MULTI-OBJECTIVE DIFFERENTIABLE NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Pareto front profiling in multi-objective optimization (MOO), i.e., finding a diverse set of Pareto optimal solutions, is challenging, especially with expensive objectives that require training a neural network. Typically, in MOO for neural architecture search (NAS), we aim to balance performance and hardware metrics across devices. Prior NAS approaches simplify this task by incorporating hardware constraints into the objective function, but profiling the Pareto front necessitates a computationally expensive search for each constraint. In this work, we propose a novel NAS algorithm that encodes user preferences to trade-off performance and hardware metrics, yielding representative and diverse architectures across multiple devices in just a single search run. To this end, we parameterize the joint architectural distribution across devices and multiple objectives via a hypernetwork that can be conditioned on hardware features and preference vectors, enabling zero-shot transferability to new devices. Extensive experiments involving up to 19 hardware devices and 3 different objectives demonstrate the effectiveness and scalability of our method. Finally, we show that, without any additional costs, our method outperforms existing MOO NAS methods across a broad range of qualitatively different search spaces and datasets, including MobileNetV3 on ImageNet-1k, an encoder-decoder transformer space for machine translation and a decoder-only space for language modelling.

1 INTRODUCTION

The ability to make good tradeoffs between predictive accuracy and efficiency (in terms of latency and/or energy consumption) has become crucial in an age of ever increasing neural networks complexity and size (Alabdulmohsin et al., 2023; Hoffmann et al., 2022; Kaplan et al., 2020; Zhai et al., 2022) and a plethora of embedded devices. However, finding the right trade-off remains a challenging task that typically requires human intervention and a lot of trial-and-error across devices. With multiple conflicting objectives, it becomes infeasible to optimize all of them simultaneously and return a single solution. Ideally, NAS should empower users to choose from a set of diverse Pareto optimal solutions that represent their preferences regarding the trade-off between objectives.

Neural Architecture Search (NAS) (White et al., 2023) provides a principled framework to search for network architectures in an automated fashion. Several works (Cai et al., 2020; Chen et al., 2021a; Elsken et al., 2019b; Wang et al., 2020b) have extended NAS for multi-objective optimization (MOO), considering performance and hardware efficiency metrics like latency and energy consumption. However, to the best of our knowledge, no existing gradient-based method returns the full Pareto front for the MOO problem at hand *without repeating their search routine multiple times with different hardware constraints*.

In this work, we propose a scalable and hardware-aware **M**ulti-**O**bjective **D**ifferentiable **N**eural **A**rchitecture **S**earch (**MODNAS**) algorithm that efficiently trains a single supernet which can be used to read off Pareto-optimal solutions for different user preferences and different target devices, without any additional search steps. To search across devices, we frame the problem as a multi-task, multi-objective optimization problem, where each task (device) has multiple (conflicting) objectives, e.g., classification accuracy and latency. The user's preferences are modelled by a *preference vector* that defines a scalarization (weighted sum), of the different objectives. This preference vector, along with features of the hardware of interest, is fed to a hypernetwork (Ha et al., 2017) that outputs continuous architectural parameters α. To search in the space of architectures, we employ a one-shot

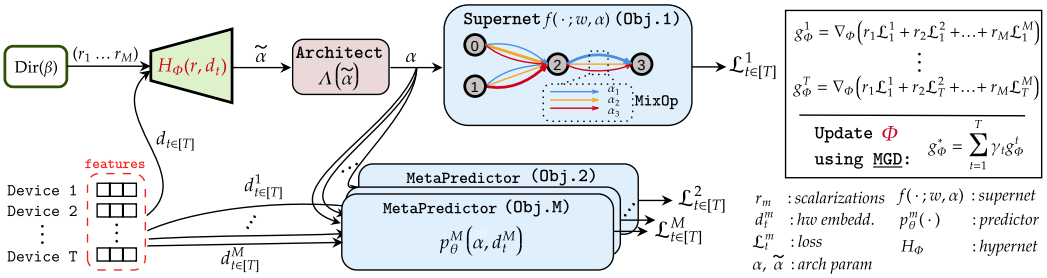


Figure 1: **MODNAS overview.** Given a set of T devices, MODNAS seeks to optimize M (potentially conflicting) objectives across these devices. To this end, it employs a `MetaHypernetwork` $H_\Phi(r, d_t)$, that takes as input a scalarization r, representing the user preferences, and a device embedding d_t, to yield an un-normalized architectural distribution $\tilde{\alpha}$. The `Architect` uses $\tilde{\alpha}$ to sample differentiable discrete architectures, used in the `Supernetwork` to estimate accuracy and in the `MetaPredictor` to estimate the other $M-1$ loss functions (e.g. latency, energy consumption) for every device. By iterating over devices and sampling scalarizations uniformly from the M-dimensional simplex, at each iteration we update the `MetaHypernetwork` using multiple gradient descent (MGD).

model and a bi-level optimization scheme, as is typically done in gradient-based NAS. In our case, however, the upper-level parameters are the hypernetwork weights, optimized in expectation over different preference vectors and hardware devices via multiple gradient descent (Désidéri, 2012).

To evaluate our method, we conduct experiments on multiple NAS search spaces, including CNN and Transformer architectures, and up to 3 objectives across 19 hardware devices. While other NAS methods that utilize hardware constraints in their search objectives require substantial search costs both for each new constraint and each new hardware, MODNAS addresses both in a zero-shot manner, without extra search cost, while yielding higher quality solutions.

Our contributions can be summarized as follows:

1. We present a principled approach for **Multi-objective Differentiable NAS**, that leverages *hypernetworks* and *multiple gradient descent* for fast Pareto-Front approximation across devices.
2. This work is the *first* to provide a global view of the Pareto solutions with **just a single search run**, without the need to repeat search or fine-tune on new target devices.
3. Extensive evaluation of our method across **4 different search spaces** (NAS-Bench-201, MobileNetV3, an encoder-decoder and a decoder-only Transformer space), **3 tasks** (image classification, machine translation and language modeling), and **up to 19 hardware devices and 3 objectives**, show both improved *efficiency* and *performance* in comparison to previous approaches that use a constrained objective in their search.

To facilitate reproducibility, we provide our code via the following anonymous link.

2 Background and Related Work

In this section, before describing our algorithm, we introduce some basic concepts, definitions and related work. Refer to Appendix A for an extended related work.

Multi-objective optimization (MOO) for Multi-Task Learning. Consider a multi-task dataset $\mathcal{D}$ consisting of N instances, where the feature vector of the $i-$th instance is denoted as $x_i \in \mathcal{X}$, and the M-many associated target variables as $y_i^1 \in \mathcal{Y}^1, \ldots, y_i^M \in \mathcal{Y}^M$. Moreover, consider there exists a family of parametric models $f(\boldsymbol{x}; \boldsymbol{w}) : \mathcal{X} \to \{\mathcal{Y}^1 \times \cdots \times \mathcal{Y}^M\}$, parameterized by $\boldsymbol{w}$, that maps the input $\boldsymbol{x}$ to the joint space of the multiple tasks. To simplify the notation, we denote the prediction of the m-th task as $f^m(\boldsymbol{x}; \boldsymbol{w}) : \mathcal{X} \to \mathcal{Y}^m$, and the respective loss $\mathcal{L}^m(\boldsymbol{w}) \triangleq \frac{1}{N}\sum_i^N \ell^m(y_i^m, f^m(x_i; \boldsymbol{w}))$. The vector of the values of all loss functions is denoted as $\mathbf{L}(\boldsymbol{w}) \triangleq (\mathcal{L}^1(\boldsymbol{w}), \ldots, \mathcal{L}^M(\boldsymbol{w}))$. MOO then seeks to find a set of Pareto-optimal solutions $\boldsymbol{w}^*$ that jointly minimize $\mathbf{L}(\boldsymbol{w})$[1]:

$$\boldsymbol{w}^* \in \operatorname*{argmin}_{\boldsymbol{w}} \mathbf{L}(\boldsymbol{w}) \tag{1}$$

[1] $\boldsymbol{w}$ can be replaced with any other parameter here, also architectural ones (see Section 3).

Definition 2.1. (Pareto Optimality): A solution $\boldsymbol{w}_2$ dominates $\boldsymbol{w}_1$ iff $\mathcal{L}^m(\boldsymbol{w}_2) \leq \mathcal{L}^m(\boldsymbol{w}_1), \forall m \in \{1, \ldots, M\}$, and $\mathbf{L}(\boldsymbol{w}_1) \neq \mathbf{L}(\boldsymbol{w}_2)$. In other words, a dominating solution has a lower loss value on at least one task and no higher loss value on any task. A solution $\boldsymbol{w}^*$ is called *Pareto optimal* iff there exists no other solution dominating $\boldsymbol{w}^*$.

Definition 2.2. (Pareto front): The sets of Pareto optimal points and their function values are called *Pareto set* ($\mathcal{P}_{\boldsymbol{w}}$) and *Pareto front* ($\mathcal{P}_{\mathbf{L}} = \{\mathbf{L}(\boldsymbol{w})_{\boldsymbol{w} \in \mathcal{P}_{\boldsymbol{w}}}\}$), respectively.

Linear Scalarization. In MOO, a standard technique to solve the M-dimensional problem is using a preference vector $\boldsymbol{r} \in \mathcal{S} \triangleq \{\mathbb{R}^M | \sum_{m=1}^{M} r_m = 1, r_m \geq 0, \forall m \in \{1, \ldots, M\}\}$ in the M-dimensional probability simplex (Lin et al., 2019; Mahapatra & Rajan, 2020; Ruchte & Grabocka, 2021). Every $\boldsymbol{r} \in \mathcal{S}$ yields a convex combination of the loss functions in Equation 1 as $\mathcal{L}_{\boldsymbol{r}}(\boldsymbol{w}) = \boldsymbol{r}^{\mathbf{T}}\mathbf{L}(\boldsymbol{w})$. Given a preference vector $\boldsymbol{r}$, one can apply standard, single-objective optimization algorithms to find a minimizer $\boldsymbol{w}^*_{\boldsymbol{r}} = \operatorname{argmin}_{\boldsymbol{w}} \mathcal{L}_{\boldsymbol{r}}(\boldsymbol{w})$. By sampling multiple $\boldsymbol{r}$ vectors, one can compute Pareto-optimal solutions $\boldsymbol{w}^*_{\boldsymbol{r}}$ that profile the Pareto front. Several methods (Hoang et al., 2023; Lin et al., 2020; Navon et al., 2021; Phan et al., 2022) employ a hypernetwork (Ha et al., 2017) to generate Pareto-optimal solutions given different preference vectors as input. In this work, we utilize a hypernetwork conditioned on scalarizations to generate Pareto-optimal architectures. Furthermore, we also extend the hypernetwork by conditioning it on different task vectors.

Multiple Gradient Descent (MGD). MOO can be solved to local optimality via MGD (Désidéri, 2012), as a natural extension of single-objective gradient descent, which iteratively updates $\boldsymbol{w}$ towards a direction that ensures that all tasks improve simultaneously (called *Pareto improvement*): $\boldsymbol{w}' \leftarrow \boldsymbol{w} - \xi g^*_{\boldsymbol{w}}$, where $g^*_{\boldsymbol{w}}$ is a vector field that needs to be determined. If we denote by $g^m_{\boldsymbol{w}} = \nabla_{\boldsymbol{w}} \mathcal{L}^m(\boldsymbol{w})$ the gradient of the m-th scalar loss function, via Taylor approximation, the decreasing direction of $\mathcal{L}^m$ when we update $\boldsymbol{w}$ towards $g^*_{\boldsymbol{w}}$ is given by $\langle g^m_{\boldsymbol{w}}, g^*_{\boldsymbol{w}} \rangle \approx -(\mathcal{L}^m(\boldsymbol{w}') - \mathcal{L}^m(\boldsymbol{w}))/\xi$. In MGD $g^*_{\boldsymbol{w}}$ is chosen to maximize the slowest update rate among all objectives:

$$g^*_{\boldsymbol{w}} \propto \operatorname*{argmax}_{g_{\boldsymbol{w}} \in \mathbb{R}^d, ||g_{\boldsymbol{w}}|| \leq 1} \Big\{ \min_{m \in [M]} \langle g_{\boldsymbol{w}}, g^m_{\boldsymbol{w}} \rangle \Big\}. \tag{2}$$

The early work of Désidéri (2012) has been extended in various settings, particularly multi-task learning, with great promise (Lin et al., 2019; Liu & Vicente, 2021; Mahapatra & Rajan, 2020; Sener & Koltun, 2018), but these approaches are applied to mainly a fixed architecture and extending them to a supernet subsuming a search space of multiple architectures is non-trivial.

One-shot NAS and Bi-Level optimization. With the architecture space being intrinsically discrete, large (often consisting of upto 10^{36} architectures) and hence expensive to search on, most existing differentiable NAS approaches leverage the weight sharing paradigm and continuous relaxation to enable gradient descent (Bender et al., 2018; Chen et al., 2021b; Dong & Yang, 2019; Liu et al., 2019; 2023; Movahedi et al., 2022; Pham et al., 2018; Xie et al., 2019; Xu et al., 2020a; Zhang et al., 2021). Typically, in these approaches, architectures are stacks of cells, where the cell structure is represented as a directed acyclic graph (DAG) with N nodes and E edges. Every transition from node i to j, i.e. edge (i, j), is associated with an operation $o^{(i,j)} \in \mathcal{O}$, where $\mathcal{O}$ is a predefined candidate operation set. Liu et al. (2019) proposed a continuous relaxation of the search space by parameterizing the discrete operation choices in the DAG edges via a learnable vector α. This enables framing the NAS problem as a bi-level optimization one, with differentiable objectives w.r.t. all variables:

$$\operatorname*{argmin}_{\alpha} \mathcal{L}^{val}(\boldsymbol{w}^*(\alpha), \alpha) \quad s.t. \quad \boldsymbol{w}^*(\alpha) = \operatorname*{argmin}_{\boldsymbol{w}} \mathcal{L}^{train}(\boldsymbol{w}, \alpha), \tag{3}$$

where $\mathcal{L}^{train}$ and $\mathcal{L}^{val}$ are the empirical losses on the training and validation data, respectively, $\boldsymbol{w}$ are the supernetwork parameters, $\alpha \in \mathcal{A}$ are the continuous architectural parameters, and $\boldsymbol{w}^*(\alpha) : \mathcal{A} \to \mathbb{R}^d$ is a best response function that maps architectures to their optimal weights.

Comparison to single-objective constrained NAS. Early NAS methods predominantly targeted high accuracy, whereas contemporary hardware-aware differentiable NAS approaches (Cai et al., 2018; Fu et al., 2020; Jiang et al., 2021; Wan et al., 2020; Wang et al., 2021; Wu et al., 2019; 2021; Xu et al., 2020b) are designed to identify architectures optimized for target hardware efficiency. Typically, these methods integrate hardware constraints within their objectives, yielding a single optimal solution and necessitating multiple search iterations to construct the Pareto front. Our proposed algorithm addresses this by profiling the entire Pareto front in a *single search iteration*. While single-objective constrained optimization is advantageous in scenarios demanding optimization of one objective under a specific constraint, practical applications often require a suite of models adaptable to varying user

preferences even on a single device. Our efficient Pareto-front approximation algorithm provides such a suite of optimal models to choose from.

3 HARDWARE-AWARE MULTI-OBJECTIVE DIFFERENTIABLE NEURAL ARCHITECTURE SEARCH

We first formalize the multi-objective bi-level optimization NAS problem across multiple hardware devices, and then introduce a scalable and differentiable method that combines MGD with linear scalarizations to efficiently solve this problem.

3.1 PROBLEM DEFINITION & SKETCH OF SOLUTION APPROACH

In multi-objective NAS, the bi-level problem described in Equation 1 becomes more difficult, since we are not only concerned with finding $\boldsymbol{w}^*$ given a fixed architecture, but we want to optimize in the space of architectures $\mathcal{A}$ as well. Assuming we have T hardware devices (target functions) and M objectives (e.g. accuracy, latency, etc.), similar to (3), for every $t \in \{1 \dots T\}$, the Pareto set can be obtained by solving the following bi-level optimization problem:

$$\underset{\alpha}{\operatorname{argmin}}\, \mathbf{L}_t^{valid}(\boldsymbol{w}^*(\alpha), \alpha) \quad s.t. \quad \boldsymbol{w}^*(\alpha) = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathbf{L}_t^{train}(\boldsymbol{w}, \alpha), \tag{4}$$

where the M-dimensional loss vector $\mathbf{L}_t(\boldsymbol{w}^*(\alpha), \alpha) \triangleq \left(\mathcal{L}_t^1(\boldsymbol{w}^*(\alpha), \alpha), \dots, \mathcal{L}_t^M(\boldsymbol{w}^*(\alpha), \alpha)\right)$ is evaluated $\forall t \in \{1, \dots, T\}$. $\mathbf{L}_t^{train}$ and $\mathbf{L}_t^{valid}$ are the vectors with all M loss functions evaluated on the train and validation splits of $\mathcal{D}$, used in the lower- and upper-level problems of (4), respectively.

Our goal is to find Pareto-optimal architectures for each target device, covering diverse and representative preferences for different objectives. However, naively solving (4) for each device t requires T independent bi-level searches, making this very inefficient for large models. To overcome this, we incorporate a single hypernetwork within the one-shot model (supernetwork) commonly used in conventional NAS (Bender et al., 2018; Liu et al., 2019; Pham et al., 2018). This allows us to generate architectures based on device embeddings and preference vectors in just *one search run*, reducing the search cost from $\mathcal{O}$(T) to $\mathcal{O}$(1).

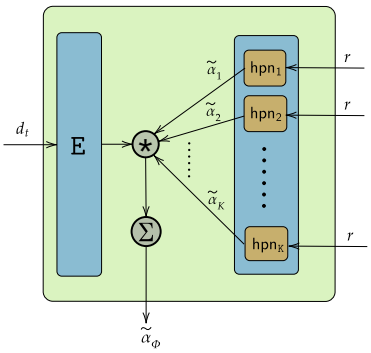


Figure 2: Architecture overview of the `MetaHypernetwork`, which gets as input a device embedding d_t (input to an embedding layer `E`) and a scalarization $\boldsymbol{r}$ (input to K hypernetworks) and yields an architecture encoding $\tilde{\alpha}$.

3.2 ALGORITHM DESIGN AND COMPONENTS

Our search procedure is composed of four core modular components (see Figure 1): (1) a `MetaHypernetwork` that generates the architectural distribution; (2) an `Architect` that samples discrete architectures from this distribution; (3) a `Supernetwork` that exploits the weight-sharing paradigm for search efficiency and serves as a proxy for the network accuracy; and (4) a `MetaPredictor` that predicts hardware metrics and enables gradient propagation back to the `MetaHypernetwork`. We now discuss each of these in detail.

`MetaHypernetwork`. In order to generate architectures across multiple devices, inspired by Wang et al. (2022) and Lin et al. (2020), we propose a `MetaHypernetwork` that can meta-learn across different hardware devices (see Figure 2). Hypernetworks are a class of neural networks that generate the parameters of another model. They were initially proposed for model compression (Ha et al., 2017) and were later adopted for NAS (Brock et al., 2018) and MOO (Lin et al., 2020; Navon et al., 2021). Here, given a preference vector $\boldsymbol{r} = (r_1, \dots, r_M)$ and a hardware device feature vector d_t, for device $t \in \{1, \dots, T\}$, we use the `MetaHypernetwork` $H_\Phi(\boldsymbol{r}, d_t)$, parameterized by Φ, to generate an un-normalized architecture distribution $\tilde{\alpha}_\Phi$ that is later used to compute the upper-level updates in (4). Similar to Lee et al. (2021b), d_t is a fixed-size feature vector that is obtained by evaluating a fixed set of reference architectures on device t. The `MetaHypernetwork` is composed of 2 main components (see Figure 2):

1. *A bank of K independent hypernetworks*: $\texttt{hpn}_1, \dots, \texttt{hpn}_K$, that parse the preference vector $\boldsymbol{r}$ and generate the architectural parameters $\tilde{\alpha}_1, \dots, \tilde{\alpha}_K$, respectively.

2. *A linear layer* `E`, that learns a similarity map from device feature vectors to the bank of `hpns`. `E` takes as input the device feature vector d_t and outputs an attention vector of size K.

The final output, $\tilde{\alpha}_\Phi$, of the `MetaHypernetwork` is computed as a weighted sum of the outputs of the K hypernetworks, where the vector of weights is the output of the linear layer `E`. For a more detailed description of the `MetaHypernetwork` we refer the reader to Appendix E.2.

