End-to-End Meta-Bayesian Optimisation with Transformer Neural Processes

Alexandre Maraval[*]
Huawei Noah's Ark Lab
alexandre.maraval1@huawei.com

Matthieu Zimmer[*]
Huawei Noah's Ark Lab
matthieu.zimmer@huawei.com

Antoine Grosnit
Huawei Noah's Ark Lab
Technische Universität Darmstadt
antoine.grosnit2@huawei.com

Haitham Bou Ammar
Huawei Noah's Ark Lab,
University College London
haitham.ammar@huawei.com

Abstract

Meta-Bayesian optimisation (meta-BO) aims to improve the sample efficiency of Bayesian optimisation by leveraging data from related tasks. While previous methods successfully meta-learn either a surrogate model or an acquisition function independently, joint training of both components remains an open challenge. This paper proposes the first end-to-end differentiable meta-BO framework that generalises neural processes to learn acquisition functions via transformer architectures. We enable this end-to-end framework with reinforcement learning (RL) to tackle the lack of labelled acquisition data. Early on, we notice that training transformer-based neural processes from scratch with RL is challenging due to insufficient supervision, especially when rewards are sparse. We formalise this claim with a combinatorial analysis showing that the widely used notion of regret as a reward signal exhibits a logarithmic sparsity pattern in trajectory lengths. To tackle this problem, we augment the RL objective with an auxiliary task that guides part of the architecture to learn a valid probabilistic model as an inductive bias. We demonstrate that our method achieves state-of-the-art regret results against various baselines in experiments on standard hyperparameter optimisation tasks and also outperforms others in the real-world problems of mixed-integer programming tuning, antibody design, and logic synthesis for electronic design automation.

1 Introduction

Bayesian optimisation (BO) techniques are sample-efficient sequential model-based solvers optimising expensive-to-evaluate black-box objectives. Traditionally, BO methods operate in isolation focusing on one task at a time, a setting that led to numerous successful applications, including but not limited to hyperparameter tuning [1], drug and chip design [2], and robotics [3]. Although successful, focusing on only one objective in isolation may increase sample complexities on new tasks since standard BO algorithms work tabula-rasa as they start optimising from scratch, ignoring previously observed black-box functions.

To improve sample efficiency on new *target* tasks, meta-BO makes use of data collected on related *source* tasks [4] and attempts to transfer knowledge in between. Those methods are primarily composed of two parts: 1) a (meta) surrogate model predicting the function to optimise and 2) a (meta) acquisition function (AF) estimating how good a queried location is. Regarding surrogate

[*] These authors contributed equally to this work

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

modelling, prior work relies on Gaussian processes (GPs) to perform transfer across tasks due to their data efficiency [5–9], or focuses on deep neural networks [10–12] gaining from their representation flexibility. As for acquisition functions, previous works assumed a fixed GP and trained a neural network to perform transfer [13, 14].

Although successful, those methods rely on the assumption that the two steps described above can be handled independently, potentially missing benefits from considering both steps together. Therefore, this paper advocates for an end-to-end training protocol for meta-BO where we train a model predicting AF values directly from observed data, without relying on a GP surrogate.

As shown in [10, 12, 15, 16], Neural processes (NP) are a good candidate for meta-learning due to their structural properties, thus we propose to use a new transformer-based NP to model the acquisition function directly [17].

Nevertheless, the lack of labelled AF data prohibits a supervised learning protocol. Here, we rely on reward functions to assess the goodness of AFs and formulate a new reinforcement learning (RL) problem. Our RL formulation attempts to learn an optimal policy that selects new evaluation probes by minimising per-task cumulative regrets.

Early on, we faced very unstable training with minimal learning due to the combination of RL with transformers [18]. Furthermore, we notice that this problem amplifies when the reward function is sparse, which is the case in our setting, as we formally show in Section 3.2. To mitigate this difficulty, we introduce an inductive bias in our transformer architecture via an auxiliary loss [19] that guides a part of the network to learn a valid probabilistic model, effectively specialising as a neural process with gradient updates. Having developed our framework, we demonstrate state-of-the-art regret performance in hyperparameter optimisation of real-world problems and combinatorial sequence optimisation for antibody and chip design (see Section 4). Those solid empirical results show the value of end-to-end training in meta-BO to further improve sample complexity.

Contributions We summarise our contributions as follows: *i)* developing the first end-to-end transformer-based architecture for meta-BO predicting acquisition function values, *ii)* identifying logarithmic reward sparsity patterns hindering end-to-end learning with RL, *iii)* introducing NP-based inductive biases for successful model updates, and *iv)* demonstrating new state-of-the-art regret results on a wide range of traditional (hyperparameter tuning) and non-traditional (chip and antibody design) real-world benchmarks. Our official code repository can be found at `https://github.com/huawei-noah/HEBO/tree/master/NAP`.

2 Background

2.1 Bayesian Optimisation

In BO, we sequentially maximise an expensive-to-evaluate black-box function $f(\boldsymbol{x})$ for variables $\boldsymbol{x} \in \mathcal{X}$. BO techniques operate in two steps. First, based on previously collected evaluation data, we fit a probabilistic surrogate model to emulate $f(\boldsymbol{x})$ allowing us to make probabilistic predictions of the function's behaviour on unobserved input points. Given the first step's probabilistic predictions, the second step in BO optimises an AF that trades off exploration and exploitation to find a new input query, $\boldsymbol{x}_{\text{new}} \in \mathcal{X}$, for evaluation. Both the model and acquisition choices play a critical role in the success of BO. Many works adopt GPs as surrogate models due to their sample efficiency and practical uncertainty estimation [20]. Moreover, on the acquisition side, the common practice is to optimise one from a set of widely adopted AFs such as Expected Improvement (EI) [21].

2.2 Transfer in Bayesian Optimisation

Transfer techniques in BO [4, 13] aim to improve the optimisation of a newly observed *target* black-box function by reusing knowledge from past experiences gathered in related *source* domains, i.e. solve $\max_{\boldsymbol{x} \in \mathcal{X}} f_{\text{Target}}(\boldsymbol{x})$ by leveraging information from K source black-boxes $f_1(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})$. We assume this information is available to our agent via $\mathcal{D}_1, \ldots, \mathcal{D}_K$ such that each dataset $\mathcal{D}_k = \{\langle \boldsymbol{x}_i^{(k)}, y_i^{(k)} \rangle\}_{i=1}^{n^{(k)}}$ consists of $n^{(k)}$ (noisy) evaluations of $f_k(\boldsymbol{x})$ for all $k \in [1 : K]$. Following the meta-learning literature [4, 5, 13], we impose no additional assumptions on the process that collects $\mathcal{D}_1, \ldots, \mathcal{D}_K$, allowing for a diverse set of source optimisation algorithms, including but not limited to BO, genetic and evolutionary algorithms [22], and sampling-based strategies [23]. Many algorithms that use source data to improve performance on target domains exist. We categorise those based on the part they customise within the BO pipeline, i.e., by registering if they affect initial points [24, 25], search spaces [26], surrogate models [5] or AFs [13, 14]. Since our work devises end-to-end meta-BO pipelines, we now elaborate on critical surrogate models needed for the rest of the paper and leave a detailed presentation of related work to Section 5.

Although GPs [20] are a prominent tool for single-task BO, high computational complexity [27], among other drawbacks (e.g. smoothness assumptions or limited scalability in terms of dimensions), can limit their application to transfer scenarios. Of course, one can generalise GPs to support transfer using multi-task kernels [28] or GP ensembles [6], for example. However, recent trends demonstrated superior performance to such extensions when using deep networks (e.g., neural processes) to meta-learn probabilistic surrogates [5, 13].

Neural Processes A popular method for effective meta-learning consists of directly inputting context points (i.e., available data to adjust to a target task) for a model to adapt to unseen tasks [15]. We can accomplish such a goal by relying on neural processes (NPs), a class of models that combine the flexibility of deep neural networks with the properties of stochastic processes [10]. Given an observation set of input-output pairs $\mathcal{D}_{\text{obs}}$ and a set of n_{pred} locations $\boldsymbol{x}^{(\text{pred})}$ at which we desire to make predictions, an NP parameterised by $\boldsymbol{\theta}$ outputs a distribution $p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}^{(\text{pred})}, \mathcal{D}_{\text{obs}})$ that approximates the true posterior over the labels $y^{(\text{pred})}$. Among the different parameterisations of $p_{\boldsymbol{\theta}}(\cdot)$, e.g., in [29–31], transformer architectures [17] recently emerged as compelling choices of NPs [11, 12]. In this work, we also adopt a transformer architecture inspired by this architecture's broad software support and state-of-the-art supervised learning results as reported in [11].

2.3 Reinforcement Learning

In Section 3, we attempt to learn an end-to-end model that predicts acquisition values. Of course, we cannot fit the parameters of such a model with supervised learning due to the lack of labelled acquisition data. RL [32] is a viable alternative for learning from delayed and non-differentiable reward signals in those cases. In RL, we formalise problems as Markov decision processes (MDP): $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the state transition model, $\mathcal{R}$ the reward function dictating the optimisation goal, and $\gamma \in [0, 1)$ a discount factor specifying the degree to which rewards are discounted over time. A policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is an action-selection rule that is defined as a probability distribution over state-action pairs, where $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ represents the probability of selecting action $\boldsymbol{a}_t$ in state $\boldsymbol{s}_t$. An RL agent aims to find an optimal policy $\pi^\star$ that maximises (discounted) expected returns. For determining $\pi^\star$, we use the state-of-the-art proximal policy optimisation (PPO) algorithm [33].

3 End-to-End Meta-Bayesian Optimisation

While transfer techniques in BO saw varying degrees of success in many applications, current approaches lack end-to-end differentiability, where model learning and acquisition discovery arise as two separate steps. Specifically, the surrogate model gradients hardly affect the acquisition network, and the acquisition's network gradients fail to back-propagate to the surrogate's updates.

