the weights for each evaluated model. For large validation sets, this step requires significant time and therefore energy. **FLAML** achieves the second lowest prediction performance in 5 min of the search-based approaches because it converges only slowly to more complex models that are more predictive and it does not leverage ensembling. However, since it starts to search models with low complexity, it achieves the lowest inference energy consumption. **TPOT** only supports search time in minutes. So, we only report two measurements for 1 and 5 min. TPOT achieved the lowest accuracy within 5 min of search time because it uses 5-fold cross-validation whereas most other systems use hold-out validation.[1]

3.2.1 Dataset-Level Predictive Performance. In addition to analyzing the average results across datasets for each AutoML system (Figure 3), we also conducted a dataset-level analysis on which AutoML system performs best across search times for each of the 39 test datasets. Due to space limitations, we provide both the raw results of all 10 runs for all search times, datasets, and systems, and the dataset-level analysis in our repository [46].

[1]For cross-validation, we split the data into k folds where each fold is used once for testing and the remaining folds are used for training - the k test scores are averaged as the cross-validation score [3].

The main findings of this analysis is that there is no clear patterns for when to use which AutoML system. However, there are trends. For small search times, i.e. 10s, FLAML (16/39) and TabPFN (12/39) achieve the highest predictive performance. FLAML is well-designed for this use case because it searches smaller models first and the few-short approach TabPFN fits this use case well too. With increasing search times, ensemble-based models achieve higher predictive performance. For 5 mins, ensemble-based approaches achieve the best predictive performance for 23 out of 39 datasets. We also analyzed the performance with respect to the data characteristics: number of rows, features, and classes. We found that TabPFN works particularly well for small datasets (<1k rows, <20 features). The reason is that this corresponds to the training domain of the transformer model. Further, FLAML performs well for large number of features (> 2k). The reason is that they designed a feature pruning strategy. For large number of classes (>100), ensemble-based systems perform well because they can cover more complexity using multiple models. Further, we also analyzed how large the energy consumption during AutoML execution varies across datasets. For instance, CAML has the lowest standard deviation across datasets for 5min (0.0007kWh). For AutoGluon, it is 0.0025kWh. The reason is that AutoGluon builds always the same ensemble. If the data is small, the ensemble is trained faster and less energy is consumed. CAML searches always until the search budget is exhausted and therefore needs the same energy independent of the dataset.

3.2.2 Energy-Focused System Recommendation. Section 3.2 showed that there are significant differences across AutoML systems in energy consumption during both execution and inference. For instance, TabPFN requires almost no energy for execution but requires a comparably large amount of energy for inference. To understand better when to use which AutoML system to achieve the least energy cost, we estimate the energy consumption in kWh for both execution and inference jointly for an increasing number of predictions and report the results in Figure 4. For fewer than 26k predictions, TabPFN is the most energy efficient because it requires almost no energy for execution. If the ML application requires more predictions, FLAML and CAML are more suited as their energy consumption during inference is significantly lower and the execution overhead compared to TabPFN is amortized.

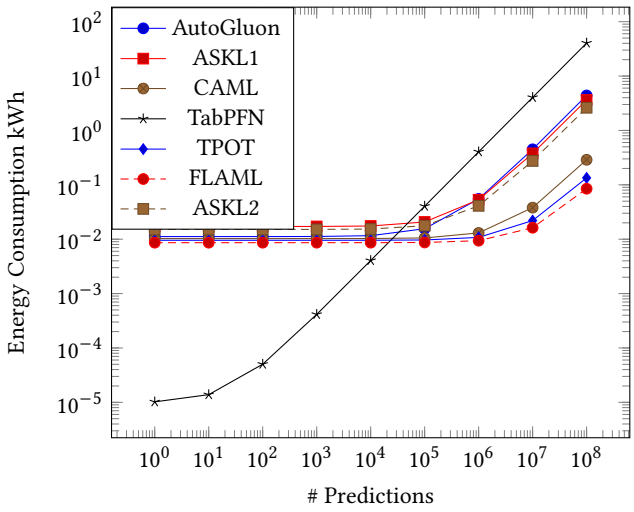

Figure 4: Comparing AutoML systems with respect to energy consumption for different scales of prediction instances.

The reason for this result is that they search smaller models first, while TabPFN uses a transformer network.

3.3 Impact of Parallelism

For multi-processor computers, the question arises whether it is more energy efficient to use one or multiple cores. To answer this question, we run AutoGluon and CAML for 10s, 30s, 1min, and 5min using 1, 2, 4, or 8 cores, where each CPU has two cores. So, each point in the graph represents one AutoML system running for a specific search time with a specific number of cores resulting execution energy consumption and a corresponding average balance accuracy. Figure 5 shows that we gain only slight improvements in average predictive performance (up to 2%) using more cores. In all cases, using more cores led to higher prediction accuracy. So, when it comes to time efficiency, leveraging more cores is always better. For CAML, we see that increasing up to 8 cores requires up to 2.7x the energy compared to 1 core. The reason for this sublinear increase in energy is that the computer can leverage caching as we use the same data. However, using 1 core is Pareto optimal because CAML leverages BO, which works sequentially. For AutoGluon, we see that more cores (4 and 8) are more energy efficient because it parallelizes the training for bagging, which is an embarrassingly parallel workload. Less runtime yields less consumed energy. The right choice of parallelism depends on the user's priorities and the AutoML system. If energy consumption is the priority then running it on one core is the best decision for CAML and running it on multiple cores for AutoGluon. If the priority is time efficiency, the user should use all cores because this approach yields the best predictive performance and requires only sublinear energy increases.

3.4 Configuring AutoML Systems for Inference

One simple way to reduce the Energy in AutoML pipelines is to limit search time because less computation time results in less energy consumption. Reducing the consumed energy during inference is not as simple and depends on the capabilities of the AutoML system at hand. Some AutoML systems provide parameters that affect the runtime and energy consumption during inference. For instance, AutoGluon builds one model for each fold to yield the bagging features. If we refit these models into one model afterward, we can save energy during inference. Therefore, AutoGluon provides settings targeted specifically to ensure faster inference: 'good quality faster inference only refit'; one will get models faster models that require less energy. For AutoGluon, refit means collapsing the bagged ensemble models into a single model that is trained on all training data [17]. CAML allows the user to provide ML application constraints, including one for inference time. Thus, by lowering the allowed limit for the inference time, one will get models faster models that require less energy.