In all experiments, we initialize the `MetaHypernetwork` to yield a uniform probability mass over all architectural parameters for all scalarizations and device embeddings. By using the preference vector $\boldsymbol{r}$ to create a linear scalarization of $\mathbf{L}_t$ and the `MetaHypernetwork` to model the architectural distribution across T devices, the bi-level problem in (4) reduces to:

$$\underset{\Phi}{\operatorname{argmin}}\, \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\big[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{valid}(\boldsymbol{w}^*(\alpha_\Phi),\alpha_\Phi)\big] \quad s.t. \quad \boldsymbol{w}^*(\alpha_\Phi) = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\big[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{train}(\boldsymbol{w},\alpha_\Phi)\big], \tag{5}$$

where α_Φ are the normalized architectural parameters obtained from the `Architect` $\Lambda(\tilde{\alpha}_\Phi)$ and $\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t(\cdot,\alpha_\Phi) = \sum_{m=1}^{M} r_m \mathcal{L}_t^m(\cdot,\alpha_\Phi)$ is the scalarized loss for device t. Conditioning the `MetaHypernetwork` on the hardware embeddings allows us to generate architectures on new test devices without extra finetuning or meta-learning steps. Iniutively, the `MetaHypernetwork`, learns to map the new test device, to the most similar device, in its learnt bank of embeddings (see also Figure 12 in the appendix). We use the *Dirichlet* distribution $Dir(\beta)$, $\beta = (\beta_1,\ldots,\beta_M)$, to sample the preference vectors and approximate the expectation over the scalarizations using Monte Carlo sampling. In our experiments, we set $\beta_1 = \cdots = \beta_M = 1$, for a uniform sampling over the $(M-1)$-simplex, however, one can set these differently based on user priors or make it a learnable parameter (Chen et al., 2021b).

`MetaPredictor`. For the cheap-to-evaluate hardware objectives, such as latency, energy consumption, we employ a regression model $p_\theta^m(\alpha, d_t^m)$ that predicts the target labels y_t^m for objective m and device t, given an architecture α and device embedding d_t^m. We use the same predictors as Lee et al. (2021b) and optimize the MSE loss: $\min_\theta \mathbb{E}_{\alpha\sim\mathcal{A},t\sim[T]}\big(y_t^m - p_\theta^m(\alpha,d_t^m)\big)^2$, as done in Lee et al. (2021a) for meta-learning performance metrics across datasets. In our experiments, we pretrain a separate `MetaPredictor` for every hardware objective m (e.g. latency, energy, etc.) on a subset of (α, y_t^m) pairs, and use its predicted value directly in (5) as $\mathcal{L}_t^m(\cdot,\alpha_\Phi) = p_\theta^m(\alpha_\Phi, d_t^m)$. Since the `MetaPredictor` is in principle a small neural network this pretraining step is inexpensive. During search, we freeze and do not further update the `MetaPredictor` parameters θ.

`Supernetwork`. For expensive objectives like neural network classification accuracy, we use a `Supernetwork` that encodes the architecture space and shares parameters between architectures, providing a best response function $\boldsymbol{w}^*(\alpha_\Phi)$ for the scalarized loss in (5). While any parametric model could estimate this function, such as performance predictors (Lee et al., 2021a), this requires an expensive prior step of creating the training dataset for the predictor, by training architectures from scratch. To reduce memory costs of `Supernetwork` training, we: (1) use a one-hot encoding of α_Φ for differentiable architecture sampling (Cai et al., 2018; Dong & Yang, 2019; Xie et al., 2019), activating only one architecture per step, and (2) entangle operation choice parameters in the `Supernetwork`, further increasing memory efficiency beyond weight sharing (Sukthanker et al., 2023).

Algorithm 1: `MODNAS`

Data: $\mathcal{D}_{train}$; $\mathcal{D}_{valid}$; `Supernetwork`; device features $\{d_t\}_{t=1}^T$; `MetaHypernetwork` H_Φ; nr. of objectives M; `Architect` Λ; learning rates ξ_1, ξ_2.

while *not converged* **do**
  for $t \in \{1,\ldots,T\}$ **do**
    Sample scalarization $\boldsymbol{r} \sim Dir(\beta)$
    Set arch params $\tilde{\alpha}_\Phi \leftarrow H_\Phi(\boldsymbol{r}, d_t)$
    Sample $\alpha_\Phi \sim \Lambda(\tilde{\alpha}_\Phi)$ from `Architect`
    $g_\Phi^t \leftarrow \sum_{m=1}^{M} r_m \nabla_\Phi \mathcal{L}_t^m(\mathcal{D}_{valid}; \boldsymbol{w}, \alpha_\Phi)$
  $\gamma \leftarrow$ `FrankWolfeSolver`$(g_\Phi^1,\ldots,g_\Phi^T)$; `// see Alg.4`
  $g_\Phi^* \leftarrow \sum_{t=1}^{T} \gamma_t \cdot g_\Phi^t$
  $\Phi \leftarrow \Phi - \xi_1 \cdot g_\Phi^*$; `// update MetaHypernetwork`
  for $t \in \{1,\ldots,T\}$ **do**
    Sample scalarization $\boldsymbol{r} \sim Dir(\beta)$
    Set arch params $\tilde{\alpha}_\Phi \leftarrow H_\Phi(\boldsymbol{r}, d_t)$
    Sample $\alpha_\Phi \sim \Lambda(\tilde{\alpha}_\Phi)$ from `Architect`
    $g_{\boldsymbol{w}}^t \leftarrow \sum_{m=1}^{M} r_m \nabla_{\boldsymbol{w}} \mathcal{L}_t^m(\mathcal{D}_{train}; \boldsymbol{w}, \alpha_\Phi)$
  $g_{\boldsymbol{w}}^* \leftarrow \frac{1}{T}\sum_{t=1}^{T} g_{\boldsymbol{w}}^t$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \xi_2 \cdot g_{\boldsymbol{w}}^*$; `// update Supernetwork`
return H_Φ

Architect. The `Architect` $\Lambda(\tilde{\alpha})$ samples discrete architectural configurations from the un-normalized distribution $\tilde{\alpha}_\Phi = H_\Phi(\boldsymbol{r}, d_t)$ and enables gradient estimation through discrete variables for $\nabla_\Phi \mathbf{L}_t(\cdot, \alpha_\Phi)$. Methods such as GDAS (Dong & Yang, 2019) utilize the Straight-Through Gumbel-Softmax (STGS) estimator (Jang et al., 2017), that integrates the Gumbel reparameterization trick to approximate the gradient. Here we employ the recently proposed *ReinMax* estimator (Liu et al., 2023), that yields second-order accuracy without the need to compute second-order derivatives. See Appendix B.1 for more details on these discrete samplers. Similar to the findings in Liu et al. (2023), in our initial experiments, ReinMax outperforms the GDAS' STGS estimator (see Figure 15 in the Appendix), therefore, we use ReinMax in all experiments that follow.

3.3 Optimizing the `MetaHypernetwork` via MGD

We denote the gradient of the scalarized loss in (5) with respect to the `MetaHypernetwork` parameters Φ, shared across all devices $t \in 1, \ldots, T$, as: $g_\Phi^t = \boldsymbol{r}^\mathbf{T} \nabla_\Phi \mathbf{L}_t(\cdot, \alpha_\Phi) = \sum_{m=1}^{M} r_m \nabla_\Phi \mathcal{L}_t^m(\cdot, \alpha_\Phi)$, where α_Φ is the discrete architectural sample from the `Architect` $\Lambda(\tilde{\alpha}\Phi)$.

Multiple Gradient Descent (MGD) (Désidéri, 2012; Sener & Koltun, 2018) provides a plausible approach to estimate the update directions for every task simultaneously by maximizing (2). Via the Lagrangian duality, the optimal solution to equation 2 is $g_\Phi^* \propto \sum_{t=1}^{T} \gamma_t^* g_\Phi^t$, where $\{\gamma_t^*\}_{t=1}^T$ is the solution of the following minimization problem:

$$\min_{\gamma_1, \ldots, \gamma_T} \left\{ \left\| \sum_{t=1}^{T} \gamma_t g_\Phi^t \right\|_2^2 \,\middle|\, \sum_{t=1}^{T} \gamma_t = 1, \gamma_t \geq 0, \forall t \right\}$$

The solution to this problem is either 0 or, given a small step size ξ, a descent direction that monotonically decreases all objectives at the same time and terminates when it finds a Pareto stationary point, i.e. $g_\Phi^t = 0, \forall t \in \{1, \ldots, T\}$. When $T = 2$, the problem above simplifies to $\min_{\gamma \in [0,1]} \left\| \gamma g_\Phi^1 + (1-\gamma) g_\Phi^2 \right\|_2^2$, which is a quadratic function of γ with a closed form solution:

$$\gamma^* = \max\left(\min\left(\frac{(g_\Phi^2 - g_\Phi^1)^\mathbf{T} g_\Phi^2}{\|g_\Phi^1 - g_\Phi^2\|_2^2}, 1 \right), 0 \right).$$

When $T > 2$, we utilize the *Frank-Wolfe* solver (Jaggi, 2013) as in Sener & Koltun (2018), where the analytical solution in for $T = 2$ is used inside the line search. We provide the full algorithm to compute γ^* in Algorithm 4 in Appendix B.2.

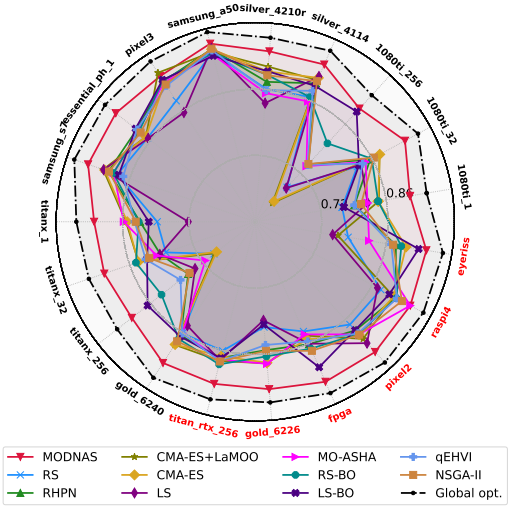


Figure 3: Hypervolume (HV) of MODNAS and baselines across 19 devices on NAS-Bench-201. For every device, we optimize for 2 objectives, namely *latency (ms)* and *test accuracy* on CIFAR-10. For each method, metric and device we report the mean of 3 independent search runs. Higher area in the radar plot indicates better HV. Test devices are colored in red around the plot.

In Algorithm 1 and Figure 1 we provide the pseudocode and an illustration of the overall search phase of MODNAS. For every mini-batch sample from $\mathcal{D}_{valid}$, we iterate over the device features d_t (line 2), sample one scalarization $\boldsymbol{r}$ and condition the `MetaHypernetwork` on both $\boldsymbol{r}$ and d_t to generate the un-normalized architectural distribution $\tilde{\alpha}_\Phi$ (lines 3-4). We then compute the device-specific gradient in line 6 which is used to estimate the γ coefficients (line 7) used from MGD to update Φ (lines 8-9). Similarly to Liu et al. (2019), we use the first-order approximation to obtain the best response function in the lower level (lines 10-14) and repeat the same procedure for the upper-level (lines 2-6), except now the `Supernetwork` weights $\boldsymbol{w}$ are updated with the mean gradient (line 15), over devices.

4 Experiments

In this section, we firstly demonstrate the scalability and generalizability of our MODNAS approach on a NAS tabular benchmark (Section 4.1). Then, we validate MODNAS on larger search spaces for Machine Translation (Section 4.2), Image Classification and Language Modeling (Section 4.3).

Search Spaces and Datasets. We evaluate MODNAS on 4 search spaces: (1) **NAS-Bench-201** (Dong & Yang, 2020; Li et al., 2021) with 19 devices and CIFAR-10 dataset; (2) **MobileNetV3** from Once-for-All (OFA) (Cai et al., 2020) with 12 devices and ImageNet-1k dataset; (3) **Hardware-Aware-Transformer (HAT)** (Wang et al., 2020b) on the machine translation benchmark WMT'14 En-De across 3 different hardware devices; (4) **HW-GPT-Bench** (Sukthanker et al., 2024) – a GPT-2 based search space used for language modeling on the OpenWebText (Gokaslan & Cohen, 2019) across 8 devices. We refer to Appendices F and G for more details on these search spaces.

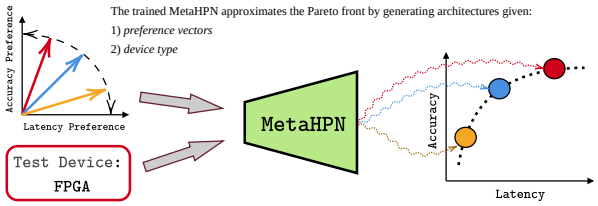


Figure 4: Illustration of MODNAS inference.

Evaluation. At test time, in order to profile the Pareto front with MODNAS on unseen devices, we sample 24 equidistant preference vectors $\boldsymbol{r}$ from the M-dimensional probability simplex and pass them through the pretrained `MetaHypernetwork` $H_\Phi(\boldsymbol{r}, d_t)$ to get 24 architectures. Here the test device feature d_t is obtained similarly as for the train devices. See Figure 4 or an illustration.

Baselines. We compare MODNAS against several baselines[2], such as Random Search (RS), Local Search (LS) and various Evolutionary Strategy and Bayesian Optimization MOO methods. Please refer to Appendix C for a more comprehensive description of each of them. Furthermore, we also evaluate the `MetaHypernetwork` with randomly initialized weights (RHPN).

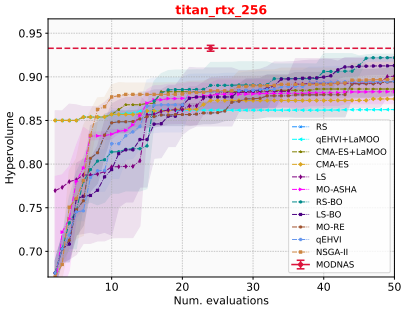


Figure 5: HV over number of evaluated architectures on NAS-Bench-201 of MODNAS and the blackbox MOO baselines on a test device. For MODNAS we only do 24 full evaluations.

Metrics. To assess the quality of the Pareto set solutions, we use the *hypervolume (HV)* indicator, which is a standard metric in MOO. Given a *reference point* $\rho = [\rho^1, \dots, \rho^m] \in \mathbb{R}_+^M$ that is an upper bound for all objectives $\{f^m(\cdot; \boldsymbol{w}, \alpha)\}_{m=1}^M$, i.e. $sup_\alpha f^m(\cdot; \boldsymbol{w}, \alpha) \leq \rho^m$, $\forall m \in [M]$, and a Pareto set $\mathcal{P}_\alpha \subset \mathcal{A}$, HV($\mathcal{P}_\alpha$) measures the region of non-dominated points bounded above from ρ:

$$\lambda\Big(\{q \in \mathbb{R}_+^M \mid \exists \alpha \in \mathcal{P}_\alpha : q \in \prod_{m=1}^M [f^m(\cdot; \boldsymbol{w}, \alpha), \rho^m]\}\Big),$$

where $\lambda(\cdot)$ is the Euclidean volume. HV can be interpreted as the total volume of the union of the boxes created by the Pareto front.

4.1 Simultaneous Pareto Set Learning across 19 devices and Ablations

We firstly validate the scalability and learning capability of MODNAS by evaluating on the NAS-Bench-201 (Dong & Yang, 2020) cell-based convolutional space. Here we want to optimize both latency and classification accuracy on all devices. We utilize the same set of 19 heterogeneous devices as Lee et al. (2021b), from which we use 13 for search and 6 at test time. For the latency predictor, we use the one from HELP, namely a graph convolutional network (GCN), which we pretrain for 3 GPU hours on the ground truth latencies on the 13 search devices as described in Section 3. We run the MODNAS search (see Appendix E for more details on the search hyperparameters), as described in Algorithm 1, for 100 epochs (22 GPU hours on a single NVidia RTX2080Ti) and show the HV in Figure 3 of the evaluated Pareto front in comparison to the baselines, for which we allocate the same search time budget across all devices equivalent to the MODNAS search + evaluation.

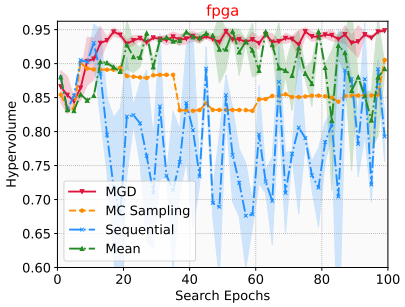


Figure 6: HV over search epochs of different gradient schemes in MODNAS.

[2]We use the implementations from SyneTune (Salinas et al., 2022): `https://github.com/awslabs/syne-tune`

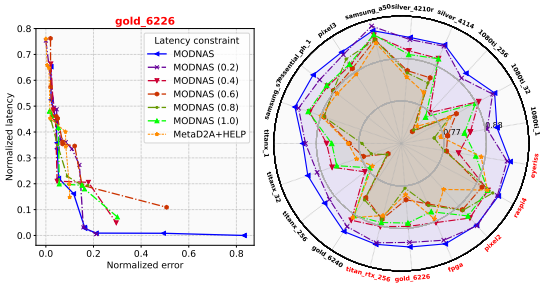


Figure 7: Pareto front on Eyeriss (*left*) and HV across devices (*right*) of MODNAS ran with various latency constraints on NAS-Bench-201. See Fig. 21 in Appendix I for all results.

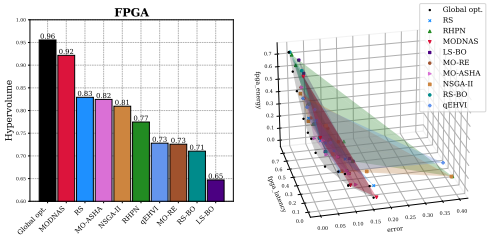


Figure 8: HV (*left*) and Pareto front (*right*) of MODNAS and baselines on FPGA with 3 normalized objectives: error, latency and energy usage. HV is computed using the $(1, 1, 1)$ reference point on the right 3D plot. See Fig. 16 for results on Eyeriss.

Most notably, MODNAS consistently outperforms all other baselines across every device. For the baselines, we conduct 19 separate search runs (one for each device), whereas MODNAS leverages meta-learning to generate the Pareto set on each device using the same `MetaHypernetwork` in a single search run. Interestingly, the trained MODNAS attention-based `MetaHypernetwork` significantly outperforms the RHPN baseline in profiling the Pareto front, demonstrating its *effectiveness in optimizing across multiple devices and conflicting objectives simultaneously*. In Figure 20a in the Appendix, we compare MODNAS with additional baselines, running them at double the budget used for the experiments in Figure 3. Figure 5 (see Figure 23 in the appendix for all devices) shows that most baselines require more than twice the number of architecture evaluations to reach the same HV as MODNAS. Results show that MODNAS remains the top performer across hardware devices on average. Furthermore, in the appendix, Figure 20 presents radar plots for four additional metrics, and Figure 18 and 17 results on NB201 when optimizing CIFAR-100 accuracy and device latency.

Reliably learnt embeddings for hardware devices. To demonstrate the effectiveness of our `MetaHypernetwork` in learning hardware device similarities, Figure 12 in the appendix shows K-means clustering of original and `MetaHypernetwork` embeddings, reduced via t-SNE. The `MetaHypernetwork` successfully clusters similar devices, confirming its efficacy.

`MetaHypernetwork` update schemes: robustness of MGD. We compare the MGD update scheme for the `MetaHypernetwork` Φ (line 9 in Alg. 1) against (1) the **mean** gradient over tasks: $\Phi \leftarrow \Phi - \xi \frac{1}{T} \sum_{t=1}^{T} g_\Phi^t$; (2) **sequential** updates with all single tasks' gradients: $\Phi \leftarrow \Phi - \xi g_\Phi^t, \forall t$; (3) single updates using gradients of **MC samples** over tasks: $\Phi \leftarrow \Phi - \xi g_\Phi^t$, $t \sim \{1, \dots T\}$. Figure 6 (see Figure 24 in Appendix I for more results) shows the HV over search epochs for these schemes. MGD, by accounting for inter-task dependencies, achieves higher final HV, better anytime performance, and faster convergence than the other schemes.

