Multi-step Planning for Automated Hyperparameter Optimization with OptFormer

Lucio M. Dery*
Carnegie Mellon University

Abram L. Friesen
Deepmind

Nando De Freitas
Deepmind

Marc'Aurelio Ranzato
Deepmind

Yutian Chen
Deepmind

Abstract

As machine learning permeates more industries and models become more expensive and time consuming to train, the need for efficient automated hyperparameter optimization (HPO) has never been more pressing. Multi-step planning based approaches to hyperparameter optimization promise improved efficiency over myopic alternatives by more effectively balancing out exploration and exploitation. However, the potential of these approaches has not been fully realized due to their technical complexity and computational intensity. In this work, we leverage recent advances in Transformer-based, natural-language-interfaced hyperparameter optimization to circumvent these barriers. We build on top of the recently proposed OptFormer which casts both hyperparameter suggestion and target function approximation as autoregressive generation thus making planning via rollouts simple and efficient. We conduct extensive exploration of different strategies for performing multi-step planning on top of the OptFormer model to highlight its potential for use in constructing non-myopic HPO strategies.

1 Introduction

The performance of a machine learning (ML) model is deeply tied to the choice of hyperparameters used for its training and deployment. As such, the problem of automated hyperparameter optimization (HPO) has become increasingly relevant as ML models have become more ubiquitous [1, 2, 3]. Given a function that describes a model's performance, $f(\boldsymbol{x})$[1], and a space of hyperparameters $\mathcal{X}$, the goal of HPO is to find $\boldsymbol{x}^* = \text{argmax}_{\{\boldsymbol{x}\in\mathcal{X}\}} f(\boldsymbol{x})$ in finite time (T) by making a sequence of decisions $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$ such that $\boldsymbol{x}^* \in \mathcal{S}$.

Popular approaches to HPO are based on Bayesian optimization (BO) [4, 5]. In BO, one builds a time-dependent surrogate model $\tilde{f}_t(\boldsymbol{x})$ using evidence collected from past evaluations of the true function $\{y_1 = f(\boldsymbol{x}_1), \ldots, y_{t-1} = f(\boldsymbol{x}_{t-1})\}$. Most methods proposed under this framework proceed as follows: at any timestep t, they suggest $\boldsymbol{x}_t \approx \text{argmax}_{\boldsymbol{x}\in\mathcal{X}} \alpha(\boldsymbol{x}; \tilde{f}_t)$ where α is called an acquisition function that measures the expected utility after one additional evaluation of f at $\boldsymbol{x}$ [5, 6, 7]. These methods are myopic. They make decisions exclusively based on the estimated function at the current timestep $\tilde{f}_t$ and can therefore be sub-optimal since they do not account for the impact of the current decision on future evaluations. Planning-based HPO attempts to avoid this by appropriately balancing exploration and exploitation during decision making. They attempt to maximize the long term reward

*Work done whilst interning at Deepmind. Correspondence to <ldery@andrew.cmu.edu> or <yutianc@deepmind.com>

[1]Here, performance captures the metric that we wish to optimize, such as accuracy, runtime, etc.

Foundation Models for Decision Making Workshop at Neural Information Processing Systems (NeurIPS 2022).

over a time horizon, h, by considering the influence of the choice of $\boldsymbol{x}_{t:t+h-1}$ on $\tilde{f}_{t+h}$ through multistep lookahead [8, 9, 10]. Such methods promise improved query efficiency – needing fewer configuration evaluations to achieve threshold performance – and improved performance of the model after all evaluations are considered when compared to myopic approaches. As recent ML models like pre-trained language and vision models become larger and thus both more expensive and time consuming to train [11, 12, 13], the above promises have become more enticing.

Despite these promises, planning-based HPO has not been widely adopted. This is because suggested approaches have been both technically complex to implement and computationally intensive to execute. In this paper, we circumvent both these challenges by leveraging recent advances in Transformer based [14], natural language interfaced HPO. Specifically, Chen et al. [2] recently proposed an approach dubbed OptFormer which presents a unified, natural language interface for hyperparameter optimization and thus allows transfer learning between HPO problems. Relevant to us is the structure of the OptFormer model: it jointly performs both hyperparameter suggestion $\boldsymbol{x}_t$ and function prediction $\tilde{f}_t(\boldsymbol{x}_t)$ as autoregressive generation.

By leveraging the ease and speed of autoregressive generation with OptFormer, we present a novel planning-based HPO algorithm that is simple and scalable; thus bypassing the typical challenges of traditional methods. We incorporate multi-step lookahead into both candidate generation (instead of doing a global search as typical of most HPO approaches) and candidate evaluation (we provide a planning-based acquisition function α) since performing rollouts with OptFormer is simply multistep generation for h timesteps. In Section 4, we present our method in detail and in Section 6 we empirically validate our approach under two settings: the BBOB dataset [15], which is a synthetic benchmark, and the RealWorldData dataset [2], which is constructed from aggregating actual hyperparameter tuning experiments. Our results highlight that incorporating planning into transformer-based hyperparameter optimization models is a promising research direction and we invite the community to explore this setting further.

