Multi-objective Differentiable Neural Architecture Search

Rhea S. Sukthanker[1*] Arber Zela[1*] Benedikt Staffler[2]
Samuel Dooley[3] Josif Grabocka[4] Frank Hutter[1]

[1]University of Freiburg [2]Bosch Center for Artificial Intelligence [3]Abacus AI [4]University of Nürnberg

February 24, 2025

GEFÖRDERT VOM

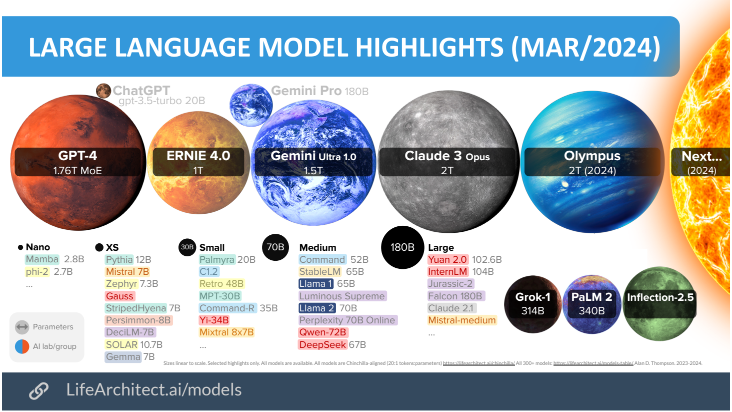
Bundesministerium für Bildung und Forschung

NAS in a world of ever growing models

- In an age of **large models**, finding architectures which are *performant*, *efficient* and with *fast* inference times is pivotal.

NAS in a world of ever growing models

- In an age of **large models**, finding architectures which are *performant*, *efficient* and with *fast* inference times is pivotal.



NAS in a world of ever growing models

- In an age of **large models**, finding architectures which are *performant*, *efficient* and with *fast* inference times is pivotal.
- Multi-objective problem with (potentially) **conflicting objectives**.
  - Optimizing all objectives simultaneously is infeasible.
  - Finding the **right trade-off** remains challenging.

NAS in a world of ever growing models

- In an age of **large models**, finding architectures which are *performant*, *efficient* and with *fast* inference times is pivotal.
- Multi-objective problem with (potentially) **conflicting objectives**.
  - Optimizing all objectives simultaneously is infeasible.
  - Finding the **right trade-off** remains challenging.
- We also need efficient search methods for these kind of spaces.
  - Conventional blackbox methods, such as ES or BO, require multiple expensive evaluations.

NAS in a world of ever growing models

- In an age of **large models**, finding architectures which are *performant*, *efficient* and with *fast* inference times is pivotal.
- Multi-objective problem with (potentially) **conflicting objectives**.
  - Optimizing all objectives simultaneously is infeasible.
  - Finding the **right trade-off** remains challenging.
- We also need efficient search methods for these kind of spaces.
  - Conventional blackbox methods, such as ES or BO, require multiple expensive evaluations.
- Multi-objective Differentiable NAS (**MODNAS**)
  - Leverages **hypernetworks** and **multiple gradient descent (MGD)** to profile the whole pareto front.
  - Scales across *multiple devices and objectives* with a **single search run**.

Optimality in Multi-objective optimization

Pareto optimality and Pareto front

MOO then seeks to find a set of Pareto-optimal solutions α^* that jointly minimize $\mathbf{L}(\alpha) \triangleq (\mathcal{L}^1(\alpha), \ldots, \mathcal{L}^M(\alpha))$:

$$\alpha^* \in \arg\min_{\alpha} \mathbf{L}(\alpha)$$

MOO then seeks to find a set of Pareto-optimal solutions α^* that jointly minimize $\mathbf{L}(\alpha) \triangleq (\mathcal{L}^1(\alpha), \ldots, \mathcal{L}^M(\alpha))$:

$$\alpha^* \in \arg\min_{\alpha} \mathbf{L}(\alpha)$$

Definition

(**Pareto Optimality**): A solution α_2 dominates α_1 iff $\mathcal{L}^m(\alpha_2) \leq \mathcal{L}^m(\alpha_1)$, $\forall m \in \{1, \ldots, M\}$, and $\mathbf{L}(\alpha_1) \neq \mathbf{L}(\alpha_2)$. 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 α^* is called *Pareto optimal* iff there exists no other solution dominating α^*.