Enabling end-to-end transfer frameworks in which we learn the surrogate and acquisition jointly hold the promise for more scalable and easier-to-deploy algorithms that are more robust to input data or task changes. Following such a framework, we also expect more accurate predictions that can lead to better regret results (see Section 4) since we optimise the entire transfer pipeline, including the intermediate probabilistic model and AF representations. Additionally, end-to-end training techniques allow us to mitigate the need for domain-specific expertise and permit stable implementations that benefit fully from GPU and computing hardware.

The most straightforward way to enable end-to-end training in Bayesian optimisation is to introduce a deep network that acquires search variables and historical evaluations of black-box functions as inputs and outputs acquisition values after a set of nonlinear transformations. Of course, it is challenging to fit the weights of such a network due, in part, to a lack of labelled acquisition data where search variables and history of evaluations are inputs and acquisition values are labels.

3.1 Reinforcement Learning for End-to-End Training

Our approach utilises RL to fit the network's parameters $\boldsymbol{\theta}$ from minimal supervision, circumventing the need for labelled acquisition data. To formalise the RL problem, we introduce an MDP where:

State: $\boldsymbol{s}_t = [\mathcal{H}_t, t, T]$ (history, BO time-step & budget) **Action**: $\boldsymbol{a}_t = \boldsymbol{x}_t$ (choice of new probe),

with $\mathcal{H}_t = \{\langle \boldsymbol{x}_1, y_1 \rangle, \ldots, \langle \boldsymbol{x}_{t-1}, y_{t-1} \rangle\}$ denoting the history of black-box evaluations up-to the current time-step t. Adding the current BO step t and the maximum budget T in our state variable $\boldsymbol{s}_t$ helps balance exploration versus exploitation trade-offs as noted in [34]. Our MDP's transition function is straightforward, updating $\mathcal{H}_t$ by appending newly evaluated points, i.e., $\mathcal{H}_{t+1} = \mathcal{H}_t \cup \{\langle \boldsymbol{x}_t, y_t \rangle\}$ and incrementing the time variable t. Regarding rewards, we follow the well-established literature [13, 14] and define $r_t = \max_{1 \le \ell \le t} y_\ell$ to correspond to simple regret. Given such an MDP, our agent attempts to find a parameterised policy $\pi_{\boldsymbol{\theta}}$ which, when conditioned on $\boldsymbol{s}_t$, proposes a new probe $\boldsymbol{x}_t$ that minimises cumulative regret (i.e., the sum of total discounted simple regrets).

Extensions to Multi-Task Reinforcement Learning The above MDP describes an RL configuration that learns $\boldsymbol{\theta}$ in a single BO task. We now extend this formulation to multi-task scenarios allowing for a meta-learning setup. To do so, we introduce a set of MDPs $\mathcal{M}_1, \ldots, \mathcal{M}_K$ with K being the total number of available tasks. This paper considers same-domain multi-task learning scenarios. As such, we assume that all MDPs share the same state and action spaces and leave cross-domain extensions as an interesting avenue for future work. We define each MDP, $\mathcal{M}_k$, as previously introduced such that:

States: $\boldsymbol{s}_t^{(k)} = [\mathcal{H}_t^{(k)}, t^{(k)}, T^{(k)}]$ **Actions:** $\boldsymbol{a}_t^{(k)} = \boldsymbol{x}_t^{(k)}$ **Rewards:** $r_t^{(k)} = \max\limits_{1 \le \ell \le t^{(k)}} y_t^{(k)} \ \forall k.$

Moreover, for each task k, the transition model updates task-specific histories with $\mathcal{H}_{t+1}^{(k)} = \mathcal{H}_t^{(k)} \sqcup \{\langle \boldsymbol{x}_t^{(k)}, y_t^{(k)} \rangle\}$ and increments $t^{(k)}$. Contrary to the single task setup, we now seek a policy $\pi_{\boldsymbol{\theta}}$ which performs well on average across K tasks:

$$\arg\max_{\pi_{\boldsymbol{\theta}}} J(\pi_{\boldsymbol{\theta}}) = \arg\max_{\pi_{\boldsymbol{\theta}}} \mathbb{E}_{k \sim p_{\text{tasks}}} \left[\mathbb{E}_{\mathcal{H}_{T^{(k)}}^{(k)} \sim p_{\pi_{\boldsymbol{\theta}}}} \left[\sum_{t=1}^{T^{(k)}} \gamma^{t-1} r_t^{(k)} \right] \right], \tag{1}$$

where p_{tasks} denotes the task distribution. Furthermore, the per-task history distribution $p_{\pi_{\boldsymbol{\theta}}}(\mathcal{H}_{T^{(k)}}^{(k)})$ is jointly parameterised by $\boldsymbol{\theta}$ and defined as:

$$p_{\pi_{\boldsymbol{\theta}}} \left(\mathcal{H}_{T^{(k)}}^{(k)} \right) = p \left(y_{T^{(k)}-1}^{(k)} \middle| \boldsymbol{x}_{T^{(k)}-1}^{(k)} \right) \pi_{\boldsymbol{\theta}} \left(\boldsymbol{x}_{T^{(k)}-1}^{(k)} | \mathcal{H}_{T^{(k)}-1}^{(k)} \right) \ldots p \left(y_1^{(k)} | \boldsymbol{x}_1^{(k)} \right) \mu_0 \left(\boldsymbol{x}_1^{(k)} \right), \tag{2}$$

with $\mu_0 \left(\boldsymbol{x}_1^{(k)} \right)$ denoting an initial (action) distribution from which we sample the first decision $\boldsymbol{x}_1^{(k)}$. While it appears that off-the-shelf PPO can tackle the problem in Equation 1, Section 3.2 details difficulties associated with such an RL formulation, noting that in BO situations, the sparsity of reward signals can impede the learning of the network's weights $\boldsymbol{\theta}$. Before presenting those arguments, we now differentiate from the closest prior art, clarifying critical MDP differences.

Connection & Differences to [13] Although we are the first to propose end-to-end multi-task MDPs for meta-BO, others have also used RL to discover AFs, for example. Our parameterisation architecture and MDP definition significantly differ from the prior art, particularly from [13], the closest method to our work. Our approach uses a transformer-based deep network model to parameterise the whole pipeline (see Section 3.4). In contrast, the work in [13] assumes a pre-trained fixed Gaussian process surrogate and uses multi-layer perceptrons only to discover acquisitions. Our state variable requires historical information from which we jointly learn a probabilistic model and an acquisition end-to-end instead of requiring posterior means and variances of a Gaussian process model. Hence, our framework is more flexible, allowing us to model non-smooth black-box functions while overcoming some drawbacks of GP surrogates, like training and inference times.

3.2 Limitations of Regret Rewards in End-to-End Training

To define Equation 1, we followed the well-established literature of meta-BO [4, 13] and utilised simple regret reward functions. Although this choice is reasonable, we face challenges in applying such rewards in end-to-end training. Apart from difficulties associated with end-to-end training of deep architectures [18], our RL algorithm is subject to additional complexities when estimating gradients from Equation 1 due to the sparsity of the reward function. To better understand this problem, we start by noticing that for a reward component $r_t^{(k)}$ to contribute to the cumulative summation $\sum_t \gamma^{t-1} r_t^{(k)}$, we need to observe a function value $y_t^{(k)}$ that outperforms all values we have seen so far, i.e., $y_t^{(k)} > \max_{1 \leq \ell < t} y_\ell^{(k)}$. During the early training stages of RL, we can quantify the average number of such informative events (when $y_t^{(k)} > \max_{1 \leq \ell < t} y_\ell^{(k)}$) by a combinatorial argument that frames this calculation as a calculation of the number of cycles in a permutation of $T^{(k)}$ elements, leading us to the following lemma.

Lemma 3.1. *Consider a task with a horizon length (budget) T, and define $r_t = \max_{1 \leq \ell \leq t} y_t$ the simple regret as introduced in Equation 1. For a history $\mathcal{H}_T$, let $m_{\boldsymbol{\mathcal{H}}}$ denote the total number of informative rewards, i.e. the number of steps t at which $y_t > \max_{1 \leq \ell < t} y_\ell$. Under a random policy $\pi_{\boldsymbol{\theta}}$, the number of informative events is logarithmic in T such that: $\mathbb{E}_{\mathcal{H} \sim p_{\pi_{\boldsymbol{\theta}}}} [m_{\mathcal{H}}] = \mathcal{O}(\log T)$, where $p_{\pi_{\boldsymbol{\theta}}}$ is induced by $\pi_{\boldsymbol{\theta}}$ as in Equation 2.*

We defer the proof of Lemma 3.1 to Appendix A due to space constraints. Here, we note that this result implies that the information contained in one sampled trajectory is sparse at the beginning of RL training when the policy acts randomly. Of course, this increases the difficulty of estimating informative gradients of Equation 1 when updating $\boldsymbol{\theta}$. One can argue that the sparsity described in Lemma 3.1 only holds under random policies during the early stages of RL training and that sparsity patterns decrease as policies improve. Interestingly, simple regret rewards do not necessarily confirm this intuition. To realise this, consider the other end of the RL training spectrum in which policies have improved to near optimality such that $\pi_{\boldsymbol{\theta}} \rightarrow \pi_{\boldsymbol{\theta}^\star}$. Because $\pi_{\boldsymbol{\theta}}$ has been trained to maximise regret, it will seek to suggest the optimal point of the current task, as early as possible in the BO trajectory. Consequently, the policy is encouraged to produce trajectories with *even sparser* rewards during later training stages, further complicating the problem of informative gradient estimates of Equation 1.

3.3 Inductive Biases and Auxiliary Tasks

Learning from sparse reward signals is a well-known difficulty in the reinforcement learning literature [32]. Many solutions, from imitation learning, [35] to exploration bonuses [36], improve reward signals to reduce agent-environment interactions and enhance gradient updates. Others [37] attempt to define more informative rewards from prior knowledge or via human interactions [38]. Unfortunately, both of those approaches are hard to use in BO. Indeed, manually engineering black-box-specific rewards is notoriously difficult and requires domain expertise and extensive knowledge of the source and target black-box functions we wish to optimise. Furthermore, learning from human feedback is data-intensive, conflicting with the goal of sample-efficient optimisation.