2 Related Work

As already discussed, most approaches to hyperparameter optimization leverage Bayesian optimization (BO). These methods typically build a Gaussian process (GP) model [5, 16] as the surrogate $\tilde{f}_t(\boldsymbol{x})$ and proceed to suggest $\boldsymbol{x}_t \approx \operatorname{argmax}_{\boldsymbol{x}\in\mathcal{X}} \alpha(\boldsymbol{x}; \tilde{f}_t)$ where α is an acquisition function. An acquisition function measures the expected utility after one additional evaluation of f at $\boldsymbol{x}$. Past work has explored a variety of acquisition functions such as probability of improvement (PI) [5], expected improvement (EI) [17, 18] and upper confidence bound (UCB) [19, 20]. These acquisition functions are myopic since they depend solely on the surrogate at the current timestep and so do not directly account for the impact of a greedy choice on downstream evaluations.

Unlike myopic HPO methods, planning based approaches fundamentally require building models of the future to assess the impact of a current decision on later timesteps. Though these methods also rely on a GP as a surrogate model, each point in multi-step planning involves *fantasizing/imagining* an updated GP posterior $\left(\tilde{f}_{t+1} \mid_{\tilde{\boldsymbol{x}}_t}\right), \ldots, \left(\tilde{f}_{t+h} \mid_{\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_{t+1}, \ldots, \tilde{\boldsymbol{x}}_{t+h-1}}\right)$ based on simulated choices from lookaheads $\{(\tilde{\boldsymbol{x}}_t, \tilde{y}_t), \ldots, (\tilde{\boldsymbol{x}}_{t+h-1}, \tilde{y}_{t+h-1})\}$ [21, 22]. Note that we use $\tilde{\boldsymbol{x}}_t$ to represent a fantasized decision, while $\boldsymbol{x}_t$ is the actual choice made at timestep t. Whilst multi-step planning is promising, constructing the posterior of a GP model requires matrix inversion which is a compute-intensive operation [23]. Even outside of this limitation, traditional planning based approaches are compute intensive due to (i) poor scaling behavior of the search tree—$\mathcal{O}(q^h)$ where q is the number of choices at each decision point for each lookahead step [9, 21]—which forces most methods to explore short horizons, typically $h \in \{1, 2\}$, and (ii) nested expectation and maximization: marginalizing future observation $\tilde{y}_{t+j}, j < h$ and global search on the acquisition function to obtain query $\tilde{\boldsymbol{x}}_{t+j}$ at every lookahead step. Another issue is complexity of implementation. In an attempt to reduce the computational burdens described above, complex approaches have been suggested that require significant expertise to execute properly. For example Jiang et al. [22] attempt to overcome the problem of nested expectation and maximization by leveraging the reparameterization trick [24] to couple sampling and optimization. They build a differentiable objective over a multi-step tree and then optimize over this objective to get a proposal for the next suggestion $\boldsymbol{x}_t$; a far from simple method. Rollout policy [25] provides a tractable alternative to approximate the nested expectation by

playing multiple rollouts using a base policy but maximization in the search space is still required at every step in the base policy.

Recently Chen et al. [2] proposed a transformer-based [14] model called OptFormer, which presents a unified, natural language interface for HPO and thus allows transfer learning between HPO problems. OptFormer jointly performs both hyperparameter suggestion $\tilde{\boldsymbol{x}}_t$ and function prediction $\tilde{f}_t(\tilde{\boldsymbol{x}}_t)$ as autoregressive generation. This means that the OptFormer model does not suffer from significant computational expense during updates in response to fantasized decisions. Also, since the OptFormer model has been trained to mimic base policies like GP-UCB, autoregressive hyperparameter generation serves as a fast and strong approximation of the inner maximization step discussed in the preceding paragraph. The model is also simple to use and, due to the widespread popularity of transformer models [26, 27], is accessible to a wide audience of ranging technical expertise. The absence of the computational and technical constraints typical of non-myopic HPO means the OptFormer model is uniquely positioned as a fertile test-bed for exploring practical planning based hyperparameter optimization.

