CAMEO: A Causal Transfer Learning Approach for Performance Optimization of Configurable Computer Systems

Md Shahriar Iqbal
University of South Carolina
USA

Ziyuan Zhong
Columbia University
USA

Iftakhar Ahmad
University of South Carolina
USA

Baishakhi Ray
Columbia University
USA

Pooyan Jamshidi
University of South Carolina
USA

Abstract

Modern computer systems are highly-configurable, with hundreds of configuration options interacting, resulting in enormous configuration space. As a result, optimizing performance goals (e.g., latency) in such systems is challenging. Worse, owing to evolving application requirements and user specifications, these systems face frequent uncertainties in their environments (e.g., hardware and workload change), making performance optimization even more challenging. Recently, transfer learning has been applied to address this problem by reusing knowledge from the offline configuration measurements of an old environment, aka, *source* to a new environment, aka, *target*. These approaches typically rely on predictive machine learning (ML) models to guide the search for finding interventions to optimize performance. However, previous empirical research showed that statistical models might perform poorly when the deployment environment changes because the independent and identically distributed (*i.i.d.*) assumption no longer holds. To address this issue, we propose Cameo—a method that sidesteps these limitations by identifying invariant causal predictors under environmental changes, enabling the optimization process to operate on a reduced search space, leading to faster system performance optimization. We demonstrate significant performance improvements over the state-of-the-art optimization methods on five highly configurable computer systems, including three MLPerf deep learning benchmark systems, a video analytics pipeline, and a database system, and studied the effectiveness in design explorations with different varieties and severity of environmental changes and show the scalability of our approach to colossal configuration spaces.

ACM Reference Format:
Md Shahriar Iqbal, Ziyuan Zhong, Iftakhar Ahmad, Baishakhi Ray, and Pooyan Jamshidi. 2023. CAMEO: A Causal Transfer Learning Approach for Performance Optimization of Configurable Computer Systems. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 18 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2023, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

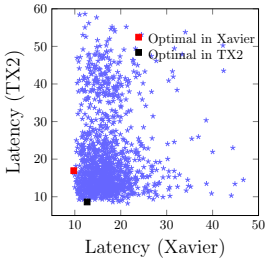


Figure 1: The optimal configuration for MLPerf Object Detection pipeline deployed on TX2 does not remain optimal on Xavier.

1 Introduction

Modern computer systems are continuously deployed in heterogeneous environments (e.g., Cloud, FPGA, SoCs) and are highly configurable across the software/hardware stack [28, 48]. In such highly configurable systems, optimizing performance indicators, e.g., latency and energy, is crucial for faster data processing, better user satisfaction, and lower application maintenance cost [61, 15]. One possible way to achieve these goals is to tune the systems with configuration options across the stack, such as *cpu frequency*, *swappiness*, and *memory growth*, to achieve optimal performance [65, 6, 10].

Finding an optimal configuration in a highly configurable system, however, is challenging [29, 62, 21, 60, 2, 8]: (i) Each component in the system stack, i.e., software, hardware, OS, etc., has many configuration options that interact with each other, giving rise to combinatorial configuration space, (ii) estimating the effect of configurations on performance is expensive as one needs to collect run-time behavior of the system for each configuration, and (iii) unknown constraints exist among configuration options, giving rise to many invalid configurations. Moreover, to meet growing user requirements and reduce service management costs, the underlying systems often undergo environmental changes, i.e., hardware updates, deployment topology change, etc. [14]. Therefore, performance optimization of such evolving systems becomes even more challenging as there is no guarantee that the optimal configurations found in one environment will remain optimal in a different environment [30, 32, 29][1].

To address these challenges, in real-world deployment scenarios, developers often use a staging (development) environment—a miniature of a production environment, for testing and debugging.

[1]we define an environment as a combination of hardware, workload, software, and deployment topology

Table 1: Comparison of CAMEO with state-of-the-art system performance optimization approaches.

Feature	SMAC	CELLO	UNICORN	RESTUNE-W/O-ML	RESTUNE	CAMEO
Detects Spurious Features	✗	✗	✓	✗	✗	✓
Handles Distribution Shift	✗	✗	✗	✗	✓	✓
Suitable for Benchmarks	✓	✓	✓	✗	✓	✓
Knowledge Reuse	✗	✗	✗	✗	✓	✓
Constrained Optimization	✗	✓	✗	✓	✓	✓

Developers collect many experimentation and performance evaluations in staging environments (hereafter, we call them source environments) to understand the performance behavior of the system (what configurations potentially produce performance anomalies, what configurations produce stable performance, or where good configurations lie). Developers then use that knowledge in the target production settings for downstream performance optimizations or debugging. However, in most cases, the result from the staging environment is completely different from the result from production, resulting in a misleading or even wrong indication about the configurations that produce optimal performance. These differences in the results mainly occur due to the hardware gap or workload differences between the development environment and the production one. For example, the workload of an ML system may surge, and as a result, the batch size behind the model server needs to increase to sustain the latency requirement; however, due to the different memory hierarchy and CPU cores between the source and the target environments, the optimal setting for inter-op parallelism of the model server would be vastly different in each environment [52].