Another prominent direction demonstrating significant gains is the introduction of auxiliary tasks (losses) within RL that allows agents to discover relevant inductive biases leading to impressive empirical successes [19, 39, 40]. Inspired by those results, we propose introducing an inductive bias in our method via an auxiliary supervised loss. Since we have at our disposal the collected source tasks datasets on which we are training our architecture $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(K)}$, we augment our objective such that our RL agent maximises not only rewards but also the likelihood of making correct predictions on these labelled datasets.

Supervised Auxiliary Loss Consider a source task k and consider that we split its corresponding dataset into an observed set and a prediction set $\mathcal{D}^{(k)} = \mathcal{D}_{\text{obs}}^{(k)} \sqcup \mathcal{D}_{\text{perd}}^{(k)}$. We define the auxiliary loss to be exactly the log-likelihood of functions values $y^{(\text{pred})}$ at predicted locations $\boldsymbol{x}^{(\text{pred})}$ given observations $\mathcal{D}_{\text{obs}}$:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{k \sim p_{\text{task}}, \mathcal{D}_{\text{obs}}^{(k)}, \mathcal{D}_{\text{perd}}^{(k)}} \Big[\log p_{\boldsymbol{\theta}}(y_k^{(\text{pred})} | \boldsymbol{x}_k^{(\text{pred})}, \mathcal{D}_{\text{obs}}^{(k)}) \Big]. \quad (3)$$

Interestingly, this part of our model specialises as a neural process (Section 2.2) with $\mathcal{D}_{\text{obs}}^{(k)}$ being the history $\mathcal{H}_t^{(k)}$ and $\boldsymbol{x}^{(\text{pred})}$ being the input points at which we wish to make predictions. To represent $p_{\boldsymbol{\theta}}(\cdot)$, we use a head of our transformer to predict multi-modal Riemannian (or bar plot probability density function) posteriors as [11, 41]. We compute Equation 3 on random iid splits of $\mathcal{D}^{(k)}$ and not directly on trajectories generated by the policy. This is because as the policy improves, the trajectories it generates are composed of less diverse (non-iid.) points. Indeed as training progresses, $\pi_{\boldsymbol{\theta}}$ becomes better and therefore finds the optimal point in $\mathcal{D}^{(k)}$ more rapidly. To fully take advantage of the labelled data at our disposal, we evaluate this auxiliary loss on iid. data, i.e. splits of $\mathcal{D}^{(k)}$ into $\mathcal{D}_{\text{obs}}^{(k)}$ and $\mathcal{D}_{\text{pred}}^{(k)}$ sampled uniformly at random. It is sensible from a BO standpoint as well since we want the introduced inductive bias to encourage our network to maximise the likelihood not only in a neighbourhood of the optimiser but also in the rest of the dataset.

3.4 Neural Acquisition Processes

We now introduce in more detail our transformer architecture and note some of its important properties. We call this architecture the neural acquisition process (NAP) because it is a new type of NPs jointly predicting AF values and distribution over actions. Similarly to other NP [11, 12], it takes a context and queried locations as input. We parameterise it by $\boldsymbol{\theta}$ and denote the acquisition prediction by $\alpha_{\boldsymbol{\theta}}(\boldsymbol{x}^{(\text{pred})}, \mathcal{H}, t, T)$.

Since $\alpha_{\boldsymbol{\theta}}$ outputs real numbers, we still need to define how we can obtain a valid probability distribution over the action space to form a policy $\pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s}_t)$. Defining such a distribution over the whole action space is hard if $\mathcal{A}$ is continuous. Hence, similarly to Volpp et al. [13], we evaluate the policy $\pi_{\boldsymbol{\theta}}$ only on the finite set of locations $\boldsymbol{x}^{(\text{pred})}$ for a given task[1]. Therefore, we have:

$$\pi_{\boldsymbol{\theta}}\left(\boldsymbol{x}_t^{(\text{pred})}|\boldsymbol{s}_t\right) \propto \frac{e^{\alpha_{\boldsymbol{\theta}}(\boldsymbol{x}_t^{(\text{pred})}, \mathcal{H}_t, t, T)}}{\sum_i^{n_{\text{pred}}} e^{\alpha_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{(\text{pred})}, \mathcal{H}_t, t, T)}}.$$

We now have all the necessary components to define the full objective combining Equation 1 and 3: $\mathcal{J}(\boldsymbol{\theta}) = J(\pi_{\boldsymbol{\theta}}) + \lambda\mathcal{L}(\boldsymbol{\theta})$, where λ is a hyperparameter to balance between the two objectives. We summarise the full training algorithm in Alg. 1 detailing each of the components.

Finally, we study some desirable properties of NAPs, explain how we can achieve them, and why they are important in the context of BO.

Algorithm 1 Neural Acquisition Process training.

```
Require: Source tasks training data $\{\mathcal{D}^{(k)}\}_{k=1}^{K}$, initial parameters $\boldsymbol{\theta}$, budgets $T^{(k)} \equiv T$, discount factor $\gamma$, learning rate $\eta$
  for each epoch do
      select task $k$ and dataset $\mathcal{D}^{(k)}$, set $\mathcal{H}_0 = \{\emptyset\}$
      for $t = 1, \ldots, T$ do
          $x_t \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s}_t)$                          ▷ predict action
          $y_t = f^{(k)}(x_t)$                                ▷ execute action
          $r_t = y^*_{\leq t}$                                   ▷ collect reward
          $\mathcal{H}_{t+1} \leftarrow \mathcal{H}_t \cup \{(x_t, y_t)\}$                 ▷ update hist.
      end for
      $R = \sum_{t=1}^{T} \gamma^t r_t$                               ▷ cumul. reward
      $\mathcal{D} \rightarrow \mathcal{D}_{\text{obs}} \sqcup \mathcal{D}_{\text{pred}}$                         ▷ split source data
      $\mathcal{L} = p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(\text{pred})}|\boldsymbol{x}^{(\text{pred})}, \mathcal{D}_{\text{obs}})$          ▷ aux. loss
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta(\ \nabla_{\boldsymbol{\theta}} R + \nabla_{\boldsymbol{\theta}} \mathcal{L}\ )$               ▷ update $\boldsymbol{\theta}$
  end for
```

Property 3.2 (History-order invariance)**.** *An NP g is history-order invariant if for any choice of permutation function ψ that changes the order of the points in history, $g(\boldsymbol{x}, \psi(\mathcal{H})) = g(\boldsymbol{x}, \mathcal{H})$.*

Unlike vanilla transformers, we do not use a positional encoding. It allows NAP to treat the history $\mathcal{H}$ as a set instead of an ordered sequence [42] and to be invariant to the order of the history. To form an $\langle \boldsymbol{x}, y \rangle$ pair in this set, we sum the embedding of $\boldsymbol{x}$ and y [11], whereas for queried locations we only use the embedding of $\boldsymbol{x}$ for a token (see Fig. 1, left). It is important for BO since the order in which we collect the points is not relevant for assessing how promising is a new queried location. Previous meta-RL algorithms [43] also shown the importance of relying on order-invariance.

Table 1: We compare the properties of different transformer architectures. L is the number of tokens needed to encode the meta-data, and D denotes the dimensionality of $\mathcal{X}$.

	History-order inv.	Query ind.	AF values	Tokens
NAP (ours)	✔	✔	✔	$t + n_{\text{pred}}$
TNPs [12]	✔	✘	✘	$t + n_{\text{pred}}$
OptFormer [41]	✘	✘	✘	$L + (D+2)(t + n_{\text{pred}})$
PFN [11]	✔	✔	✘	$t + n_{\text{pred}}$

[1] With a continuous action space, one could optimise $\boldsymbol{x}^{(\text{pred})}$ to maximise the NAP outputs before forming the distribution over the discrete actions.

Property 3.3 (Query independence)**.** *A NP g is query independent if for any choice of n queried locations $\boldsymbol{x}^{(pred)} = (\boldsymbol{x}_1^{(pred)}, \dots, \boldsymbol{x}_n^{(pred)})$, we have $g(\boldsymbol{x}^{(pred)}, \mathcal{H}) = (g(\boldsymbol{x}_1^{(pred)}, \mathcal{H}), \dots, g(\boldsymbol{x}_n^{(pred)}, \mathcal{H}))$.*

In NAP, every token in the history can access each other through the self-attention mask, whereas the elements of $\boldsymbol{x}^{(\text{pred})}$ can only attend to themselves and to tokens in the history $\mathcal{H}$ (see Fig. 1, right). Because the tokens inside $\boldsymbol{x}^{(\text{pred})}$ cannot access each other and we do not use positional encoding, NAP is query independent, which is important to make consistent predictions of AF values for BO as they should not depend on the other queried locations. Additionally, NAP is fully differentiable enabling end-to-end training but also optimisation of the queried locations via gradient ascent for continuous action spaces. In Table 1, we highlight the differences with other state-of-the-art models regarding those properties.