MOO then seeks to find a set of Pareto-optimal solutions α^* that jointly minimize $\mathbf{L}(\alpha) \triangleq (\mathcal{L}^1(\alpha), \ldots, \mathcal{L}^M(\alpha))$:

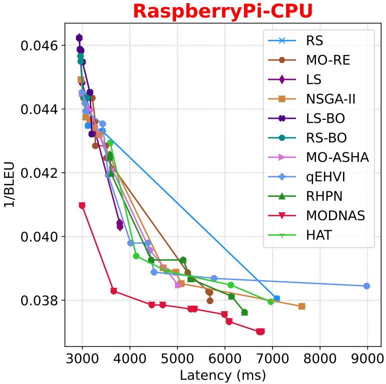
$$\alpha^* \in \arg\min_{\alpha} \mathbf{L}(\alpha)$$

Definition

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

MOO then seeks to find a set of Pareto-optimal solutions α^* that jointly minimize $\mathbf{L}(\alpha) \triangleq (\mathcal{L}^1(\alpha), \ldots, \mathcal{L}^M(\alpha))$:

$$\alpha^* \in \arg\min_{\alpha} \mathbf{L}(\alpha)$$



Problem Formulation

Multi-objective NAS as bi-level optimization

Assuming we have **T hardware devices** (target functions) and **M objectives** (e.g. accuracy, latency, energy usage, etc.), the Pareto set $\mathcal{P}_{\alpha_t}$ of the multi-objective NAS problem is obtained by solving the following bi-level optimization problem:

$$\begin{aligned} &\arg\min_{\alpha} \mathbf{L}_t^{valid}(\boldsymbol{w}^*(\alpha), \alpha) \\ s.t. \quad &\boldsymbol{w}^*(\alpha) = \arg\min_{\boldsymbol{w}} \mathbf{L}_t^{train}(\boldsymbol{w}, \alpha), \end{aligned}$$

where the M-dimensional loss vector
$\mathbf{L}_t(\boldsymbol{w}^*, \alpha) \triangleq \left(\mathcal{L}_t^1(\boldsymbol{w}^*, \alpha), \ldots, \mathcal{L}_t^M(\boldsymbol{w}^*, \alpha)\right)$ is evaluated $\forall t \in \{1, \ldots, T\}$.

Problem Formulation

Multi-objective NAS as bi-level optimization

Assuming we have **T hardware devices** (target functions) and **M objectives** (e.g. accuracy, latency, energy usage, etc.), the Pareto set $\mathcal{P}_{\alpha_t}$ of the multi-objective NAS problem is obtained by solving the following bi-level optimization problem:

$$\begin{aligned} &\arg\min_{\alpha} \mathbf{L}_t^{valid}(\boldsymbol{w}^*(\alpha), \alpha) \\ s.t. \quad &\boldsymbol{w}^*(\alpha) = \arg\min_{\boldsymbol{w}} \mathbf{L}_t^{train}(\boldsymbol{w}, \alpha), \end{aligned}$$

where the M-dimensional loss vector $\mathbf{L}_t(\boldsymbol{w}^*, \alpha) \triangleq \left(\mathcal{L}_t^1(\boldsymbol{w}^*, \alpha), \dots, \mathcal{L}_t^M(\boldsymbol{w}^*, \alpha)\right)$ is evaluated $\forall t \in \{1, \dots, T\}$.

- Still expensive...
  - Need to run the search T times.
  - Cannot be solved exactly due to the expensive lower problem.

MODNAS

Algorithmic components

1. `MetaPredictor`: regression model to predict cheap-to-evaluate hardware objectives (e.g. latency, energy usage, etc.)
2. `Supernetwork`: proxy to approximate the lower level best response function $\boldsymbol{w}^*(\alpha)$
3. `MetaHypernetwork`: hypernetwork to generate unnormalized architectural distribution conditioned on preference vectors and hardware device type
4. `Architect`: samples from the architectural distribution discrete architectures

- For the cheap-to-evaluate hardware objectives, such as latency, energy consumption.