Scalability to three objectives. We show the scalability of MODNAS to 3 objectives, namely, accuracy, latency and energy consumption. For this experiment we use the FPGA and Eyeriss tabular energy usage values from HW-NAS-Bench (Li et al., 2021). In addition to the `MetaPredictor` for latency, we pretrain a second predictor on the energy usage objective. We then run MODNAS and the MOO baselines with the same exact settings as for 2 objectives. Results shown in Figure 8 indicate that MODNAS can scale to $M > 2$ without additional search costs or hyperparameter tuning and yet achieves HV close to the global optimum front of the NAS-Bench-201 space.

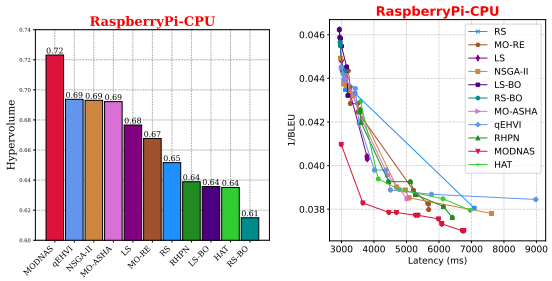


Figure 9: HV and Pareto fronts of MODNAS and baselines across devices on the HAT space.

MODNAS vs. constrained single-objective optimization. To compare against single-objective NAS with hardware constraints in the objective, we run **MetaD2A+HELP** (Lee et al., 2021b). Since MetaD2A + HELP is not able to profile the Pareto front directly, we run the NAS search 24 times with different constraints, which we compute by denormalizing the same 24 equidistant preference vectors we use to evaluate MODNAS. We also extend MODNAS to incorporate user prior constraints over the multiple objectives being optimized during search. Namely, we add a normalized constraint c^m, such that if the predicted value from the `MetaPredictor` during search

satisfies this constraint, i.e. $p_\theta^m(\alpha_\Phi, d_t^m) \leq c^m$, we remove the gradient w.r.t. to that objective in lines 6 and 14 of Algorithm 1. In Figure 7 (other devices in Figure 21) we can see that when increasing the latency constraint to 1 (only cross-entropy optimized), though the HV decreases, MODNAS returns Pareto sets with more performant architectures. MetaD2A+HELP, despite multiple search runs, prioritizes performance over diversity, resulting in less varied solutions.

4.2 Pareto Front Profiling on Transformer Space

To demonstrate its effectiveness beyond image classification and CNN spaces, we apply MODNAS to the hardware-aware Transformer (HAT) search space from Wang et al. (2020b) on the WMT'14 En-De (Jean et al., 2015; Macháček & Bojar, 2014) machine translation task. We pretrain the `MetaPredictor` (details in Appendix E.1) for 5 GPU hours on 2000 architecture samples from the search space and then conduct the search for 110 epochs (6 days on 8 NVIDIA RTX A6000 GPUs) using 2 search devices, adhering to the same hyperparameters as Wang et al. (2020b) to optimize for *latency* and *validation cross entropy loss*. We allocate to each baseline 2.5× more runtime budget than MODNAS, resulting in 1300 (RS-BO) to 6000 (MO-ASHA) total architecture evaluations, whereas MODNAS evaluates only 24 generated architectures. Details on the HAT search space and search hyperparameters are in Appendix F. We evaluate MODNAS on all 3 devices (2 search and 1 test) using the BLEU score, and results in Figure 9 show that MODNAS outperforms all baselines, achieving a higher hypervolume (left plot) of the generated Pareto fronts (right plot). For HAT, we evaluate the architectures provided in their paper. Additional results on other training devices and evaluation metrics are presented in Figures 26, 27 and 28 in the Appendix.

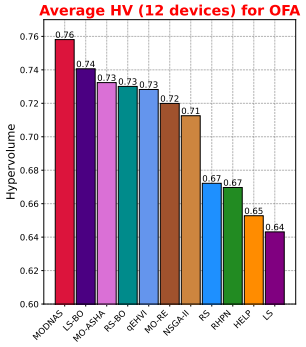


Figure 10: Average HV of MODNAS and baselines across 12 devices on OFA space. For every device we optimize for 2 objectives, namely *latency (ms)* and *test accuracy* on ImageNet-1k.

4.3 Efficient Differentiable MOO starting from Pretrained Supernetworks

Image Classification on ImageNet-1k. We now evaluate MODNAS on ImageNet-1k using the MovileNetV3 search space from Once-for-All (OFA) (Cai et al., 2020). For this experiment, we run MODNAS using 11 search (and 1 test) devices starting with the pretrained OFA supernetwork and run the search further for 1 day on 8 RTX2080Ti GPUs. During the search, we only update the `MetaHypernetwork` weights and keep the pretrained `Supernetwork` weights frozen. Details on the search space and hyperparameters are in Appendices F and E.3. We use the simple MLP from Lee et al. (2021b) as our `MetaPredictor`, pretraining it for 6 hours on 5000 sampled architecture-latency pairs. To evaluate the 24 points generated by our `MetaHypernetwork` and baselines, we use the OFA pretrained `Supernetwork`. Results in Figure 10 show that MODNAS achieves a higher average HV across all devices compared to baselines, which we run for 192 hours using the OFA pretrained accuracy predictor (see Figure 31 for all results and Figure 30 for the Pareto fronts).

Comparison to Zero-Cost Proxies. We also compare the HV of the Pareto front obtained by MODNAS to that produced by NSGA-II (Deb et al., 2002), which uses a zero-cost proxy (ZCP) (Abdelfattah et al., 2021) for performance estimation instead of the actual accuracy. We select Zico (Li et al., 2023) since it is one of the few ZCPs evaluated in the MobileNetV3 search space and on large datasets like ImageNet-1k. Table 1 presents the results of this experiment on two devices. As shown, despite its improved runtime efficiency, the ZCP-guided search underperforms compared to both the existing baselines and MODNAS, which optimize for accuracy directly.

Table 1: HV of MODNAS and baselines on the OFA search space. For every device we optimize for 2 objectives: *latency (ms)* and validation accuracy on ImageNet-1k.

Device Name	RS	RHPN	HELP	EHVI	LS	LS-BO	MO-ASHA	RS-BO	MO-REA	NSGA-II	Zico-NSGA-II	MODNAS
v100_64	0.677	0.683	0.638	0.748	0.697	0.749	0.740	0.747	0.750	0.744	0.689	**0.757**
titan_rtx_64	0.722	0.698	0.663	0.751	0.734	0.755	0.736	0.753	0.752	0.744	0.690	**0.763**

Language Modeling with GPT-2. With the rapid growth of language model sizes, it is crucial to identify transformer variants that are efficient during inference (latency) while maintaining competitive

performance. We apply MODNAS to the GPT-S space from HW-GPT-Bench (Sukthanker et al., 2024), which features a non-convex Pareto front between perplexity and hardware metric objectives. Using pretrained `Supernetwork` weights from HW-GPT-Bench, we conduct a single 6-hour search on 4 Nvidia A100 GPUs, optimizing for energy consumption (Wh) and perplexity across 8 different GPU devices. See Appendix E for details on the `MetaHypernetwork` architecture and search hyperparameters. The `Supernetwork` weights are kept frozen while updating the `MetaHypernetwork`. Figure 11 shows that, with the same time budget, MODNAS matches or surpasses other MOO baselines, demonstrating its effectiveness in optimizing beyond convex Pareto fronts.

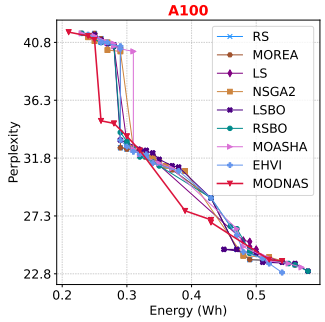


Figure 11: Pareto front of MODNAS and baselines on the HW-GPT-Bench, A100 GPU.

4.4 Computational Complexity

Ignoring the cost to train final architectures in the Pareto set, methods like MetaD2A + HELP (Lee et al., 2021a;b) have a worst-case time complexity of $\mathcal{O}$(CT) to build the Pareto set, where T is the number of devices and C is the number of constraints. MODNAS reduces this to $\mathcal{O}(1)$ by conditioning a single `MetaHypernetwork` on both device types and constraints. Methods like LEMONADE (Elsken et al., 2019a) and ProxylessNAS (Cai et al., 2018) apply constraints during the search phase, requiring an independent search per device. Black-box methods such as LEMONADE, NSGA-II (Deb et al., 2002), or qEHVI (Daulton et al., 2020) train $\mathcal{O}$(NT) architectures or a surrogate based on $\mathcal{O}$(N) architectures in the case of MetaD2A + HELP. In contrast, MODNAS and OFA have a cost of $\mathcal{O}(1)$ as they train a single supernetwork. Although MODNAS iterates over T devices to compute g_Φ^* and $g_{\boldsymbol{w}}^*$, Figure 25 in Appendix I.2 shows that MODNAS generalizes well on 17 test devices with only 2 search devices due to its meta-learning capabilities. See Tables 2 and 5 in the Appendix for more details.

Table 2: Cost of MODNAS in comparison to other methods. N is the number of trained architectures during search, T the number of devices and C the number of constraints.

Method	Search Cost	Pareto Set Build Cost
LEMONADE (Elsken et al., 2019a)	$\mathcal{O}$(NT)	$\mathcal{O}(1)$
Blackbox MOO (Daulton et al., 2020; Zhao et al., 2022)	$\mathcal{O}$(NT)	$\mathcal{O}(1)$
ProxylessNAS (Cai et al., 2018)	$\mathcal{O}$(CT)	$\mathcal{O}(1)$
MetaD2A + HELP (Lee et al., 2021a;b)	$\mathcal{O}$(N)	$\mathcal{O}$(CT)
OFA (Cai et al., 2020) + HELP (Lee et al., 2021b)	$\mathcal{O}(1)$	$\mathcal{O}$(CT)
MODNAS (Ours)	$\mathcal{O}(1)$	$\mathcal{O}(1)$

5 Broader Impact and Limitations

Broader Impact. In an era of large-scale models (e.g. foundation models), speeding up the search and training cost for inference-optimal neural architectures is an important aspect of responsible research (Cai et al., 2024; Muralidharan et al., 2024; Zhang et al., 2024a). The main goal of this work is to improve the search costs, as well as the efficiency of the found architectures in terms of various hardware metrics, therefore reducing the energy consumption and CO_2 footprint.The energy savings of these architectures will be amplified as they might be deployed on a large number of devices.

Limitations. While our differentiable multi-objective search method shows promising results, there are potential limitations. MODNAS inherits challenges common to gradient-based search, such as the risk of failure without proper tuning or regularization (Zela et al., 2020). For example, gradients may favor one objective, leading to local optima that hinder exploration of the full Pareto front. Additionally, the method relies on differentiable proxies for objectives, which may not always align with ground truth values.

6 Conclusion

In this paper, we propose a novel hardware-aware differentiable NAS algorithm for profiling the Pareto front in multi-objective problems. In contrast to constraint-based NAS methods, ours can generate Pareto optimal architectures across multiple devices with a single hypernetwork that is conditioned on preference vectors encoding the trade-off between objectives. Experiments across various hardware devices (up to 19), objectives (accuracy, latency and energy usage), search spaces (CNNs and Transformers), and applications (classification, machine translation, language modeling) demonstrate the effectiveness and efficiency of our method.

REFERENCES

M. Abdelfattah, A. Mehrotra, L. Dudziak, and N. Lane. Zero-cost proxies for lightweight NAS. In *Proceedings of the International Conference on Learning Representations (ICLR'21)*, 2021. Published online: iclr.cc. 9

Ibrahim Alabdulmohsin, Xiaohua Zhai, Alexander Kolesnikov, and Lucas Beyer. Getting vit in shape: Scaling laws for compute-optimal model design. *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 1

G. Bender, P-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le. Understanding and simplifying one-shot architecture search. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80. Proceedings of Machine Learning Research, 2018. 3, 4, 17

Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. Hardware-aware neural architecture search: Survey and taxonomy. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 4322–4329, 8 2021. Survey Track. 17

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. 19

Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018. 4

H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-All: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR)*, 2020. 1, 7, 9, 10, 19, 24, 40

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2018. 3, 5, 10, 17

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. *arXiv preprint arXiv:2406.10260*, 2024. 10

Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12270–12280, 2021a. 1, 19

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In *International conference on machine learning*, pp. 1554–1565. PMLR, 2020. 40

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. DrNAS: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021b. 3, 5

Richeek Das and Samuel Dooley. Fairer and more accurate tabular models through nas. *Algorithmic Fairness through the Lens of Time Workshop at NeurIPS*, 2023. 17

S. Daulton, M. Balandat, and E. Bakshy. Differentiable expected hypervolume improvement for parallel Multi-Objective Bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9851–9864. Curran Associates, Inc., 2020. 10, 20

Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Multi-objective bayesian optimization over high-dimensional search spaces. In *Uncertainty in Artificial Intelligence*, pp. 507–517. PMLR, 2022. 17

Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *Parallel Problem Solving from Nature PPSN VI: 6th International Conference Paris, France, September 18–20, 2000 Proceedings 6*, pp. 849–858. Springer, 2000. 17

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002. 9, 10, 17, 19

Jean-Antoine Désidéri. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350:313–318, 2012. 2, 3, 6, 17, 40

X. Dong and Y. Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 3, 5, 6, 27, 29, 40

X. Dong and Y. Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, 2020. Published online: `iclr.cc`. 7, 20, 24

Samuel Dooley, Rhea Sanjay Sukthanker, John P Dickerson, Colin White, Frank Hutter, and Micah Goldblum. Rethinking bias mitigation: Fairer architectures make for fairer face recognition. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 17

Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns. *Advances in Neural Information Processing Systems*, 33:10480–10490, 2020. 17, 20

T. Elsken, J. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019a. 10

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019b. 1, 17

Yonggan Fu, Wuyang Chen, Haotao Wang, Haoran Li, Yingyan Lin, and Zhangyang Wang. AutoGAN-distiller: Searching to compress generative adversarial networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3292–3303. PMLR, 13–18 Jul 2020. 3, 17

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019. 7, 24

Nyoman Gunantara. A review of multi-objective optimization: Methods and its applications. *Cogent Engineering*, 5(1):1502242, 2018. 17

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pp. 544–560. Springer, 2020. 17

D. Ha, A. Dai, and Q. Le. Hypernetworks. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. 1, 3, 4

Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11990–11999, 2020. 17, 29

Long P Hoang, Dung D Le, Tran Anh Tuan, and Tran Ngoc Thang. Improving pareto front learning via multi-sample hypernetworks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 7875–7883, 2023. 3, 17

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030. Curran Associates, Inc., 2022. 1

Chi-Hung Hsu, Shu-Huan Chang, Jhao-Hong Liang, Hsin-Ping Chou, Chun-Hao Liu, Shih-Chieh Chang, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Da-Cheng Juan. Monas: Multi-objective neural architecture search using reinforcement learning. *arXiv preprint arXiv:1806.10332*, 2018. 17

C. Igel, Nikolaus Hansen, and Stefan Roth. Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation*, 15:1–28, 2007. 19

Rafael C Ito and Fernando J Von Zuben. Ofa ^{2}: A multi-objective perspective for the once-for-all neural architecture search. *arXiv preprint arXiv:2303.13683*, 2023. 17

Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning*, 2013. 6, 18

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`. 6, 18

Sébastien Jean, Orhan Firat, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. Montreal neural machine translation systems for wmt'15. In *Proceedings of the tenth workshop on statistical machine translation*, pp. 134–140, 2015. 9

Qian Jiang, Xiaofan Zhang, Deming Chen, Minh N Do, and Raymond A Yeh. Eh-dnas: End-to-end hardware-aware differentiable neural architecture search. *arXiv preprint arXiv:2111.12299*, 2021. 3, 17

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 1

Sunghoon Kim, Hyunjeong Kwon, Eunji Kwon, Youngchang Choi, Tae-Hyun Oh, and Seokhyeong Kang. Mdarts: Multi-objective differentiable neural architecture search. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1344–1349. IEEE, 2021. 17

Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. In *International Conference on Learning Representations*, 2021a. 5, 10

Hayeon Lee, Sewoong Lee, Song Chong, and Sung Ju Hwang. Hardware-adaptive efficient latency prediction for nas via meta-learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 27016–27028. Curran Associates, Inc., 2021b. 4, 5, 7, 8, 9, 10, 20, 21, 22, 24, 27

Jaeseong Lee, Duseok Kang, and Soonhoi Ha. S3nas: Fast npu-aware neural architecture search methodology. *arXiv preprint arXiv:2009.02009*, 2020. 17

Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan (Celine) Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021. 7, 8, 24

Guihong Li, Yuedong Yang, Kartikeya Bhardwaj, and Radu Marculescu. Zico: Zero-shot NAS via inverse coefficient of variation on gradients. In *The Eleventh International Conference on Learning Representations*, 2023. 9

L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. In J. Peters and D. Sontag (eds.), *Proceedings of The 36th Uncertainty in Artificial Intelligence Conference (UAI'20)*, pp. 367–377. PMLR, 2020. 17, 19

Liam Li, Kevin G. Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. *ArXiv*, abs/1810.05934, 2018. 19

X. Lin, H. Zhen, Z. Li, Q. Zhang, and S. Kwong. Pareto multi-task learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 3, 17

Xi Lin, Zhiyuan Yang, Qingfu Zhang, and Sam Tak Wu Kwong. Controllable pareto multi-task learning. *ArXiv*, abs/2010.06313, 2020. 3, 4, 17, 22

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 3, 4, 6, 17

Liyuan Liu, Chengyu Dong, Xiaodong Liu, Bin Yu, and Jianfeng Gao. Bridging discrete and backpropagation: Straight-through and beyond. *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 3, 6, 18, 40

Suyun Liu and Luís Nunes Vicente. The stochastic multi-gradient algorithm for multi-objective optimization and its application to supervised machine learning. *Annals of Operations Research*, pp. 1572–9338, 2021. 3, 17

Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pp. 35–51. Springer, 2020. 17

Matouš Macháček and Ondřej Bojar. Results of the WMT14 metrics shared task. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pp. 293–301, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3336. 9, 24

D. Mahapatra and V. Rajan. Multi-task learning with user preferences: Gradient descent with controlled ascent in pareto optimization. In *Proceedings of the 36th International Conference on Machine Learning (ICML'20)*, pp. 6597–6607. Proceedings of Machine Learning Research, 2020. 3, 17

Natalia Martinez, Martin Bertran, and Guillermo Sapiro. Minimax pareto fairness: A multi objective perspective. In *International Conference on Machine Learning*, pp. 6755–6764. PMLR, 2020. 17

Michinari Momma, Chaosheng Dong, and Jia Liu. A multi-objective/multi-task learning framework induced by pareto stationarity. In *International Conference on Machine Learning*, pp. 15895–15907. PMLR, 2022. 17

Sajad Movahedi, Melika Adabinejad, Ayyoob Imani, Arezou Keshavarz, Mostafa Dehghani, Azadeh Shakery, and Babak N Araabi. λ-darts: Mitigating performance collapse by harmonizing operation selection among cells. *The Eleventh International Conference on Learning Representations*, 2022. 3

Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation. *arXiv preprint arXiv:2407.14679*, 2024. 10

Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. Learning the pareto front with hypernetworks. *International Conference on Learning Representations*, 2021. 3, 4, 17

Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. A flexible framework for multi-objective bayesian optimization using random scalarizations. In *Uncertainty in Artificial Intelligence*, pp. 766–776. PMLR, 2020. 19

H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, 2018. 3, 4, 17

Hoang Phan, Ngoc Tran, Trung Le, Toan Tran, Nhat Ho, and Dinh Phung. Stochastic multiple target sampling gradient descent. *Advances in neural information processing systems*, 35:22643–22655, 2022. 3, 17

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, AAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. 19

Binxin Ru, Clare Lyle, Lisa Schut, Miroslav Fil, Mark van der Wilk, and Yarin Gal. Speedy performance estimation for neural architecture search. In *Advances in Neural Information Processing Systems*, 2021. 20

Michael Ruchte and Josif Grabocka. Scalable pareto front approximation for deep multi-objective learning. In *2021 IEEE international conference on data mining (ICDM)*, pp. 1306–1311. IEEE, 2021. 3, 23

David Salinas, Valerio Perrone, Olivier Cruchant, and C. Archambeau. A multi-objective perspective on jointly tuning hardware and hyperparameters. *ArXiv*, abs/2106.05680, 2021. 19

David Salinas, Matthias Seeger, Aaron Klein, Valerio Perrone, Martin Wistuba, and Cedric Archambeau. Syne tune: A library for large scale hyperparameter tuning and reproducible research. In *International Conference on Automated Machine Learning, AutoML 2022*, 2022. 7

Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. *Advances in neural information processing systems*, 29, 2016. 17

Robin Schmucker, Michele Donini, Muhammad Bilal Zafar, David Salinas, and C. Archambeau. Multi-objective asynchronous successive halving. *ArXiv*, abs/2106.12639, 2021. 19

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Neural Information Processing Systems*, 2018. 3, 6, 17

Albert Shaw, Daniel Hunter, Forrest Landola, and Sammy Sidhu. Squeezenas: Fast neural architecture search for faster semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019. 17

Samuel L Smith, Benoit Dherin, David Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. In *International Conference on Learning Representations*, 2021. 40

Rhea Sanjay Sukthanker, Arjun Krishnakumar, Mahmoud Safari, and Frank Hutter. Weight-entanglement meets gradient-based neural architecture search. *arXiv preprint arXiv:2312.10440*, 2023. 5, 40

Rhea Sanjay Sukthanker, Arber Zela, Benedikt Staffler, Joerg K.H. Franke, and Frank Hutter. Hw-gpt-bench: Hardware-aware architecture benchmark for language models. *arXiv preprint arXiv:2405.10299*, 2024. 7, 10, 21, 24, 27

M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 17

Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12965–12974, 2020. 3

Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenas: Improving neural architecture search via attentive sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6418–6427, 2021. 3, 17

H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv:2005.14187[cs.CL]*, 2020a. 24

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7675–7688, Online, July 2020b. Association for Computational Linguistics. 1, 7, 9, 21, 24

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 139–149, 2022. 4, 22

Colin White, Mahmoud Safari, Rhea Sanjay Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *ArXiv*, abs/2301.08727, 2023. 1

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019. 3, 17

Yan Wu, Zhiwu Huang, Suryansh Kumar, Rhea Sanjay Sukthanker, Radu Timofte, and Luc Van Gool. Trilevel neural architecture search for efficient single image super-resolution. *arXiv preprint arXiv:2101.06658*, 2021. 3, 17

S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. 3, 5

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020a. 3

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Bowen Shi, Qi Tian, and Hongkai Xiong. Latency-aware differentiable neural architecture search. *arXiv preprint arXiv:2001.06392*, 2020b. 3, 17

Xinmin Yang, Wei Yao, Haian Yin, Shangzhi Zeng, and Jin Zhang. Gradient-based algorithms for multi-objective bi-level optimization. *Science China Mathematics*, 2024. 40

Feiyang Ye, Baijiong Lin, Xiaofeng Cao, Yu Zhang, and Ivor W. Tsang. A first-order multi-gradient algorithm for multi-objective bi-level optimization. In *ECAI*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pp. 2621–2628. IOS Press, 2024. 40

Mao Ye and Qiang Liu. Pareto navigation gradient descent: a first-order algorithm for optimization in pareto set. In *Uncertainty in Artificial Intelligence*, pp. 2246–2255. PMLR, 2022. 17

A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1gDNyrKDS`. 10, 40

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12104–12113, 2022. 1

Dingkun Zhang, Sijia Li, Chen Chen, Qingsong Xie, and Haonan Lu. Laptop-diff: Layer pruning and normalized distillation for compressing diffusion models. *arXiv preprint arXiv:2404.11098*, 2024a. 10

Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 692–693, 2020. 17

Miao Zhang, Steven Su, Shirui Pan, Xiaojun Chang, Ehsan Abbasnejad, and Reza Haffari. idarts: Differentiable architecture search with stochastic implicit gradients. In *International Conference on Machine Learning*, 2021. 3

Qi Zhang, Peiyao Xiao, Shaofeng Zou, and Kaiyi Ji. Mgda converges under generalized smoothness, provably. 2024b. 40

Yiyang Zhao, Linnan Wang, Kevin Yang, Tianjun Zhang, Tian Guo, and Yuandong Tian. Multi-objective optimization by learning space partition. In *International Conference on Learning Representations*, 2022. 10, 19

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. 17

A EXTENDED RELATED WORK

Multi-objective optimization. Multi-objective optimization (MOO) (Gunantara, 2018) is a crucial field in optimization theory, tackling decision-making scenarios with multiple conflicting objectives. MOO techniques can be categorized into gradient-based and gradient-free approaches. *Gradient-free* MOO approaches, such as evolutionary algorithms and dominance-based methods like NSGA-II (Deb et al., 2000), often suffer from sample inefficiency and are typically unsuitable for deep learning applications. On the other hand, *gradient-based* MOO methods leverage gradients. The foundational work by Désidéri (2012) has been significantly extended in multi-task learning contexts, demonstrating considerable potential (Lin et al., 2019; Liu & Vicente, 2021; Mahapatra & Rajan, 2020; Sener & Koltun, 2018). However, these methods are primarily applied to fixed architectures, and adapting them to architecture search spaces is complex. This adaptation would require retraining each architecture with multiple objectives, which is impractically expensive for large search spaces. Another major challenge in MOO is balancing the different objectives. To address this, preference vectors have been proposed to guide the prioritization of objectives on the Pareto Front (Momma et al., 2022; Ye & Liu, 2022). An emerging approach to mitigate the retraining issue involves hypernetworks, which determine the weights of the main network in MOO scenarios (Lin et al., 2020), often incorporating preference vector (Hoang et al., 2023; Navon et al., 2021; Phan et al., 2022).

Neural Architecture Search. A major challenge in the automated design of neural network architectures is the efficient exploration of vast search spaces. Early NAS methods relied on Reinforcement Learning (Zoph & Le, 2017), evolutionary algorithms (Deb et al., 2002; Elsken et al., 2019b; Lu et al., 2020), and other black-box optimization techniques (Daulton et al., 2022) to train and evaluate numerous architectures from scratch. The advent of one-shot NAS introduced weight sharing among architectures by training an over-parameterized network, known as a supernet, to expedite the evaluation of individual networks within the search space (Bender et al., 2018; Liu et al., 2019; Pham et al., 2018; Saxena & Verbeek, 2016). Differentiable one-shot NAS methods (Cai et al., 2018; Fu et al., 2020; He et al., 2020; Wu et al., 2019; 2021) further improved efficiency by applying a continuous relaxation to the search space, enabling the use of gradient descent to identify optimal sub-models within the supernet. In contrast, two-stage NAS methods initially train a supernet, often through random sampling of subnetworks, and subsequently employ black-box optimization to identify optimal subnetworks (Bender et al., 2018; Guo et al., 2020; Li & Talwalkar, 2020).

Hardware-aware and Multi-objective Neural Architecture Search. Early NAS methods primarily focused on maximizing accuracy for a given task. In contrast, hardware-aware NAS aims to optimize architectures for efficient performance on specific hardware devices (Benmeziane et al., 2021; Lee et al., 2020; Shaw et al., 2019; Zhang et al., 2020), naturally leading to multi-objective NAS (Hsu et al., 2018; Kim et al., 2021; Tan et al., 2019). Two-stage NAS methods can be adapted to this context by incorporating a multi-objective search in the second stage (Cai et al., 2018; Ito & Von Zuben, 2023). However, most two-stage methods depend on random sampling during supernet training, which doesn't prioritize promising architectures. Differentiable NAS methods, such as those in Cai et al. (2018); Fu et al. (2020); Jiang et al. (2021); Wang et al. (2021); Wu et al. (2019; 2021); Xu et al. (2020b), use latency proxies like layer-wise latencies and FLOPS (Dudziak et al., 2020) to evaluate hardware performance, combining task and hardware objectives with fixed weighting to find a single optimal solution. However, changing the objective weighting requires a complete search rerun, which is computationally demanding.

In contrast, our proposed search algorithm offers the entire Pareto Front of objectives in a single run, making it more efficient. While our focus is on multi-objective NAS for hardware constraints, our technique is applicable to other objectives such as fairness (Das & Dooley, 2023; Dooley et al., 2023; Martinez et al., 2020), suggesting promising avenues for future research.

B ALGORITHMIC COMPONENTS

In this section, we provide the pseudocodes for some of the algorithmic components we use in MODNAS.

B.1 DISCRETE SAMPLERS

Given the architecture parameters $\tilde{\alpha}_\Phi$ from the `MetaHypernetwork`, we obtain a differentiable discrete architecture sample from the `Architect` as $\alpha_\Phi \leftarrow \pi - \texttt{stop_g}(\pi) + \alpha_\Phi$, where $\alpha_\Phi \sim \texttt{Cat}\big(\texttt{softmax}_1(\tilde{\alpha}_\Phi)\big)$ and

$$\pi \leftarrow 2 \cdot \texttt{softmax}_1\Big(\texttt{stop_g}\big(\texttt{ln}(\frac{\alpha_\Phi + \texttt{softmax}_\tau(\tilde{\alpha}_\Phi)}{2}) - \tilde{\alpha}_\Phi\big) + \tilde{\alpha}_\Phi\Big) - \frac{\texttt{softmax}_1(\tilde{\alpha}_\Phi)}{2}.$$

Here, `Cat` is the categorical distribution, τ is the temperature in the tempered softmax $\texttt{softmax}_\tau(\alpha)_i = \frac{exp(\alpha_i/\tau)}{\sum_{j=1}^{|\mathcal{O}|} exp(\alpha_j/\tau)}$, and $\texttt{stop_g}(\cdot)$ duplicates its input and detaches it from backpropagation. Refer to the ReinMax paper (Liu et al., 2023) for more details. The algorithm pseudocode on how a one-hot encoded (discrete) architecture is sampled given an unnormalized architectural distribution $\tilde{\alpha}$ is given in Algorithm 2 and Algorithm 3, for the Straight-Through (Jang et al., 2017) and ReinMax (Liu et al., 2023) gradient estimators, respectively.

Algorithm 2: `Straight-Through` (Jang et al., 2017)

Data: $\tilde{\alpha}$: softmax input, τ : temperature
Result: α: one-hot samples

$\pi_0 \leftarrow \texttt{softmax}_1(\tilde{\alpha})$
$\alpha \sim \texttt{Cat}(\pi_0)$
$\pi_1 \leftarrow \texttt{softmax}_\tau(\tilde{\alpha})$
$\alpha \leftarrow \pi_1 - \texttt{stop_g}(\pi_1) + \alpha$
return α

Algorithm 3: `ReinMax` (Liu et al., 2023)

Data: $\tilde{\alpha}$: softmax input, τ : temperature
Result: α: one-hot samples

$\pi_0 \leftarrow \texttt{softmax}_1(\tilde{\alpha})$
$\alpha \sim \texttt{Cat}(\pi_0)$
$\pi_1 \leftarrow \frac{\alpha + \texttt{softmax}_\tau(\tilde{\alpha})}{2}$
$\pi_1 \leftarrow \texttt{softmax}_1\big(\texttt{stop_g}\big(\texttt{ln}(\pi_1) - \tilde{\alpha}\big) + \tilde{\alpha}\big)$
$\pi_2 = 2 \cdot \pi_1 - \frac{1}{2} \cdot \pi_0$
$\alpha \leftarrow \pi_2 - \texttt{stop_g}(\pi_2) + \alpha$
return α

B.2 FRANK-WOLFE SOLVER

In this section, we provide the pseudocode of the Frank-Wolfe solver (Jaggi, 2013) used to compute the gradient coefficients used for the MGD updates. To solve the constrained optimization problem, the Frank-Wolfe solver uses analytical solution for the line search with $T = 2$ (Algorithm 5).

Algorithm 4: `FrankWolfeSolver` (Jaggi, 2013)

Data: $g_\Phi^1, \ldots, g_\Phi^T$
Result: $\gamma = (\gamma_1, \ldots, \gamma_T)$

`Initialize` $\gamma \leftarrow (\frac{1}{T}, \ldots, \frac{1}{T})$
`Precompute` $\mathcal{M}$ s.t. $\mathcal{M}_{i,j} = (g_\Phi^i)^{\mathbf{T}}(g_\Phi^j)$
repeat
  $\hat{t} \leftarrow \text{argmin}_r \sum_{t=1}^{T} \gamma_t \mathcal{M}_{rt}$
  $e_{\hat{t}} \leftarrow \mathcal{M}_{\hat{t},\cdot}$; `// ` $\hat{t}$`-th row of` $\mathcal{M}$
  $\hat{\delta} \leftarrow \text{argmin}_\delta \big((1-\delta)\gamma + \delta e_{\hat{t}}\big)^{\mathbf{T}} \mathcal{M}\big((1-\delta)\gamma + \delta e_{\hat{t}}\big)$; `// using Algorithm 5`
  $\gamma \leftarrow (1-\hat{\delta})\gamma + \hat{\delta} e_{\hat{t}}$
until $\hat{\delta} \sim 0$ *or Number of Iterations Limit*;
return γ

Algorithm 5: Solver $\min_{\delta \in [0,1]} ||\delta\theta + (1-\delta)\bar{\theta}||_2^2$

if $\theta^{\mathbf{T}}\bar{\theta} \geq \theta^{\mathbf{T}}\theta$ **then**
  $\delta \leftarrow 1$
else if $\theta^{\mathbf{T}}\bar{\theta} \geq \bar{\theta}^{\mathbf{T}}\bar{\theta}$ **then**
  $\delta \leftarrow 0$
else
  $\delta \leftarrow \frac{(\bar{\theta}-\theta)^{\mathbf{T}}\bar{\theta}}{||\theta-\bar{\theta}||_2^2}$
return δ

C MULTI-OBJECTIVE NAS ALGORITHMS

This section elaborates on the multi-objective NAS methods we utilize as baselines in Section 4.

- **Random Search (RS)** is a robust baseline for both single-objective (Bergstra & Bengio, 2012; Li & Talwalkar, 2020) and multi-objective (Cai et al., 2020; Chen et al., 2021a) architecture searches. This baseline involves randomly sampling architectures from the search space and computing the Pareto front from these samples. While RS is computationally efficient and often effective, it may not always find the optimal architectures, especially in larger search spaces.
- **Local Search (LS)** is adapted to refine solutions near Pareto-optimal points in multi-objective optimization, iteratively improving solutions within defined neighborhoods.
- **Multi-objective Asynchronous Successive Halving (MO-ASHA)** (Schmucker et al., 2021) is a multi-fidelity method that utilizes an asynchronous successive halving scheduler (Li et al., 2018) and non-dominating sorting for budget allocation. MO-ASHA uses the NSGA-II selection mechanism and the ϵ-net (Salinas et al., 2021) exploration strategy that ranks candidates in the same Pareto set by iteratively selecting the one with the largest Euclidian distance from the previous set of candidates.
- **Multi-Objective Regularized Evolution (MO-RE)** builds on Regularized Evolution (RE) (Real et al., 2019), which evolves a population of candidates through mutation and periodically removes the oldest individuals, thus regularizing the population. MO-RE adapts this by using multi-objective non-dominated sorting to score candidates, with parents sampled based on these scores.
- **Non-dominated Sorting Genetic Algorithm II (NSGA-II)** (Deb et al., 2002) is a multi-objective evolutionary algorithm designed to find a Pareto set of architectures. It ranks architectures using non-dominated sorting and maintains diversity with crowding distance. Through selection, crossover, and mutation, NSGA-II evolves populations towards the Pareto front, although it is known for being sample inefficient.
- **Covariance Matrix Adaptation Evolution Strategy (CMA-ES)** (Igel et al., 2007) is an evolutionary algorithm particularly effective in continuous optimization problems. In a multi-objective context, it adapts its covariance matrix to the shape of the search space, iteratively updating its sampling distribution to favor promising regions. This method efficiently handles complex, non-linear optimization landscapes and can be adapted to multi-objective scenarios by using techniques such as Pareto-based selection to maintain a diverse set of solutions.
- **Latent Action MOO (LaMOO)** (Zhao et al., 2022) uses a parametric model and Monte Carlo Tree Search (MCTS) to learn to partition the objective space based on the dominance number, which indicates the vicinity of a point to the Pareto front relative to the other samples. qEHVI+LaMOO and CMA-ES+LaMOO use the original qEHVI and CMA-ES, respectively, as an inner routine in the learned subspaces.
- **Bayesian Optimization with Random Scalarizations (RS-BO)** (Paria et al., 2020) uses an acquisition function based on random linear scalarizations of objectives across multiple points to find the Pareto-optimal set that minimizes Bayesian regret.
- **Bayesian Optimization with Linear Scalarizations (LS-BO)** is similar to RS-BO but optimizes a single objective derived from a fixed linear combination of two objectives instead of using randomized linear scalarizations.

- **Expected Hypervolume Improvement (qEHVI)** (Daulton et al., 2020) is a Bayesian optimization acquisition function that explores the Pareto front by quantifying potential hypervolume improvement. This approach measures the volume dominated by Pareto-optimal solutions and guides the search towards regions likely to offer better trade-offs, aiding in the discovery of diverse Pareto-optimal solutions.

D EVALUATION DETAILS

D.1 OTHER METRICS

For NAS-Bench-201, in addition, we evaluate the *generational distance* (GD) and *inverse generational distance* (IGD) (see Appendix D). See Figure 20 for the results complementary to the hypervolume radar plot in Figure 3 of the main paper.

Generational Distance (GD) and Inverse Generational Distance (IGD). Given a *reference set* $\mathcal{S} \subset \mathcal{A}$ and a Pareto set $\mathcal{P}_\alpha \subset \mathcal{A}$ with $dim(\mathcal{A}) = K$, the GD indicator is defined as the distance between every point $\alpha \in \mathcal{P}_\alpha$ and the closest point in $s \in \mathcal{S}$, averaged over the size of $\mathcal{P}_\alpha$:

$$GD(\mathcal{P}_\alpha, \mathcal{S}) = \frac{1}{|\mathcal{P}_\alpha|}\left(\sum_{\alpha \in \mathcal{P}_\alpha} \min_{s \in \mathcal{S}} d(\alpha, s)^2\right)^{1/2},$$

where $d(\alpha, s) = \sqrt{\sum_{k=1}^{K}(\alpha_k - s_k)^2}$ is the Euclidean distance from α to its nearest reference point in $\mathcal{S}$.

The inverted generational distance (IGD) is computed as $IGD(\mathcal{P}_\alpha, \mathcal{S}) = GD(\mathcal{S}, \mathcal{P}_\alpha)$.

Generational Distance Plus (GD^+) and Inverse Generational Distance Plus (IGD^+). $GD^+(\mathcal{P}_\alpha, \mathcal{S}) = IGD^+(\mathcal{S}, \mathcal{P}_\alpha)$ replaces the euclidean distance $d(\alpha, s)$ in GD with:

$$d^+(\alpha, s) = \sqrt{\sum_{k=1}^{K}(\max\{\alpha_k - s_k, 0\})^2}$$

D.2 MODNAS-SOTL

On the NAS-Bench-201 search space, since the architectures evaluated with the supernetwork weights are not highly correlated to the ones trained independently from scratch, we employ the Sum of Training Losses (SoTL) proxy from Ru et al. (2021). To profile the Pareto front with SoTL, we firstly evaluate the 24 architectures using the exponential moving average of the sum of training losses for the initial 12 epochs of training as $\sum_{e=1}^{12} 0.9^{12-e}\mathcal{L}^{train}(\boldsymbol{w}, \alpha)$, and then train from scratch only the subset of architectures in the Pareto set built using the SoTL evaluations. We present the results of MODNAS-SoTL in Figure 20, where we compare to the other baselines as well. As we see, we can further decrease the evaluation cost via MODNAS-SoTL, by trading off the number of solutions in the Pareto set with HV.

E EXPERIMENTAL DETAILS

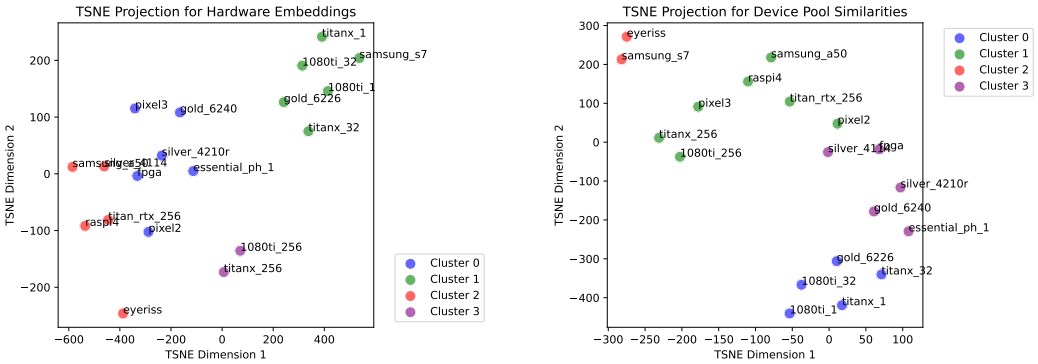
E.1 `MetaPredictor` ARCHITECTURES

For all search spaces we set the dimensionality of the hardware embedding to 10. This corresponds to latency evaluations on a set of 10 reference architectures, which are the same used by Lee et al. (2021b).