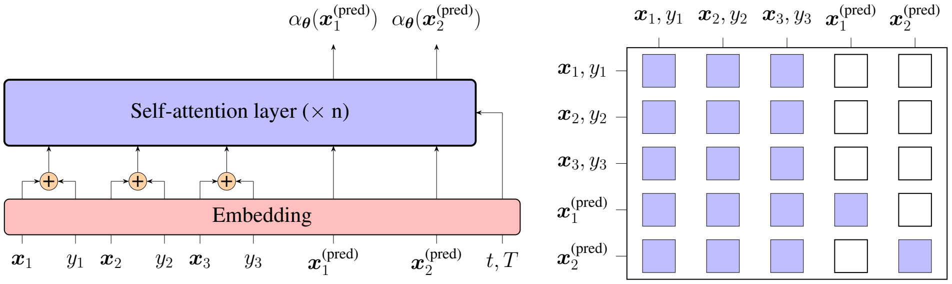
$\alpha_{\boldsymbol{\theta}}(\boldsymbol{x}_1^{(\text{pred})})$
$\alpha_{\boldsymbol{\theta}}(\boldsymbol{x}_2^{(\text{pred})})$
Self-attention layer (× n)
+
Embedding
$\boldsymbol{x}_1$
y_1
$\boldsymbol{x}_2$
y_2
$\boldsymbol{x}_3$
y_3
$\boldsymbol{x}_1^{(\text{pred})}$
$\boldsymbol{x}_2^{(\text{pred})}$
t, T
$\boldsymbol{x}_1, y_1$
$\boldsymbol{x}_2, y_2$
$\boldsymbol{x}_3, y_3$
$\boldsymbol{x}_1^{(\text{pred})}$
$\boldsymbol{x}_2^{(\text{pred})}$

Figure 1: Our proposed NAP architecture (left) and an example of the masks applied during inference (right). We apply independent embedding on $\boldsymbol{x}_i$, y_i, t and T. The colored squares mean that the tokens on the left can attend the tokens on the top in the self-attention layer.

4 Experiments

We conduct experiments on hyperparameter optimisation (HPO) and sequence optimisation tasks to assess NAP's efficiency. Regarding HPO, we run our algorithm on datasets from the HPO-B benchmark [44] and in real-world settings of tuning hyperparameters for Mixed-Integer Programming (MIP) solvers. For sequence optimisation, we test NAP on combinatorial black-box problems from antibody design and synthesis flow optimisation for electronic design and automation (EDA).

Baselines We compare our method against popular meta-learning baselines, including few-shot Bayesian optimisation (FSBO) [5], MetaBO [13] as well as OptFormers [41] when applicable. Moreover, we show how classical GP-based BO, equipped with an adequate kernel [2] and an EI acquisition function, which we title GP-EI, performs across domains. We first fit a GP on the meta-training datasets to enable a fair comparison. We initialise the GP model with those learnt kernel parameters at test time. This way, the GP baseline can benefit from the information provided in the source tasks. We also explore training a neural process directly on source task datasets and then using a fixed EI acquisition. For that, we introduce NP-EI that combines the same base architecture from [11] with EI. Additionally, we contrast NAP against random search (RS).

Following the standard practice in meta-BO, we report our results in terms of normalised regrets for easier comparison across all tasks. We attempt to re-implement all baselines across all empirical domains, extending previous implementations as needed, e.g., developing MetaBO [13] and FSBO [5] versions for combinatorial and mixed spaces to enable a fair comparison in MIP solver tuning, antibody and EDA design tasks.

Remark on OptFormer We re-run OptFormer with an EI AF on hyperparameter tuning tasks by extending the implementation in [41] to the discrete search space version of HPO-B. However, the lack of a complete open-source implementation and interpretable meta-data in the other benchmarks (e.g., in antibody design and EDA) prohibited successful execution.

[2] We adapt the GP kernel to each task depending on the nature of the optimisation variable, e.g., Matérn in continuous domains, categorical for combinatorial search space, and mixed when having both.

4.1 Hyperparameter Optimisation Results

Hyperparameter Optimisation Benchmarks We experiment on the HPO-B benchmark [44], which contains datasets of (classification) model hyperparameters and the models' corresponding accuracies across multiple types and search spaces. Due to resource constraints, we selected six representative search spaces. Nonetheless, we chose the search spaces to represent all underlying classification models in the experiment. We also always picked the ones with the least points to focus on the low data regime performance; see Appendix C.1 for more details. The results of our tests in Figure 2 demonstrate that NAP and OptFormer outperform all other baselines. Surprisingly, although NAP uses a much smaller architecture than OptFormer (around 15 million parameters vs 250 million for OptFormer) and trains on much less data (around 80k original points on average versus more than 3 million points for OptFormer), its regret performance is statistically similar to that of the OptFormer after 100 steps. Moreover, on the same GPU, to perform the same inference, NAP only uses 2% of OptFormer's compute time and around 40% of its memory usage.

Tuning MIP Solvers Apart from HPO-B, we consider another real-world example of hyperparameter tuning that requires finding the optimal parameters of MIP solvers. We use the open-source `SCIP` solver [45] and the `Benchmark` suite from the `MIPLib2017` [46] that consists of a collection of 240 problems. The objective is to find a set of hyperparameters so that `SCIP` can solve MIP instances in minimal time. Our high-dimensional search space comprises 135 hyperparameters with mixed types, including boolean, integers, categories and real numbers. We train our model on data collected from BO traces on 103 MIPs and test on a held-out set of 42 instances. Our results in Figure 2 demonstrate that NAP is capable of outperforming all other baselines reaching low regret about an order of magnitude faster than FSBO [5]. Figure 2 further demonstrates the importance of end-to-end training where NAP again outperforms NP-EI.

4.2 Sequence Optimisation Experiments

Now, we demonstrate NAP's abilities beyond hyperparameter tuning tasks in two real-world combinatorial black-box optimisation problems.

Antibody CDRH3-Sequence Optimisation This experiment focuses on finding antibodies that can bind to target antigens. Antigens are proteins, i.e., sequences of amino acids that fold into a 3D shape giving them specific chemical properties. A protein region called CDRH3 is decisive in the antibody's ability to bind to a target antigen. Following the work in [47], we represent CDRH3s as a string of 11 characters, each character being the code for a different amino acid in an alphabet of cardinality 22. The goal is to find the optimal CDRH3 that minimises the binding energy towards a specific antigen. Binding energies can be computed using state-of-the-art simulation software like Absolut! [48]. We collected datasets of CDRH3 sequences and their respective binding energies (with Absolut!) across various antigens from the protein database bank [49]. We then formed a transfer scenario across antigens where we meta-learn on 109 datasets, validate on 16, and test NAP on 32 new antigens. Our results in Figure 2 indicate that NAP is not limited to hyperparameter tuning tasks but can also outperform all other baselines in combinatorial domains.

Electronic Design Automation (EDA) Logic synthesis (LS) is an essential step in the EDA pipeline of the chip design process. At the beginning of LS, we represent the circuit as an AIG (an And-Inverter-Graph representation of Boolean functions) and seek to map it to a netlist of technology-dependent gates (e.g., 6-input logic gates in FPGA mapping). The goal in LS is to find a sequence of graph transformations such that the resulting netlist meets an objective that trades off the number of gates (area) and the size of the longest directed path (delay) in the netlist. We perform a sequence of logic synthesis operators dubbed a synthesis flow to optimise the AIG.

Following [50], we consider length 20 LS flows and allow an alphabet of 11 such operators, e.g., {`refactor`, `resub`, . . . , `balance`} as implemented in the open-source `ABC` library [51]. We collected datasets for 43 different circuits. Each dataset consisted of 500 sequences (collected via a Genetic algorithm optimizer) and their associated area and delay. Additionally, we applied the well-known heuristic sequence `resyn2` on each circuit to get a reference area and delay. For this task, the black-box takes a sequence as input and returns the sum of area and delay ratios with respect to the reference ones, as detailed in Appendix B.1. We train all methods on 30 circuits from `OpenABC` [50], validate on 4 and test on 9. Our results in Figure 2 again demonstrate that NAP outperforms all other baselines by a significant margin.

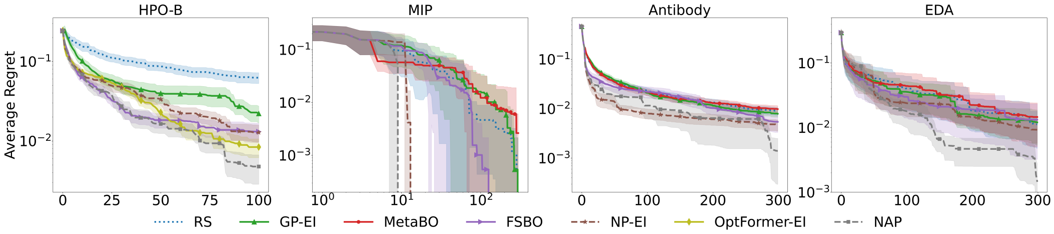

Figure 2: Average regret vs. BO iterations with 5 initial points. (Left) Results on 6 search spaces on the HPO-B benchmark. (Middle-left) Results tuning `SCIP` for solving 42 different MIPs. (Middle-right) Antibody CDR3 sequence optimisation on 32 test datasets corresponding to 32 different antigens. (Right) Logic synthesis operator sequence optimisation on 9 test datasets corresponding to 9 different circuits. For each method, error bars show confidence intervals computed across 5 runs on HPO-B and 10 runs on all the others.

5 Related Work

Meta-Learning Paradigms Meta-learning aims to learn how to quickly solve a new task based on the experience gained from solving or observing data from related tasks [15]. To achieve this goal, we can follow two main directions. In the first one, the meta-learner uses source data to learn a good initialisation of its model parameters, such that it is able to make accurate predictions after only few optimisation steps on the new target task [52]. The second one is more ambitious as it endeavours to directly learn, from the related tasks, a general rule to predict a helpful quantity to suggest the next point (black-box value, acquisition function value, etc.) from a context of observations (i.e. our history $\mathcal{H}^{(k)}$). Works following that direction, which include ours, can rely on different types of models to learn this general rule, such as NPs [10], recurrent neural networks [53], HyperNetworks [16] or transformers [12, 11]. Orthogonal to these two directions is the learning of a hyper-posterior on the datasets to meta-train on. In that line of research, Rothfuss et al. [54] suggest the use of Stein Variational Gradient Descent (SVGD) to estimate uncertainty with multiple models better, and Hsieh et al. [14] extend the use of SVGD to meta-learn AFs. We consider those last two works as an orthogonal direction to our, as NAP could also benefit from SVGD updates.