[1] HELP: Hardware-adaptive Efficient Latency Prediction for NAS via Meta-Learning, Lee et al. NeurIPS 2021

- For the cheap-to-evaluate hardware objectives, such as latency, energy consumption.
- A parametric regression model (e.g. MLP) $p_{\theta}^{m}(\alpha, d_t^m) : \mathcal{A} \times \mathcal{H} \rightarrow \mathbb{R}$.

[1] HELP: Hardware-adaptive Efficient Latency Prediction for NAS via Meta-Learning, Lee et al. NeurIPS 2021

MODNAS

- For the cheap-to-evaluate hardware objectives, such as latency, energy consumption.
- A parametric regression model (e.g. MLP) $p_\theta^m(\alpha, d_t^m) : \mathcal{A} \times \mathcal{H} \to \mathbb{R}$.
- We use the same predictors as in [1] and optimize the MSE loss:

$$\min_\theta \mathbb{E}_{\alpha \sim \mathcal{A}, t \sim [T]} \left(y_t^m - p_\theta^m(\alpha, d_t^m)\right)^2$$

[1] HELP: Hardware-adaptive Efficient Latency Prediction for NAS via Meta-Learning, Lee et al. NeurIPS 2021

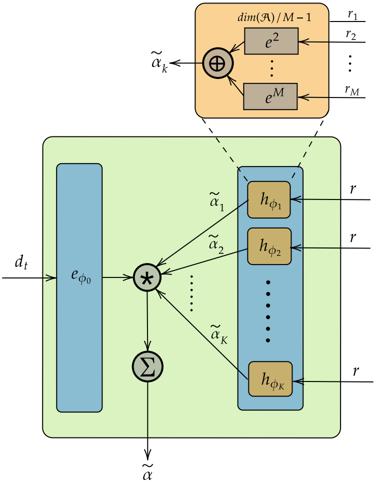
- For the cheap-to-evaluate hardware objectives, such as latency, energy consumption.
- A parametric regression model (e.g. MLP) $p_\theta^m(\alpha, d_t^m) : \mathcal{A} \times \mathcal{H} \to \mathbb{R}$.
- We use the same predictors as in [1] and optimize the MSE loss:

$$\min_\theta \mathbb{E}_{\alpha \sim \mathcal{A}, t \sim [T]} \big(y_t^m - p_\theta^m(\alpha, d_t^m)\big)^2$$

- Use $\mathcal{L}_t^m(\cdot, \alpha_\Phi) = p_\theta^m(\alpha_\Phi, d_t^m)$ as the objective function.
  - During the search we freeze and do not update further the `MetaPredictor` parameters θ.

[1] HELP: Hardware-adaptive Efficient Latency Prediction for NAS via Meta-Learning, Lee et al. NeurIPS 2021

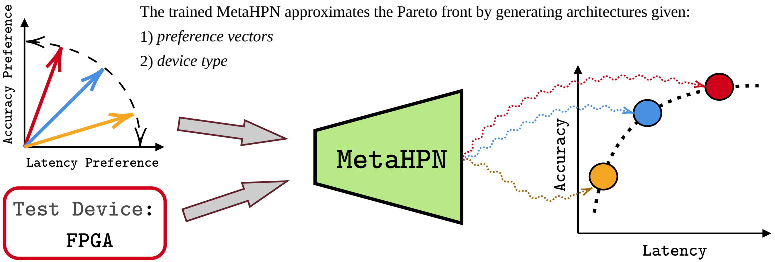
- We use a hypernetwork $H_\Phi(\boldsymbol{r}, d_t)$ that takes as input a **device embedding** d_t and a **preference vector** $\boldsymbol{r} \in \mathbb{R}^M$ to yield an architecture distribution $\tilde{\alpha}$.
  - Just a forward pass to generate an architecture.
  - Scalable across different hardware devices.