NAS-Bench-201. For the NAS-Bench-201 (Dong & Yang, 2020) search space we use a Graph Convolutional Network (GCN) as proposed in Dudziak et al. (2020). Furthermore, in addition to the *one-hot operation encoding* and *adjacency matrix* corresponding to the architecture cells, we also input the hardware embedding to this predictor, as done by Lee et al. (2021b). The number of nodes in the GCN is 8 and the dimensionality of the layers is set to 100 following HELP (Lee et al., 2021b). In order to show the effectiveness of our `MetaHypernetwork` to learn the hardware

device similarities, in Figure 12 we cluster the original device embedding vectors and the learned `MetaHypernetwork` embeddings using K-means clustering after reducing their dimensionality using t-SNE. As we can see, the `MetaHypernetwork` learns to cluster similar devices together in latent space, demonstrating the efficacy of our algorithm.

MobileNetV3 (OFA). Following HELP (Lee et al., 2021b), we employ a simple feedforward neural network in the MobileNetV3 search space. The input dimension of the `MetaPredictor` is set to 160, matching the concatenated architecture encoding dimension. We set the size of the hidden layers to 100. Specifically, the `MetaPredictor` comprises 2 linear layers with ReLU activation for processing the 160-dimensional one-hot architecture encoding and 2 linear layers for processing the hardware embedding. The outputs from these two paths are concatenated and passed through a final linear layer to predict the latency.



(a) Original hardware embeddings

(b) `MetaHypernetwork` embeddings

Figure 12: K-means clustering on the t-SNE projections of the original hardware device embeddings and learned embeddings from the `MetaHypernetwork` on NB201.

Seq-Seq Transformer (HAT). HELP [3] does not release the architecture or the meta-learned pretrained predictor for HAT(Wang et al., 2020b). However, HAT [4] releases code and pretrained models for each of the devices and tasks trained independently. Hence, we build our single per-task `MetaPredictor` based on the architecture of the HAT predictor, i.e. a simple feedforward neural network. The input dimension corresponds to the one-hot architecture encoding of the candidate Transformer architecture. Additionally, to condition on the hardware embedding, we include 2 extra linear layers for processing the hardware embedding, which is then concatenated with the processed architecture encoding to produce the final latency prediction. The hidden dimension of the `MetaHypernetwork` is set to 400, with 6 hidden layers. The predictor's input feature dimension is 130.

HW-GPT-Bench. We utilize the raw energy observations released in (Sukthanker et al., 2024) to train a single hardware-aware meta-predictor across energy observations from eight GPU types. Our meta-predictor is a simple MLP, similar to the one in HAT, with 4 hidden layers, 2 layers for processing the hardware embedding (which the network is conditioned on). The MLP's hidden dimension is 256, and the input feature dimension matches the one-hot encoded architecture feature map for this space, i.e., 80.

E.2 `MetaHypernetwork` Architecture

Given a preference vector $\boldsymbol{r} \in \mathbb{R}^M$, we use the hypernetwork $h_\phi(\boldsymbol{r}) : \mathbb{R}^M \to \mathcal{A}$, parameterized by $\phi \in \mathbb{R}^n$, to generate an un-normalized architecture distribution $\tilde{\alpha}$ that is later used to compute the upper-level updates in (4). In our experiments, h_ϕ is composed of $M - 1$ [5] embedding layers

[3] `https://github.com/HayeonLee/HELP`

[4] `https://github.com/mit-han-lab/hardware-aware-transformers`

[5] $m = 1$ (CE loss) does not have an hardware embedding.

e^m, $m \in \{2, \dots, M\}$ with n_m possible learnable vectors of size $\frac{dim(\mathcal{A})}{M-1}$. The output of h_ϕ is the concatenation of all $M-1$ outputs of e^m, such that its size matches $dim(\mathcal{A})$. See Figure 13 for details.

In order to enable the hypernetwork to generate architectures across multiple devices, inspired by Wang et al. (2022) and Lin et al. (2020), we propose a `MetaHypernetwork` $H_\Phi(\boldsymbol{r}, d_t) : \mathbb{R}^M \times \mathcal{H}^{M-1} \rightarrow \mathcal{A}$ that can meta-learn across T different hardware devices (see Figure 1). The input to H_Φ is a concatenation of device feature vectors across all metrics, i.e. $d_t = \oplus_{m=2}^{M} d_t^m$. Similar to Lee et al. (2021b), $d_t^m \in \mathcal{H}$ is a fixed-size feature vector representative of device $t \in \{1, \dots, T\}$ and objective $m \in \{2, \dots, M\}$, that is obtained by evaluating a fixed set of reference architectures for a given metric. The `MetaHypernetwork`, with $\Phi = \cup_{k=0}^{K} \phi_k$ parameters, contains a bank of $K > T$ hypernetworks $\{h_{\phi_k}(\boldsymbol{r})\}_{k=1}^{K}$ and an additional linear layer $e_{\phi_0}(d_t) : \mathcal{H}^{M-1} \rightarrow \mathbb{R}^K$ at the beginning, that learns a similarity map for every device feature to the hypernetworks' bank. If we denote by $h_{\phi_{1:k}} = (h_{\phi_1} \cdots h_{\phi_k})^{\mathbf{T}}$ the vector of all hypernetworks in the bank, then, given a preference vector $\boldsymbol{r}$, to obtain $\tilde{\alpha}$ for device t, we compute a weighted mixture of predictions of all h_ϕ in the hypernetwork bank as follows:

$$\tilde{\alpha}_\Phi = H_\Phi(\boldsymbol{r}, d_t) = \sum_{k=1}^{K} e_{\phi_0}(d_t)[k] \cdot h_{\phi_k}(\boldsymbol{r}) = e_{\phi_0}(d_t) \cdot h_{\phi_{1:k}}(\boldsymbol{r}).$$

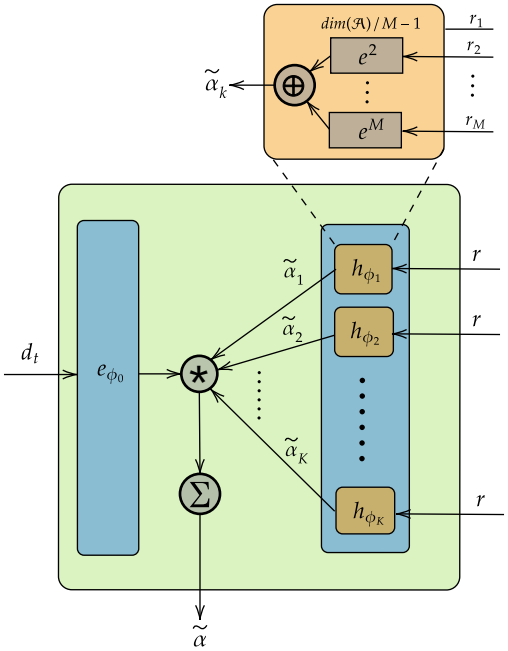


Figure 13: `MetaHypernetwork` architecture overview in the case of M objectives. Note that $m = 1$ is reserved for the accuracy objective, which we model through the cross-entropy loss in the `Supernetwork`. The initial linear layer e_{ϕ_0} gets the d_t hardware embedding and outputs a weight that scales each of the K hypernetworks' (orange boxes) outputs from the hypernetwork bank. The scaled architectural parameters are then summed up element-wise. All individual hypernetwork h_{ϕ_k} get as input the same scalarization $\boldsymbol{r}$. Each of them has $M-1$ embedding layers with dimensions $n_m \times \frac{dim(\mathcal{A})}{M-1}, \forall m \in \{2, \dots, M\}$ that gets as input the scalarizations for objectives $m = 2, \dots, m = M$, and yields a vector of size $\frac{dim(\mathcal{A})}{M-1}$. The output from the $M-1$ embedding layers are concatenated to give the architecture encoding $\tilde{\alpha}$.

We keep the `MetaHypernetwork` architecture similar across search spaces. The only thing we adapt is the output dimensionality of the hypernetwork (in the hypernetwork bank of `MetaHypernetwork`), which corresponds to the dimensionality of the architecture parameters of the respective search space. We set the size of the initial hardware embedding layer and the hypernetwork bank to 50 for all search spaces. Furthermore, each hypernetwork has 100 possible learnable embeddings e^m, for every objective $m \in \{2, \dots, M\}$, to map the scalarization vector to an architecture. We quantize the continuous sampled $r_m \in [0, 1]$ to the discrete $[0, 1, \dots 100]$ interval before indexing the respective embedding layers. See Figure 2 for an illustration of the `MetaHypernetwork` architecture.

For the **NAS-Bench-201** search space, we use a single embedding layer of dimensionality 30, corresponding to the dimensionality of the architecture space: 6×5 (6 edges and 5 operation choices on each edge). For the 3-objective experiment, we include an additional embedding for the energy usage objective, concatenated with the latency embedding before passing it to the `MetaHypernetwork`. The individual hypernetworks in the `MetaHypernetwork` bank have 2 embedding layers with dimensionality 15, whose outputs are concatenated to match the architecture space dimensions.

In the **MobileNetV3** space, we use 4 embedding layers – for depth, expansion ratio, kernel size, and resolution. The space comprises 5 blocks, each with 3 depth choices, making the depth embedding layer dimensionality 5×3. The kernel and expansion embedding layers have dimensions $5 \times 4 \times 3$, corresponding to 5 blocks with a maximum depth of 4 and 3 possible kernel size or expansion ratio

choices. The resolution embedding layer has a dimension of 25, representing 25 possible resolution choices.

In the **Seq-Seq Transformer (HAT)** space, the individual hypernetworks of the `MetaHypernetwork` utilize 9 embedding layers (the encoder layer count is fixed; see Table 3):

- 2 embedding layers of size 2 for the encoder and decoder blocks to map the scalarization to the embedding dimension architecture parameter, held constant throughout the encoder or decoder block.
- 2 embedding layers with dimensions 6×3 (6 encoder/decoder layers, 3 choices) for the linear layer size in every attention block for both encoder and decoder.
- 2 embedding layers with dimensions 6×2 for the number of heads in each attention block.
- 1 embedding layer of size 6 to encode the 6 possible choices for the number of layers in the decoder.
- 1 embedding layer of size 6×3 (6 encoder layers, 3 choices) for the arbitrary encoder layer choice for attention.
- 1 embedding layer of size 6×2 (6 encoder layers, 2 choices) for the number of heads in the encoder-decoder attention.

For the **HW-GPT-Bench** space, the individual hypernetworks of the `MetaHypernetwork` contain 5 embedding layers:

- 1 embedding layer of dimension 1×3 for mapping the scalarization to the embedding dimension architecture parameter of the language model, with 3 choices.
- 1 embedding layer of dimension 1×3 for mapping the scalarization to the layer number dimension architecture parameter of the language model, with 3 choices.
- 1 embedding layer of dimension 12×3 for mapping the scalarization to the mlp_ratio dimension architecture parameter of the language model, with 12 layers and 3 mlp_ratio choices per layer.
- 1 embedding layer of dimension 12×3 for mapping the scalarization to the num_heads dimension architecture parameter of the language model, with 12 layers and 3 choices per layer.
- 1 embedding layer of dimension 2 for toggling the bias in linear layers on or off.

E.3 MODNAS Hyperparameter Configurations

In Table 6, we show the search hyperparameters and their corresponding values we use to conduct our experiments with MODNAS. For the convolutional spaces we subtract a cosine similarity penalty from the scalarized loss following (Ruchte & Grabocka, 2021):

$$g_\Phi^t \leftarrow \boldsymbol{r}^{\mathbf{T}} \nabla_\Phi \mathbf{L}_t(\mathcal{D}_{valid}, \boldsymbol{w}, \alpha_\Phi) - \lambda \nabla_\Phi \frac{\boldsymbol{r}^{\mathbf{T}} \mathbf{L}_t(\mathcal{D}_{valid}, \boldsymbol{w}, \alpha_\Phi)}{||\boldsymbol{r}|| \; ||\mathbf{L}_t(\mathcal{D}_{valid}, \boldsymbol{w}, \alpha_\Phi)||}, \tag{6}$$

where $||\cdot||$ is the l_2 norm. We set λ to 0.001. Empirically we did not observe significant differences on disabling the cosine penalty term.

E.4 Normalization of objectives

Since our method relies on a *scalarization* of different objectives, it is important that the objectives being optimized are on the same scale. For simplicity, lets consider the scenario where the two objectives of interest are the *cross-entropy* loss and *latency*. Since we pretrain and freeze our `MetaPredictor`, the latency-scale remains constant throughout the search, while the cross-entropy loss of the `Supernetwork` (likely) decreases over time. To this end, we use the following max-min normalization to normalize the objectives:

$$\mathcal{L}_t^m(\cdot, \alpha_\Phi) = \frac{\mathcal{L}_t^m(\cdot, \alpha_\Phi) - \min(\bar{\mathbf{L}})}{\max(\bar{\mathbf{L}}) - \min(\bar{\mathbf{L}})}, \tag{7}$$

where $\bar{\mathbf{L}} = \bigcup_{i=1}^{N} \texttt{stop_g}\big(\mathcal{L}_t^m(\cdot, \alpha_i)^i\big)$ is the set of losses evaluated on N architectures and potentially N previous steps. For the latency objective, we precompute these sample-statistics using N samples (ground-truth for NAS-Bench-201 and predicted for OFA and HAT spaces) from the search space, whilst for the cross-entropy loss we compute them throughout the search. Furthermore, to take into account the decreasing cross-entropy, we reset the cross-entropy loss statistics after every epoch.

F DETAILS ON SEARCH SPACES

NAS-Bench-201 (Dong & Yang, 2020) is a convolutional, cell-based search space. The search space consists of 3 stages, each with number of channels 16, 32 and 64, respectively. Each stage contains a convolutional cell repeated 5 times. Here, every cell is represented as a directed acyclic graph (DAG) which has 4 nodes, densely connected with 6 edges. Each edge has 5 possible operation choices: a skip connection, a zero operation, a 3×3 convolution, a 5×5 convolution or an average pooling operation. NAS-Bench-201 is a tabular benchmark exhaustively constructed, where the objective is finding the optimal cell for the given macro skeleton.

Table 3: Encoder-Decoder Search Space for HAT.

Module	Searchable Dim	Choices
Encoder	No. of Layers	[6] (fixed)
	Embedding dim	[640, 512]
	No. of heads	[8, 4]
	FFN dim	[3072, 2048, 1024]
Decoder	No. of layers	[6, 5, 4, 3, 2, 1]
	Embedding dim	[640, 512]
	No. of heads	[8, 4]
	FFN dim	[3072, 2048, 1024]
	Arbitrary-Encoder-Layer	[-1, 1, 2]
	Enc-Dec attention num heads	[8, 4]

MobileNetV3 proposed in OFA (Cai et al., 2020) is a macro convolutional search space. The different searchable dimensions in the search space are the depth (per block), the kernel size (for every layer in every block) and the channel expansion ratio (for every layer in every block). There are a total of 5 blocks, each with 3 possible depth choices and every layer in this block has 3 possible kernel sizes and channel expansion ratio choices. This amounts to a total search space size of $((3\times3)^2+(3\times3)^3+(3\times3)^4)^5 \approx 2\times10^{19}$. Additionally, every architecture has 25 possible choices for the size of the input resolution. The 3 possible choices for depth, kernel size and expansion ratio are $\{2,3,4\}$, $\{3,5,7\}$ and $\{3,4,6\}$, respectively. The input resolution choices are {128, 132, 136, 140, 144, 148, 152, 156, 160, 164, 168, 172, 176, 180, 184, 188, 192, 196, 200, 204, 208, 212, 216, 220, 224}. We use a width factor of 1.2 similar to OFA (Cai et al., 2020).

Seq-Seq Encoder-Decoder Transformer (HAT) (Wang et al., 2020a) for the En-De machine translation task has a searchable number of layers, embedding dimension, feedforward expansion layer dim per-layer, number of heads per-layer for both the encoder and the decoder sub-modules. In addition to this, the number of encoder layers the decoder attends to, and the number of attention heads in the encoder-decoder attention is also searchable. We present the details of the search space in Table 3.

HW-GPT-Bench (Sukthanker et al., 2024) is a decoder-only transformer space designed for autoregressive language modeling. The search space includes choices for embedding dimensions {768, 384, 192}, the number of layers from {10, 11, 12}, the MLP expansion ratio per layer from {2,3,4}, the number of heads per layer from {12,8,4}, and the option to toggle the bias parameter on or off in the layers.

G DATASETS AND DEVICES

This section describes the hardware devices and tasks used to evaluate MODNAS and the MOO baselines throughout the paper. We assess our methods across small- and large-scale image classification datasets, including CIFAR-10 and ImageNet-1K. For the machine translation task, we evaluate our method on the WMT'14 En-De dataset (Macháček & Bojar, 2014), and we use the OpenWebText (Gokaslan & Cohen, 2019) dataset for language modeling. Furthermore, we evaluate MODNAS across 19 devices on NAS-Bench-201, 12 devices on MobileNetV3, three devices on Seq-Seq Transformer, and eight devices from HW-GPT-Bench (Sukthanker et al., 2024), with zero-shot generalization to test devices. Table 4 lists the devices used. For more details on the devices, we refer readers to Lee et al. (2021b), Cai et al. (2020), Wang et al. (2020b), Li et al. (2021), and Sukthanker et al. (2024).

H RUNTIME COMPARISON

In Table 5 we provide the number of GPU hours we ran MODNAS and baselines on every search space. We ran the search on NAS-Bench-201, OFA, together with the evaluations on Nvidia RTX2080Ti, while for HAT we used NVidia A6000. For both OFA and HAT, we used 8 GPUs in parallel. Similar as in Sukthanker et al. (2024), on the HW-GPT-Bench space we ran the MODNAS search and evaluations on 4 Nvidia A100 GPUs.

Table 4: Search-test split for hardware devices and datasets for different search spaces.

Search Space	Train-devices	Test devices	Dataset
NAS-Bench-201	1080ti_1, 1080ti_32, 1080ti_256, silver_4114, silver_4210r, samsung_a50, pixel3, essential_ph_1, samsung_s7, titanx_1, titanx_32, titanx_256, gold_6240	titan_rtx_256, gold_6226, fpga, pixel2, raspi4, eyeriss	CIFAR10
MobileNetV3 (OFA)	2080ti_1, 2080ti_32, 2080ti_64, titan_xp_1, titan_xp_32, titan_xp_64, v100_1, v100_32, v100_64, titan_rtx_1, titan_rtx_32	titan_rtx_64	ImageNet-1k
Seq-Seq Transformer (HAT)	titanxp gpu, cpu xeon	cpu raspberrypi	WMT14.en-de
HW-GPT-Bench	a40, v100, rtx2080, rtx3080	a100, h100, P100, a6000	OpenWebText

Table 5: Total amount of GPU hours required to run MODNAS' and baselines' search on every search space.