Learning a Meta-Model Chen et al. [55] and TV et al. [56] train an RNN to predict what should be the next suggestion in BO instead of predicting the acquisition function value. We do not compare to their method given the lack of available implementation. Still, we compare to the outperforming approach developed by Chen et al. [41] based on a transformer architecture designed to do meta-BO on hyperparameter tuning problems. Their OptFormer is trained on related tasks over different search spaces to output the next point to evaluate and predict its value. The next point is sequentially decoded, one token at a time, and exploits the hyperparameters' names or descriptions to improve generalisation across tasks. Contrary to NAP, OptFormer is not designed to predict acquisition value at any point and does not meet the two properties 3.2 and 3.3, and therefore needs much more training data to predict the proper sequence of tokens. We note that NAP does not rely on variable descriptions, making it easily deployable on various tasks and still very competitive in the hyperparameter optimisation context.

Rather than learning an entire predictive model from scratch, prior works [5, 7–9] learn deep kernel surrogates, i.e. a warping functions mapping the input space with a neural network before it is given to a GP. Learning only the input transformation allows the authors to rely on the closed-form posterior prediction capacity of standard GP models. To perform the transfer, Feurer et al. [6] rely on an ensemble of GPs. Iwata [57] is an approach close to FSBO [5] as it learns a meta-model by meta-training a Deep Kernel GP. Notably, it does so in an end-to-end fashion using RL to propagate gradients through the acquisition function and the GP back to the deep kernel. It does not, however, learn a meta-acquisition.

Learning a Meta-Acquisition Function Hsieh et al. [14] and Volpp et al. [13] choose to perform transfer through the acquisition function. They use directly GP surrogates and define the acquisition as a neural network that is meta-trained on related source tasks. They first pre-train GP surrogates on all source tasks and fix their kernel parameters. They then rely on RL training to meta-learn the neural acquisition function that takes as inputs the posterior mean and variance of the GP surrogate (which is itself trained online at test time). At test time, they allow for update in the GP but keep the weights of the neural acquisition fixed.

In summary, the methods meta-learning AFs do not do so in an end-to-end fashion and still rely on trained GP surrogates. While these methods are principled and competitive, they suffer from the cost of inverting the GP kernel matrix (cubic in the number of observations). In comparison, NAP can make predictions through a simple forward pass. On the other hand, the methods that learn a meta-model, either use a Deep Kernel GP (suffering the same cost) or have to learn a large model from scratch, costing a lot of budgets for collecting data beforehand, as well a pre-training time. Both of them use standard acquisition functions, missing the potential benefits of doing transfer in acquisition between tasks.

The performance of some of those algorithms on HPO-B is presented in Appendix-C showing that despite learning an architecture from scratch, NAP achieves a lower regret. For a more detailed survey on transfer and meta-learning in BO, we refer the reader to Bai et al. [4].

6 Conclusion

We proposed the first end-to-end training protocol to meta-learn acquisition functions in meta-BO. Our method predicts acquisition values directly from a history of points with meta-reinforcement learning and auxiliary losses. We demonstrated new state-of-the-art results compared to popular baselines in a wide range of benchmarks.

Limitations & Future Work Our architecture suffers from the usual quadratic complexity of the transformer in terms of the number of tokens which limits the budget of BO steps. Nevertheless, it can still handle around 5000 steps, which is enough for most BO scenarios. Another limitation of our architecture is that we need to train a new model for each search space. In future, we plan to enable our method to leverage meta-training from multiple search spaces and investigate how we could design data-driven BO-specific augmentation to further mitigate meta-overfitting [58].

Acknowledgements

We would like to thank Prof. Frank Hutter and his team for their constructive feedback and for the availability of their benchmark code and datasets. We also would like to thanks Massimiliano Patacchiola for his comments during the writing phase of the paper.

This work is supported by the CSTT on Generalisable Robot Learning via Machine Learning Models 2100332-GB and by the 2030 "New Generation of AI" - Major Project of China under grant No. 2022ZD0116408.

References

[1] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2960–2968, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html.

[2] Nate Gruver, Samuel Stanton, Polina Kirichenko, Marc Finzi, Phillip Maffettone, Vivek Myers, Emily Delaney, Peyton Greenside, and Andrew Gordon Wilson. Effective surrogate models for protein design with bayesian optimization. In *ICML Workshop on Computational Biology*, 2021.

[3] Fabio Muratore, Christian Eilers, Michael Gienger, and Jan Peters. Data-efficient domain randomization with bayesian optimization. *IEEE Robotics and Automation Letters*, 6(2): 911–918, 2021.

[4] Tianyi Bai, Yang Li, Yu Shen, Xinyi Zhang, Wentao Zhang, and Bin Cui. Transfer learning for bayesian optimization: A survey, 2023.

[5] Martin Wistuba and Josif Grabocka. Few-shot bayesian optimization with deep kernel surrogates. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=bJxgv5C3sYc`.

[6] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable meta-learning for bayesian optimization using ranking-weighted gaussian process ensembles. In *AutoML Workshop at ICML*, volume 7, 2018.

[7] Massimiliano Patacchiola, Jack Turner, Elliot J. Crowley, Michael O' Boyle, and Amos J Storkey. Bayesian meta-learning for the few-shot setting via deep kernels. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16108–16118. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/b9cfe8b6042cf759dc4c0cccb27a6737-Paper.pdf`.

[8] Zi Wang, George E. Dahl, Kevin Swersky, Chansoo Lee, Zachary Nado, Justin Gilmer, Jasper Snoek, and Zoubin Ghahramani. Pre-trained gaussian processes for bayesian optimization, 2023.

[9] Wenlin Chen, Austin Tripp, and José Miguel Hernández-Lobato. Meta-learning adaptive deep kernel gaussian processes for molecular property prediction. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=KXRSh0sdVTP`.

[10] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Jimenez Rezende, and S. M. Ali Eslami. Conditional neural processes. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1690–1699. PMLR, 2018. URL `http://proceedings.mlr.press/v80/garnelo18a.html`.

[11] Samuel Müller, Noah Hollmann, Sebastian Pineda-Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=KSugKcbNf9`.

[12] Tung Nguyen and Aditya Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 16569–16594. PMLR, 2022. URL `https://proceedings.mlr.press/v162/nguyen22b.html`.

[13] Michael Volpp, Lukas P. Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in bayesian optimization. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=ryeYpJSKwr`.

[14] Bing-Jing Hsieh, Ping-Chun Hsieh, and Xi Liu. Reinforced few-shot acquisition function learning for bayesian optimization. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 7718–7731. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/3fab5890d8113d0b5a4178201dc842ad-Paper.pdf`.

[15] Timothy M. Hospedales, Antreas Antoniou, Paul Micaelli, and Amos J. Storkey. Meta-learning in neural networks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(9):5149–5169, 2022. doi: 10.1109/TPAMI.2021.3079209. URL `https://doi.org/10.1109/TPAMI.2021.3079209`.

[16] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[18] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphaël Lopez Kaufman, Aidan Clark, Seb Noury, Matthew Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7487–7498. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/parisotto20a.html.

[19] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJ6yPD5xg.

[20] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian Processes for Machine Learning*, volume 2. MIT press Cambridge, MA, 2006.

[21] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129):2, 1978.

[22] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[23] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[24] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 1128–1135. AAAI Press, 2015. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/10029.

[25] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Mach. Learn.*, 107(1):43–78, 2018. doi: 10.1007/s10994-017-5684-y. URL https://doi.org/10.1007/s10994-017-5684-y.

[26] Valerio Perrone and Huibin Shen. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12751–12761, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/6ea3f1874b188558fafbab78e8c3a968-Abstract.html.

[27] Pavel Izmailov, Alexander Novikov, and Dmitry Kropotov. Scalable gaussian processes with billions of inducing inputs via tensor train decomposition. In Amos J. Storkey and Fernando Pérez-Cruz, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, volume 84 of *Proceedings of Machine Learning Research*, pages 726–735. PMLR, 2018. URL http://proceedings.mlr.press/v84/izmailov18a.html.

[28] Edwin V. Bonilla, Kian Ming Adam Chai, and Christopher K. I. Williams. Multi-task gaussian process prediction. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 153–160. Curran Associates, Inc., 2007. URL `https://proceedings.neurips.cc/paper/2007/hash/66368270ffd51418ec58bd793f2d9b1b-Abstract.html`.

[29] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Jimenez Rezende, and S. M. Ali Eslami. Conditional neural processes. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1690–1699. PMLR, 2018. URL `http://proceedings.mlr.press/v80/garnelo18a.html`.

[30] Jonathan Gordon, Wessel P. Bruinsma, Andrew Y. K. Foong, James Requeima, Yann Dubois, and Richard E. Turner. Convolutional conditional neural processes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=Skey4eBYPS`.

[31] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, S. M. Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=SkE6PjC9KX`.

[32] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN 978-0-262-19398-6. URL `https://www.worldcat.org/oclc/37293240`.

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

[34] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 1015–1022. Omnipress, 2010. URL `https://icml.cc/Conferences/2010/papers/422.pdf`.

[35] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[36] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[37] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287. Citeseer, 1999.

[38] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[39] Xingyu Lin, Harjatin Baweja, George Kantor, and David Held. Adaptive auxiliary task weighting for reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

[40] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. Agent modeling as auxiliary task for deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, volume 15, pages 31–37, 2019.

[41] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc Aurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards learning universal hyperparameter optimizers with transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 32053–32068. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/cf6501108fced72ee5c47e2151c4e153-Paper-Conference.pdf`.

[42] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 2019. URL `http://proceedings.mlr.press/v97/lee19d.html`.

[43] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5331–5340. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/rakelly19a.html`.

[44] Sebastian Pineda-Arango, Hadi S. Jomaa, Martin Wistuba, and Josif Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on openml. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL `https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/ec8956637a99787bd197eacd77acce5e-Abstract-round2.html`.

[45] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, et al. The scip optimization suite 8.0. *arXiv preprint arXiv:2112.08872*, 2021.