Table 3: Experiments with and without GPU acceleration. For each metric, we report the fraction GPU result/CPU only result. We mark metrics green if the GPU setup is better and red if the CPU setup is better.

System	Execution		Inference	
	Energy	Time	Energy	Time
AutoGluon	1.35	1.03	2.39	1.96
TabPFN	1.37	0.96	0.13	0.07

Figure 6 shows the results of these optimizations. For CAML, we simply set the inference time constraints with the values 0.001s, 0.002s, and 0.003s. For AutoGluon, we use the described inference-optimized setting. For both systems, optimizing the inference speed comes at the cost of less predictive performance. The reason is that higher inference speed requires lower complexity that reduces the descriptiveness of the models and therefore predictive performance. Using the most strict inference time of CAML constraint of 0.001s, we can save up to 69% of the consumed energy. At the same time, the average predictive performance is up to 6% lower compared to without constraints. Depending on the requirements of the ML application, the user can adjust the constraint accordingly. The inference-optimized AutoGluon can save up to 79% of the inference energy. At the same time, the average predictive performance is up to 5% lower compared to no optimization. The inference speed-up is significant but even the optimized AutoGluon requires more energy than CAML without constraints because AutoGluon still uses ensembling. In conclusion, decisions in the execution stage affect the performance in the later stage - inference.

3.5 Impact of Modern Hardware

Specialized hardware accelerators, such as GPUs or TPUs, improve the energy efficiency of modern ML workloads [68]. However, most AutoML systems, such as ASKL, TPOT, and CAML, are based on the scikit-learn stack that does not support GPUs or TPUs. TabPFN is an exception because it employs a transformer model. AutoGluon supports a few models with GPU support. Therefore, we executed both systems with and without GPU support and report for execution time/energy and inference time/energy the quotient $GPU/CPUonly$. The results are presented in Table 3. For AutoGluon, we compare the two setups with a budget of 5 min. For TabPFN, the inference requires both significantly less time (16x) and energy (8x). The reason is that the transformer

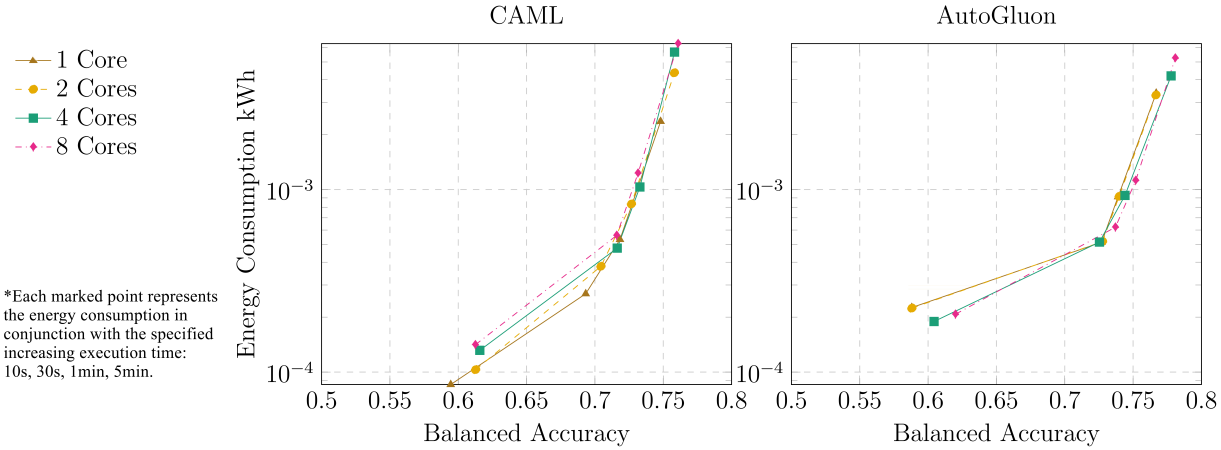

Figure 5: Average balanced accuracy and CPU energy consumption (kWh) during execution of CAML and AutoGluon across CPU cores.

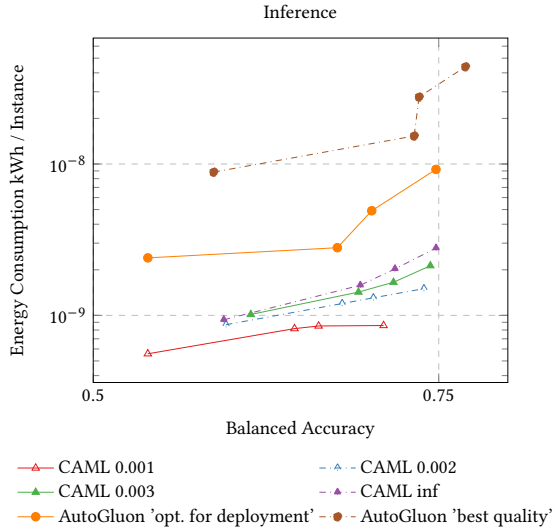

Figure 6: Average balanced accuracy and energy consumption (kWh): We run CAML with inference constraints ranging from 0.001-0.003s/instance. We execute the deployment-optimized configuration of AutoGluon. Each marked point represents the energy consumption in conjunction with the specified execution time: 10s, 30s, 1min, and 5min.

model is highly optimized for GPUs. For AutoGluon, GPU support increases both time and energy requirements. One reason is that most models that AutoGluon supports cannot use the GPU. Therefore, the GPU consumes energy in idle mode.