Search Spaces	Method	Lat/En/Mem Pred.	Supernet	Acc./Ppl Pred.	Search	Total Time
NASBench201	MetaD2A+HELP	25	-	8629	0.3	8654.3
	MOO Baselines	-	-	-	370.5	370.5
	MODNAS	3	22	-	0.05	**25.25**
Once-For-All	OFA+HELP	6	1200	356	10	1572
	MOO Baselines	6	1200	356	192	1754
	MODNAS	6	1392	-	0.05	**1398.25**
HAT	HAT	15	346.7	-	210.9	572.6
	MOO Baselines	15	346.7	-	576	937.7
	MODNAS	5	576	-	0.05	**581.25**
HW-GPT-Bench	MOO Baselines	1	192	-	48	241
	MODNAS	1	216	-	0.05	**217.25**

Table 6: Hyperparameters used on different search spaces

Search Space	Hyperparameter Type		Value
NAS-Bench-201	`MetaHypernetwork`	learning rate	3e-4
		weight decay	1e-3
		embedding layer size	100
		hypernetwork bank size	50
		optimizer	Adam
		ReinMax temperature	1
	`Supernetwork`	learning rate	0.025
		momentum	0.9
		weight decay	0.0027
		learning rate scheduler	cosine
		epochs	100
		batch size	256
		gradient clipping	5
		cutout	true
		cutout length	16
		initial channels	16
		optimizer	SGD
		train portion	0.5
MobileNetV3 (OFA)	`MetaHypernetwork`	learning rate	1e-5
		weight decay	1e-3
		embedding layer size	100
		hypernetwork bank size	50
		optimizer	Adam
		ReinMax temperature	1
	`Supernetwork`	learning rate	1e-3
		momentum	0.9
		weight decay	3e-5
		learning rate scheduler	cosine
		epochs	50
		batch size	32
		bn_momentum	0.1
		bn_eps	1e-5
		dropout	0.1
		width	1.2
		optimizer	SGD
		train portion	1.0
Seq-Seq Transformer (HAT)	`MetaHypernetwork`	learning rate	3e-4
		weight decay	1e-3
		embedding layer size	100
		hypernetwork bank size	50
		optimizer	Adam
		ReinMax temperature	1
	`Supernetwork`	learning rate	1e-7
		momentum	0.9
		weight decay	0.0
		learning rate scheduler	cosine
		epochs	110
		batch size/max-tokens	4096
		criterion	label_smoothed_cross_entropy
		attention-dropout	0.1
		dropout	0.3
		precision	float32
		optimizer	Adam
		train portion	1.0
HW-GPT-Bench	`MetaHypernetwork`	learning rate	1e-5
		weight decay	1e-3
		embedding layer size	100
		hypernetwork bank size	50
		optimizer	Adam
		ReinMax temperature	1
	`Supernetwork`	learning rate	0.000316
		momentum	-
		weight decay	0.1
		learning rate scheduler	cosine
		steps	800k
		batch size/max-tokens	32768
		criterion	cross_entropy
		attention-dropout	0.0
		dropout	0.0
		precision	bfloat16
		optimizer	AdamW
		train portion	1.0

I ADDITIONAL EXPERIMENTS

I.1 PREDICTED V/S GROUND-TRUTH LATENCIES

In Figure 8, we present the scatter plots of the predictions of our hardware-aware `MetaPredictor` vs. the ground-truth latencies of different architectures. In the figure title we also report the kendall-tau correlation coefficient for every device. As observed, our predictor achieves high kendall-τ correlation coefficient across all devices.

I.2 ADDITIONAL RESULTS ON NAS-BENCH-201

In Figure 19, we present the Pareto fronts obtained by our method in comparison to different baselines on the NAS-Bench-201 search space. In Figure 20, we present different additional metrics, such as GD and IGD (see Section D), to evaluate the quality of the Pareto fronts obtained on NAS-Bench-201. Figure 21 presents the Pareto front MODNAS yields when applying different latency constraints during the search phase. Figure 15a compares our method using the ReinMax gradient estimator to the GDAS estimator (Dong & Yang, 2019). As we can see, ReinMax obtains a qualitatively better hypervolume coverage compared to GDAS. Figure 16 presents the 3D Pareto front and hypervolume obtained by MODNAS compared to other baselines when optimizing for accuracy, latency and energy usage on NAS-Bench-201. Figure 24 presents the comparison of MODNAS with MGD to other gradient aggregation schemes, such as mean, sequential and MC sampling (see Section 4.1), across multiple hardware devices. Finally, in Figure 25 we present the robustness of MODNAS to the fraction of devices used for the predictor training and the search phase. In addition, in Figure 18 and Figure 17, we compare MODNAS against different MO baselines on the CIFAR-100 dataset on two different devices.

I.3 ADDITIONAL RESULTS ON HARDWARE-AWARE TRANSFORMERS (EN-DE)

We show the Pareto fronts of MODNAS compared to baselines for the Transformer space in Figure 26, as well as their comparison with respect to hypervolume for the SacreBLEU metric in Figure 28. These results demonstrate the superior performance of our method compared to the other baselines on this benchmark. All evaluations are done by inheriting the weights of a pretrained supernet.

I.4 ADDITIONAL RESULTS ON THE HW-GPT SPACE

In figure 29, we present the Pareto fronts on all the 8 GPU types for MODNAS and different baselines. The Pareto fronts are obtained using the perplexity and energy predictors trained on data collected in the HW-GPT-Bench (Sukthanker et al., 2024).

I.5 ADDITIONAL RESULTS ON MOBILENETV3

In Figure 30, we present the Pareto fronts of our method compared to different baselines for 12 different hardware devices on the MobileNetV3 space. We show as well the Pareto front of OFA+HELP (Lee et al., 2021b), ran with the original setting.

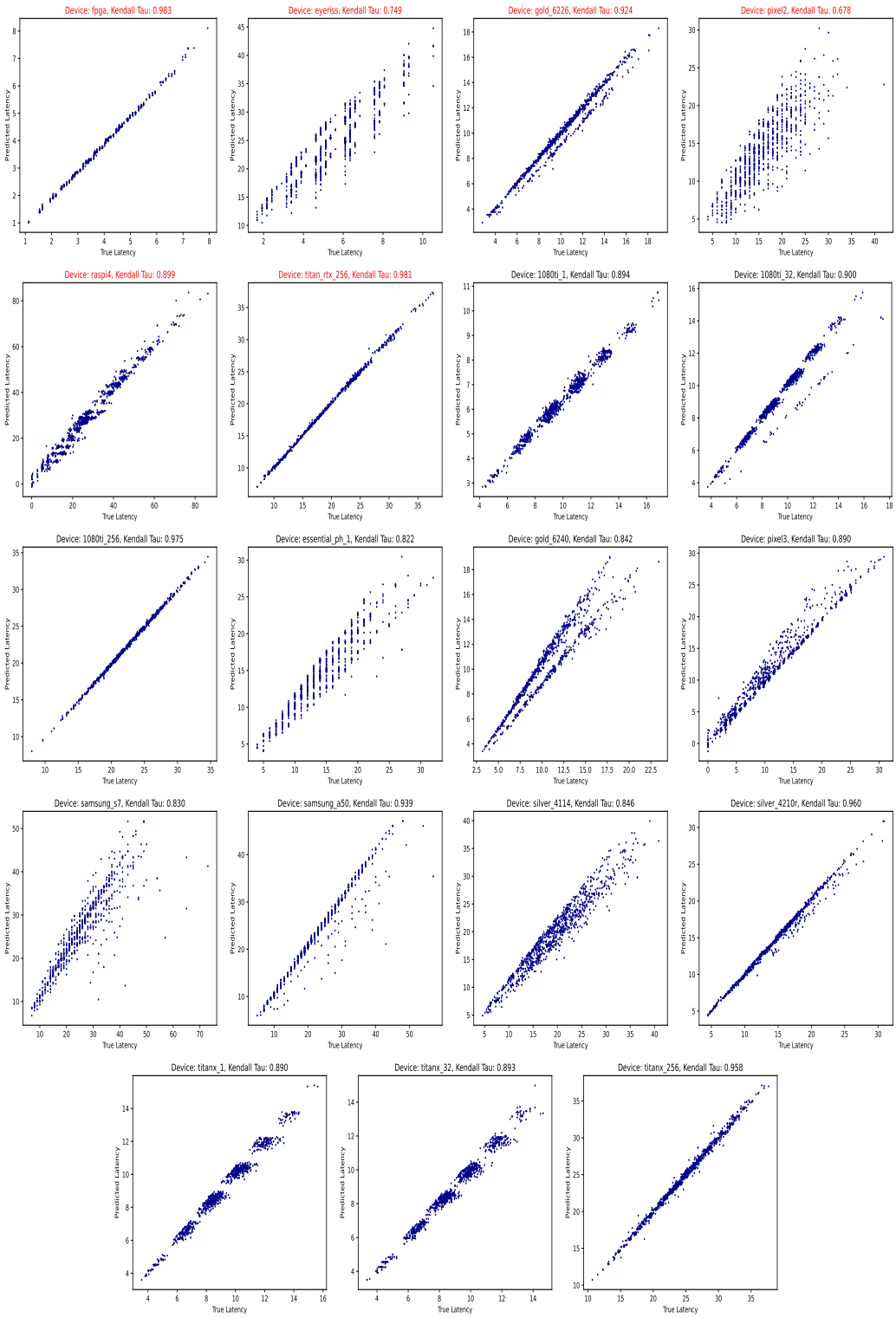
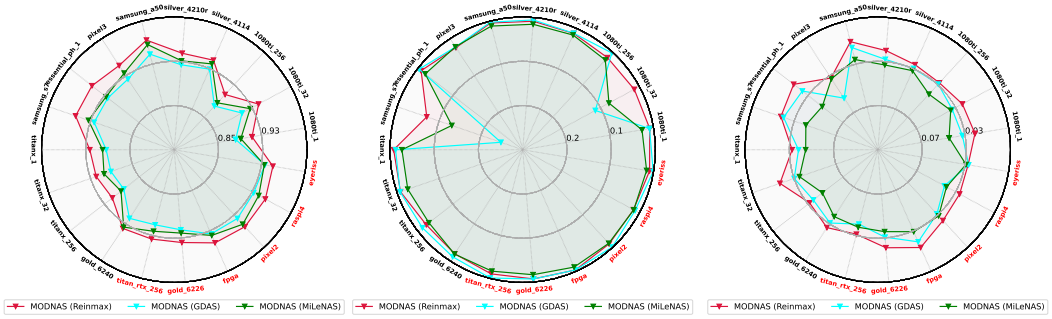


Figure 14: Scatter plots of predicted latencies from our pretrained `MetaPredictor` vs. ground-truth latencies (test devices in red).



(a) Hypervolume (b) GD+ (c) IGD+

Figure 15: Hypervolume, GD+ and IGD+ of MODNAS with Reinmax as gradient estimator in the `Architect` vs. the one from GDAS (Dong & Yang, 2019) and MiLeNAS (He et al., 2020) across 19 devices on NAS-Bench-201. Higher area in the radar indicates better performance for every metric. Test devices are colored in red around the radar plot.

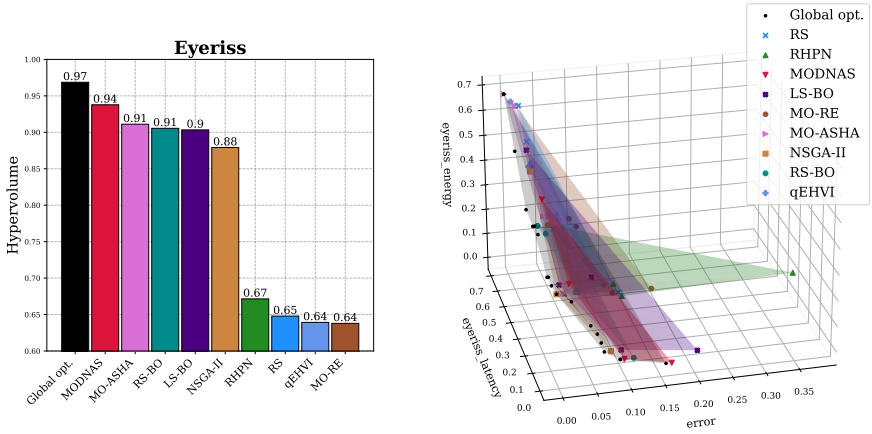


Figure 16: HV (*left*) and Pareto front (*right*) of MODNAS and baselines on Eyeriss with 3 normalized objectives: error, latency and energy usage. HV was computed using the $(1, 1, 1)$ reference point on the right 3D plot.

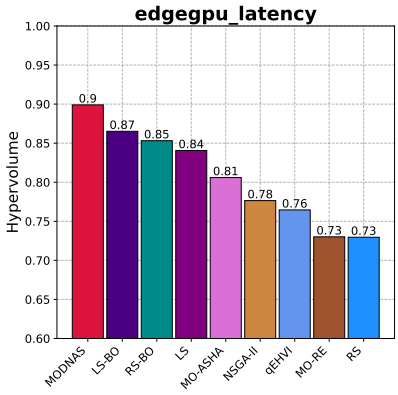


Figure 17: Hypervolume on CIFAR-100 and edgegpu device.

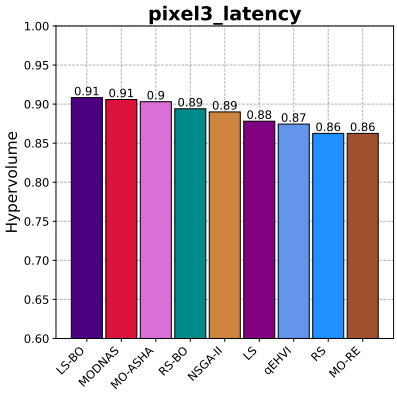


Figure 18: Hypervolume on CIFAR-100 and Pixel3 device.

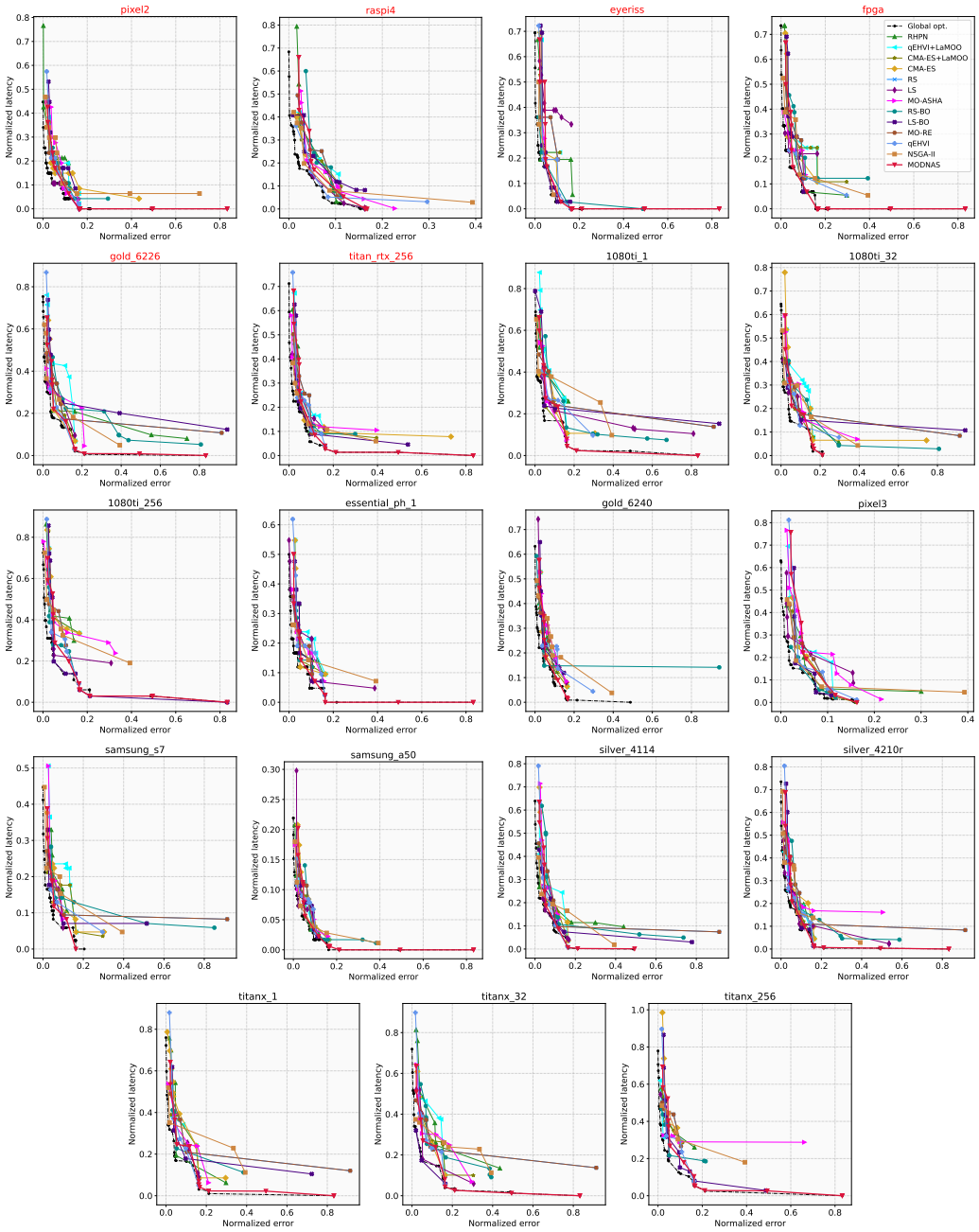
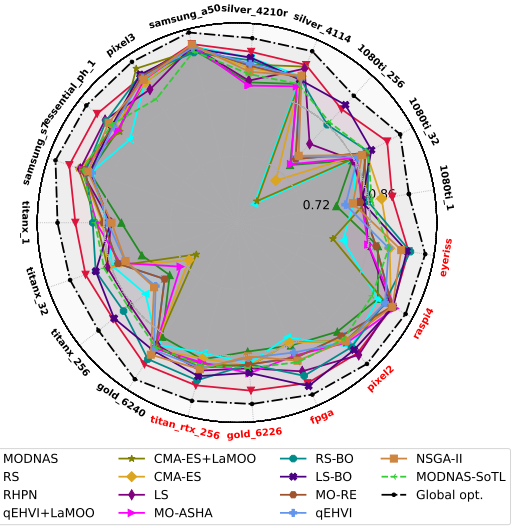
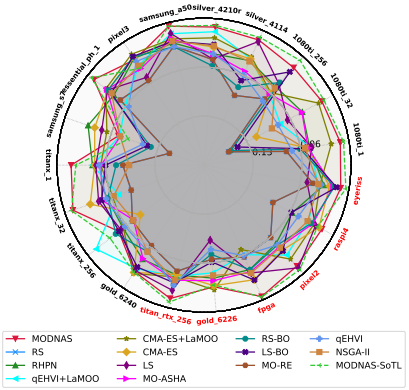


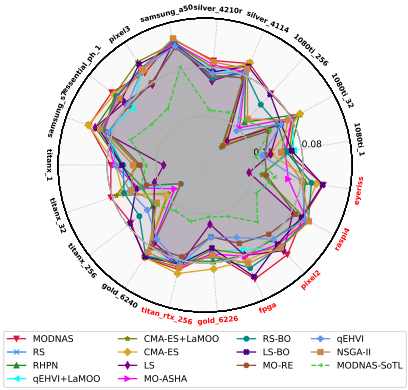
Figure 19: Pareto fronts of MODNAS and baselines on NAS-Bench-201. MODNAS-SoTL is not shown for better visibility.



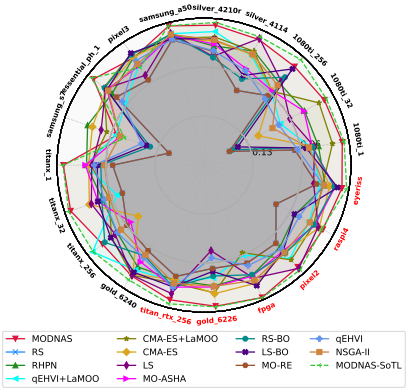
(a) HV



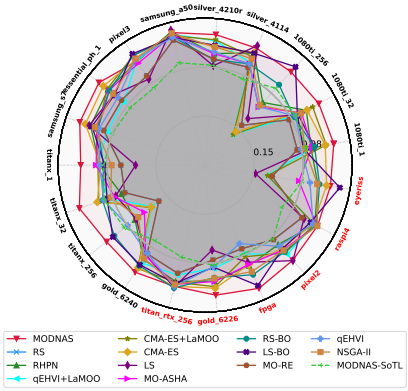
(b) GD



(c) IGD



(d) GD+



(e) IGD+

Figure 20: HV, GD, GD+, IGD and IGD+ of MODNAS and baselines across 19 devices on NAS-Bench-201. For every device we optimize for 2 objectives, namely *latency (ms)* and *test accuracy* on CIFAR-10. For method, metric and device we report the mean of 3 independent search runs. Higher area in the radar indicates better performance for every metric. Test devices are colored in red around the radar plot. Here we allocate double the budget to baselines, i.e. we run all baselines for 50 function evaluations.