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 reduces to:

$$\begin{aligned} &\arg\min_{\Phi} \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\left[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{valid}(\boldsymbol{w}^*(\alpha_\Phi), \alpha_\Phi)\right] \\ s.t. \quad &\boldsymbol{w}^*(\alpha_\Phi) = \arg\min_{\boldsymbol{w}} \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\left[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{train}(\boldsymbol{w}, \alpha_\Phi)\right], \end{aligned}$$

where $\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t(\cdot, \alpha_\Phi) = \sum_{m=1}^{M} r_m \mathcal{L}_t^m(\cdot, \alpha_\Phi)$ is the scalarized loss for device t.



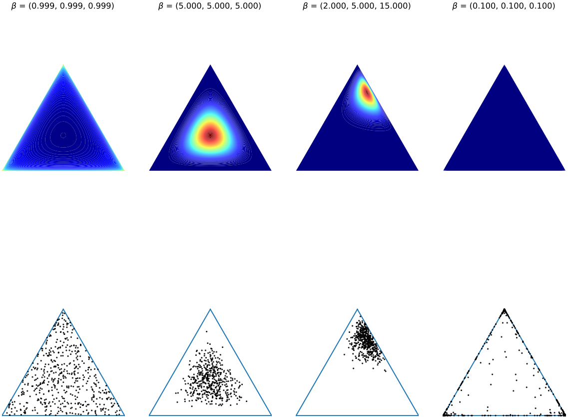
Using the **preference vector** $\boldsymbol{r}$ to create a linear scalarization of $\mathbf{L}_t$ and the `MetaHypernetwork` to model the architectural distribution across T devices, the bi-level problem reduces to:

$$\arg\min_{\Phi} \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\left[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{valid}(\boldsymbol{w}^*(\alpha_\Phi), \alpha_\Phi)\right]$$
$$s.t. \quad \boldsymbol{w}^*(\alpha_\Phi) = \arg\min_{\boldsymbol{w}} \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\left[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{train}(\boldsymbol{w}, \alpha_\Phi)\right],$$

where $\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t(\cdot, \alpha_\Phi) = \sum_{m=1}^{M} r_m \mathcal{L}_t^m(\cdot, \alpha_\Phi)$ 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.

We sample the preference vector $\boldsymbol{r}$ from a Dirichlet distribution with concentration parameters $\beta_1, \ldots, \beta_M = 1$.



MODNAS

Optimizing the `MetaHypernetwork` with MGD

- Multiple Gradient Descent (MGD) seeks to simultaneously optimize the `MetaHypernetwork` parameters (shared across all devices) $\Phi \leftarrow \Phi - \xi g_{\Phi}^{*}$, where: $g_{\Phi}^{*} = \sum_{t=1}^{T} \gamma_t^{*} g_{\Phi}^{t}$

- Multiple Gradient Descent (MGD) seeks to simultaneously optimize the `MetaHypernetwork` parameters (shared across all devices) $\Phi \leftarrow \Phi - \xi g_{\Phi}^{*}$, where: $g_{\Phi}^{*} = \sum_{t=1}^{T} \gamma_t^{*} g_{\Phi}^{t}$
- What are the optimal γ_t^{*}?

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

Optimizing the MetaHypernetwork with MGD

- Multiple Gradient Descent (MGD) seeks to simultaneously optimize the `MetaHypernetwork` parameters (shared across all devices) $\Phi \leftarrow \Phi - \xi g_{\Phi}^*$, where: $g_{\Phi}^* = \sum_{t=1}^{T} \gamma_t^* g_{\Phi}^t$
- What are the optimal γ_t^*?

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

- $T = 2$:

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

- $T > 2$:

  Frank-Wolfe solver [Jaggi, 2013]

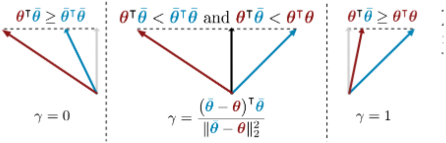


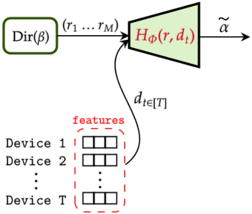
Figure 1: Visualisation of the min-norm point in the convex hull of two points ($\min_{\gamma\in[0,1]} \|\gamma\boldsymbol{\theta} + (1-\gamma)\bar{\boldsymbol{\theta}}\|_2^2$). As the geometry suggests, the solution is either an edge case or a perpendicular vector.

Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization, Jaggi, ICML 2013

Figure from: Multi-Task Learning as Multi-Objective Optimization. Sener and Koltun, NeurIPS 2018

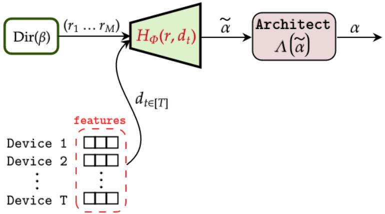
MODNAS

All pieces together



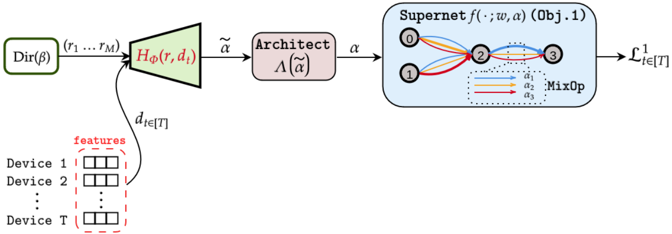
MODNAS

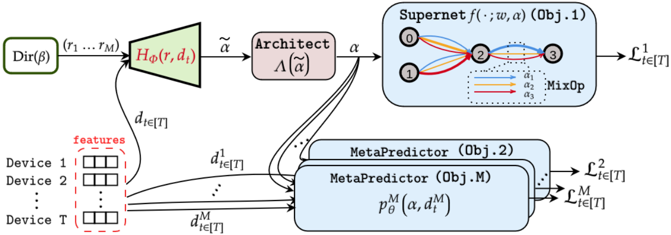
All pieces together



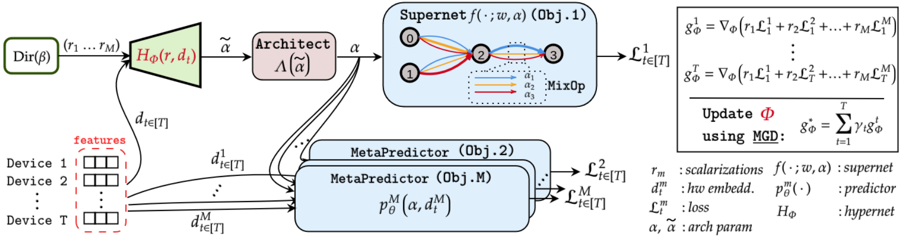
MODNAS

All pieces together





Dir(β)
$(r_1 \dots r_M)$
$H_\Phi(r, d_t)$
$\tilde{\alpha}$
Architect
$\Lambda(\tilde{\alpha})$
α
Supernet $f(\cdot; w, \alpha)$ (Obj.1)
MixOp
α_1
α_2
α_3
$\mathcal{L}^1_{t \in [T]}$
$d_{t \in [T]}$
features
Device 1
Device 2
Device T
$d^1_{t \in [T]}$
$d^M_{t \in [T]}$
MetaPredictor (Obj.2)
MetaPredictor (Obj.M)
$p^M_\theta(\alpha, d^M_t)$
$\mathcal{L}^2_{t \in [T]}$
$\mathcal{L}^M_{t \in [T]}$

