BLISS: Auto-tuning Complex Applications using a Pool of Diverse Lightweight Learning Models

Authors: Rohan Basu Roy, Tirthak Patel, Vijay Gadepally, Devesh Tiwari
Presented by: Jonathan Lorray

Introduction

01

What is it all about?

What is Auto-Tuning

"Why do it yourself, when robots do it better?"

Hyperparameters (or knobs in the case of HPCs) are variables that control the overall behavior of a system and are defined before you even run it.

Auto-tuning is the systematic exploration of the hyperparameter space to find the best combination of hyperparameter values.

The goal of auto-tuning is to minimize the amount of manual trial and error required to find the best hyperparameter configuration.

Auto-Tuning: Context and Motivation

As parallel applications become more complex, auto-tuning becomes more desirable, challenging, and time-consuming

The Search space of tunable parameters for complex systems are often prohibitively large, and expensive to explore

Most state-of-the-art methods rely on variants of learning based strategies that produce suboptimal results:

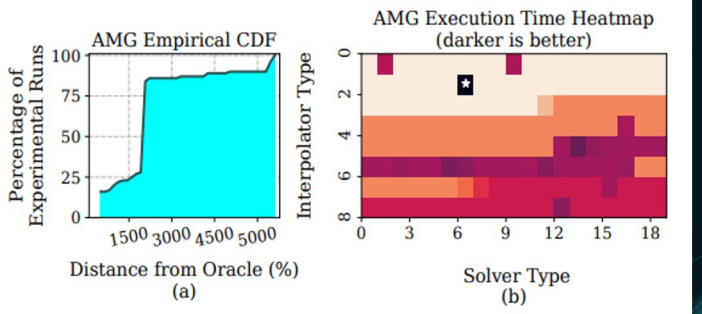
- Traditional learning strategies are prone to get stuck in local minima
- Most ML-based approaches need to build complex and powerful models
    - Require extensive training with a prohibitively large number of samples before they're useful
    - A single ml-model may not be able to effectively capture the diversity within and across an applications' search space of tunable parameters

Auto-tuning Challenges: Search Space

Challenge 1: Finding the optimal set of parameters requires exhaustively exploring a large multidimensional search space.

Exploring such search spaces can be prohibitively expensive due to having to run on real hardware. (see Fig. a)

Requires quick elimination of low interest areas to keep costs low, but you run the risk of missing the optimal configuration. (see Fig. b)



(a) The CDF of the execution times of 500 randomly selected AMG configurations shows that the execution times are far from the Oracle

(b) AMG's execution times vary even when only two parameters are varied (others are fixed)

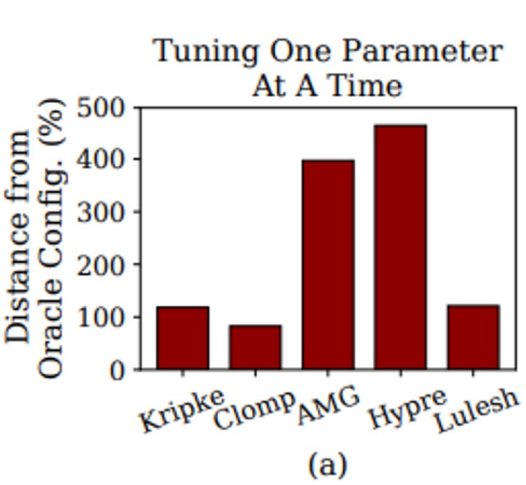
Note: Oracle refers to the best (lowest runtime) configuration available

Application	Description	# Parameters	Application Parameters (all parameters are discrete integers unless otherwise stated)	Hardware Parameters (same range/ values for all applications)	Size of Search Space (# Configs.)
Kripke [43]	Discrete ordinates S_n transport code	4 S/W, 3 H/W	# OpenMP Threads, Nesting Order, Group Set, Direction Set	Uncore Freq. (0.8 GHz-3 GHz), Core Freq. (0.8 GHz-2.2 GHz), Hyperthreading (On/ Off)	1, 112, 832
Clomp [11]	Modeling of a multi-physics problem	6 S/W, 3 H/W	# OpenMP Threads, Parts/Thread, Zones/Part, Zone Size, Flop Size	Uncore Freq., Core Freq., Hyperthreading	618, 240, 000
AMG [56]	3D Laplacian	5 S/W, 3 H/W	Solver Type (categorical), Smoother Type, Interpolator Type, Coarsening Type, Elements/Row	Uncore Freq., Core Freq., Hyperthreading	5, 873, 280
Hypre [27]	Convection-diffusion	5 S/W, 3 H/W	Solver Type (categorical), Smoother Type, Interpolator Type, Coarsening Type, Elements/Row	Uncore Freq., Core Freq., Hyperthreading	3, 297, 280
Lulesh [41]	Hydrodynamics modelling (version 2)	2 S/W, 3 H/W	Elements in a Mesh, Materials in a Region	Uncore Freq., Core Freq., Hyperthreading	49, 680

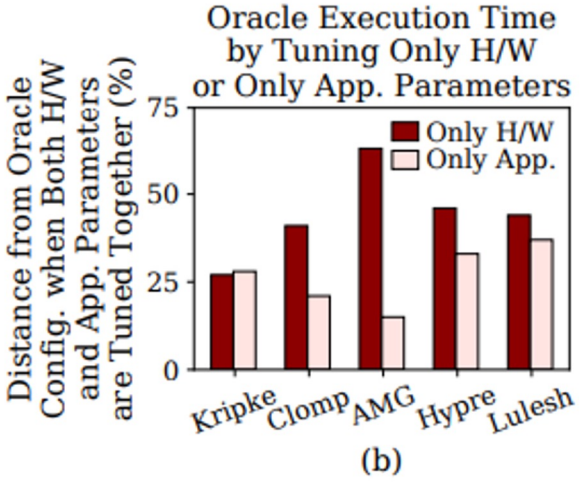
Number of tunable parallel application and hardware-level parameters, and the corresponding search space size for different parallel applications used in this study.