3.6 Example: Trillion prediction workload

Wu et al. [69] describe that Meta makes trillions of predictions per day. To understand the impact of energy consumption during inference better, we compute the consumed energy for one trillion predictions for each AutoML model considering the model with the highest predictive performance reported in Figure 3. Table 4 describes this cost with respect to energy (kWh), monetary (), and environment (kg CO2). To estimate the cost in Euro, we assume current average electricity prices in Europe of 0.20/kWh [12]. To

Table 4: Cost of 1 trillion predictions across AutoML systems

AutoML	Energy (kWh)	CO2 (kg)	Cost ()
TabPFN	404,649	89,832	80,930
AutoGluon	43,887	9,743	8,777
AutoSklearn1	36,090	8,012	7,218
AutoSklearn2	25,975	5,766	5,195
CAML	2,783	618	557
TPOT	1,245	276	249
FLAML	**762**	**169**	**152**

estimate the CO2 emissions, we assume the current emissions in Germany of 0.222kg/kWh [50]. The data shows that at this scale choosing small models as done by FLAML and CAML saves significant amounts of costs - up to 129k and 89t CO2 per day.

3.7 Analyzing the Energy Consumption of the AutoML Development Stage and its Impact

To analyze how investing more energy into the development stage of AutoML systems impacts the downstream execution and the inference stage, we analyze to which extent optimizing AutoML system parameters affects execution and inference. As many factors such as human interference can influence the energy consumption during this stage, we provide an approximation based on a CAML that automatically adjust the parameter space for specified application constraints. To carry out our analysis, we apply the optimization approach discussed in Section 2.5 for CAML for the search times 10s, 30s, 1min, and 5min using the top-20 most representative datasets from a set of 124 binary classification datasets from OpenML [66]. We start by examining the approach for binary classification. In total, we optimize 192 AutoML system parameters, incl. the design of the configuration space. In total, we optimize 186 AutoML system parameters that correspond to the ML hyperparameter search space of CAML and 6 other AutoML system parameters: hold-out validation fraction, evaluation fraction (the maximum time before one evaluation is stopped), sampling (number of instances that are used for the AutoML training run), refit (whether to refit the model on the

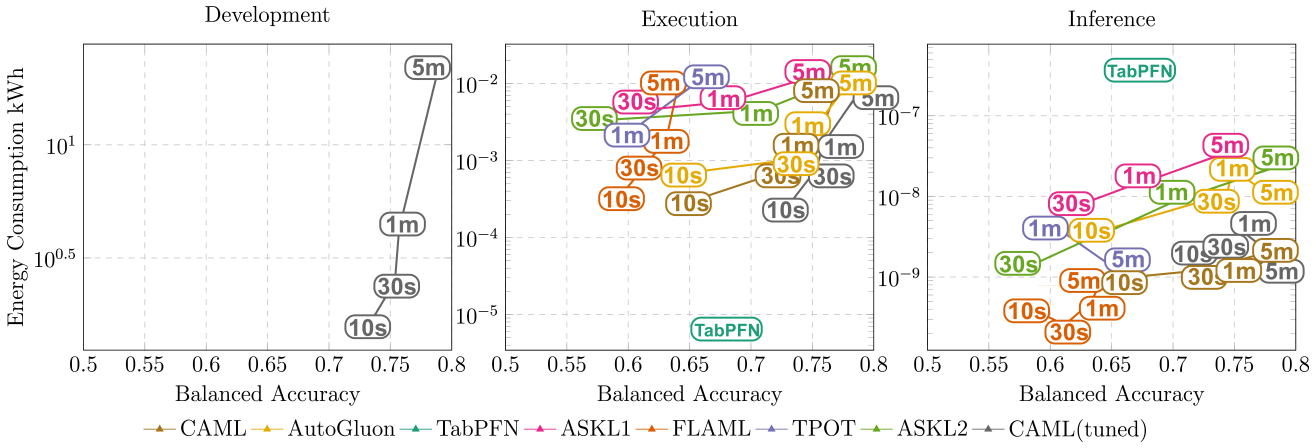

Figure 7: Search time, average balanced accuracy, and their impact on the energy consumption (kWh) during development, execution, and inference for each AutoML system. Each marked point represents one of these execution time budgets. Note that the shown configured runtime might differ from the actual runtime.

merged training and validation set), whether to use random validation set splitting for each BO iteration, and whether to use incremental training similar to successive halving.

Figure 7 shows the results of these experiments. First, CAML(tuned) that uses CAML based on optimization proposed in Section 2.5 outperforms all other AutoML systems with regard to balanced accuracy. Further, it requires the least amount of energy for execution. For instance, for 10s search time, CAML(tuned) outperforms the next best AutoML systems TabPFN and AutoGluon by 5% and 10% average balanced accuracy correspondingly. The inference cost is similar to the one achieved by CAML with default parameters. In the first chart of Figure 7, we report the energy consumed by our AutoML system parameter tuning process, i.e., the development stage. To yield the AutoML system parameters for a search time of 5 min, we spend 21kWh. The consumed time amortizes when the tuned AutoML system has run 885 times on new datasets. This amount would fit the scenario of AutoML-as-a-service on big cloud providers [15].

All in all, this experiment shows that we can reduce energy consumption for execution and inference if we invest energy into the development stage. Therefore, decisions in the early stage (development) affect later stages - both execution and inference.

To better understand why the tuned CAML outperforms all other systems, we report the corresponding AutoML system parameters for 30s, and 5min in Table 5 , which shows that AutoML parameters highly dependend on the search time. Different search times lead to different AutoML parameters that perform well. First, with increasing search time, the ML hyperparameter spaces also grow because there is more time to explore different ML pipelines. Further, we emphasize frequently chosen ML hyperparameters with blue (occur 2 times). The decision tree classifier was chosen for both search spaces. The reason is that decision trees can be both simple (shallow and narrow) and complex (deep and wide). More complex models, such as MLP and Random Forest, are only chosen for the longer search time of 5 min. For 5 min, the evaluation fraction increases to 17% (51s) - likely because one can spend more time training one model to avoid overfitting caused by too many BO iterations. Another interesting finding is that our tuning process always ends up sampling upfront. This search-time-specific sampling step is not implemented by any AutoML system - users always have to make

Table 5: AutoML system parameters for 30s, 1min, and 5min search time. For each budget, we report the ML hyperparameter search space as a tree and the list of initialized AutoML system parameters. We color parameters with blue if they were chosen 2 times.