[46] Ambros M. Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff T. Linderoth, Marco E. Lübbecke, Hans D. Mittelmann, Derya B. Özyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Math. Program. Comput.*, 13(3):443–490, 2021. doi: 10.1007/s12532-020-00194-3. URL `https://doi.org/10.1007/s12532-020-00194-3`.

[47] Asif Khan, Alexander I. Cowen-Rivers, Antoine Grosnit, Derrick-Goh-Xin Deik, Philippe A. Robert, Victor Greiff, Eva Smorodina, Puneet Rawat, Rahmad Akbar, Kamil Dreczkowski, Rasul Tutunov, Dany Bou-Ammar, Jun Wang, Amos Storkey, and Haitham Bou-Ammar. Toward real-world automated antibody design with combinatorial bayesian optimization. *Cell Reports Methods*, page 100374, 2023. ISSN 2667-2375. doi: https://doi.org/10.1016/j.crmeth.2022.100374. URL `https://www.sciencedirect.com/science/article/pii/S2667237522002764`.

[48] Philippe A. Robert, Rahmad Akbar, Robert Frank, Milena Pavlović, Michael Widrich, Igor Snapkov, Andrei Slabodkin, Maria Chernigovskaya, Lonneke Scheffer, Eva Smorodina, Puneet Rawat, Brij Bhushan Mehta, Mai Ha Vu, Ingvild Frøberg Mathisen, Aurél Prósz, Krzysztof Abram, Alex Olar, Enkelejda Miho, Dag Trygve Tryslew Haug, Fridtjof Lund-Johansen, Sepp Hochreiter, Ingrid Hobæk Haff, Günter Klambauer, Geir Kjetil Sandve, and Victor Greiff. Unconstrained generation of synthetic antibody–antigen structures to guide machine learning methodology for antibody specificity prediction. *Nature Computational Science*, 2(12):845–865, 2022. ISSN 2662-8457. doi: 10.1038/s43588-022-00372-4. URL `https://www.nature.com/articles/s43588-022-00372-4`.

[49] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000.

[50] Animesh Basak Chowdhury, Benjamin Tan, Ramesh Karri, and Siddharth Garg. Openabc-d: A large-scale dataset for machine learning guided integrated circuit synthesis, 2021.

[51] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 24–40, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14295-6.

[52] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017. URL `http://proceedings.mlr.press/v70/finn17a.html`.

[53] Yan Duan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1087–1098, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/ba3866600c3540f67c1e9575e213be0a-Abstract.html`.

[54] Jonas Rothfuss, Vincent Fortuin, Martin Josifoski, and Andreas Krause. PACOH: bayes-optimal meta-learning with pac-guarantees. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9116–9126. PMLR, 2021. URL `http://proceedings.mlr.press/v139/rothfuss21a.html`.

[55] Yutian Chen, Matthew W. Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P. Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to learn without gradient descent by gradient descent. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 748–756. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/chen17e.html`.

[56] Vishnu TV, Pankaj Malhotra, Jyoti Narwariya, Lovekesh Vig, and Gautam Shroff. Meta-learning for black-box optimization. In Ulf Brefeld, Élisa Fromont, Andreas Hotho, Arno J. Knobbe, Marloes H. Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II*, volume 11907 of *Lecture Notes in Computer Science*, pages 366–381. Springer, 2019. doi: 10.1007/978-3-030-46147-8_22. URL `https://doi.org/10.1007/978-3-030-46147-8_22`.

[57] Tomaharu Iwata. End-to-end learning of deep kernel acquisition functions for bayesian optimization, 2021.

[58] Huaxiu Yao, Long-Kai Huang, Linjun Zhang, Ying Wei, Li Tian, James Zou, Junzhou Huang, and Zhenhui Li. Improving generalization in meta-learning via task augmentation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 11887–11897. PMLR, 2021. URL `http://proceedings.mlr.press/v139/yao21b.html`.

[59] V. L. Goncharov. Sur la distribution des cycles dans les permutations. *Acad. Sci. URSS (N.S.)*, 35:267–269, 1944.

[60] Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor H. Awad, Marius Lindauer, and Frank Hutter. Hpobench: A collection of reproducible multi-fidelity benchmark problems for HPO. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL `https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/93db85ed909c13838ff95ccfa94cebd9-Abstract-round2.html`.

A Proof of Lemma 3.1

Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ denote the set of points associated to a task, and a untrained policy π that collects a trajectory of length T by iteratively drawing points in $\mathcal{D}$ uniformly without replacement. We note $(z_1, \dots, z_T)$ the sequence of function values observed during the trajectory, and $(r_1, \dots, r_T)$ the sequence of rewards obtained. We remind that $r_t = \max_{1 \leq \ell \leq t} z_\ell$, and we consider that r_t (and z_t) is informative if $z_t = \max_{1 \leq \ell \leq t} z_\ell$. We want to compute the probability of obtaining exactly m informative rewards by sampling a trajectory from π. As each sequence is equiprobable, we do so by counting the number of sequences leading to m informative rewards. We assume for now that the values in $\{y_i\}_{i=1}^n$ are pairwise distinct.

We first note that there are $\binom{n}{T}$ ways to choose the $T \leq n$ points composing a trajectory of length T from $\mathcal{D}_n$. We now consider that the set of T sampled value functions is fixed (without loss of generality we note this set $V = \{v_\ell\}_{1 \leq \ell \leq T}$). For $1 \leq k \leq T$, we note $C(k, T)$ the number of ways to order them such that the resulting trajectory $(z_1, \dots, z_n)$ contains exactly k informative values, and we will give a recurrent formula for $C(k, T)$.

We can see that $k = 1$ necessarily implies that $z_1 = \max_{1 \leq \ell \leq T} v_\ell$. Thus there are $(T-1)!$ ways to order the remaining elements of the trajectory $\{z_\ell\}_{2 \leq \ell \leq n}$, hence $C(1, T) = (T-1)!$. On the other hand, $k = T$ informative rewards are only obtained for when the element of V are sorted in increasing order, i.e. with ($z_1 < z_2 < \dots < z_T$), and therefore $C(T, T) = 1$.

Finally, for $2 \leq k < T$ we can establish a recurrence relation by reasoning on z_n, the last element of the sequence:

- If $z_T = \max_{1 \leq \ell \leq n} v_\ell$, then z_T is informative, and there remains to count the number of ways to order the first $T-1$ elements $V \backslash \{z_T\}$ to get $k-1$ informative steps, which is by definition $C(k-1, T-1)$.
- If $z_T = v_j < \max_{1 \leq \ell \leq T} v_\ell$, then z_T is not informative. We note that there are $T-1$ choices for such v_j, and for each choice of v_j there remains to order $V \backslash \{v_j\}$ such that the resulting sub-trajectory $(z_1, \dots, z_{T-1})$ has exactly k informative values. There are therefore $(T-1)C(k, T-1)$ trajectories with k informative rewards such that $z_T \neq \max_{1 \leq \ell \leq T} v_\ell$

From this analysis we get that $C(k, T) = C(k-1, T-1) + (T-1)C(k, T-1)$.

This relation, along with boundary values $C(1, T) = (T-1)!$ and $C(T, T) = 1$, allow to identify $C(k, T)$ as the Stirling number of first kind $\left[{T \atop k}\right]$. This number notably corresponds to the number of permutations in S_T made of exactly k cycles.

Finally, the number of trajectories of length T that can be obtained from $\mathcal{D}$ and having k informative rewards, is given by $\frac{\binom{n}{T} \times C(k,T)}{\binom{n}{T} \times T!} = \frac{1}{T!}\left[{T \atop k}\right]$, which is equal to the probability of getting k cycles in a permutation sampled randomly in S_T. Therefore the expected number of informative rewards in a trajectory of length T is equal to the average number of cycles in a permutation sampled uniformly in S_T, which is a known result [59], thus is equal to the T^{th} harmonic number $H_T = \log T + \mathcal{O}(1)$.

B Experimental setup

In this section we give more details about the experimental setup. In particular, we add details about the experiments environments and baselines where needed. We also give some more intuition on the NAP training and data augmentation scheme used. Finally we list all important hyperparameters as well as the hardware used in our experiments.

B.1 Tasks

HPO-B As mentioned in Section 4 for this experiment we choose a subset of tasks from the HPO-B benchmark set. HPO-B is a collection of HPO datasets first grouped by search space. Each search space corresponds to the hyperparameters of a particular model, e.g. SVM, XGBoost, ... Each such search space then has multiple associated datasets split into a set for training, validating and for testing. The multi-task RL setting from Section 3.1 states that we limit ourselves to MDPs sharing state and action spaces across tasks hence we don't train NAP on multiple search spaces at the same time. We train one model per search space, being careful to choose a search space for each type of underlying model. When multiple search spaces related to the same underlying model we choose the search space with the least amount of total data in order to focus on the low-data regime as much as possible. We pick the following search spaces: 5860 (glmnet), 4796 (rpart.preproc), 5906 (xgboost), 5889 (ranger), 5859 (rpart), 5527 (svm). Refer to Table 3 in Pineda-Arango et al. [44] for more details.

Electronic Design Automation Following the description in Section 4 we search for the sequence of operators to optimise an objective combining two metrics associated with circuit performance, the area and the delay. The area is the number of gates in the mapped netlist, while the delay corresponds to the length of the longest directed path in the mapped netlist. As these two metrics are not directly commensurable, we normalise them by the area and delay obtained when running twice the reference sequence `resyn2` [51] made of 10 operators, as follows:

$$f_{\text{EDA}}(\texttt{seq}) = \frac{\text{Area}(\texttt{seq})}{\text{Area}(2 \times \texttt{resyn2})} + \frac{\text{Delay}(\texttt{seq})}{\text{Delay}(2 \times \texttt{resyn2})}$$

where `seq` is a sequence of operators from `ABC`.