Auto-tuning Challenges: Naïve Solutions

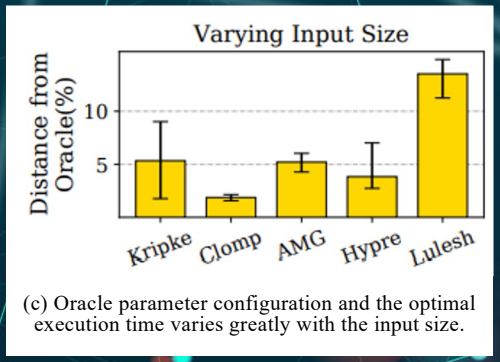
Challenge 2: Naïve auto-tuning solutions result far from optimal configurations, and capturing the interactions between application and hardware-level tuning knobs is the key to achieving the highest performance.



(a) Distance from the Oracle when the best configuration is reached by one-parameter-at-a-time tuning is high



(b) The Oracle by tuning only hardware (application params are fixed at mid range values) and only application params (hardware params are fixed at mid range values)



(c) Oracle parameter configuration and the optimal execution time varies greatly with the input size.

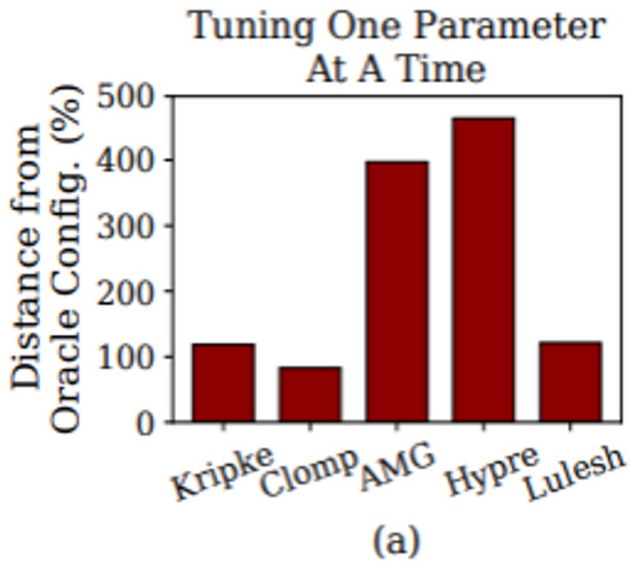
Note: Oracle refers to the best (lowest runtime) configuration available

Auto-tuning Challenges: Naïve Solutions

“One-Parameter-at-a-Time Tuning”

Appears promising as it drastically reduces the search space size

However, using this method results in configs 100% or more from oracle, meaning a slowdown of 2x (see Fig. a)



(a) Distance from the Oracle when the best configuration is reached by one-parameter-at-a-time tuning is high

Note: Oracle refers to the best (lowest runtime) configuration available

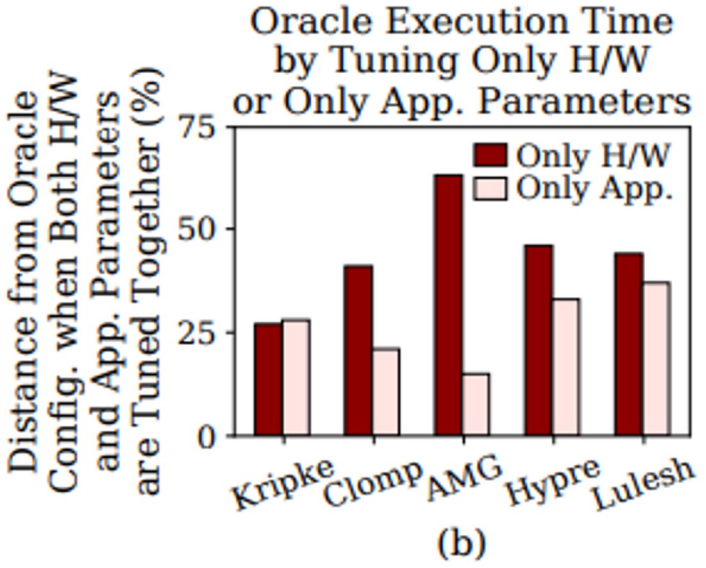
Auto-tuning Challenges: Naïve Solutions

Exclusive Hardware/Application Tuning

Expedite the optimal configuration search process by restricting the search space.

- For example, targeting only one type of knob (hardware level or application-level) to reduce the search space

Trying to tune the knobs in this way, consistently results in oracle configurations that are worse than when you try to tune both types jointly (see Fig. b)



(b) The Oracle by tuning only hardware (application params are fixed at mid range values) and only application params (hardware params are fixed at mid range values)

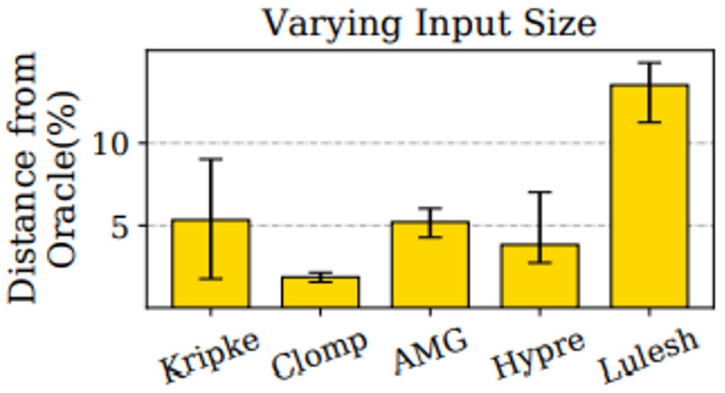
Note: Oracle refers to the best (lowest runtime) configuration available

Auto-tuning Challenges: Naïve Solutions

Straw-Man Solution

Auto-tune an application on a smaller problem size, and use the same solution (optimal values of parameters) when running the application with a larger input

Although it performs far better than previously mentioned Naïve methods, this method can result in configurations that deviate up to 15% from oracle. (see Fig. c)



(c) Oracle parameter configuration and the optimal execution time varies greatly with the input size.