30s	5min
root ├ oversampling │ ├ Identity │ ├ RandomOverSampler │ ├ SMOTE │ │ └ n_neighbors │ └ ADASYN ├ classifier │ ├ DecisionTree │ │ ├ min_samples_split │ │ └ min_samples_leaf │ ├ HistGradientBoosting │ │ ├ l2_regularization │ │ └ validation_fraction │ ├ KNeighbors │ │ └ k │ ├ LinearSVC │ │ ├ tol │ │ └ C │ └ PassiveAggressive │ ├ C │ ├ tol │ └ average │ └ True └ scaler ├ Identity ├ Normalizer ├ RobustScaler └ StandardScaler	root ├ classifier │ ├ RandomForest │ │ └ min_samples_split │ ├ BernoulliNB │ ├ DecisionTree │ │ └ max_depth_factor │ ├ ExtraTrees │ │ ├ max_features │ │ ├ min_samples_split │ │ ├ min_samples_leaf │ │ └ bootstrap │ │ └ False │ ├ MultinomialNB │ │ ├ alpha │ │ └ fit_prior │ │ ├ True │ │ └ False │ ├ PassiveAggressive │ │ ├ loss │ │ │ ├ hinge │ │ │ └ squared_hinge │ │ └ tol │ ├ SGD │ │ ├ penalty │ │ │ ├ l1 │ │ │ ├ l2 │ │ │ └ elasticnet │ │ ├ l1_ratio │ │ ├ tol │ │ ├ epsilon │ │ └ learning_rate │ ├ MLP │ │ ├ hidden_layer_depth │ │ ├ num_nodes_per_layer │ │ └ learning_rate_init │ └ LDA ├ imputation │ └ median └ categorical_encoding ├ FrequencyEncoding └ LabelEncoder
eval. time fraction: 0.1 presample: 22,610 refit: Yes holdout: 0.12 random shuffle: Yes incremental: Yes	eval. time fraction: 0.17 presample: 109,094 refit: No holdout: 0.19 random shuffle: Yes incremental: Yes

this decision by themselves. Further, the incremental training approach was always chosen across search times. This approach is very robust because it can handle any number of instances as it starts off by training 10 instances per class and step-wise increases the training set size. Further, our tuning process prefers to leverage random validation set splitting for each BO iteration.

Table 6: Number of times that AutoML systems achieve worse accuracy for 5min than 1min execution time.

System	FLAML	AutoGluon	AutoSklearn1	AutoSklearn2	TPOT	CAML
|Overfitting|	9	11	4	4	5	9

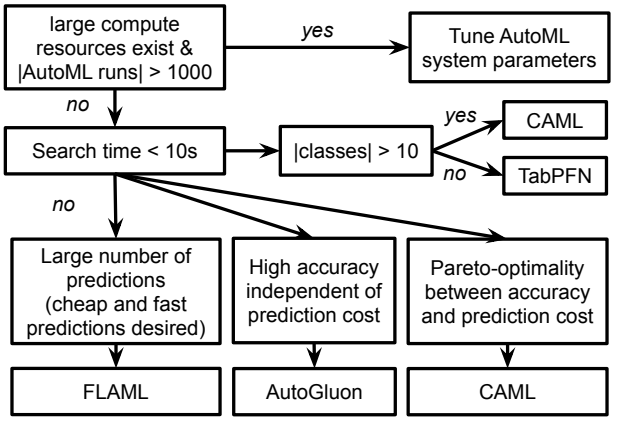

Figure 8: Guideline for picking the most energy-efficient solution depending on the task parameters and requirements.

This approach reduces the likelihood of overfitting to the validation set. Finally, CAML(tuned) does not leverage ensembling and still outperforms all systems that do, such as AutoGluon and AutoSklearn. Ensembling pays off if there are a number of mediocre predictive models. But if one manages to find one highly predictive model, ensembling cannot easily outperform it. By pruning the ML hyperparameter space, we significantly raise the odds of finding such a highly predictive model.

The AutoML parameters also explain the fact that models trained for 5min require less energy for inference than the models trained for 1min. The reason is that our tuning process does not choose to refit the models on the joined training and validation set as it does for 1min. Therefore, some models are potentially smaller and require less energy on average.

3.8 Overfitting & Early Stopping

One way of reducing energy consumption is to stop the AutoML system execution once it reaches the optimal performance [44]. To evaluate this approach's potential, we evaluate for how many datasets, AutoML systems achieve worse balanced accuracy running for 5min compared to 1min - in other words, how many times they overfit. Table 6 shows that up to 11 out of 39 datasets, AutoML systems overfit for a small search time of 5min. The datasets that are most often overfitted are kc1, cnae-9, and blood-transfusion-service-center - all have less than 3k rows. So, especially for smaller datasets, early stopping should be enforced to save energy.

3.9 Guideline

Based on the experiments shown in Figures 3 and 7, we distill a guideline for when to choose which AutoML system as a flowchart in Figure 8. The first question is whether the user has access to large CPU compute resources, e.g. at least 1 machine (28 Œ Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz cores, 264 GB RAM or similar) over more than a week, (development cost) and intends to perform thousands of AutoML system executions. For this case, we recommend picking any system that is tunable and tuning the AutoML system parameters because a tuned AutoML system requires the least energy for both the execution and inference stages. In all other cases, the search time is a deciding factor. For search budgets smaller than 10s, we should use TabPFN (with GPU support) or CAML depending on the number of classes. TabPFN provides zero-shot AutoML and CAML leverages incremental training and can find ML pipelines even for very large datasets. If there is a bigger search budget, the AutoML system choice depends on the user's priority. If the user wants fast inference at the cost of accuracy, they should use FLAML, which was designed with this goal. If users prioritize predictive accuracy, they should choose AutoGluon. If Pareto-optimal solutions between predictive performance and inference cost are desired, CAML should be the choice.