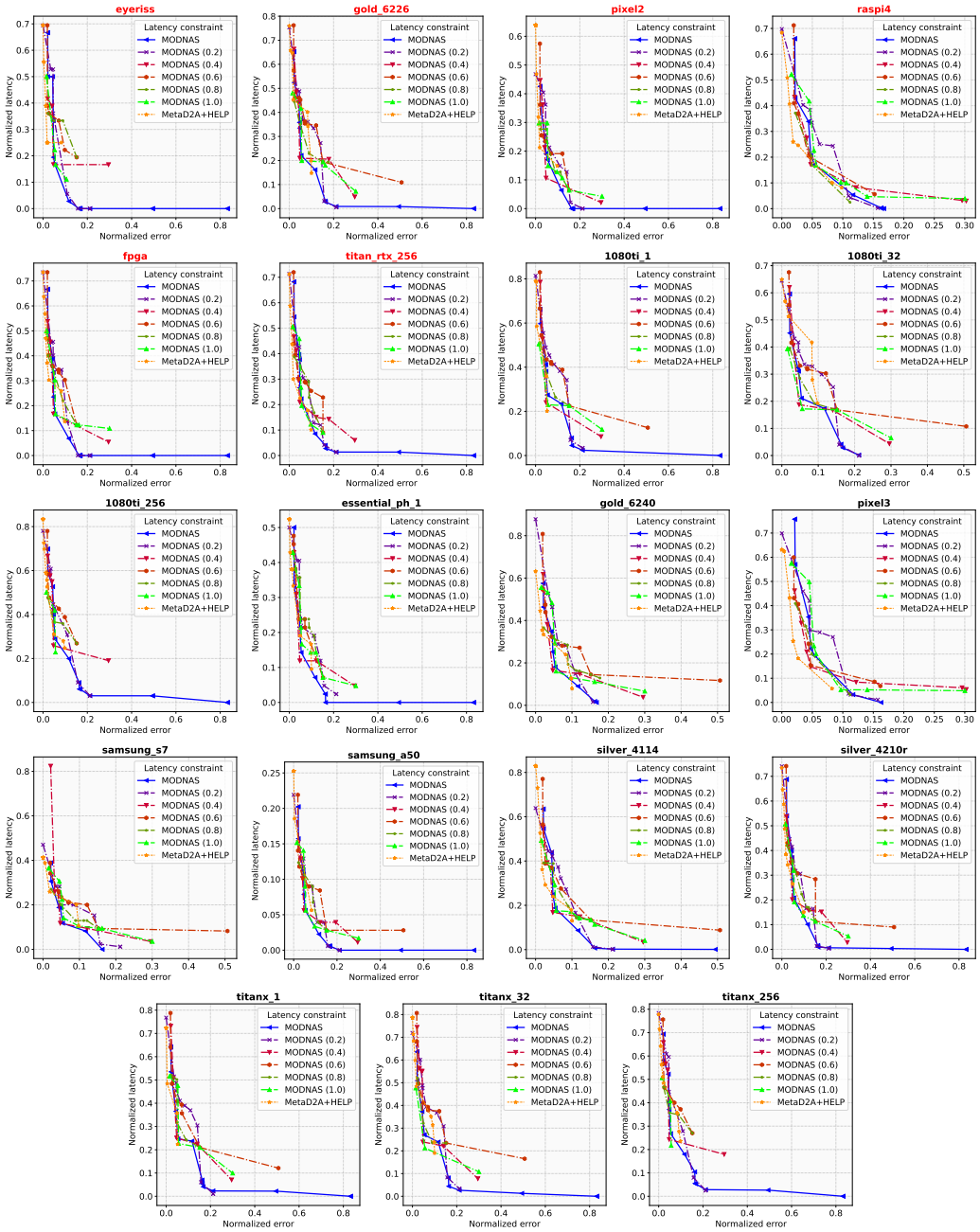
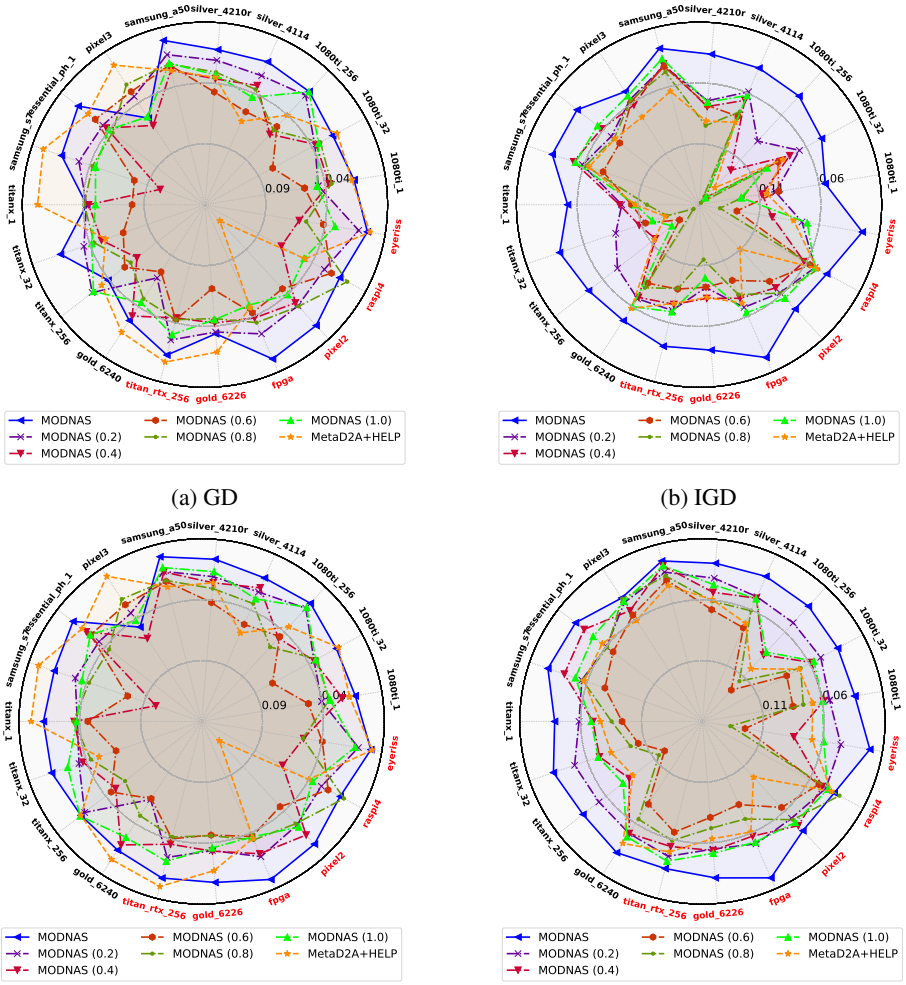


Figure 21: Pareto fronts of MODNAS ran with different latency constraints during search.



(a) GD (b) IGD

(c) GD+ (d) IGD+

Figure 22: GD, GD+, IGD and IGD+ of MODNAS with different latency constraints during search across 19 devices on NAS-Bench-201. Higher area in the radar indicates better performance for every metric. Test devices are colored in red around the radar plot.

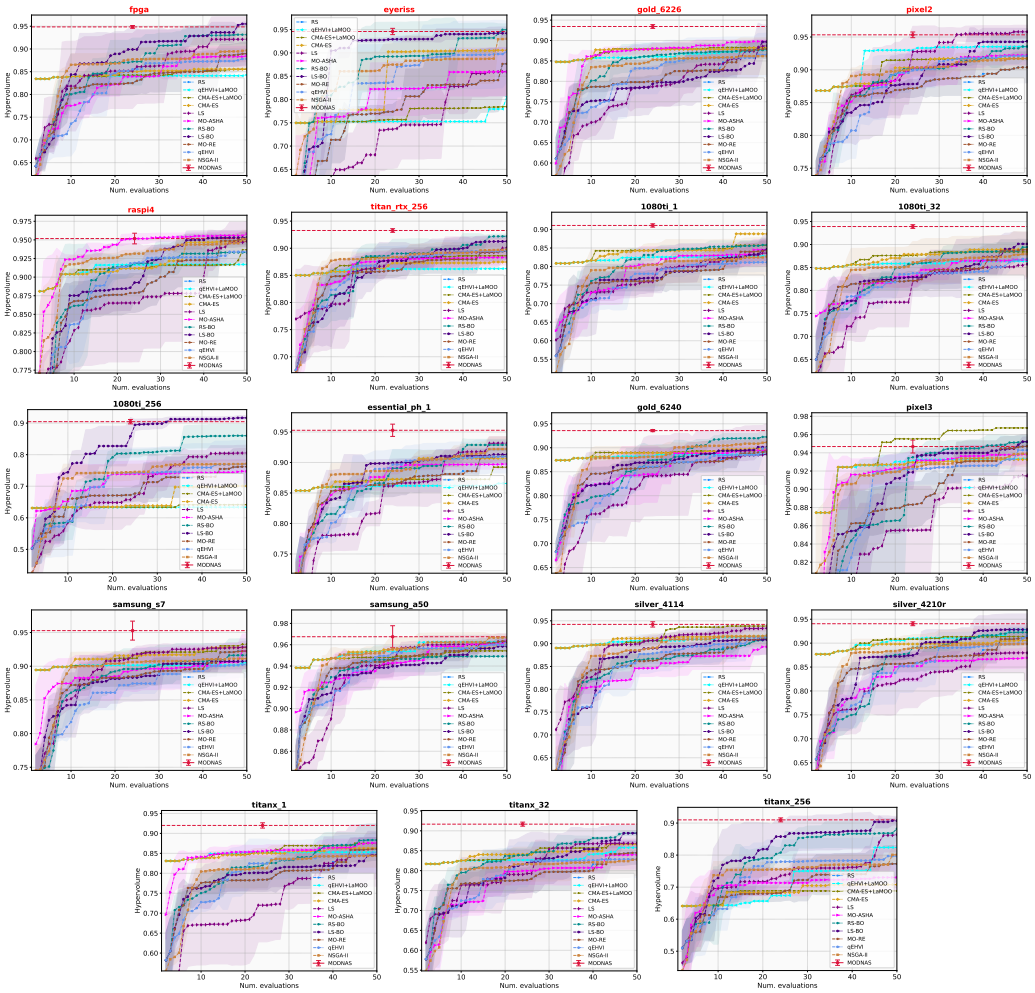


Figure 23: HV over number of evaluated architectures on NAS-Bench-201 of MODNAS and the blackbox MOO baselines. Note that for MODNAS we only have 24 evaluations in the end.

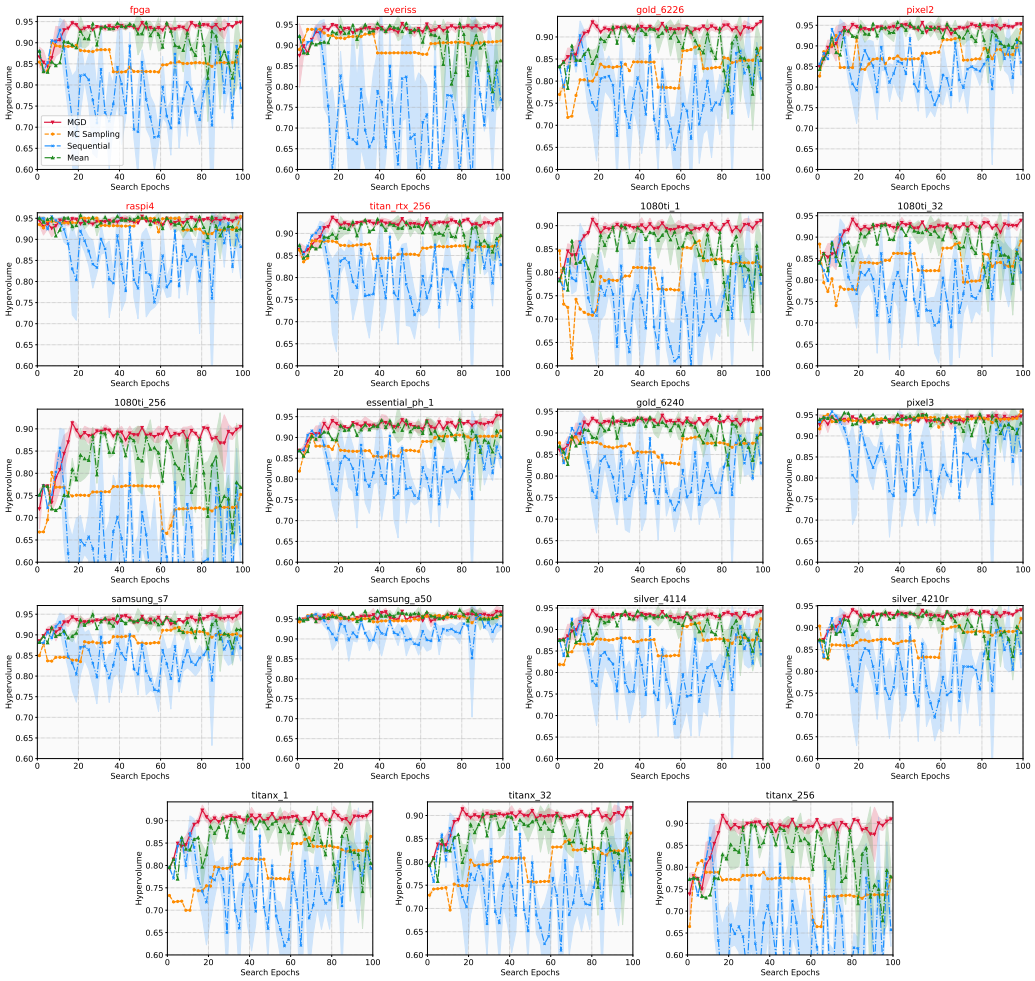


Figure 24: HV over time on NAS-Bench-201 of MODNAS with different gradient update schemes.

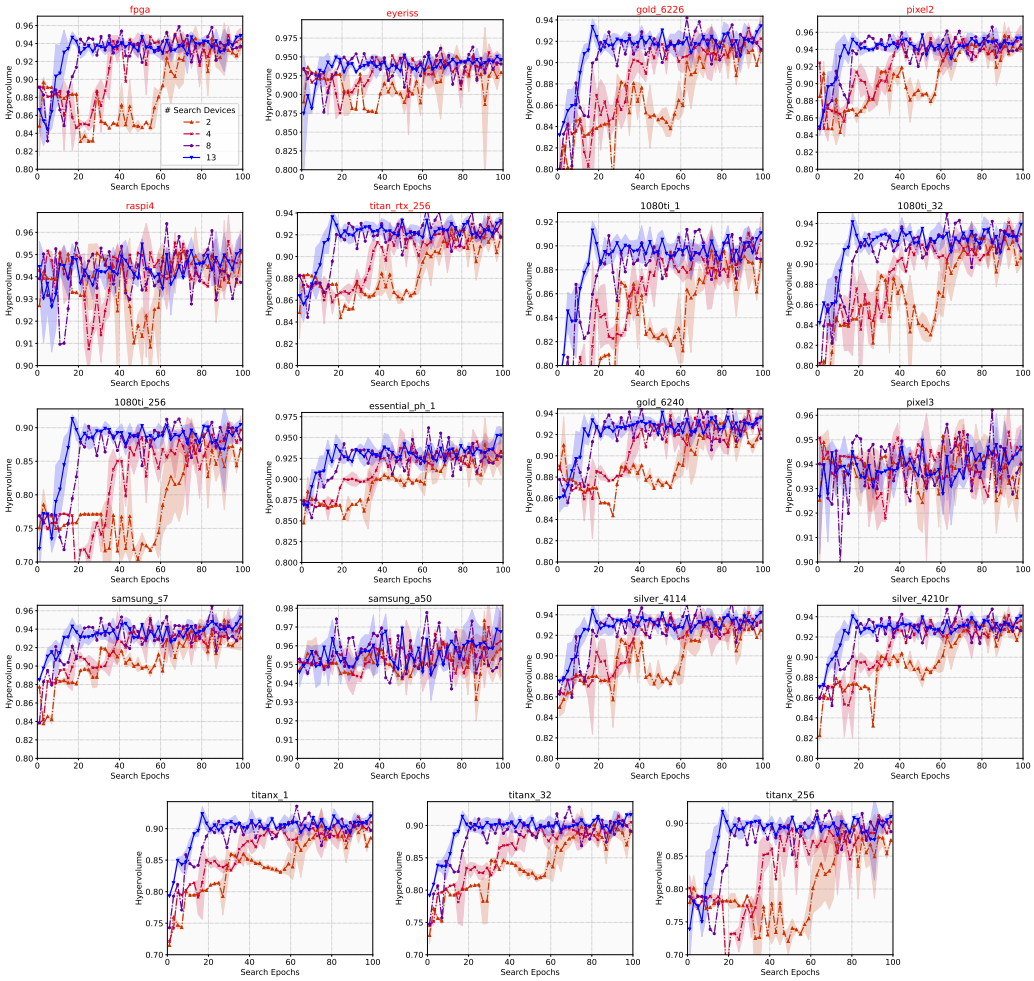


Figure 25: HV over time on NAS-Bench-201 of MODNAS with different number of devices during search. For number of devices less than 13 (default one) we randomly select a subset from these 13 devices.

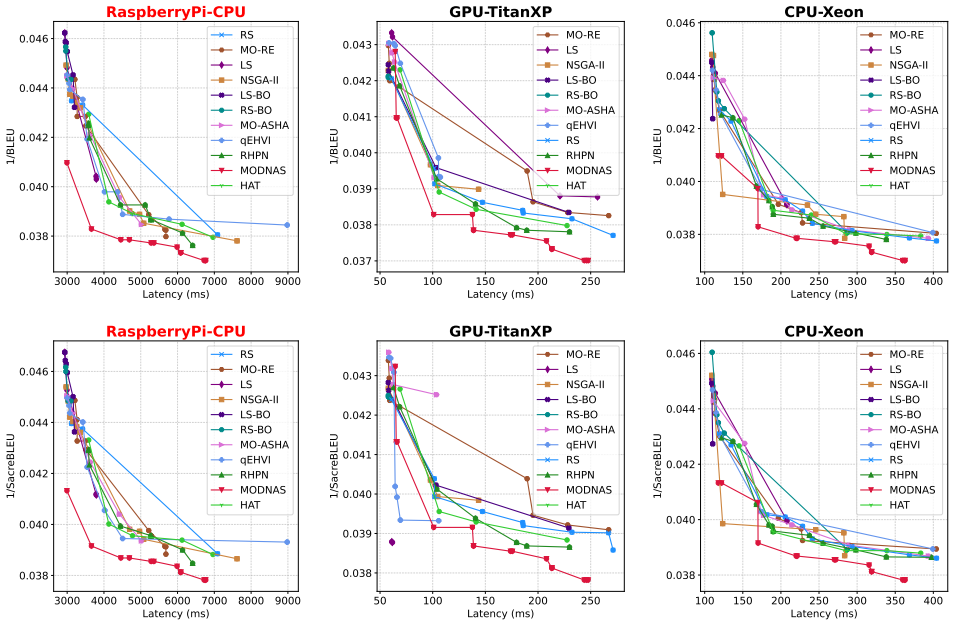


Figure 26: Pareto fronts of MODNAS and baselines on the HAT space for the WMT' En-De task. All performance metrics are obtained from the inherited supernet weights.

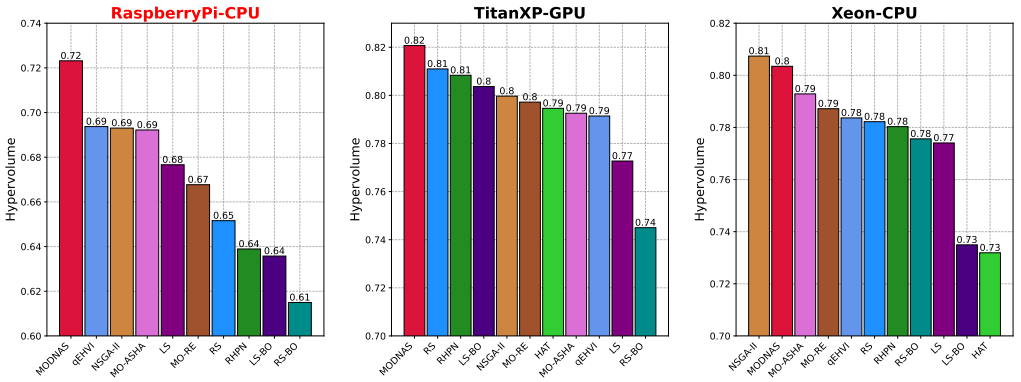


Figure 27: Hypervolume (HV) of MODNAS and baselines across devices on the HAT space. The objectives used to compute the HV are latency and BLEU score. Leftmost plot is for the test device.