Note: Oracle refers to the best (lowest runtime) configuration available

The Latest Solution

02

What the whole paper is about

BLISS Auto-tuner

BLISS is a novel solution for auto-tuning parallel applications, that explores the large configuration space efficiently using a Bayesian Optimization (BO) based approach.

Compared to most state-of-the-art methods, BLISS...

- Doesn't require prior knowledge of the application and its parameters
- Doesn't require prior knowledge of the instrumentation or hardware parameters
- Doesn't require prior knowledge specific to the domain

Algorithm 1 BLISS' multiple BO model based auto-tuner.

1: **Input:** Application to auto-tune A, search space $\mathcal{X}$, maximum number of iterations N_{iter}, N number of BO models with surrogate models M.
2: **Initialize:** Sampled Configurations $S_{\text{conf}} = \{\emptyset\}$, Observed Runtimes $T_{\text{obs}} = \{\emptyset\}$, Selection Probability Vector $P_{\text{select}} = [\frac{1}{M}]^{M\times 1}$.
3: **Start:** Get runtime T of a random sample x: $S_{\text{conf}} \Leftarrow S_{\text{conf}} \bigcup \{x\}$, $T_{\text{obs}} \Leftarrow T_{\text{obs}} \bigcup \{T\}$.
4: **for** $i = 0$ to N_{iter} **do**
5: Probabilistically select a BO model (BO_{select}) using the distribution P_{select}.
6: Update the surrogate model of BO_{select} with (S_{conf}, T_{obs}).
7: Update the acquisition function of BO_{select}.
8: Find the next configuration to sample x.
9: Run A with x and observe the execution time T.
10: $S_{\text{conf}} \Leftarrow S_{\text{conf}} \bigcup \{x\}$, $T_{\text{obs}} \Leftarrow T_{\text{obs}} \bigcup \{T\}$
11: Update P_{select} for the BO model with the best configuration (minimum execution time).
12: **end for**
13: **Output** Best configuration x^*

BLISS Auto-tuner

Leverages a **pool of diverse lightweight Bayesian Optimization models** to find the near-optimal parameter settings for auto-tuning complex applications

Builds a model that captures the shape and distribution of the underlying configuration search space by incrementally **“sampling”** different configurations, and refines this model over time to find the **optimal solution**

Does not rely on deep learning, transfer learning, or any pre-trained models, making it **application-independent**

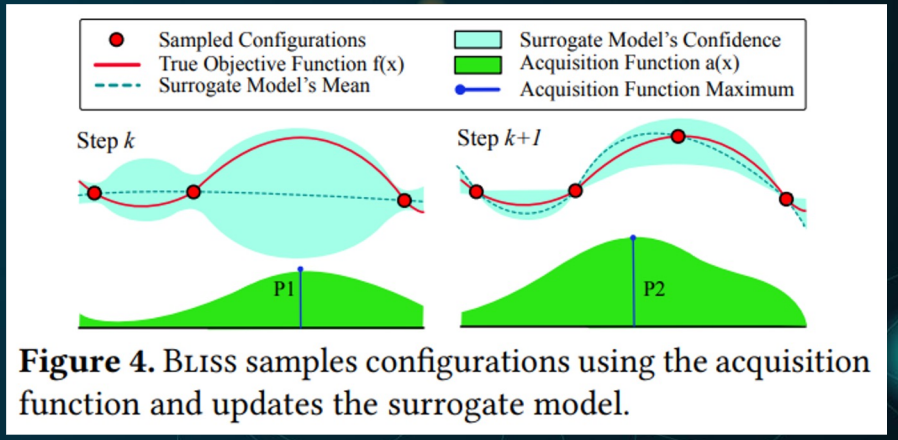


Figure 4. BLISS samples configurations using the acquisition function and updates the surrogate model.

Bliss Evaluation

BLISS compared to State-of-the-Art techniques:

- Oracle (offline optimal)
- OpenTuner
- Active Harmony (v4.6)
- GEIST

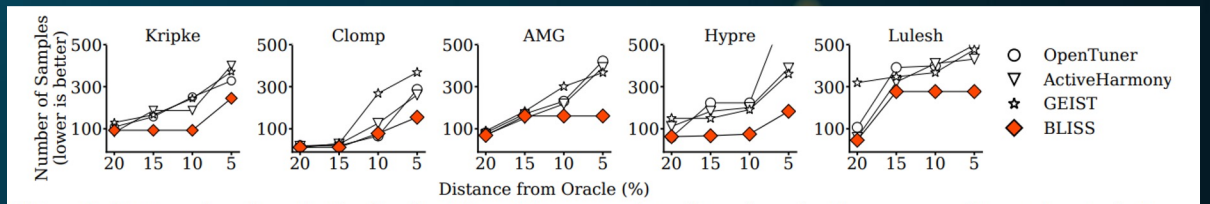


Figure 9. BLISS reaches closest to the Oracle with a minimum number of sample evaluations compared to previous techniques.

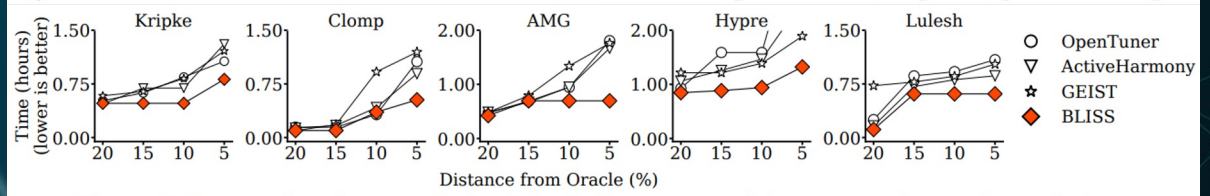


Figure 10. BLISS reaches closest to the Oracle in the least amount of time compared to previous techniques.