3 Background on OptFormer

OptFormer is a transformer-based hyperparameter optimization method proposed in Chen et al. [2]. It converts historical hyperparameter tuning experiments, each consisting of a task description, m, and optimization trajectory, $\boldsymbol{h}_t = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_t, y_t))$, into a large dataset of over 700K text sequences, and then trains a transformer to learn (1) $P(\boldsymbol{h}_t|m)$, the prior distribution of the data generative process; (2) $P(\boldsymbol{x}_t|m, \boldsymbol{h}_{t-1})$, the prior conditional distribution, which corresponds to the distribution of the next query made by the base HPO algorithms (eg. GP-UCB) that generated the tuning trajectory data (equivalently, the behavior policy in reinforcement learning); and (3) $P(y_t|m, \boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$, the predicted distribution of the unknown function value f at input $\boldsymbol{x}_t$ given observations $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{t-1}, y_{t-1})\}$—the surrogate model $\tilde{f}_t(\boldsymbol{x})$ required by traditional BO methods. Note that all OptFormer distributions are modelled autoregressively. Thus, generating $\boldsymbol{x}_t$ is achieved by simply prompting OptFormer with a text trajectory $(m; \boldsymbol{h}_{t-1})$ to produce the distribution $P(\boldsymbol{x}_t|m, \boldsymbol{h}_{t-1})$ at the model's output and then sampling from it. Once $\boldsymbol{x}_t$ is generated, it is appended to the trajectory $(m; \boldsymbol{h}_{t-1}; \boldsymbol{x}_t)$ which we autoregressively feed back into OptFormer to obtain $P(y_t|m, \boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$.

OptFormer proposes two policies for hyperparameter optimization. First, a **prior policy** that samples hyperparameters $\boldsymbol{x}_t$ autoregressively from the prior conditional distribution directly. Second, a two-stage **augmented policy** as follows:

1. **Candidate generation**: proposes a set of n hyperparameter candidates from the prior policy. The OptFormer model performs autoregressive generation conditioned on the evidence and problem meta-data m, producing a set of n possible candidates $C_t = \{\tilde{\boldsymbol{x}}_t^{(i)} \mid \tilde{\boldsymbol{x}}_t^{(i)} \sim P(\boldsymbol{x}_t|m, \boldsymbol{h}_{t-1})\}_{i \in [n]}$. This step is absent from traditional BO approaches. Since OptFormer has been trained to mimic prior policies, C_t is typically a high quality set of suggestions.
2. **Candidate evaluation**: computes a 1-step acquisition function $\alpha(\boldsymbol{x}_t^{(i)}; \tilde{f}_t(\boldsymbol{x}))$ such as UCB or expected improvement (EI) based on its own surrogate model $\tilde{f}_t(\boldsymbol{x}) = P(y_t|m, \boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$. OptFormer suggests the candidate $\boldsymbol{x}_t = \mathrm{argmax}_{\boldsymbol{x} \in C_t} \alpha(\boldsymbol{x}, \tilde{f}_t(\boldsymbol{x}))$.

Whilst the prior policy just mimics the base HPO algorithm, the augmented policy constitutes a 1-step policy improvement over the base algorithm and empirically yields better performance compared to the prior policy [2] (Figure 6).

The OptFormer procedure described above uses a myopic acquisition function and does not leverage simulations to gauge the effects of the current choice on future evaluations. The improvements achieved by the OptFormer augmented policy over its base prior engender hope that further performance gains can be realised by incorporating planning.

4 Planning-based HPO with OptFormer

In this work, we develop approaches that are able to plan over a given horizon h. Below, we describe the modifications we make to the two-stage OptFormer process to perform multi-step planning. For

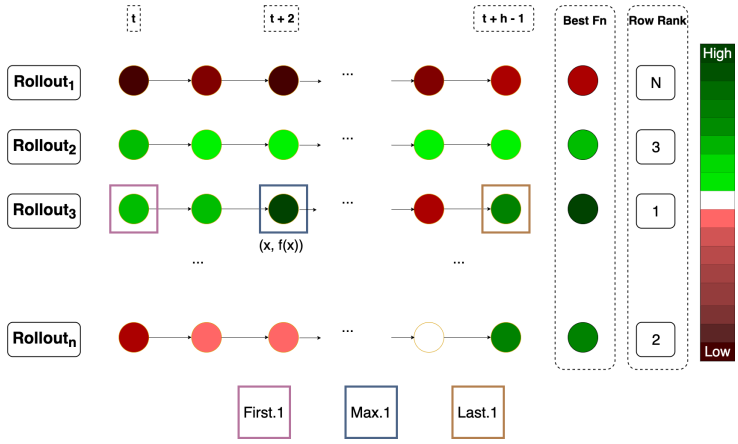


Figure 1: We refine C_t by performing roll outs with each candidate. We rank trajectories according to the highest sampled $\tilde{y}$ within the trajectory – this gives us the trajectory's row rank. Given the trajectories generated, we propose 3 criteria for refining C_t. (1) First.k picks k of the seed hyperparameters in C_t that have the highest row rank. (2) Max.k picks the top k $\tilde{\boldsymbol{x}}$ that led to the highest $\tilde{y}$ within the top k row ranked trajectories. (3) Last.k picks the top k members of column $t + h - 1$, sorted according to row rank

notational clarity, we use $\tilde{\boldsymbol{b}}_t$ to represent a fantasized/imagined quantity at timestep t and $\boldsymbol{b}_t$ for the actual observed or decided quantity.