Table 7: Actual execution time for specified search times

AutoML	10s	30s	1min	5min
TabPFN	**0.29 ± 0.01**	**0.29 ± 0.01**	**0.29 ± 0.01**	**0.29 ± 0.01**
CAML	10.47 ± 0.05	30.89 ± 0.10	61.06 ± 0.10	301.42 ± 0.10
CAML(tuned)	11.31 ± 0.07	31.10 ± 0.27	60.34 ± 0.46	293.07 ± 3.84
FLAML	12.88 ± 0.19	33.27 ± 0.30	71.16 ± 1.30	329.32 ± 4.43
AutoGluon	22.32 ± 0.36	51.23 ± 0.13	81.13 ± 0.14	303.02 ± 5.60
TPOT	-	-	100.17 ± 7.74	368.17 ± 7.30
AutoSklearn2	-	128.72 ± 20.64	132.80 ± 13.51	405.17 ± 23.92
AutoSklearn1	-	176.46 ± 30.09	201.11 ± 31.06	451.98 ± 29.15

3.10 Runtime vs. Energy consumption

Longer runtime naturally leads to higher energy consumption. However, in general, single-core runtime alone is not a good proxy for energy consumption because the AutoML system might use other resources, such as memory and accelerators, such as GPUs [62]. To be consistent with previous AutoML benchmarks [21, 22], we consider the search time as the common termination criteria. As shown in Table 7, it turns out that AutoSklearn, TPOT, and AutoGluon do not strictly follow the specified search time. For small search times, AutoGluon's execution time is almost twice as long as specified because it has to learn a stacked model and it does not know how long the training of the different stacking levels will take. AutoSklearn takes even more time than AutoGluon and requires up to 5 times more time than specified because it considers the search time as the time it takes to evaluate different ML pipelines. After searching it still has to calculate the ensemble weights, which might take a significant amount of time, especially for large validation sets. Currently, AutoSklearn does not consider the ensembling time as search time. Nevertheless, Figure 3 clearly shows that some systems achieve the same predictive performance with significantly different energy consumption levels.

For the energy consumption of the execution of different AutoML systems, we have to define a termination criterion. In all available search-based systems, the termination criterion is defined by a limit for search time. Yet, many of the systems consider the defined search time as a soft criterion that cannot be strictly enforced. Autogluon uses the search time parameter to estimate its training strategy. If its estimation is too generous, it will take longer than defined. While AutoSklearn strictly adheres to the defined search time, it does not count in the ensembling step, which can lead to longer runtime than defined. CAML strictly adheres to the search time. FLAML finishes evaluating the last model that was started before hitting the time limit. TabPFN

Table 8: Comparison of different numbers of top-k datasets for a search time budget of 10s

top-k Datasets	Balanced Accuracy (%)	Energy (kWh)	Time (h)
10	68.60 ± 2.91	**0.43**	**3.5**
20	73.49 ± 1.09	2.38	17.9
40	**73.50 ± 0.57**	4.88	39.6

Table 9: Comparison of different numbers of BO iterations for a search time budget of 10s

top-k Datasets	Balanced Accuracy (%)	Energy (kWh)	Time (h)
75	72.91 ± 1.24	**0.74**	**5.6**
150	72.75 ± 1.46	1.29	9.9
300	**73.49 ± 1.09**	2.38	17.9
600	72.75 ± 2.24	3.46	26.1

almost requires no time for execution because most computation happens in the inference stage. Search-based approaches, such as CAML, AutoSklearn, and FLAML, follow the any-time AutoML approach, meaning that we can stop the search at any time, and if at least one model was evaluated successfully, they can provide predictions.

3.11 AutoML Development Results

To find optimal parameters for the tuned AutoML approach, as described in Section 2.5, we evaluate different numbers of representative datasets and numbers of BO iterations. First, we run the optimization approach for a search time of 10s using top-10, 20, and 40 datasets for 300 BO iterations. Table 8 reports the results of this experiment. With an increasing number of used datasets, the achieved average balance accuracy also increases. The reason is that the optimization generalizes better using more datasets whereas when using fewer datasets, the optimization might overfit easier to few datasets. However, the cost of using more datasets is significant both with respect to energy and time. Therefore, we choose to run all previous experiments with k=20 as it resembles the best trade-off between accuracy and cost. Second, we run the optimization approach for a search time of 10s using 75, 150, 300, and 600 BO iterations using the top-20 datasets. Table 9 reports the results of this experiment. Surprisingly, the highest number of iterations (600) achieves a lower accuracy than 300 iterations. The reason might be that the optimization starts to overfit to the datasets at hand and finds better and better AutoML system parameters that work particularly well for the specified 20 datasets. To avoid this overfitting, we could increase the number of datasets, but this results in an increase in cost. So, for the experiments in Section 3.7, we leverage 300 BO iterations.

4 RELATED WORK

Our work combines research from various areas of Green Data Processing, Green AI, and AutoML.

Green Data Processing. Guo et al. [23] survey existing energy management techniques for database systems. For instance, Lang et al. [35, 71] propose energy-aware query processing that allows the query optimizer to reason about the energy cost. Further, Lang et al. [34] evaluate the energy trade-off of scaling out database systems. While these works focus on energy consumption for databases, which allow a user to query the data for instance using SQL, we focus on energy consumption for AutoML systems, which allow a user to extract patterns/models from the data that can be leveraged for queries on new incoming data. We follow the same measurement best practices that were proposed in this line of work.

Green AI. Research in Green AI focuses on both improving the CO_2 footprint of AI applications and developing AI applications to positively impact the environment. Aimee van Wynsberghe [65] explains the differentiation between AI for sustainability and the sustainability of AI. We focus on the sustainability of AI - more specifically AutoML. Schwartz et al. [56] criticize ML advances that originate from a mere increase in computation. They promote focusing on efficiency and avoiding optimizing accuracy at any cost. Inspired by this work, we analyze the trade-off between accuracy and efficiency for AutoML systems, specifically. Patterson et al. [52] and Wu et al. [69] describe the resource consumption ML workloads at Google and Meta. They describe that energy consumption was reduced by focusing on efficient models. They argue that optimization of neural networks does require significant resources but the resulting models are more efficient and are used many times [53]. Both focus on company-wide ML, whereas we focus on AutoML. Luccioni et al. [43] discuss the carbon footprint of training and deploying large language models, whereas we focus on AutoML applications. Lacoste et al. [33] developed an emission calculator that estimates the CO2 emissions based on the cloud provider/region, hardware, and runtime. We assume fixed hardware and measure the consumed energy as a proxy for CO2 emissions.