BLISS outperforms existing state-of-the-art techniques in:

- Fewer samples/configurations evaluated
- Less time needed to reach high-performing configuration (near-optimal performance)

Figures 9 and 10 illustrate the **number of samples** and the **amount of time needed** for BLISS to find configurations that are within 20%, 15%, 10%, and 5% of Oracle performance

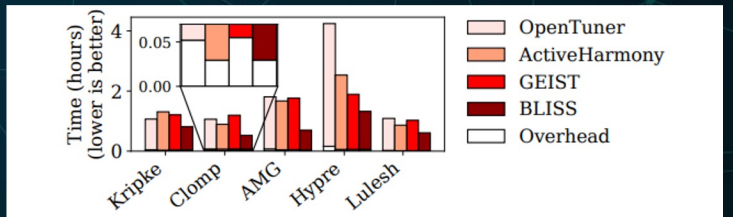


Figure 11. BLISS' overhead is similar to other competing techniques, and is included in the total auto-tuning time.

Bliss Evaluation

BLISS compared to Deep Neural Network based solutions:
Bliss strikes a balance between the quality of the solution and the training overhead

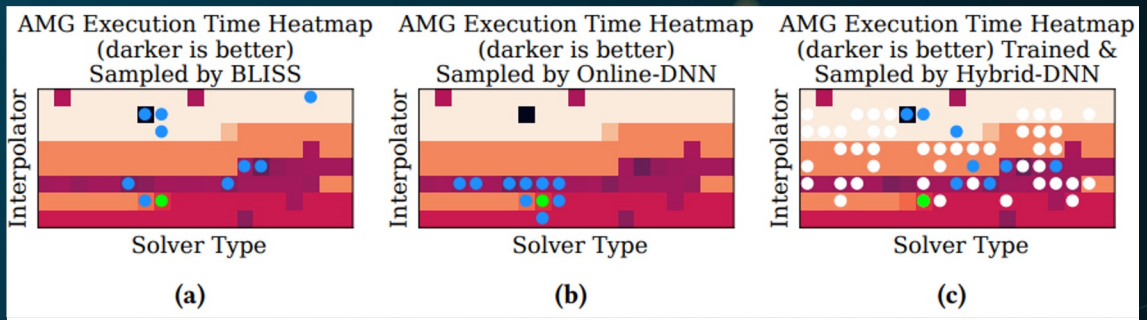
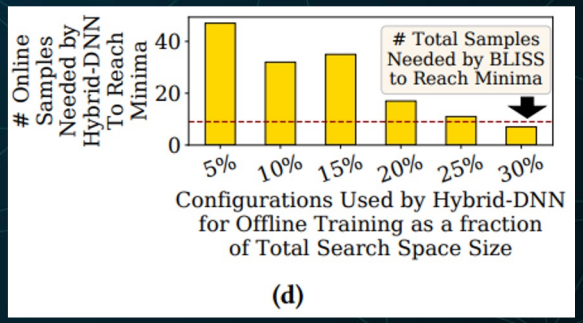


Figure 15. (a) - (b) BLISS reaches global minimum while Online-DNN gets stuck at a local minimum (blue dots denote samples, green dot denotes initial sampling configuration). (c) Hybrid-DNN trained with 30% configurations (denoted by white dots). (d) DNN outperforms BLISS only when pre-trained with 30% of the configurations.

Online-DNN: an online technique where the DNN model is built purely using online configuration samples without any pre-training (no offline overhead, similar to Bliss)
Hybrid-DNN: a predetermined portion of the configurations are sampled offline to initialize the DNN model and the model continues to be updated as online samples are collected (high offline overhead)



Bliss Evaluation

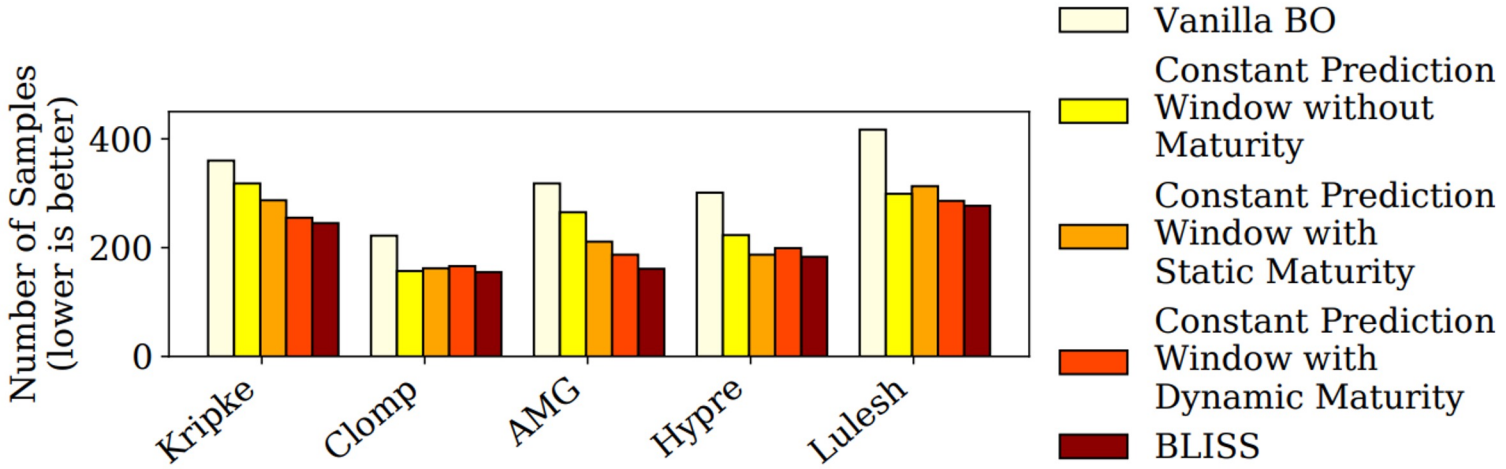


Figure 14. BLISS performs better than other non-enhanced variants of BO including vanilla BO.