4.1 Candidate Generation

As mentioned above, the set C_t from the OptFormer prior policy is a strong baseline of potential suggestions. However, we can further leverage our ability to autoregressively prompt the OptFormer model to generate a new horizon-aware set C_t^h, that refines C_t by fantasizing future time-steps. Specifically, given $\tilde{\boldsymbol{x}}_t \in C_t$, we can roll out the prior policy to generate *a single* length h sequence of fantasized hyperparameters and function values $\tilde{\boldsymbol{h}}(\tilde{\boldsymbol{x}}_t) = \{(\tilde{\boldsymbol{x}}_t, \tilde{y}_t), \ldots, (\tilde{\boldsymbol{x}}_{t+h-1}, \tilde{y}_{t+h-1})\}$ by repeatedly sampling from the OptFormer and then autoregressively prompting it with the sampled values:

$$\tilde{\boldsymbol{x}}_{t+j} \sim P(\boldsymbol{x}_{t+j} \mid m, \boldsymbol{h}_{t-1}, \tilde{\boldsymbol{x}}_t, \ldots, \tilde{y}_{t+j-1}) \quad \text{where } j \in [1, \ldots h]$$
$$\tilde{y}_{t+j} \sim P(y_{t+j} \mid m, \boldsymbol{h}_{t-1}, \tilde{\boldsymbol{x}}_t, \ldots, \tilde{y}_{t+j-1}, \tilde{\boldsymbol{x}}_{t+j}) \quad \text{where } j \in [1, \ldots h]$$

Given these sequences (one for each member of C_t), we can select $k \leq n$ rollout-aware hyperparameter candidates. Consider a rollout trajectory $\tilde{\boldsymbol{h}}(\tilde{\boldsymbol{x}}_t^i)$ seeded by some $\tilde{\boldsymbol{x}}_t^i \in C_t$. Let $\mathcal{Y}(\tilde{\boldsymbol{h}})$ be the set of sampled function values $\tilde{y}$ that appear in the trajectory and $\mathcal{X}(\tilde{\boldsymbol{h}})$ be the corresponding set of sampled hyperparameter suggestions. We can rank each rollout based on the highest fantasized function value $\tilde{y}^*(\tilde{\boldsymbol{x}}_t^i) = \max\left(\mathcal{Y}(\tilde{\boldsymbol{h}}(\tilde{\boldsymbol{x}}_t^i))\right)$ to obtain the row rank of the rollout (See Figure 1). We can then define the following set of rollout trajectories based on which we will extract C_t^h:

$$\tilde{\mathbf{H}}(k) = \text{argtopk}_{\{\tilde{\boldsymbol{h}}(\tilde{\boldsymbol{x}}_t^i)\}} \; \tilde{y}^*(\tilde{\boldsymbol{x}}_t^i)$$

Interpreted in terms of Figure 1, this corresponds to taking the top k trajectories as defined by row rank (and we define the row-rank by comparing the best fantasized $\tilde{y}$ achieved by each rollout). Having constructed $\tilde{\mathbf{H}}$, we suggest three different criteria as follows:

1. **First.k:** $C_t^{h,F} = \{\tilde{\boldsymbol{x}}_t^i \in C_t \mid \tilde{\boldsymbol{h}}(\tilde{\boldsymbol{x}}_t^i) \in \tilde{\mathbf{H}}\}$.

Intuitively, this approach selects the top k entries *in the original set* C_t that led to the highest fantasized function value $\tilde{y}$ during rollouts. We posit that these candidates may be the most promising to explore further within C_t. Figure 1 is a pictorial rendering of the approach.

2. **Max.k:** $C_t^{h,M} = \{\tilde{\boldsymbol{x}}^*(\tilde{\boldsymbol{x}}_t^i) \in \mathcal{X}(\tilde{\boldsymbol{h}}_t(\tilde{\boldsymbol{x}}_t^i) \mid \tilde{\boldsymbol{h}}(\tilde{\boldsymbol{x}}_t^i) \in \tilde{\mathbf{H}}\}$

Here $\tilde{\boldsymbol{x}}^*(\tilde{\boldsymbol{x}}_t^i)$ is the hyperparameter suggestion that directly led to $\tilde{y}^*(\tilde{\boldsymbol{h}})$ within its trajectory. The Max.k approach selects the hyperparameter suggestion that led to the highest fantasized function value $\tilde{y}$ within each of the highest ranked trajectories $\tilde{\mathbf{H}}$ (See Figure 1).