Green AutoML. To the best of our knowledge, Tornede et al. [62] were the first to introduce the paradigm of Green AutoML. They describe how to measure the environmental footprint of AutoML and survey optimizations to improve the efficiency of executing AutoML search, but they do not conduct any experiments. Motivated by this work, we conducted an experimental study to evaluate AutoML systems empirically and holistically. There is additional work that focuses on leveraging AutoML systems to address applications that can help the environment. For instance, Tu et al. [63] promote to focus on building AutoML systems to fight climate change, e.g. wind power forecasting. Theodorakopoulos et al. propose an energy-efficient AutoML solution for plastic litter detection [60]. To make AutoML system development more resource efficient, Ying et al. [74] proposed to evaluate all possible combinations of a large search space of five million DNN architectures. This way, AutoML system developers can use this database to evaluate new ideas for novel search strategies without training or evaluating the ML pipelines from scratch. However, this approach assumes a fixed search space that cannot be extended and a fixed validation strategy. We focus on analyzing the environmental footprint of AutoML. Further, Lale [1] provides an interface to specify search spaces that are then optimized using optimizers, such as Hyperopt, grid search, and SMAC [41]. So, users can leverage domain knowledge to specify a tailored search space that can be optimized faster and more efficiently than full search spaces or choose between predefined or extracted search spaces. This is another avenue of reducing computation.

5 LIMITATIONS

We summarize our assumptions and discuss how robust your results are w.r.t these assumptions. First, we limit our evaluation to tabular data (incl. categorical and numerical features). Other data modalities, such as text, sound, and images, are out of scope

of this work. Therefore, we also limit the set of evaluated AutoML systems to AutoML systems tailored to tabular data and leave neural architecture search [14] for future work. There has been extensive work on improving the inference performance and therefore energy consumption, e.g. model distillation [25], quantization [7], or model compression in general [24]. These can be combined with AutoML tools, e.g., distilling the large stacking models of AutoGluon with a DNN [17]. We aim to provide a snapshot of the current state of energy efficiency of all current state-of-the-art AutoML systems as most users would use it out of the box. Our limitation with regards to measuring energy consumption is that we use the well-known library CodeCarbon [16]. Measuring energy physically is more accurate with regard to the absolute numbers, but would require physical access to the devices and abilities to isolate the energy measurement of the relevant systems.

Threats to validity and generalizability. ML and therefore also AutoML is highly non-deterministic. To ensure the validity of our results, we evaluate each experiment with 10 runs. We report all results of all runs in our repository [46]. Further, the results depend on the data. Therefore, we decided to leverage the AutoML benchmark datasets [22] that represent 39 diverse ML tasks. The results might also differ depending on the hardware. Therefore, we conducted experiments with different numbers of CPU cores and conducted additional experiments on a GPU environment. For completely new hardware, e.g. AutoML systems that leverage FPGAs, one would need to rerun the evaluation. We open-sourced all our source code for all experiments to make this easier. For new AutoML systems, one can look at the underlying implementation and compare it to the evaluated AutoML systems or rerun the experiments for this system to understand its efficiency.

We chose a representative subset of state-of-the-art AutoML systems and focused our analysis on the major differences between these systems. Some of our results, therefore, might only be valid in this very specific context. For example, ensembling models of a fixed set of base models will lead to increased energy consumption compared to a single model in general, but a single (complex) DNN might be more energy-consuming than an ensemble of linear models. Therefore, we emphasize that our results provide guidance toward Green AutoML, but eventually, the concrete combination of different approaches will decide the overall energy efficiency.

6 CONCLUSION

We analyzed the energy consumption of the state-of-the-art AutoML systems during all three stages of development, execution, and inference. Based on our results, we compiled a guideline on when to use which type of AutoML system. In the evaluation, we showed that there is an inherent trade-off between predictive performance and energy consumption. While AutoML systems that focus on predictive performance, such as AutoGluon, achieve outstanding predictive performance, they also require a significant amount of energy during inference. Other systems, such as FLAML, focus on reducing model complexity. It yields significantly lower predictive performance compared to AutoGluon but also requires orders of magnitudes less energy during inference. The zero-shot AutoML system TabPFN requires almost no energy during execution but requires significantly more energy during inference. We also showed that one can invest energy in the system development stage to reduce the energy consumption in the succeeding stages. However, the resulting AutoML system's execution must be a recurrent task to amortize the tuning cost. We urge AutoML systems engineers to report the energy consumption during development to make AutoML system engineering more efficient. Further, we showed how we can reduce the energy consumption during inference by changing the AutoML systems parameter or enforcing constraints, such as inference constraints. Optimizing efficiency and performance is one of the cores of database research. Therefore, improving the efficiency of AutoML systems by well-known database optimization strategies, such as exploiting modern hardware, leveraging efficient parallelization/distribution/caching strategies, and code generation, are all promising avenues of future research and a great opportunity for the database and AutoML community.