Bayesian Optimization

03

The Backbone of BLISS

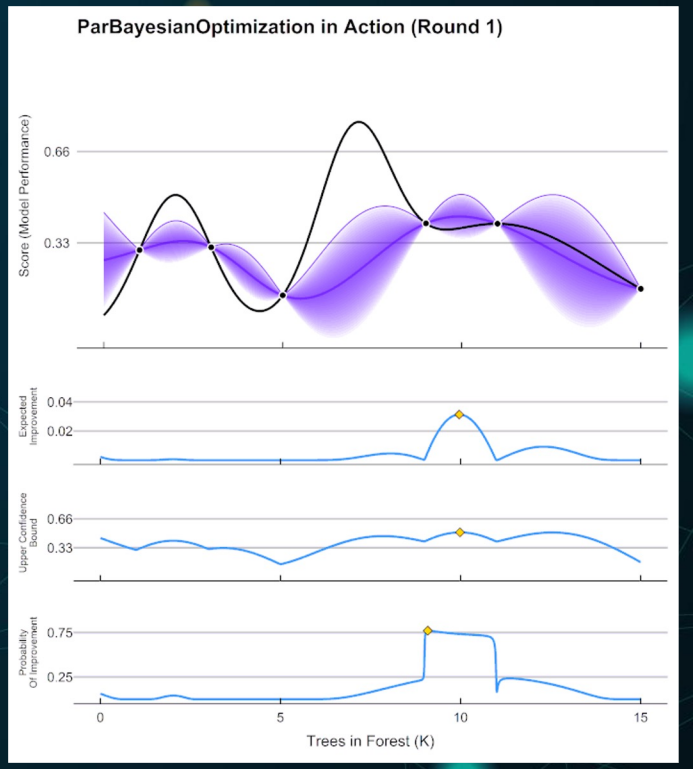
Bayesian Optimization (BO)

BO is a black-box optimization method for minimizing an unknown objective function, and is used in cases where it is costly to find the value of the objective function for all values in the search space.

Initially, BO has zero knowledge of the objective function and its relation to the input, but by intelligently querying the objective function for selected inputs, BO develops an approximation of the objective function over the entire search space.

Overall Steps

1. Define the **Objective Function**
2. Build/Choose a **Surrogate Model**
3. Build/Choose an **Acquisition Function**
4. Initialize the whole process and the **Iterative optimization loop**

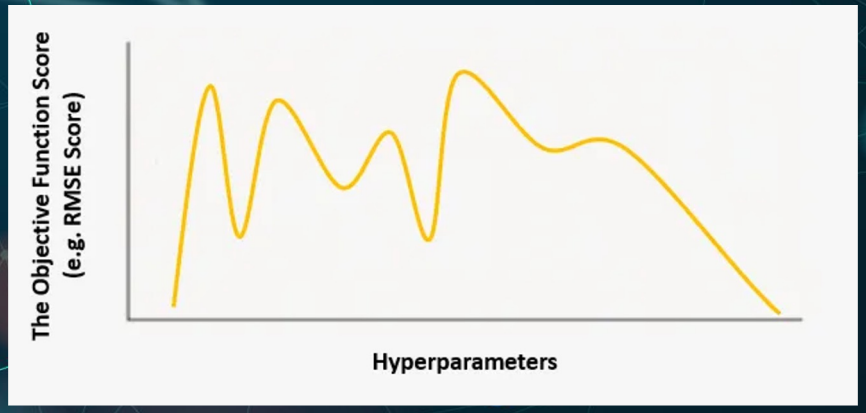


Bayes Optimizer Step 1: The Objective Function

First, determine the Objective Function you wish to optimize, even if its exact details are unknown.

The objective function should accept a set of inputs and return a scalar value that you aim to maximize or minimize.

In this scenario, the objective function represents the runtime, while the hyper parameter inputs consist of hardware and application settings.



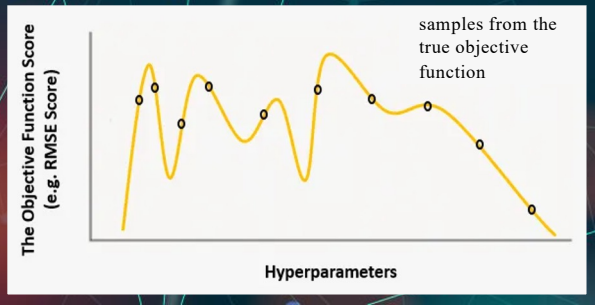
Bayes Optimizer Step 2: The Surrogate Model

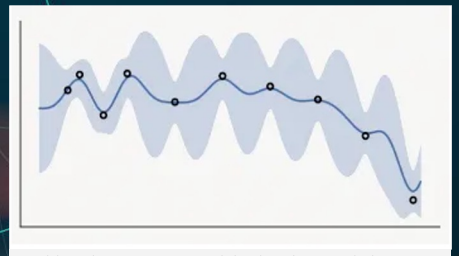
Bayesian Optimization employs a Surrogate Model to approximate the objective function.

A Surrogate Model is, by definition, "a probabilistic representation of the objective function," essentially training on (input, true objective function score) pairs.

The Gaussian Process (GP) is the most prevalent choice for a surrogate model, including in BLISS:

- It is a probabilistic model that offers a function distribution based on a set of observations.
- Capable of modeling intricate functions with uncertainty estimates, GPs are advantageous for balancing exploration and exploitation.





Initiate the Surrogate Model using the sampled objective function (input, output) pairs.

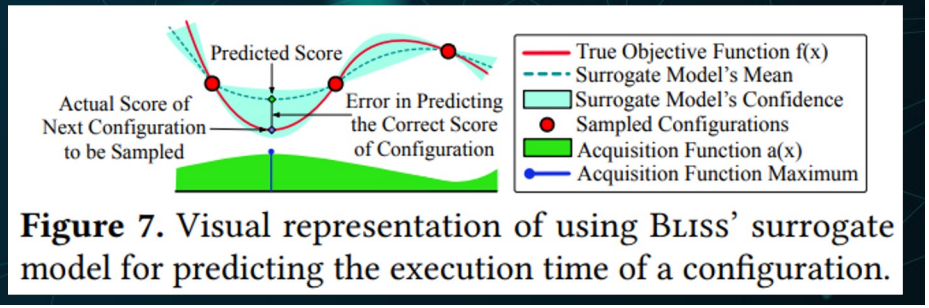


Figure 7. Visual representation of using BLISS' surrogate model for predicting the execution time of a configuration.