3. **Last.k:** $C_t^{h,L} = \{\tilde{\boldsymbol{x}}_{t+h-1}(\tilde{\boldsymbol{x}}_t^i) \in \mathcal{X}(\tilde{\boldsymbol{h}}_t(\tilde{\boldsymbol{x}}_t^i) \mid \tilde{\boldsymbol{h}}(\tilde{\boldsymbol{x}}_t^i) \in \tilde{\mathbf{H}}\}$
$C_t^{h,L}$ contains the last sampled $\tilde{\boldsymbol{x}}_{t+h-1}$ during rollouts within the highest ranked trajectories $\tilde{\mathbf{H}}$. This criterion is motivated by the fact that OptFormer was trained to mimic a base prior policy. Since the prior policy used to construct OptFormer training data improves over time (due to access to more observations), we posit that hyperparameter suggestions made by OptFormer from later rollout timesteps are more promising candidates.

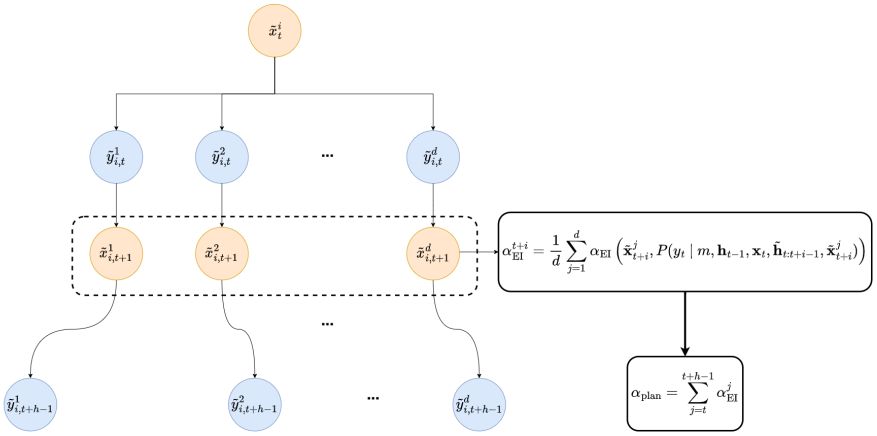


Figure 2: An example instantiation of our planning-based acquistion function. For each candidate $\tilde{\boldsymbol{x}}_t^i \in C_t^h$, we perform d roll outs of length h each. At each timstep during rollouts, we compute the expected improvement (EI) and average over all trajectories. The final acquisition function is the sum of the averaged EI over the full trajectory horizon h.

4.2 Candidate Evaluation

Having constructed a refined set C_t^h, it remains to decide which member of the set to suggest. Here, we investigate two acquisition functions:

1. Expected improvement (EI): We suggest $\boldsymbol{x} = \text{argmax}_{\tilde{\boldsymbol{x}}_t \in C_t^h} \alpha_{\text{EI}}(\tilde{\boldsymbol{x}}_t, P(y_t \mid m, \boldsymbol{h}_{t-1}, \tilde{\boldsymbol{x}}_t))$ where $\alpha_{\text{EI}} = \mathbb{E}_{\tilde{y} \sim P(y_t \mid m, \boldsymbol{h}_{t-1}, \tilde{\boldsymbol{x}}_t)}\left[\max(\tilde{y} - y_{t-1}^*, 0)\right]$ is the expected improvement acquisition function and $y_{t-1}^* = \max\left(\mathcal{Y}(\boldsymbol{h}_{t-1})\right)$ is the highest observed return so far. $\mathcal{Y}(\boldsymbol{h}_{t-1})$ is as defined in Section 4.1. We can easily compute the expectation above given that the transformer outputs a distribution over possible values of $\tilde{y}_t$. Note that EI on its own is myopic, we use $C_t^{h,*}$ as our conduit for incorporating planning information.

2. Planning based EI: For each candidate $\tilde{\boldsymbol{x}}_t$ we perform d rollouts each for h timesteps. At each timestep, we can compute α_{EI} based on the simulated trajectory so far and average across rollouts.

$$\alpha_{\text{Plan}}(\tilde{\boldsymbol{x}}_t) = \sum_{i=0}^{h-1} \frac{1}{d} \sum_{j}^{d} \alpha_{\text{EI}}\left(\tilde{\boldsymbol{x}}_{t+i}^j, P(y_t \mid m, \boldsymbol{h}_{t-1}, \tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{h}}_t^i, \tilde{\boldsymbol{x}}_{t+i}^j)\right)$$

   We then suggest $\boldsymbol{x} = \text{argmax}_{\tilde{\boldsymbol{x}}_t \in C_t^h} \alpha_{\text{Plan}}(\tilde{\boldsymbol{x}}_t)$ as the next hyperparameter to evaluate. Figure 2 demonstrates the details of computing α_{Plan}. By performing d rollouts, we can average α_{EI} at every rollout timestep and thus get a lower variance estimate. Also note that whilst α_{Plan} generates $\mathcal{O}(dh)$ trajectories per candidate, classic tree based multistep planning HPO methods would require $\mathcal{O}(d^h)$.