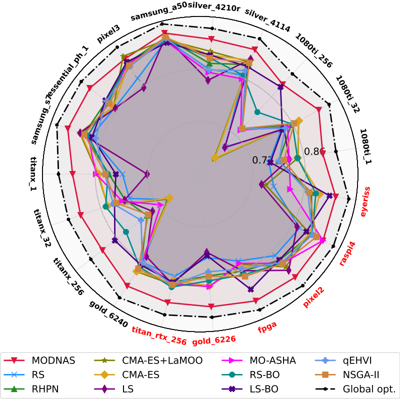


Dir(β)
(r_1 … r_M)
H_Φ(r, d_t)
α̃
Architect
Λ(α̃)
α
Supernet f(·; w, α) (Obj.1)
0
1
2
3
α_1
α_2
α_3
MixOp
L^1_{t∈[T]}
d_{t∈[T]}
features
Device 1
Device 2
Device T
d^1_{t∈[T]}
d^M_{t∈[T]}
MetaPredictor (Obj.2)
MetaPredictor (Obj.M)
p^M_θ(α, d^M_t)
L^2_{t∈[T]}
L^M_{t∈[T]}
g^1_Φ = ∇_Φ(r_1 L^1_1 + r_2 L^2_1 + … + r_M L^M_1)
g^T_Φ = ∇_Φ(r_1 L^1_T + r_2 L^2_T + … + r_M L^M_T)
Update Φ
using MGD:
g*_Φ = Σ_{t=1}^{T} γ_t g^t_Φ
r_m : scalarizations
d^m_t : hw embedd.
L^m_t : loss
α, α̃ : arch param
f(·; w, α) : supernet
p^m_θ(·) : predictor
H_Φ : hypernet

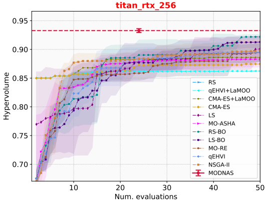
Experimental results

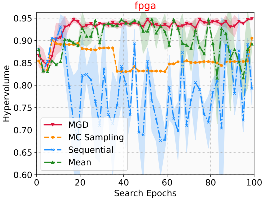
Simultaneous Pareto Set Learning across 19 devices on NB201

- **Metric**: Hypervolume (HV) indicator.
- **Baselines**:
  - Random baselines
  - Evolutionary strategies
  - Bayesian Optimization
- **Evaluation**: Sample 24 preference vectors and get the MAP architecture from the `MetaHypernetwork` output for each of them.



- **Metric**: Hypervolume (HV) indicator over time.
- **Baselines**:
  - Mean grad update
  - Sequential grad updates
  - Grad samples updates
- **Evaluation**: Sample 24 preference vectors and get the MAP architecture from the `MetaHypernetwork` output for each of them.

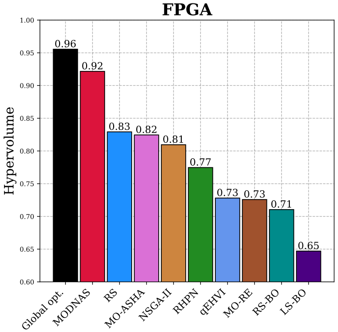


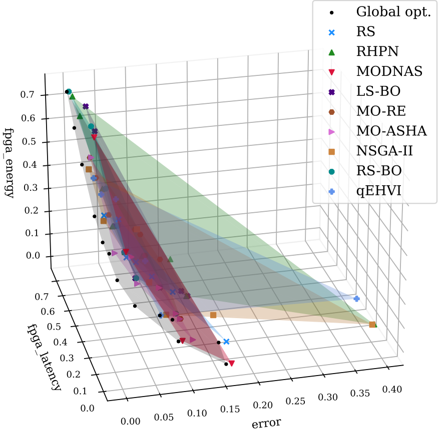


Experimental results

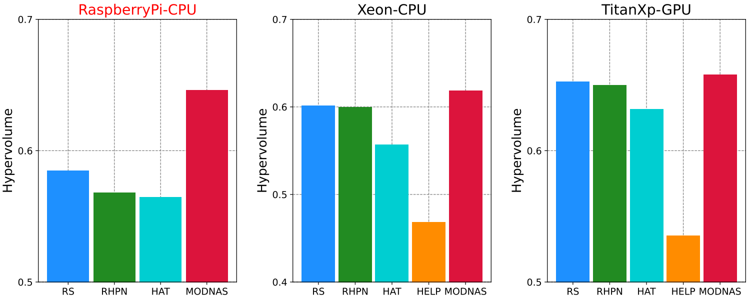
Scalability to 3 objectives

- Optimize for *latency*, *energy usage* and *accuracy* simultaneously across devices.