Bayes Optimizer Step 3: The Acquisition Model

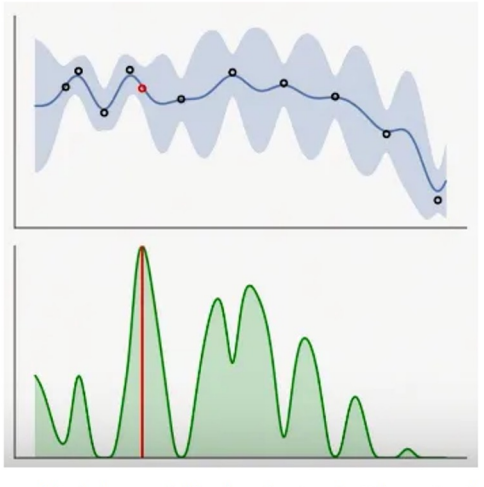
The Acquisition Function directs the search for optimal input values by assessing the anticipated value of each potential point when aiming to minimize the objective function.

The acquisition function strikes a balance between:

- Exploration (searching unexplored regions of the search space)
- Exploitation (focusing on regions with high predicted performance)

The optimal acquisition function depends on the search space's dynamics and how adjacent configurations interact.

For BLISS, the acquisition function is selected during runtime after assessing the search space.



Maximize acquisition function to select the next point

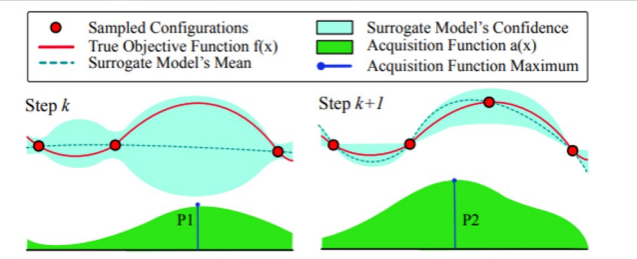


Figure 4. BLISS samples configurations using the acquisition function and updates the surrogate model.

Bayes Optimizer Step 4: The Optimization Loop

Once you have your Objective function, Surrogate Model, and Acquisition function, you initialize and perform the **Iterative Optimization Loop**:

1. Fit the **Surrogate Model** to the current set of input-output pairs
2. Optimize Pick the next input to evaluate the Objective Function with by finding the argmax of the **Acquisition Function**
3. Sample a true value of the **Objective Function** using the chosen input from the previous step.
4. Add the new input-output pair to the existing dataset and refit the **Surrogate Model**.
5. Check the **Stopping Criterion**, and Repeat steps 1-5 if the predefined stopping criterion is not satisfied

Algorithm 1 BLISS' multiple BO model based auto-tuner.

Input: Application to auto-tune A, search space $\mathcal{X}$, maximum number of iterations N_{iter}, N number of BO models with surrogate models M.
Initialize: Sampled Configurations $S_{\text{conf}} = \{\emptyset\}$, Observed Runtimes $T_{\text{obs}} = \{\emptyset\}$, Selection Probability Vector $P_{\text{select}} = [\frac{1}{M}]^{M\times 1}$.
Start: Get runtime T of a random sample x: $S_{\text{conf}} \Leftarrow S_{\text{conf}} \bigcup \{x\}$, $T_{\text{obs}} \Leftarrow T_{\text{obs}} \bigcup \{T\}$.
for $i = 0$ to N_{iter} **do**
  Probabilistically select a BO model (BO_{select}) using the distribution P_{select}.
  Update the surrogate model of BO_{select} with (S_{conf}, T_{obs}).
  Update the acquisition function of BO_{select}.
  Find the next configuration to sample x.
  Run A with x and observe the execution time T.
  $S_{\text{conf}} \Leftarrow S_{\text{conf}} \bigcup \{x\}$, $T_{\text{obs}} \Leftarrow T_{\text{obs}} \bigcup \{T\}$
  Update P_{select} for the BO model with the best configuration (minimum execution time).
end for
Output Best configuration x^*

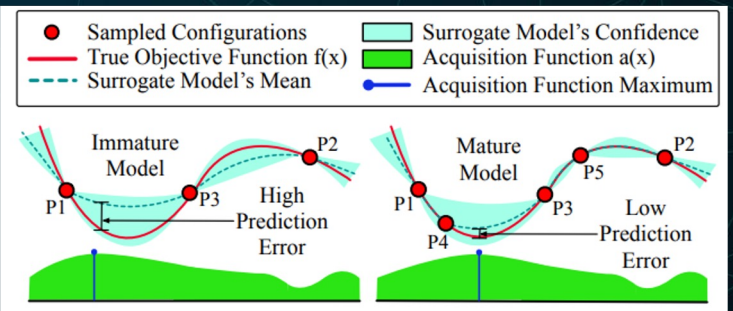


Figure 8. Prediction error decreases as more configurations are sampled and as the surrogate model matures.