In this section, we have presented augmentations to the OptFormer model that enable it to perform multistep planning. Next, we will setup an experimental framework within which we will seek to validate our proposed changes.

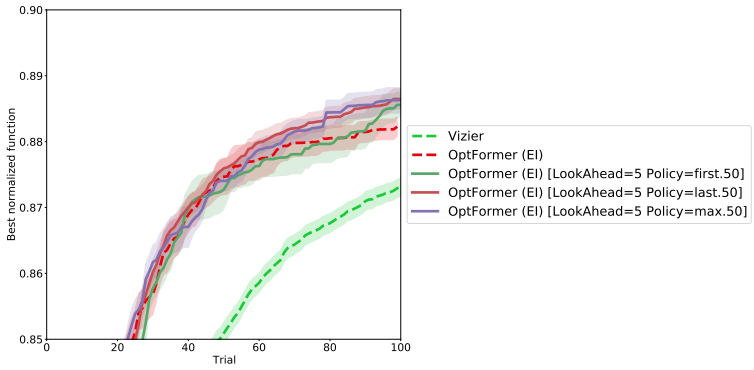


Figure 3: We evaluate augmenting OptFormer (EI) with our candidate generation strategies from Section 4.1. We use a lookahead horizon $h = 5$ which we find to be a reasonable length from ablations. We observe the following performance ranking towards the end of the trial : Max.50 $\approx$ Last.50 $>$ First.50 $\approx$ OptFormer (EI) $>$ Vizier

5 Experimental Setting

Datasets: OptFormer leverages past HPO trajectories for supervised training of the transformer-based model. Chen et al. [2] introduce the RealWorldData dataset, based on the Google Vizier [28] database, which they use to train OptFormer. The RealWorldData has 750K trajectories—each with 300 trials on average —and covers many HPO tuning experiments run at Google across domains like vision, speech, NLP, and robotics. We will use this dataset along with the blackbox optimization benchmark (BBOB) dataset [15], which consists of 24 types of synthetic functions with customizable properties. For evaluation, we use the same *test* set of functions used by Chen et al. [2] based on both datasets.
Model and Training Details: We use models provided by Chen et al. [2] which were pre-trained on the RealWorldData and BBOB datasets respectively. For specific details about the hyperparameters used to train these models and the model architecture, please see Appendix D.2 of Chen et al. [2]. In our experiments, we generate an initial seed set C_t of 100 candidates that we sample from the OptFormer prior policy. We then use this set to obtain our rollout-generated horizon-aware candidate set, C_t^h. We fix $k = |C_t^h| = 50$ and experiment with rollout horizons $h = \{2, 3, 5, 10\}$. For the BBOB dataset, we generate plots based on averaging 100 random functions per function type whilst for RealWorldData we average over 5 runs per test task. All plots are displayed with confidence intervals highlighted.
Baselines: We compare our approaches to two baselines.

1. Vizier: The Google Vizier Service [28] implements a GP-UCB (Gaussian process surrogate with upper confidence bound acquisition function) and is a baseline used in [2]. We choose this because it was the strongest performing algorithm amongst non-OptFormer models in Chen et al. [2].
2. OptFormer (EI): This is the base OptFormer policy augmented with the expected improvement acquisition function. This is the best policy found by Chen et al. [2] on the two datasets we consider.

6 Results and Discussion

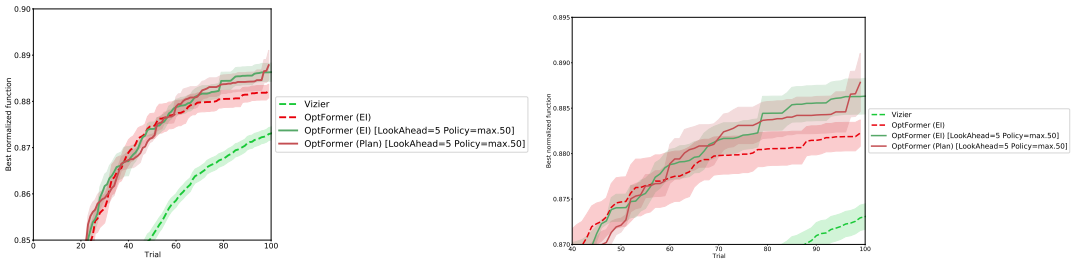
In this section, we demonstrate that the changes that we have made on top of the OptFormer model lead to modest improvements in downstream performance.

6.1 Rollout-aware candidate generation improves upon OptFormer (EI)

Figure 3 shows how refining $C_t \to C_t^h$ impacts the performance of our baseline model OptFormer (EI) on the RealWorldData dataset. As can be seen, we get modest improvements from using Max.50 and Last.50. Our First.50 however mostly mimics the baseline OptFormer (EI) policy and does not show any gains until (arguably) the last few trials.