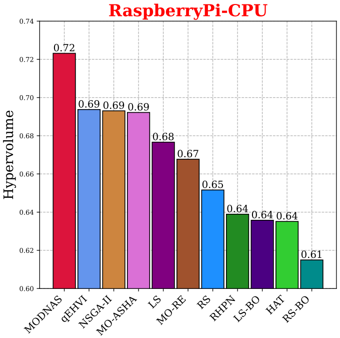


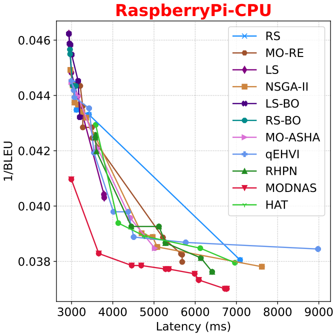


- We run MODNAS on the **Hardware-Aware Transformer (HAT)** [Wang et al. 2020] search space on the WMT'14 En-De machine translation task.
- **Search costs:** 6 days on 8 NVidia RTX A6000



- We run MODNAS on the **Hardware-Aware Transformer (HAT)** [Wang et al. 2020] search space on the WMT'14 En-De machine translation task.
- **Search costs:** 6 days on 8 NVidia RTX A6000

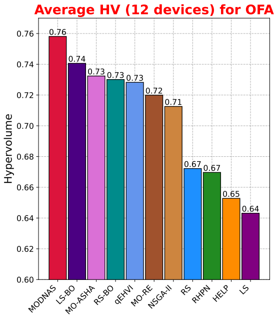


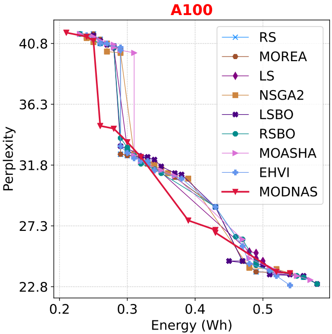


Experimental results

Efficient MOO on ImageNet-1k and OpenWebText starting from Pretrained Supernetworks

- We use the **Once-for-All (OFA)** `[Cai et al. 2020]` pretrained supernet on ImageNet and run MODNAS for 1 day on 8 GPUs.
- We also use **HW-GPT-Bench** texttt[Sukthanker et al. 2024] to run MODNAS on a GPT-2 search space.
- **Higher HV** compared to baselines





Summary and future directions

- MODNAS can **profile the pareto front** of various objective spaces.

Summary and future directions

- MODNAS can **profile the pareto front** of various objective spaces.
- Via a **hypernetwork** and **MGD**, MODNAS can optimize simultaneously across many devices (up to 19) and multiple objectives (up to 3).

Summary and future directions

- MODNAS can **profile the pareto front** of various objective spaces.
- Via a **hypernetwork** and **MGD**, MODNAS can optimize simultaneously across many devices (up to 19) and multiple objectives (up to 3).
- We show **improved hypervolume on test devices** across various spaces, tasks and datasets, without additional fine-tuning and with less search costs.

Summary and future directions

- MODNAS can **profile the pareto front** of various objective spaces.
- Via a **hypernetwork** and **MGD**, MODNAS can optimize simultaneously across many devices (up to 19) and multiple objectives (up to 3).
- We show **improved hypervolume on test devices** across various spaces, tasks and datasets, without additional fine-tuning and with less search costs.
- `https://arxiv.org/pdf/2402.18213`

In the future:

Summary and future directions

- MODNAS can **profile the pareto front** of various objective spaces.
- Via a **hypernetwork** and **MGD**, MODNAS can optimize simultaneously across many devices (up to 19) and multiple objectives (up to 3).
- We show **improved hypervolume on test devices** across various spaces, tasks and datasets, without additional fine-tuning and with less search costs.
- `https://arxiv.org/pdf/2402.18213`

In the future:

- Extend MODNAS to work in the Few-shot NAS settings with subspace partitions.

Summary and future directions

- MODNAS can **profile the pareto front** of various objective spaces.
- Via a **hypernetwork** and **MGD**, MODNAS can optimize simultaneously across many devices (up to 19) and multiple objectives (up to 3).
- We show **improved hypervolume on test devices** across various spaces, tasks and datasets, without additional fine-tuning and with less search costs.
- `https://arxiv.org/pdf/2402.18213`

In the future:

- Extend MODNAS to work in the Few-shot NAS settings with subspace partitions.
- More control on the preference vector sampler during search.

Thank you for your attention.
Questions?