Multi-Model Auto-tuning

04

What makes it special

BLISS: Multi-Model Auto-Tuning

BLISS implements a new auto-tuning technique that's based on **multiple** Bayesian Optimization (BO) models

Instantiates multiple BO models, with **varying** surrogate model kernels & acquisition functions

Iteratively samples new configurations, updates BO models

Goal: Ensure the **most suitable** BO model guides the auto-tuning process

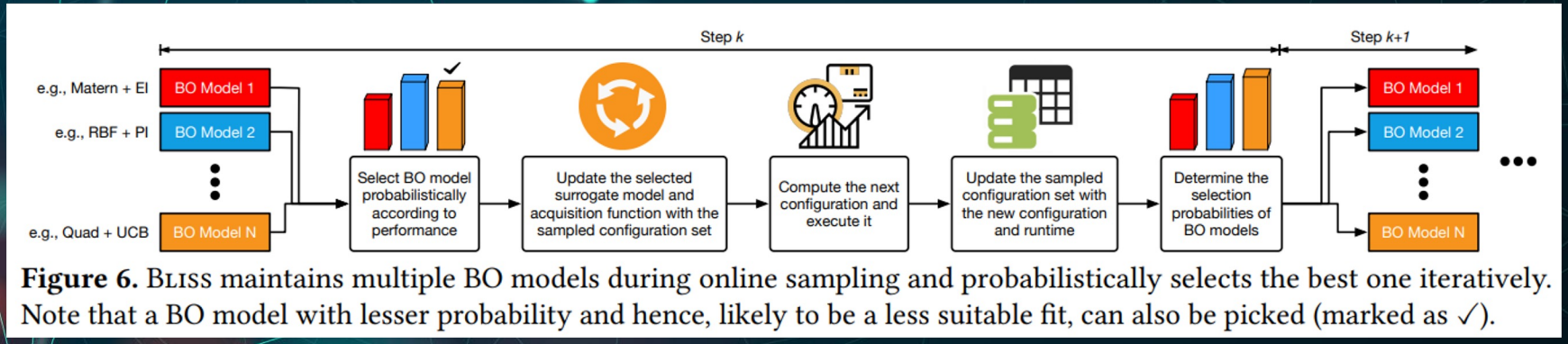
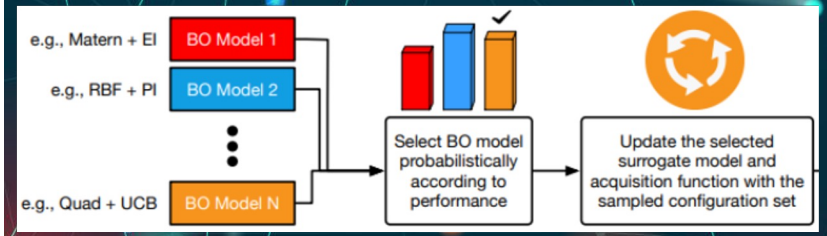


Figure 6. BLISS maintains multiple BO models during online sampling and probabilistically selects the best one iteratively. Note that a BO model with lesser probability and hence, likely to be a less suitable fit, can also be picked (marked as ✓).

BLISS: Multi-Model Auto-Tuning

At each iteration (configuration selection & evaluation):

- BLISS considers **all** existing BO models in the pool
- **Probabilistically** selects a BO model
  - The BO model corresponding to the best configuration so far simply has a higher chance of selection, and is **not** an ***automatic*** pick.
- The selected BO model has its surrogate model **refitted** to the sampled points



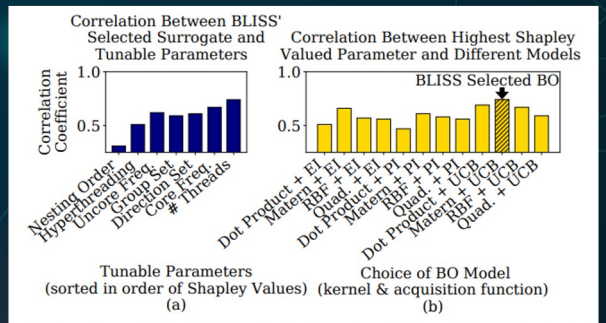


Figure 12. BLISS (a) identifies the most important parameter and (b) selects the best BO model.

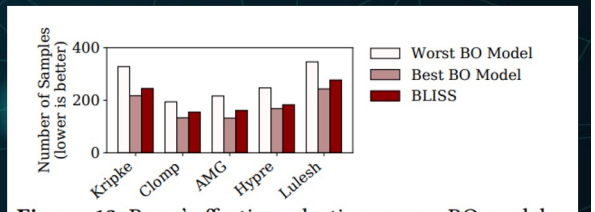


Figure 13. BLISS' effective selection among BO models.

BLISS: Multi-Model Auto-Tuning

At each iteration (configuration selection & evaluation):

- The Chosen BO model's **acquisition function** picks the next configuration to evaluate
- The next configuration is evaluated, and is added with its runtime to the sampled configuration set
- The probabilities of the BO models are updated
- Repeat

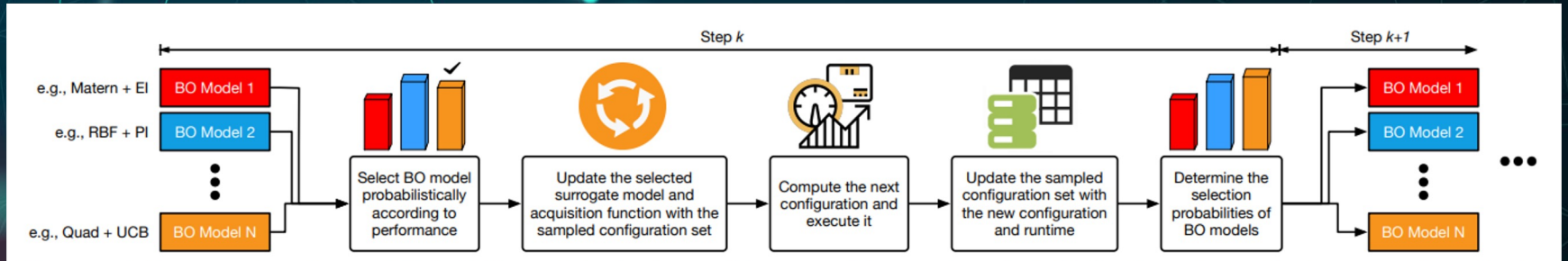


Figure 6. BLISS maintains multiple BO models during online sampling and probabilistically selects the best one iteratively. Note that a BO model with lesser probability and hence, likely to be a less suitable fit, can also be picked (marked as ✓).