ACKNOWLEDGMENTS

Felix Neutatz and Ziawasch Abedjan acknowledge funding by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A). Marius Lindauer was supported by the German Federal Ministry of the Environment, Nature Conservation, Nuclear Safety and Consumer Protection (GreenAutoML4FAS project no. 67KI32007A)

REFERENCES

[1] Guillaume Baudart, Martin Hirzel, Kiran Kate, Parikshit Ram, Avraham Shinnar, and Jason Tsay. 2021. Pipeline Combinators for Gradual AutoML. In *NeurIPS*. 19705–19718.
[2] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305.
[3] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. 2023. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data. Mining. Knowl. Discov.* 13, 2 (2023).
[4] Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie N. Lindstaedt, Arnab Phani, Benjamin Rath, Berthold Reinwald, Shafaq Siddiqui, and Sebastian Benjamin Wrede. 2020. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. In *CIDR*.
[5] Matthias Boehm, Michael W. Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R. Reiss, Prithviraj Sen, Arvind C. Surve, and Shirish Tatikonda. 2016. SystemML: Declarative Machine Learning on Spark. *PVLDB* 9, 13 (2016), 14251436.
[6] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble selection from libraries of models. In *ICML*, Vol. 69.
[7] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *CoRR* abs/1602.02830 (2016).
[8] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6, 2 (2002), 182–197.
[9] Behrouz Derakhshan, Alireza Rezaei Mahdiraji, Ziawasch Abedjan, Tilmann Rabl, and Volker Markl. 2020. Optimizing Machine Learning Workloads in Collaborative Environments. In *SIGMOD*.
[10] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. 2016. A Validation of DRAM RAPL Power Measurements. In *MEMSYS*.
[11] Thomas G. Dietterich. 2000. Ensemble Methods in Machine Learning. In *MCS*, Vol. 1857.
[12] ec.europa.eu. 2023. Electricity price statistics. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Electricity_price_statistics.
[13] Katharina Eggensperger, Marius Lindauer, and Frank Hutter. 2019. Pitfalls and Best Practices in Algorithm Configuration. *J. Artif. Intell. Res.* 64 (2019).
[14] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *J. Mach. Learn. Res.* 20 (2019), 55:1–55:21.
[15] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander J. Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *CoRR* abs/2003.06505 (2020).
[16] Benoit Courty et al. 2023. Code Carbon. https://github.com/mlco2/codecarbon.
[17] Rasool Fakoor, Jonas Mueller, Nick Erickson, Pratik Chaudhari, and Alexander J. Smola. 2020. Fast, Accurate, and Simple Models for Tabular Data via Augmented Distillation. In *NeurIPS*.

[18] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *ICML*, Vol. 80. 1436–1445.

[19] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2022. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *JMLR* 23, 261 (2022).

[20] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *NeurIPS*.

[21] Pieter Gijsbers, Marcos L. P. Bueno, Stefan Coors, Erin LeDell, Sébastien Poirier, Janek Thomas, Bernd Bischl, and Joaquin Vanschoren. 2022. AMLB: an AutoML Benchmark. *CoRR* abs/2207.12560 (2022).

[22] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An Open Source AutoML Benchmark. *ICML* (2019).

[23] Binglei Guo, Jiong Yu, Dexian Yang, Hongyong Leng, and Bin Liao. 2023. Energy-Efficient Database Systems: A Systematic Survey. *ACM Comput. Surv.* 55, 6 (2023).

[24] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.

[25] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* abs/1503.02531 (2015).

[26] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2023. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. In *ICLR*.

[27] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *J. Artif. Intell. Res.* 36 (2009).

[28] Kevin G. Jamieson and Ameet Talwalkar. 2016. Non-stochastic Best Arm Identification and Hyperparameter Optimization. In *AISTATS*, Vol. 51.

[29] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. 2010. ISAC - Instance-Specific Algorithm Configuration. In *ECAI*, Vol. 215.

[30] Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Saravanan Thirumuruganathan, Sanjay Chawla, and Divy Agrawal. 2017. A cost-based optimizer for gradient descent optimization. In *SIGMOD*. 977–992.

[31] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. 2018. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3, 2 (2018).

[32] Arun Kumar, Matthias Boehm, and Jun Yang. 2017. Data Management in Machine Learning: Challenges, Techniques, and Systems. In *SIGMOD*. 17171722. https://doi.org/10.1145/3035918.3054775

[33] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. 2019. Quantifying the Carbon Emissions of Machine Learning. *CoRR* abs/1910.09700 (2019).

[34] Willis Lang, Stavros Harizopoulos, Jignesh M. Patel, Mehul A. Shah, and Dimitris Tsirogiannis. 2012. Towards Energy-Efficient Database Cluster Design. *VLDB* 5, 11 (2012).

[35] Willis Lang, Ramakrishnan Kandhan, and Jignesh M. Patel. 2011. Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race. *IEEE Data Eng. Bull.* 34, 1 (2011), 12–23.

[36] Teddy Lazebnik, Amit Somech, and Abraham Itzhak Weinberg. 2022. SubStrat: A Subset-Based Optimization Strategy for Faster AutoML. *Proc. VLDB Endow.* 16, 4 (2022), 772–780.

[37] Julien-Charles Lévesque. 2018. Bayesian hyperparameter optimization: overfitting, ensembles and conditional spaces. (2018).

[38] Keqin Li. 2016. Improving Multicore Server Performance and Reducing Energy Consumption by Workload Dependent Dynamic Power Management. *IEEE Trans. Cloud Comput.* 4, 2 (2016), 122–137.

[39] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* 18 (2017), 185:1–185:52.

[40] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Yaliang Li, Bolin Ding, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, and Bin Cui. 2021. VolcanoML: Speeding up End-to-End AutoML via Scalable Search Space Decomposition. *Proc. VLDB Endow.* 14, 11 (2021), 2167–2176.

[41] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *J. Mach. Learn. Res.* 23 (2022), 54:1–54:9.

[42] Marius Lindauer, Matthias Feurer, Katharina Eggensperger, André Biedenkapp, and Frank Hutter. 2019. Towards Assessing the Impact of Bayesian Optimization's Own Hyperparameters. *CoRR* abs/1908.06674 (2019).

[43] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. 2023. Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model. *JMLR* 24, 253 (2023).

[44] Anastasia Makarova, Huibin Shen, Valerio Perrone, Aaron Klein, Jean Baptiste Faddoul, Andreas Krause, Matthias W. Seeger, and Cédric Archambeau. 2022. Automatic Termination for Hyperparameter Optimization. In *ICML (Proceedings of Machine Learning Research)*, Vol. 188. 7/1–21.