We posit that candidate generation leads to improvement in performance via the following mechanisms:

- Improved diversity of candidates: Members of C_t and $C_t^{h,F}$ are generated from the same distribution $P(\boldsymbol{x}_t|m, \boldsymbol{h}_{t-1})$. However for $C_t^{h,M}$ and $C_t^{h,L}$, we get higher diversity of samples since candidates are generated from different prompting sequences obtained from a diverse set of rollouts. More diverse candidates means we can explore more of the search space and thus increase our chances of finding a good hyperparameter configuration. Note that the First.k criterion produces a subset of C_t and so does not increase the diversity of candidates as compared to the two other criteria. This could explain its failure to improve upon the baseline.
- Improved quality of candidates: As discussed in Section 4.1, taking the $\mathrm{argtopk}$ based on fantasized $\tilde{y}$ amongst rollout can be seen as a policy improvement step thus resulting in a better set of candidates. This improvement is only possible because OptFormer was trained to mimic a strong prior policy and its function approximation $P(y_t|m, \boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$ is well calibrated [2].



(a) Full Plot (b) Zoomed In

Figure 4: We replace the EI acquisition function with our Planning-based acquisition function presented in 4.2. For each candidate, we generated $d = 64$ rollouts which we use to compute α_{Plan}. We do not observe any improvement from introducing this change.

6.2 Planning-based acquisition function does not improve over rollout-aware candidate generation

Encouraged by the gains from rollout-aware candidate generation, we introduce even more planning information during the candidate evaluation stage. We replace the expected improvement (EI) acquisition function (α_{EI}) with α_{Plan} defined in Section 4.2 but keep the Max.k candidate generation strategy. We compare these two approaches on the RealWorldData dataset in Figure 4.

From Figure 4, we do not see significant gain in changing $\alpha_{\mathrm{EI}} \rightarrow \alpha_{\mathrm{Plan}}$. One hypothesis for why this might be the case is that the set $C_t^{h,M}$ already contains significant planning information thus rendering the new acquisition function irrelevant for taking into account future evaluations. We leave a more thorough investigation to future work.

6.3 Long rollout horizon leads to compounding bias

In order to understand the impact of the rollout horizon on performance, we fix our policy to [OptFormer (EI) - Max.k] but vary $h \in \{2, 5, 10\}$. We find in Figure 5 that using a very long horizon degrades performance. Note that whilst $P(\tilde{y}_t|m, \boldsymbol{h}_{t-1}, \tilde{\boldsymbol{x}}_t)$ is primarily based on an actual, observed trajectory $\boldsymbol{h}_{t-1}$, sampling at timestep ℓ of a rollout $P(\tilde{\boldsymbol{x}}_{t+i}|m, \boldsymbol{h}_{t-1}, \tilde{\boldsymbol{h}}_t^\ell)$ requires a simulated trajectory $\tilde{\boldsymbol{h}}_t^\ell$. The longer $\tilde{\boldsymbol{h}}_t^\ell$ is, the more likely it deviates from the true $\boldsymbol{h}_t^\ell$ thus leading to inaccurate decisions. Our ablation suggests a horizon of 2-5 works best with the OptFormer model. Note that previous approaches to HPO have to restrict $h \in \{1, 2\}$ in order to keep compute feasible. Our horizon is not limited by computation but by compounding bias. For future work, we could train OptFormer not just to imitate the base policy but also to predict rolled-out values more accurately so as to combat the issue of long horizon bias.

6.4 Performance on BBOB dataset

We have demonstrated improvements to OptFormer on the RealWorldDataset. To validate the generality of our approach, we further test it on the BBOB dataset. Unfortunately we do not see

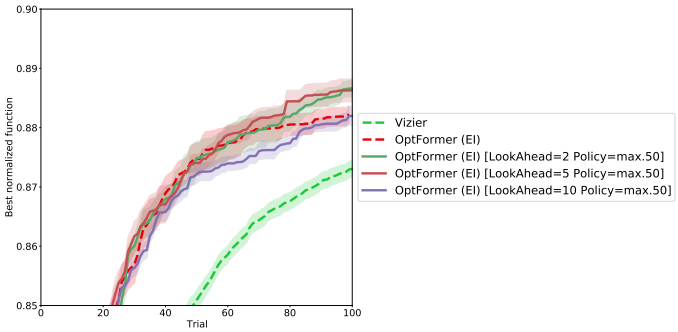


Figure 5: Ablation of h on the RealWorldData dattset. Comparing look ahead horizons at k=50 for different Max.k based policies. In general, looking ahead too far into the future is not beneficial and actually degrades performance. A look ahead horizon of 2-5 seems to produce the best performance.

much gain from our methods over OptFormer (EI) in this setting. As we conducted limited tuning of $h \in \{3, 5\}, k \in \{50\}$ in this setting we believe more extensive tuning of these configurations could yield better results.