Interesting and Positive Points

05

Random Interesting things

Predicting when to Skip Sampling

- Bliss skips sampling and instead uses the surrogate model's prediction as a **proxy** for runtime when the prediction of the surrogate model is **close enough** to the true objective function.
- Bliss skips sampling if the difference between the predicted performance metric and the true performance metric is less than a **certain threshold**.
- An inaccurate prediction can misguide the acquisition function. To address this, Bliss estimates the prediction inaccuracy for a **window** of previously sampled configurations.
- The prediction window is defined as the **number of consecutive configurations** for which samples can be skipped and replaced with the predicted value.

the length of the prediction window (l) is:

$$l = \begin{cases} \frac{\sum_{W_p} \left\lceil (1 - \frac{|T_{\text{pred}} - T_{\text{true}}|}{T_{\text{true}}}) * l_{max} \right\rceil}{W_p}, & \text{if } |T_{\text{pred}} - T_{\text{true}}| < T_{\text{true}} \\ 0, & \text{otherwise} \end{cases}$$

T_{pred}: Predicted values
T_{true}: True Runtime
W_p: Length of the Prediction history window
$L_{\max}$: Maximum number of samples BLISS can skip

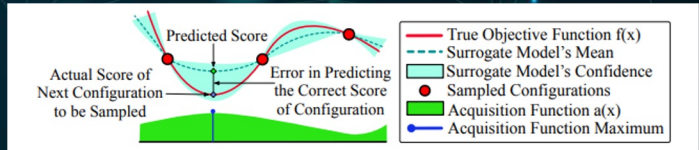


Figure 7. Visual representation of using Bliss' surrogate model for predicting the execution time of a configuration.

Allowing Surrogate Models to Mature

- A major flaw with skipping is that it relies on the surrogate model having low prediction error
- To address the flaw, BLISS iteratively updates surrogate models over time to increase their accuracy, and **intentionally delays** predicting the execution times of configurations until the surrogate functions are **mature enough** to make accurate predictions
- To **measure** the maturity of a surrogate model, Bliss checks the differences between the predicted values (T_{pred}) of the objective function for a certain number of configurations (W_m) and the true values (T_{true}) at those configurations

the number of samples by which Bliss should delay its prediction (d) is:

$$d = \begin{cases} \dfrac{\sum_{W_m} \left\lceil \left(\frac{|T_{pred} - T_{true}|}{T_{true}}\right) * d_{max} \right\rceil}{W_m}, & \text{if } |T_{pred} - T_{true}| < T_{true} \\ d_{max}, & \text{otherwise} \end{cases}$$

T_{pred}: Predicted values
T_{true}: True Runtime
W_m: Maturity history window
d_{max}: Number of samples

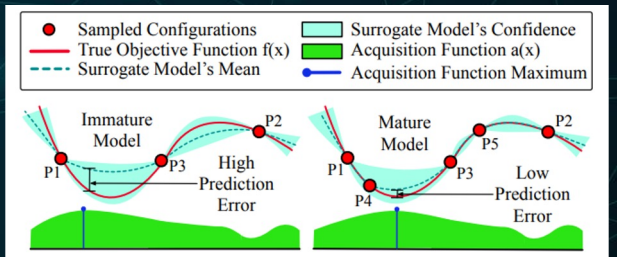


Figure 8. Prediction error decreases as more configurations are sampled and as the surrogate model matures.

Portability Aid

- A desirable feature for auto-tuning is the ability to reduce the auto-tuning time when the underlying hardware platform changes
- Bliss employs a simple portability-aid to accomplish this goal:
  - Prunes the search space on the new platform by remembering the combinations of the values of software parameters that resulted in long runtimes
  - Simply filters the parameter values of application-level knobs that are not promising
  - Sets a sufficiently high runtime threshold (> 3x Oracle) for high confidence in suggested prunings

Some positive Points

- **Github Repository**: The authors have a public Github repo that contains the BLISS script, and the sample applications used in the papers
- **Good Explanations of the Results**: The paper provides sufficient explanations of the methodology used, and explains the results in an easy-to-understand manner. The figures are also informative and easy to visualize the results with.

Limitations & Negative Points

06

Random less Interesting things

Limitations and Suboptimal Situations

Limited to Single-Objective Optimization: BLISS is designed for single-objective optimization, and may perform suboptimally if used in a situation where you need to optimize and strike a balance between multiple metrics

Limited to Expensive Evaluations: Bliss is designed for applications where configuration evaluation is expensive, meaning it may not be cost or time effective for applications where the evaluation of each configuration is cheap

Limited to Small Search Spaces: Although Bliss has been shown to scale well when increasing the number of possible configurations by adding new parameters, it may still face challenges when dealing with excessively large search spaces

Despite these limitations, BLISS outperforms state-of-the-art solutions by a factor of **1.6x** in total time to reach near-optimal solutions on average, and significantly lower computation cost for auto-tuning complex applications.

Negative Points

Limited Evaluation on Real-World Applications: The paper only evaluates BLISS on a small and limited set of real-world applications, which may be under representative of its possible use cases.

Limited Discussion about Limitations or Challenges: The paper discusses some, but not many of the possible limitations of BLISS, and it doesn't go into much detail about the ones that it does mention.

Small Communication Errors: Although the paper as a whole is relatively well written, there are some sentences that have a confusing flow to them, or talks about something in a weird way. There are also small typos and random dashes scattered throughout the paper.

Conclusion

07

Random less Interesting things

Why the Paper should be Accepted

A Novel Approach: Bliss presents an approach to auto-tuning large-scale and complex applications using Bayesian Optimization, which hasn't had much exploration up to this point compared to other more popular methods.

Advancements in the Use of Bayesian Optimizers: It introduces a novel technique to leverage multiple Bayesian Optimizers, rather than relying on only one, like you would normally do when using Bayesian Optimization for hyperparameter tuning

Improved Performance: The paper shows that Bliss outperforms other state-of-the-art solutions, both in terms of total time taken and number of samples needed to reach near-optimal solutions, while requiring significantly less computational cost compared to other methods for auto-tuning complex applications.