[45] Mark Mazumder, Colby R. Banbury, Xiaozhe Yao, Bojan Karlas, William Gaviria Rojas, Sudnya Frederick Diamos, Greg Diamos, Lynn He, Douwe Kiela, David Jurado, David Kanter, Rafael Mosquera, Juan Ciro, Lora Aroyo, Bilge Acun, Sabri Eyuboglu, Amirata Ghorbani, Emmett D. Goodman, Tariq Kane, Christine R. Kirkpatrick, Tzu-Sheng Kuo, Jonas Mueller, Tristan Thrush, Joaquin Vanschoren, Margaret Warren, Adina Williams, Serena Yeung, Newsha Ardalani, Praveen K. Paritosh, Ce Zhang, James Zou, Carole-Jean Wu, Cody Coleman, Andrew Y. Ng, Peter Mattson, and Vijay Janapa Reddi. 2022. DataPerf: Benchmarks for Data-Centric AI Development. *CoRR* abs/2207.10062 (2022).

[46] Felix Neutatz. 2024. Green AutoML Benchmark. https://github.com/FelixNeutatz/GreenAutoML.

[47] Felix Neutatz, Felix Biessmann, and Ziawasch Abedjan. 2021. Enforcing Constraints for Machine Learning Systems via Declarative Feature Selection: An Experimental Study. In *SIGMOD*. 1345–1358.

[48] Felix Neutatz, Binger Chen, Yazan Alkhatib, Jingwen Ye, and Ziawasch Abedjan. 2022. Data Cleaning and AutoML: Would an Optimizer Choose to Clean? *Datenbank-Spektrum* 22, 2 (2022), 121–130. https://doi.org/10.1007/S13222-022-00413-2

[49] Felix Neutatz, Marius Lindauer, and Ziawasch Abedjan. 2023. AutoML in Heavily Constrained Applications. *VLDB J.* (2023).

[50] Nowtricity. 2023. CO2 emissions per kWh in Germany. https://www.nowtricity.com/country/germany/.

[51] Randal S. Olson and Jason H. Moore. 2019. TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. In *AutoML - Methods, Systems, Challenges*.

[52] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R So, Maud Texier, and Jeff Dean. 2022. The carbon footprint of machine learning training will plateau, then shrink. *Computer* 55, 7 (2022).

[53] David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. 2021. Carbon Emissions and Large Neural Network Training. *CoRR* abs/2104.10350 (2021).

[54] Michael J. Pawlish, Aparna S. Varde, Stefan A. Robila, and Anand Ranganathan. 2014. A call for energy efficiency in data centers. *SIGMOD Rec.* 43, 1 (2014), 45–51.

[55] David Salinas and Nick Erickson. 2024. TabRepo: A Large Scale Repository of Tabular Model Evaluations and its AutoML Applications. In *Proceedings of the third international AutoML Conference*.

[56] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. 2020. Green AI. *Commun. ACM* 63, 12 (2020).

[57] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing data science through interactive curation of ml pipelines. In *SIGMOD*. 1171–1188.

[58] Shafaq Siddiqi, Roman Kern, and Matthias Boehm. 2024. Saga: A Scalable Framework for Optimizing Data Cleaning Pipelines for Machine Learning Applications. In *SIGMOD*.

[59] Evan R Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J Franklin, and Benjamin Recht. 2017. Keystoneml: Optimizing pipelines for large-scale advanced analytics. In *ICDE*. 535–546.

[60] Daphne Theodorakopoulos, Christoph ManSS, Frederic Stahl, and Marius Lindauer. 2023. Green AutoML for Plastic Litter Detection. In *ICLR*.

[61] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *KDD*.

[62] Tanja Tornede, Alexander Tornede, Jonas Hanselle, Felix Mohr, Marcel Wever, and Eyke Hüllermeier. 2023. Towards Green Automated Machine Learning: Status Quo and Future Directions. *J. Artif. Intell. Res.* 77 (2023).

[63] Renbo Tu, Nicholas Roberts, Vishak Prasad, Sibasis Nayak, Paarth Jain, Frederic Sala, Ganesh Ramakrishnan, Ameet Talwalkar, Willie Neiswanger, and Colin White. 2022. AutoML for Climate Change: A Call to Action. *CoRR* abs/2210.03324 (2022).

[64] Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. 2020. Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*.

[65] Aimee van Wynsberghe. 2021. Sustainable AI: AI for sustainability and the sustainability of AI. *AI Ethics* 1, 3 (2021).

[66] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013).

[67] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. FLAML: A Fast and Lightweight AutoML Library. In *MLSys*.

[68] Yu Wang, Gu-Yeon Wei, and David Brooks. 2019. Benchmarking TPU, GPU, and CPU Platforms for Deep Learning. *CoRR* abs/1907.10701 (2019).

[69] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga Behram, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin S. Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim M. Hazelwood. 2022. Sustainable AI: Environmental Implications, Challenges and Opportunities. In *MLSys*.

[70] Doris Xin, Stephen Macke, Litian Ma, Jialin Liu, Shuchen Song, and Aditya Parameswaran. 2018. Helix: Holistic optimization for accelerating iterative machine learning. *PVLDB* 12, 4 (2018), 446–460.

[71] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. 2010. Exploring power-performance tradeoffs in database systems. In *ICDE*.
[72] Anatoly Yakovlev, Hesam Fathi Moghadam, Ali Moharrer, Jingxiao Cai, Nikan Chavoshi, Venkatanathan Varadarajan, Sandeep R Agrawal, Sam Idicula, Tomas Karnagel, Sanjay Jinturkar, et al. 2020. Oracle AutoML: a fast and predictive AutoML pipeline. *PVLDB* 13, 12 (2020), 3166–3180.
[73] Junwen Yang, Yeye He, and Surajit Chaudhuri. 2021. Auto-Pipeline: Synthesize Data Pipelines By-Target Using Reinforcement Learning and Search. *Proc. VLDB Endow.* 14, 11 (2021), 2563–2575.
[74] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 2019. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *ICML*, Vol. 97.