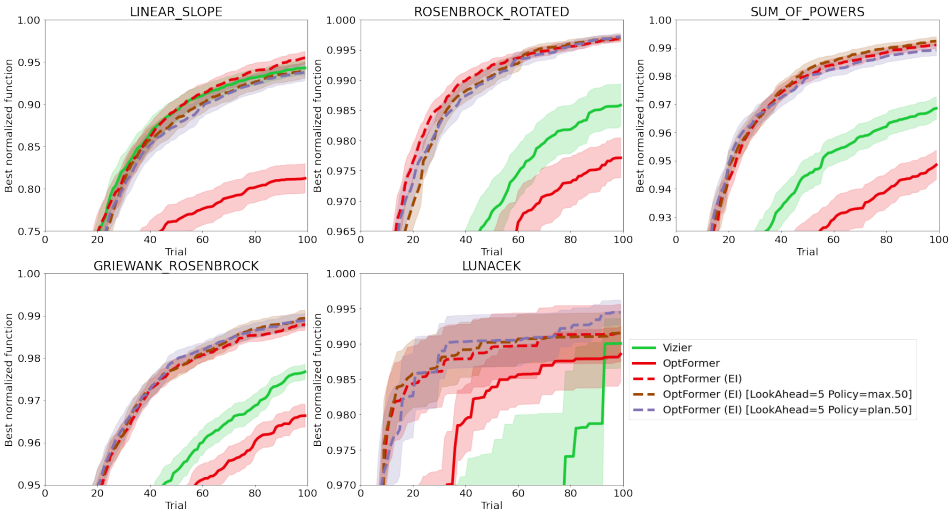


Figure 6: Performance on 5 test functions from the BBOB dataset. OptFormer is the unaugmented prior policy (we just sample $\boldsymbol{x}_1 = P(\tilde{\boldsymbol{x}}_t | m, \boldsymbol{h}_{t-1})$ which we play).

7 Conclusion

In this work, we have explored incorporating multistep planning into the recently proposed OptFormer HPO algorithm. We began by motivating why OptFormer is a suitable test-bed for exploring planning-based HPO and then presented different strategies to enable planning with OptFormer. Empirically, we show that our *planning-horizon-aware* candidate generation strategies can lead to improvements when coupled with the expected improvement acquisition function. However, we note that our specialized acquisition function α_{Plan} did not lead to desired levels of improvement over OptFormer. For future work, we hope to investigate the reason for this and use it to guide new proposals for planning-based acquisition functions.

8 Acknowledgements

We would like to thank Bobak Shahriari, Kazuya Kawakami, Caglar Gulcehre, Xingyou Song, Chansoo Lee, Zi Wang and Arnaud Doucet for helpful discussions at various points in producing this work.

References

[1] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020.

[2] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Qiuyi Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'aurelio Ranzato, et al. Towards learning universal hyperparameter optimizers with transformers. *arXiv preprint arXiv:2205.13320*, 2022.

[3] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

[4] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[5] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[6] Jian Wu and Peter Frazier. The parallel knowledge gradient method for batch bayesian optimization. *Advances in neural information processing systems*, 29, 2016.

[7] Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635. PMLR, 2017.

[8] Javier González, Michael Osborne, and Neil Lawrence. Glasses: Relieving the myopia of bayesian optimisation. In *Artificial Intelligence and Statistics*, pages 790–799. PMLR, 2016.

[9] Remi Lam and Karen Willcox. Lookahead bayesian optimization with inequality constraints. *Advances in Neural Information Processing Systems*, 30, 2017.

[10] Jian Wu and Peter Frazier. Practical two-step lookahead bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[12] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[13] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[15] Ouassim Elhara, Konstantinos Varelas, Duc Nguyen, Tea Tusar, Dimo Brockhoff, Nikolaus Hansen, and Anne Auger. Coco: the large scale black-box optimization benchmarking (bbob-largescale) test suite. *arXiv preprint arXiv:1903.06396*, 2019.

[16] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[17] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

[18] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[19] Rajeev Agrawal. Sample mean based index policies by o (log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995.

[20] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[21] Remi Lam, Karen Willcox, and David H Wolpert. Bayesian optimization with a finite budget: An approximate dynamic programming approach. *Advances in Neural Information Processing Systems*, 29, 2016.

[22] Shali Jiang, Daniel Jiang, Maximilian Balandat, Brian Karrer, Jacob Gardner, and Roman Garnett. Efficient nonmyopic bayesian optimization via one-shot multi-step trees. *Advances in Neural Information Processing Systems*, 33:18039–18049, 2020.

[23] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[24] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[25] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific, 2012.

[26] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *arXiv preprint arXiv:2106.04554*, 2021.

[27] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Computing Surveys (CSUR)*, 2021.

[28] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1487–1495, 2017.