B.2 Baselines

OptFormer The results reported by Chen et al. [41] are for the continuous HPO-B benchmark where XGBoost models approximate the black-box functions from the discrete points. To evaluate every approach in a fairer and more robust manner, we instead focused on the original discrete setup of HPO-B which uses only true values from the black-boxes. We relied on the shared OptFormer checkpoint trained and validated on HPO-B. We adapted the inference code for the discrete setting and noticed that the default parameters were set to NaPolicy.DROP which drop the missing values in the HPO-B benchmark and removes the additionnal "na" columns. We switched it to NaPolicy.CONTINUOUS to keep every column leading to better performances. We also had to increase the maximum number of tokens possible in a trajectory from 1024 to 2048.

B.3 NAP training

We conduct our experiments in a consistent manner. We define a training , validation and test sets that are kept the same for each method. We also ensure reproducibility by ensuring random seeds are similar across experiments and initial points too. Finally we run the tests on 10 different random seeds in all experiments and 5 for HPO-B as there are only 5 seeds available for other baselines.

Data augmentation When training on tasks with low number of points in each dataset, we perform data augmentation using Gaussian processes. On each dataset we fit an exact GP and during training we sample a new dataset directly from the posterior of that GP. This has an interpolating effect such that between original data points, the GP can make predictions that are roughly realistic. Training on these augmented datasets sampled from GP posteriors is handy in practice because it helps to create datasets of the desired size (we can sample as many data points as we like) and datasets that resemble the original one, provided we don't sample from the GP posterior too far from the original inputs. In practice we sample inputs points from the original dataset, add to them a random uniform perturbation and sample from the GP posterior at those new points.

Value function The value function takes as input t/T and the best y value observed in $\mathcal{H}_t$.

B.4 Hyperparameters

We share in Table 2 a comprehensive list of the hyperparameters used during training and inference. More details can be found in the associated code repository. We want to underline that none of these presented hyperparameters were tuned. This is only fair as we did also not optimise any

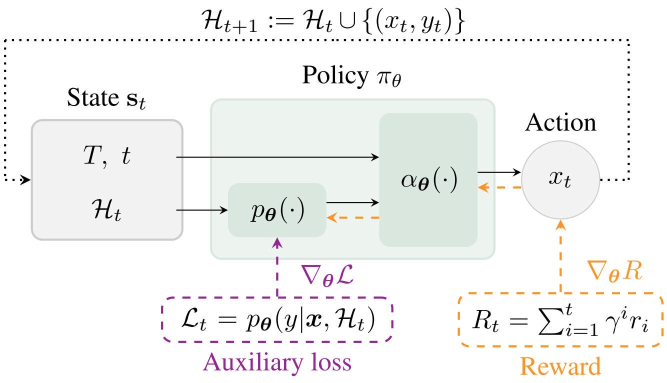
$\mathcal{H}_{t+1} := \mathcal{H}_t \cup \{(x_t, y_t)\}$
Policy π_θ
State $\mathbf{s}_t$
$T,\ t$
$\mathcal{H}_t$
$p_{\boldsymbol{\theta}}(\cdot)$
$\alpha_{\boldsymbol{\theta}}(\cdot)$
Action
x_t
$\nabla_{\boldsymbol{\theta}}\mathcal{L}$
$\nabla_{\boldsymbol{\theta}}R$
$\mathcal{L}_t = p_{\boldsymbol{\theta}}(y|\boldsymbol{x}, \mathcal{H}_t)$
Auxiliary loss
$R_t = \sum_{i=1}^{t} \gamma^i r_i$
Reward

Figure 3: Summary of our proposed Neural Acquisition Process (NAP) architecture. At iteration $t = 1, \dots, T$, the state consists of $s_t\mathbf{s}_t = \{\mathcal{H}_t, t, T\}$, respectively the history of collected points, the current iteration index and the total budget. The action is sampled from the policy $x_t \sim \pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t)$. For a set of locations $\boldsymbol{x} \subseteq \mathcal{A}$, the gradients flow back to parameters $\boldsymbol{\theta}$ from both the cumulative regret returns R_t and the auxiliary likelihood loss $\mathcal{L}_t$.

hyperparameters from the other baseline methods. Hyperparameters that are shared between our method and previous baselines are simply taken as is from their respective codebases as we assume the authors have already tuned them. We use the same for all experiments. Note that the ablation study in C.1 can be seen as a simple tuning of the parameter λ introduced to weight both losses in NAP.

Table 2: List of used hyperparameters in NAP.

PPO	
Learning rate for gradient descent	$3 \cdot 10^{-5}$
Learning rate decay	Linear decay to 0 over 2000 iterations
Number of training PPO iterations	2000
Horizon of episodes used in training	24
Trajectories collected per iteration	60
Total numbers of transformer updates	90,000
Minibatch size	32
Weight of auxiliary loss in total loss (λ)	1.0
Weight of the value function loss in total loss	1.0
Generalised Advantage Estimator-λ	0.98
Discount factor γ	0.98
Clip of importance sampling ratio ϵ	0.15
L2 gradient clipping	0.5
BO environment	
Range of Uniform perturbation for data augmentation	$[-0.05, 0.05]$
Number of random initial points	0 during training, 5 during validation
Architecture	
Number of buckets in the output histogram	1000
Point-wise feed-forward dimension of Transformer	1024
Embedding dimension of Transformer	512
Number of self-attention layers of Transformer	6
Number of self-attention heads of Transformer	4
Dropout rate of Transformer	0.0
Softmax temperature to compute π from α	0.1 for training, argmax otherwise
Value function network	Linear(2, 512), TanH, Linear(512, 1)

B.5 Hardware

We train our model on a machine with 4 GPUs Tesla V100-SXM2-16GB and an Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz with 88 threads with an average training time of approximately 10 hours per experiment.

C Additional Results

C.1 Ablation

In this section we perform an ablation study to answer some of the questions that arise naturally from the proposed framework. Notably, is end-to-end training useful? Does the auxiliary loss help? Is it useful to learn the acquisition function and not simply a surrogate model? We run an additional experiment on a dataset from the HPOBench benchmark [60]. For reference, we detail the methods of this ablation in Table 3 for easier comparison.

Table 3: Variations of NAP and their components.

	$p_{\boldsymbol{\theta}}$	$\alpha_{\boldsymbol{\theta}}$	RL	Supervision	End-to-end
NAP (ours)	✓	✓	✓	✓	✓
Pre-NAP	✓	✓	✓	✓	✗
NAP-RL	✓	✓	✓	✗	✓
NP-EI	✓	✗	✗	✓	✗

This study uses datasets of HPOBench [60] for XGBoost hyperparameters, similarly to Volpp et al. [13]. It consists of hyperparameter configurations and their associated accuracy of the XGBoost model on a classification task.

We analyse the effects of training end-to-end with the introduced auxiliary loss to better understand the method's strengths and limitations. Six hyperparameters (learning rate, regularisation, etc.) and 48 classification tasks exist. We have 1000 hyperparameter configurations evaluated for each task and the corresponding XGBoost model accuracy, creating 48 datasets of different black-box functions. We meta-train on 20 datasets, validate on 13 and test on 15.

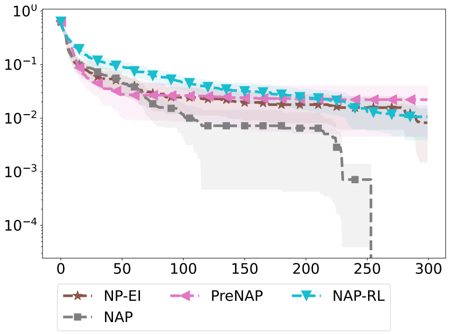

Figure 4: Average regret vs iterations on HPOBench dataset for XGBoost. Error bars are confidence intervals across ten runs.

Is it worthwhile to learn a meta-acquisition function? A valid question to ask is whether or not we need to meta-learn an acquisition function. There exist popular methods for meta-learning models, as discussed in the main paper, so one could use such a surrogate model and apply it directly in BO, using its posterior distribution to compute an acquisition function. Such an approach matches our baseline NP-EI, where we train a PFN [11] on the same training data and apply it directly in BO. Comparing NAP and NP-EI, Figure 4 reveals the benefits of learning the acquisition function as part of the end-to-end architecture instead of using a pre-defined expected improvement acquisition function.

Does training end-to-end using the auxiliary loss help? Training end-to-end, we check that using supervised information through the auxiliary loss helps compared to end-to-end training with only the reinforcement learning reward. We name the latter NAP-RL and show the results in Figure 4 which suggests that indeed, the inductive bias introduced with the auxiliary loss is beneficial for downstream performance.

Is end-to-end training beneficial? To investigate this question, we first pre-train the probabilistic model part of a NAP with the supervised auxiliary loss and then use PPO to update the meta-acquisition function while keeping the rest of the weights frozen. We denote this method as PreNAP as the architecture is partly pre-trained. Figure 4 shows that PreNAP . Thus, training jointly end-to-end NAP with both objectives translate into improved regret at test time, validating our hypothesis in Section 3.

C.2 HPO-B per search space results

Additionally to the aggregated regret plot in Figure 2 of Section 4 we show ranks and regrets per search space.

Figure 5 shows the regret for each method per search space, aggregated on all the test datasets of that search space and across 5 random seeds. Figure 6 shows the relative rank of each method (lower is better) per search space, aggregated across all the test datasets of that search space and 5 random seeds.