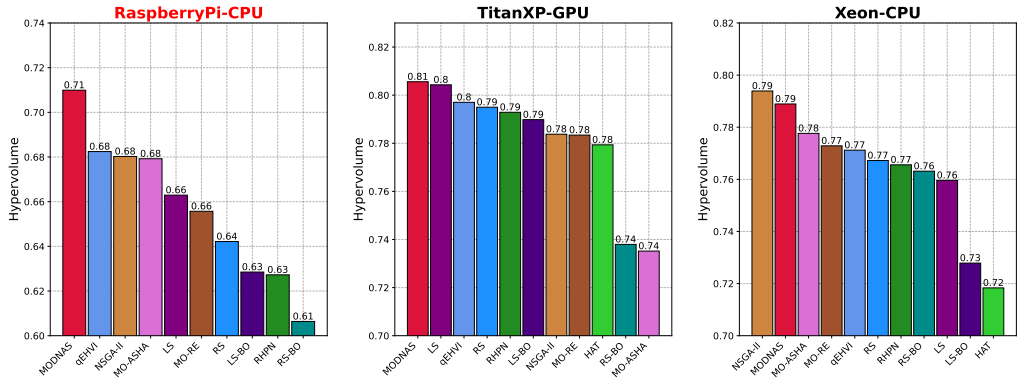


Figure 28: Hypervolume (HV) of MODNAS and baselines across devices on the HAT space. The objectives used to compute the HV are latency and SacreBLEU score. Leftmost plot is for the test device. MODNAS is the best or on par to the baselines across all three devices.

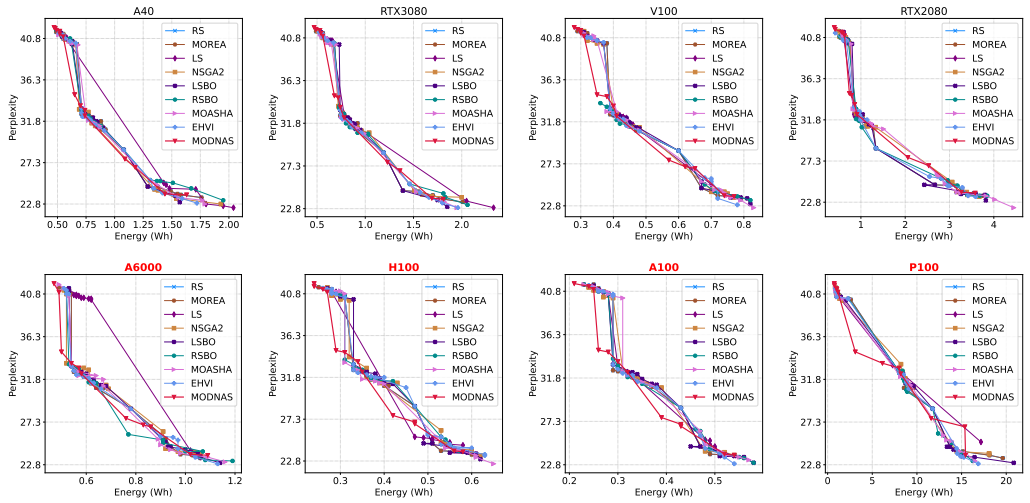


Figure 29: Pareto fronts of MODNAS and baselines optimizing for GPU energy consumption (Wh) and perplexity on the HW-GPT-Bench space.

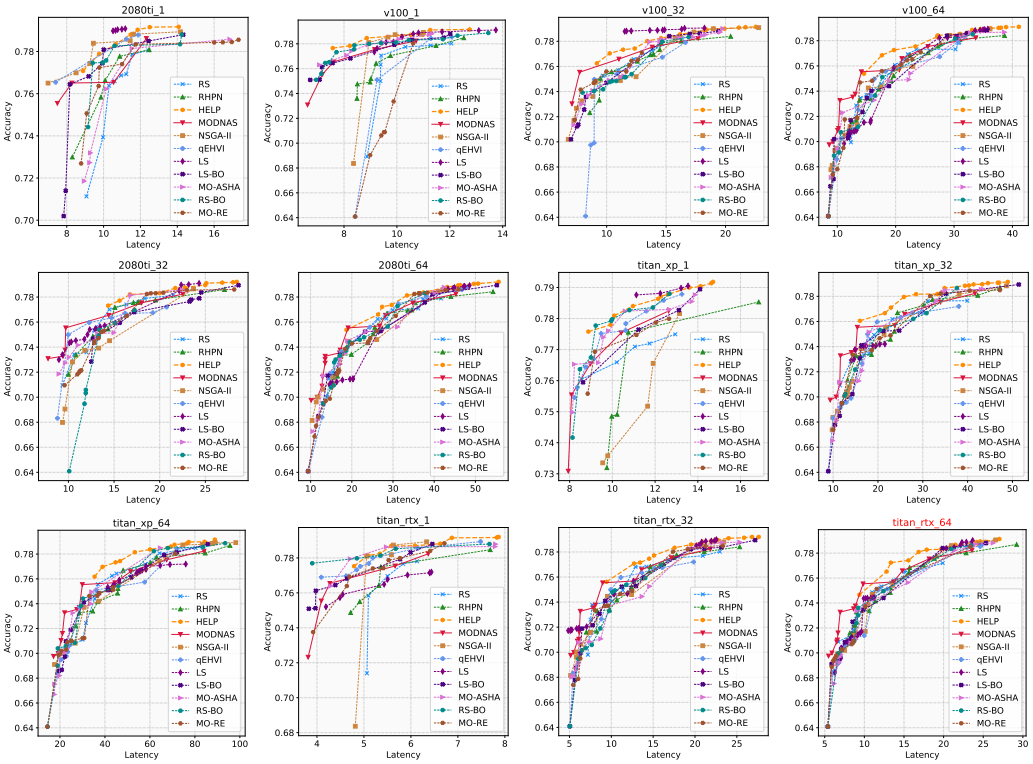


Figure 30: Pareto fronts of MODNAS and baselines on the MobileNetV3 space.

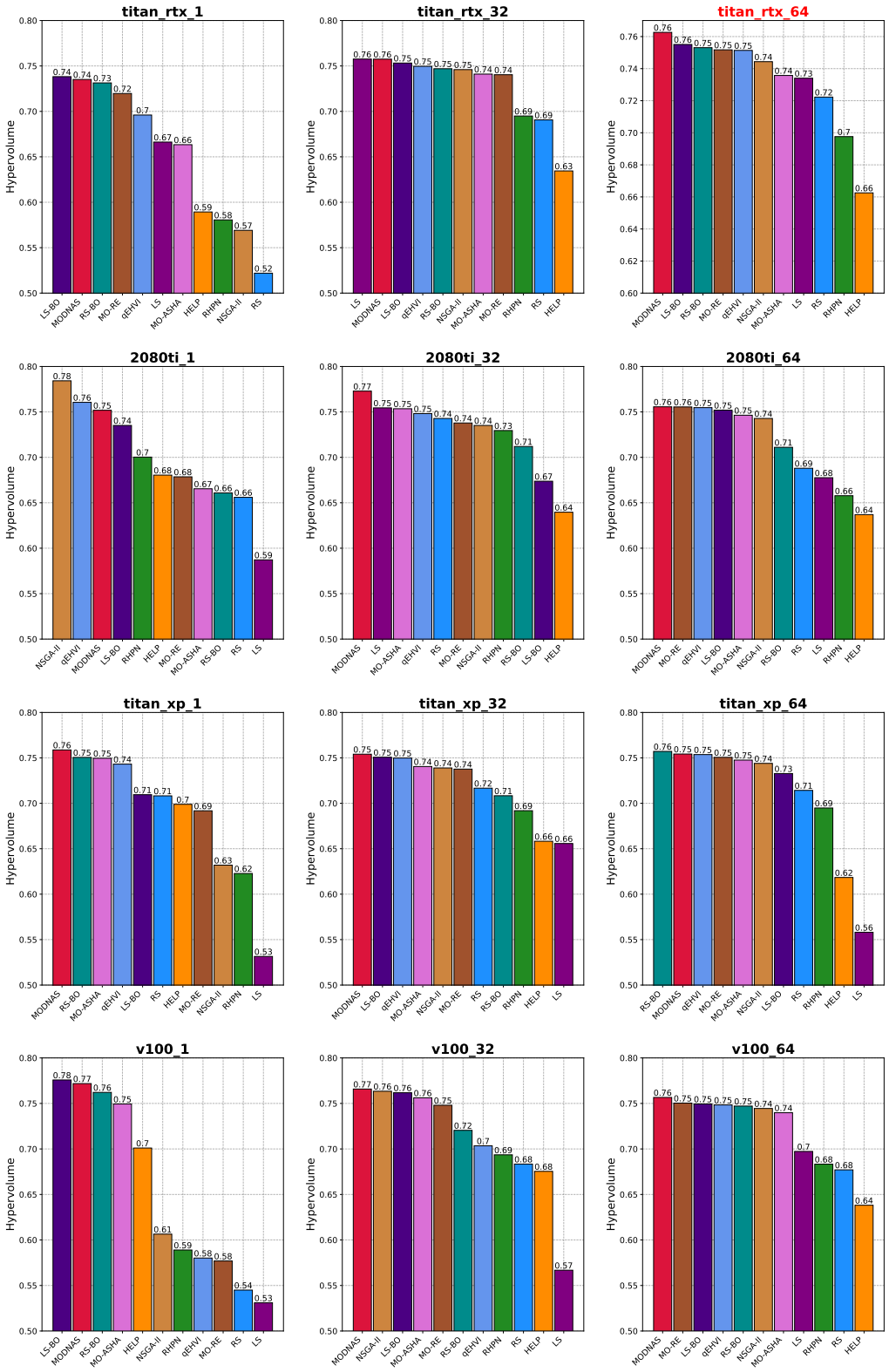


Figure 31: Hypervolume across devices on the MobileNetV3 search space of MODNAS and baselines. Here the Nvidia Titan RTX is the test device.

J ADDITIONAL DISCUSSION ON THE ROBUSTNESS OF MODNAS

Initially observed by (Zela et al., 2020), differentiable NAS methods can be very sensitive to their hyperparameter choices, especially the regularization ones responsible for the loss landscape in the upper level problem. In our experiments, there were three crucial components that made MODNAS robust and to work reliably across benchmarks:

1. **Choice of `MetaHypernetwork` update scheme:** this played a pivotal role in the performance of MODNAS. Although other gradient update strategies underperformed or started diverging (Figure 6), MGD converged relatively quickly to a hypervolume close to that of the global Pareto front. The convergence of MGD to a pareto stationary point is discussed in Désidéri (2012) and more recently in Zhang et al. (2024b). The convergence of MGD in bilevel optimization is an open research topic (see recent results from Ye et al. (2024) and Yang et al. (2024)). One potential scenario when MGD could fail is when the gradient directions of the objectives it is optimizing point in different opposing directions; however, this becomes practically unlikely, especially as the number of objectives grows (in our case we use it to find the common gradient across devices, which is for instance 13 on NAS-Bench-201).
2. **Choice of gradient estimation method in the `Architect`**: In Section 3, we discuss our choice for the method that enables gradient estimation through discrete variables (since architectures are discrete variables). We noticed that the ReinMax (Liu et al., 2023) estimator always outperformed previous estimators such as the one in GDAS (Dong & Yang, 2019) (Figure 15a), so we believe this choice is crucial.
3. **Weight entanglement vs. weight sharing in the `Supernetwork`**: In early experiments on NB201 we noticed that weight sharing in the `Supernetwork`, was not only more expensive, but much more unstable as well when compared to weight entanglement (Cai et al., 2020; Sukthanker et al., 2023), even yielding diverging solutions quite often (common pattern seen in differentiable NAS with shared weights as you mention; see Zela et al. (2020) for instance).

We hypothesize that all design choices mentioned above play an implicit regularization effect on the upper level optimization in the bi-level problem, leading to a faster convergence and robustness (Chen & Hsieh, 2020; Smith et al., 2021; Zela et al., 2020).

K ALIGNMENT OF PREFERENCE VECTORS WITH PARETO FRONT

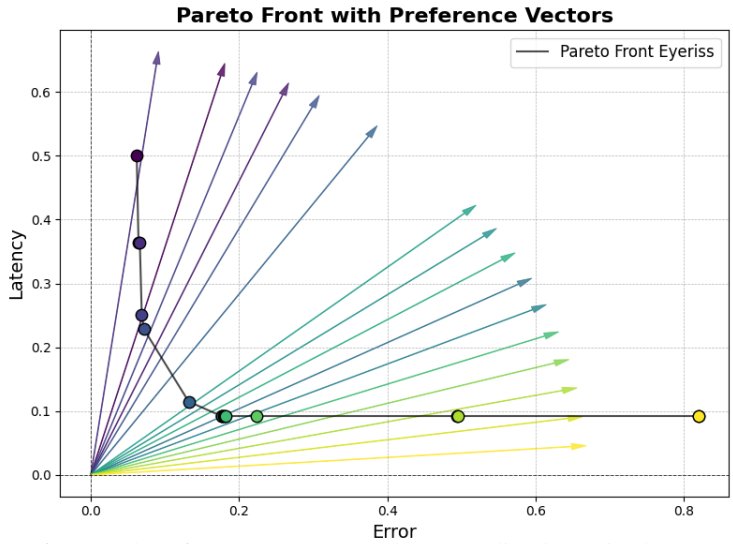


Figure 32: Pareto front and preference vectors on the normalized Eyeriss latency and test error of NAS-Bench-201.

In this section, we provide empirical evidence that the solutions generated using the `MetaHypernetwork` align well with the preference vectors. To this end, we utilize one of our

runs on the NAS-Bench-201 test devices, namely Eyeriss. In Figure 32, we show the Pareto front of the normalized test error and latency on Eyeriss. Note that of the 24 sampled preference vectors, 17 generate solutions that are in the Pareto set. Each point in the Pareto front with a certain color corresponds to the preference vector with the same color. In the figure, there are actually 17 points in the Pareto front; however, some of them are really close to each other or are the same, since the function mapping preference vectors to architectures is a many-to-one function. Nevertheless, we can visually notice that the preference vectors starting from the origin align very well with the generated solutions. The missing vectors are mainly in the center, where there are not many solutions available for this particular device.

L TRAINING AND VALIDATION LOSS CURVES

In addition to the hypervolume indicator, in this section, we provide the training and validation loss curves in Figure 33. At each mini-batch iteration we plot the average cross entropy loss across all devices. As expected both training and validation cross-entropy go down and we do not notice any overfitting. The high noise is common for sample-based NAS optimizers, since a sampled different architecture is activate at each mini-batch iteration. In the plot, for visualization purposes, we have used a running average with a window size of 100 to smooth out the noise.

M MULTI-OBJECTIVE OPTIMIZATION BASELINES WITH MORE BUDGET

Black-box multi-objective optimizers can potentially reach the global Pareto front if the compute resources are not a concern and given enough time. However, it is not practical to train or even evaluate these architectures, especially for larger model sizes (e.g. Transformer spaces from HW-GPT-Bench). Sometimes in practice, the user wants to get a quick estimation of the Pareto front instead the global optimum, and this is the use-case where MODNAS shines. Given enough budget, even a random search (RS) will find a near-optimal solution. For example, in NAS-Bench-201, the size of the search space is $K = 15625$ architectures. The optimal theoretical number of RS steps n to achieve a success probability α is approximately: $n \geq Kln(1/1-\alpha)$, therefore, for random search to have a success probability higher than 0.5 it requires $n \geq 10781$ iterations in theory. For the other guided search methods, this number is even smaller, though similar to MODNAS, they have the same limitation that they can converge to a local minimum. We conducted the same experiment as the one in Figure 3, but this time with baselines given 4 times more budget than MODNAS. We show the result in Figure 34. As we can see, some of the methods such as LS-BO can reach results closer to the global Pareto front compared to MODNAS.

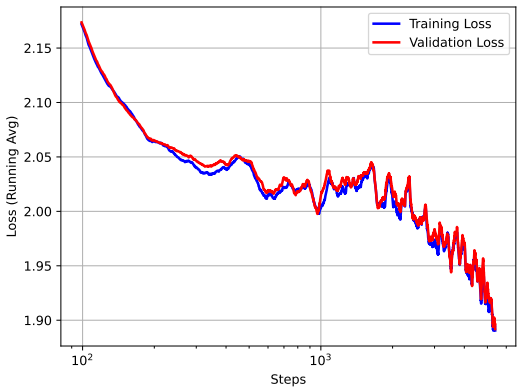


Figure 33: Average training and validation cross-entropy loss across devices during the MODNAS search on NAS-Bench-201.

N ADDITIONAL DETAILS ON THE `Architect`

In this section, we provide additional details on how the `Architect` utilizes the Straight-Through Estimator (STE) to backpropagate through the sampling of discrete architectural parameters.

Forward pass:

1. The `MetaHypernetwork` parameterizes the unnormalized architectural distribution: $\tilde{\alpha} = H_\Phi$, where Φ are the `MetaHypernetwork` parameters.
2. $\tilde{\alpha}$ is passed to `Architect` and it does the following steps:
   (a) Normalizes $\tilde{\alpha}$ and samples a one-hot (discrete) α: $\alpha \sim Categorical(Softmax(\tilde{\alpha}))$.

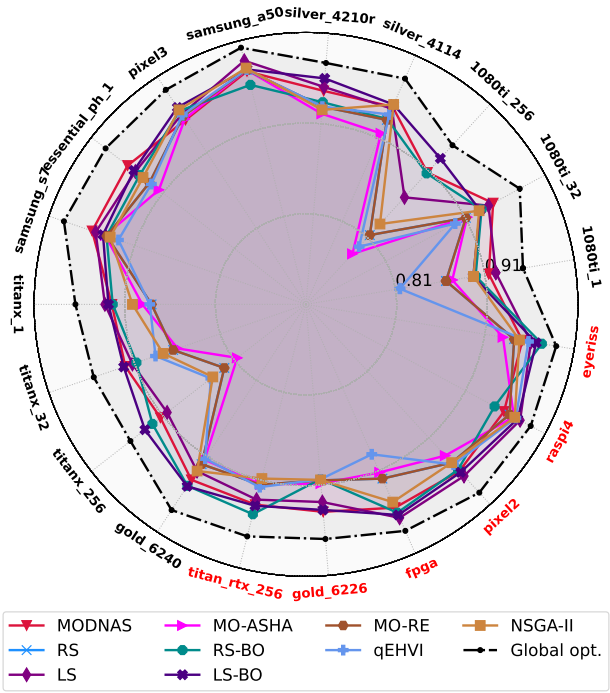


Figure 34: HV of MODNAS and baselines across 19 devices on NAS-Bench-201. For every device we optimize for 2 objectives, namely *latency (ms)* and *test accuracy* on CIFAR-10. For method, metric and device we report the mean of 3 independent search runs. Higher area in the radar indicates better performance for every metric. Test devices are colored in red around the radar plot. Here we allocate 4 times the budget to baselines, i.e. we run all baselines for 100 function evaluations.

   (b) Sets the `Supernetwork` architectural parameters to the one-hot α, i.e. resulting in a single subnetwork by masking the `Supernetwork`.
   (c) Passes α as input to `MetaPredictor`.
3. The `Supernetwork` and `MetaPredictor` do a forward pass using the training data (e.g., images) and hardware embedding, respectively.
4. Compute the scalarized loss function.

The main problem now is that we cannot directly backpropagate the gradient computation through the `Architect` to update the `MetaHypernetwork` parameters Φ. This is due to the sampling from the Categorical distribution in step 2/(a) above being non-differentiable. The STE approximates the gradient for the discrete architectural parameters by ignoring this actual non-differentiable sampling operation.

Backward pass:

1. Calculate the gradient of the scalarized loss with respect to the discrete architectural parameters α: $\partial\mathcal{L}/\partial\alpha$.
2. Propagate this gradient back to Φ (`MetaHypernetwork` parameters) via the probability distribution:
$$\nabla_\Phi \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \alpha} \frac{\partial \alpha}{\partial Softmax} \nabla_\Phi Softmax(H_\Phi).$$
STE backpropagates "through" a proxy that treats the non-differentiable function (sampling of α) as an identity function (as a result $\frac{\partial \alpha}{\partial Softmax} = 1$) and computes the gradient w.r.t. to the `MetaHypernetwork` parameters:
$$\nabla_\Phi \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \alpha} \nabla_\Phi Softmax(H_\Phi)$$
.

To recap, during the forward pass the `Architect` samples a discrete architecture from an architecture distribution parameterized by the `MetaHypernetwork`, and during backpropagation the STE is utilized to propagate back through the sampling operation and update the `MetaHypernetwork` parameters, hence the distribution from which the discrete architectures in the next iteration will be sampled.