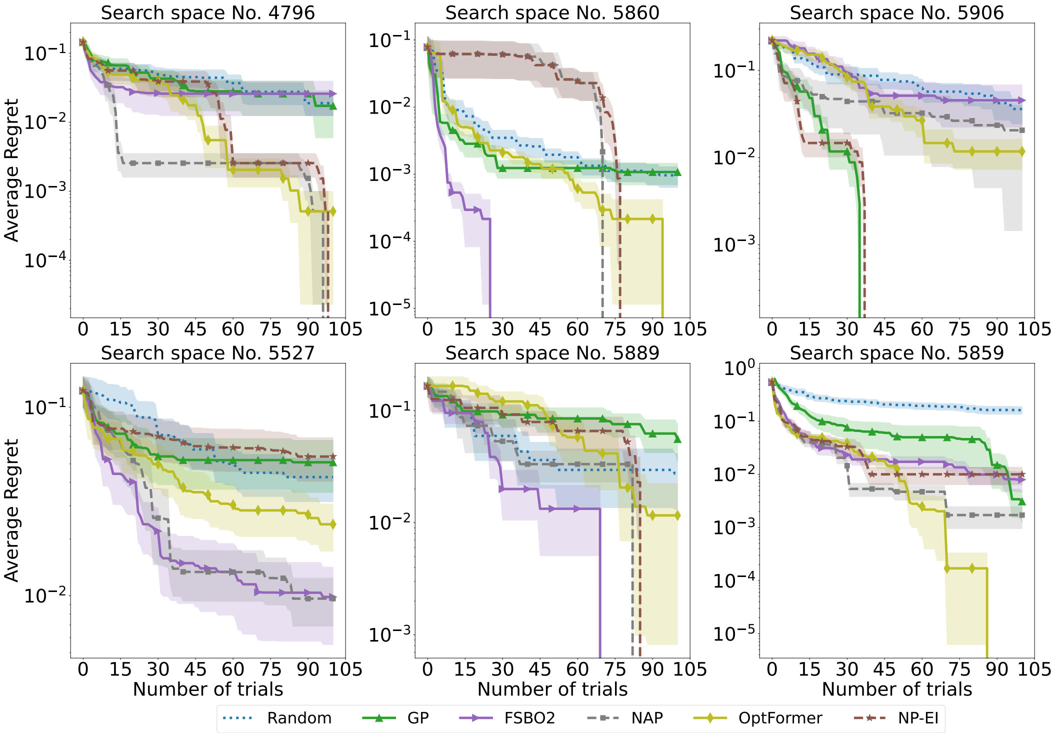

Figure 5: Average regret vs. BO iterations on each search space with 5 initial points. For each method, error bars show confidence intervals computed across 5 runs.

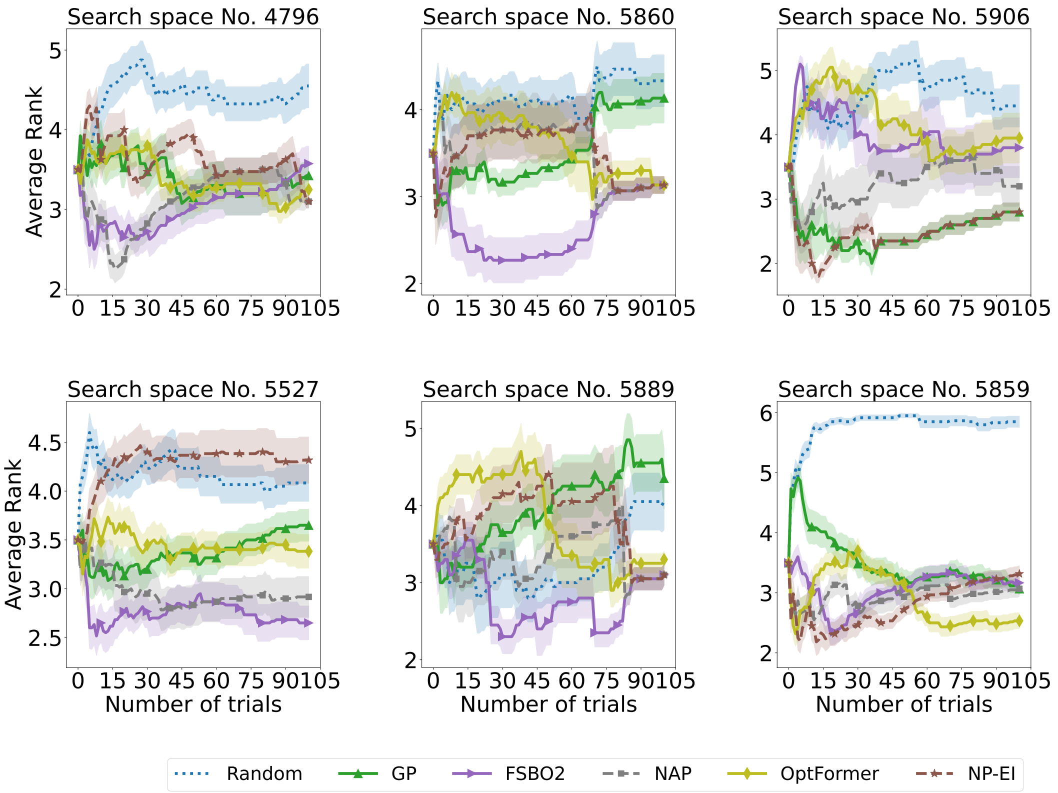

Figure 6: Average rank (lower is better) vs. BO iterations on each search space with 5 initial points. For each method, error bars show confidence intervals computed across 5 runs.

C.3 Time Comparison

While in our BO setting we assume that querying the black-box objective is the main bottleneck (hence our focus on sample efficiency), it is also interesting to analyse the time efficiency of algorithms to gain another perspective on various methods. In this section we summarise some average test running time results to provide a different point of view. In short, we see that even though some methods require offline pre-training (e.g. MetaBO, FSBO, NAP), the time required to evaluate points in the objective function out-weights this cost. Hence when measuring the total running time, it makes little difference if some methods require this pre-training.

For example, in the antibody experiment, evaluating the objective is costly. This is true both in terms of monetary and time costs as evaluating the objective could mean manufacturing the molecule and testing it in a wet-lab experiment. In our experiments we use a simulator as a proxy. The HPO-B black-box function is also very expensive to evaluate as it relies on training and testing several models. We use result files posted on the authors' repository which only contain black-box values for some baselines.

However we do have at our disposal the true black-boxes for the MIP and EDA experiments. By design of the experiment, evaluating one set of hyperparameters on the MIP experiment takes 2 hours. Compared to that, the time to train a GP model or doing a forward pass in NAP at test time is negligible. On EDA, the black-box time depends on the circuit so we approximate an average running time of 1 minute per circuit on open-source circuits, but this can take several hours on industrial circuits.

Table 4 and 5 compare the average test time of one seed across all methods. In the first column, we can see that methods which have to fit a GP during the BO loop (FSBO, MetaBO and GP-EI) are considerably slowed down compared to methods like NAP that only do forward passes through their network. This is because fitting the GP surrogate at each BO step is time consuming, and increasingly so, as its dominant computational cost is cubic in the number of observed points. Note also that

FSBO not only fits a GP at each step but also fine tunes the MLP of its deep kernel, hence the extra time. The second column, with the black-box time taken into account, further underlines that even though NAP is faster at test time than e.g. FSBO or GP-EI, this time gain it is negligible compared to the black-box evaluations. The third column takes into account the pre-training time for methods that require it. Note that for different test functions within the same search space, we can reuse the same model for NAP, NP-EI, MetaBO and FSBO without having to redo the pre-training, so we divided the pre-training time by the number of seeds and test functions. Hence, it does not add much time to the total.

It should be underlined that this way of presenting BO results is less readable than presenting regret vs BO steps as the more seeds and test tasks we have, the more negligible the pre-training time becomes compared to the black-box evaluation time.

Table 4: Average test time of 1 seed on the MIP experiment.

Method	**without bbox**	**with bbox**	**with bbox & pretrain**
GP-EI	585sec	25d 0hr 9min 45sec	25d 0hr 9min 45sec
FSBO	330sec	25d 0hr 5min 30sec	25d 0hr 10min
MetaBO	30sec	25d 0hr 0min 30sec	25d 0hr 12min
NP-EI	2sec	25d 0hr 0min 2sec	25d 0hr 36min
NAP	3sec	25d 0hr 0min 3sec	25d 1hr

Table 5: Average test time of 1 seed on the EDA experiment.

Method	**without bbox**	**with bbox**	**with bbox & pretrain**
GP-EI	17sec	1hr 5min 17sec	1hr 5min 17sec
FSBO	516sec	1hr 13min 36sec	1hr 14min 6sec
MetaBO	35sec	1hr 5min 35sec	1hr 12min 5sec
NP-EI	8sec	1hr 5min 8sec	1hr 7min 34sec
NAP	9sec	1hr 5min 9sec	1hr 7min 42sec

D Discussions

We give a more detailed explanation of Table 1 below.

OptFormer OptFormer encodes a history as follows: a short meta-data sequence describing the variables taken as input and a sequence of trials (a trial corresponds to an $\langle \boldsymbol{x}, y \rangle$ pair). Each trial is composed of the values present in $\boldsymbol{x}$, the value of y and a separator to mark the end of a trial $(D + 2)$. They need to use positional encoding to keep the sequence consistent (for instance the order of the dimensions is important to identify which dimension of x it is). Because of that, their architecture is not history-order invariant.

The query independence of their model is debatable. We can achieve it by splitting the queries across batches, but it is not doable in a single batch without substantial modifications of their code, their masks and their positional encoding. Note that splitting queries across batches results in a very slow inference. To evaluate OptFormer on HPO-B in a fair manner we had to do it, and it took us more than 2 weeks to obtain the results with 16 GPUs.

Transformer NP Nguyen and Grover [12] propose several transformer-based neural process architecture. Omitting the fact that they predict a Gaussian distribution over function values and not acquisitions, their TNP-D transformer architecture is the closest to ours. It does not rely on positional encoding, hence it is history-order invariant.

However, instead of summing the embeddings of $\boldsymbol{x}$ and y as we do, they concatenate them in a fixed representation, forcing them to set $y = 0$ in the queries. Hence, the only way to achieve query independence is to train on the same queries during testing and training. As we cannot assume we know what the queries will be during testing, Property 3.3 is not respected for **any** queries.

Prior Fitted Transformer PFN satisfies the two properties 3.2 and 3.3 since they do not rely on positional encoding and sum the embeddings of $\boldsymbol{x}$ and y.

The main differences with them are that our input is different and that we predict AF values with reinforcement learning.