Existing works and gap. *Performance Optimization in Configurable Systems.* Several approaches have been proposed for performance optimization of configurable systems, e.g., Bayesian optimization (BO) [25, 66, 64, 4, 44, 29, 32], BO with regression [15], prediction models [7], search space modification [24], online few-shot learning [6], and uniform random sampling and random search algorithms [47]. However, using these approaches in a production environment requires many queries, which are often too expensive to collect or maybe infeasible to perform. The optimal configuration found by these methods in a source environment is also suboptimal for the targets, as the optimal configuration determined in the source environment usually no longer remains optimal in the other (see Figure 1 for an example).

Transfer Learning for Performance Analysis. In real-world deployment scenarios, developers typically have access to performance evaluations of different configurations from a staging environment. Exploiting such additional information using transfer learning can result in efficient optimization, as demonstrated by recent work [32, 39, 26, 43, 40, 33]. For example, searching for the optimized performance in the target setting can leverage the summary statistics of the models built using source performances [67]. However, each environmental change can potentially incur a distribution shift. The ML models used in these transfer learning methods are vulnerable to spurious correlations, which do not hold across distribution shifts and result in inferior performance [68, 45, 27] (see Section 2.1 for an example).

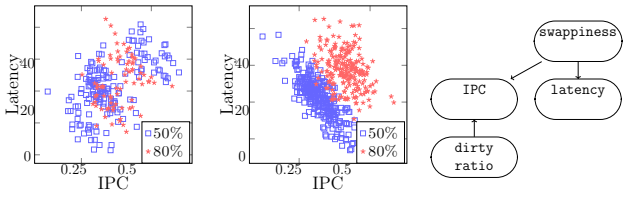
Usage of Causal Analysis in Configurable Systems. To address the problem of spurious correlations, recent work has leveraged causal inference [27, 16, 53] to build a causal performance model[2] that captures the dependencies (a.k.a. causal structures) among configuration options, system events, and performance objectives. However, the causal graphs in the source and target can still have some differences (see Figure 3 for an example). Recent work [27] shows that the source causal model could be reused for performance debugging in the target environment; however, further measurements are needed for performance model learning and optimization.

In summary, all these existing works are suboptimal for performance optimization when the environment changes because the knowledge extracted by these methods from the source (i.e., optimal configuration) has changed and cannot be directly applied to the target, the model (i.e., ML-based transfer learning model) may capture spurious correlations, or the model (i.e., causal model) is mostly stable but need further adaptation in the target environment (see Table 1).

Our approach. An ideal optimization approach should leverage the knowledge derived from the source, which is a close replica of the target environment with a cheaper experimentation cost. Our key insight is, using causal reasoning, we should be able to identify the non-spurious invariances across environments that truly impact the performance behavior of the system. These invariances can then be transferred to the target environment for performance optimization tasks, thus reducing the need for many observational data in the production environment. Therefore, we will reduce the cost of optimization tasks without compromising accuracy.

To this end, we propose CAMEO (**Ca**usal **M**ulti **E**nvironment **O**ptimization), a causal transfer-based optimization algorithm that aims to overcome the limitation of prior approaches. Our approach builds on top of two previous works, JUMBO (a multi-task BO method) [19] and CBO (a causal BO method) [3]. A typical BO approach consists of two main elements: the surrogate model and the acquisition function. The surrogate model tries to predict the performance objective when given a configuration, and the acquisition function assigns a score to each configuration and chooses the one with the highest score to query for the next iteration. In CAMEO, we first build two causal performance models to learn the dependency among configuration options, system events, and performance objectives for each environment using the previous performance measurements from the source environment and a considerably smaller number of measurements from the target environment. After that, we simultaneously train two Causal Gaussian Processes (CGPs) (which leverage the causal performance models when estimating means and variances) as two surrogate models: a warm CGP in the source and a cold CGP in the target. The acquisition function combines the individual acquisition functions of both CGPs to leverage knowledge from both source and target. This way of combining individual acquisition functions of both CGPs allows to only to rely on the core features from the source environment that remain stable across environments and update belief about the environment specific features in the target, making the optimization more effective.

[2]A causal performance model is an acyclic-directed mixed graph, with nodes being variables and arrows being causal connections. It represents the dependencies (a.k.a. causal structures) among configuration options, system events, and performance objectives.



(a) source (TX2) (b) target (Xavier) (c) true relationship

Figure 2: (a)-(b) The relationship between IPC and latency reverse from source (TX2) to target (Xavier) while the relationship between swappiness (the values are denoted as colors) and latency stays invariant. (c) The true causal relationship among the relevant variables.

Evaluation. We evaluated Cameo in terms of its *effectiveness, sensitivity*, and *scalability*, and compared it with four state-of-the-art performance optimization techniques (Smac [25], ResTune-w/o-ML and ResTune [67], cello [15], and Unicorn [27]) using five real-world highly configurable systems, including three MLperf pipelines (object detection, natural language processing, and speech recognition), a video analytics pipeline, and a database system, deployed on edge and cloud under different environmental changes. Our results indicate that Cameo improves latency by 3.7× and energy by 5.6× on average than the best baseline optimization approach, ResTune.

Contributions. Our contributions are the following:

- We propose Cameo, a novel causal transfer-based approach that allows faster optimization of software systems when the environment changes. To the best of our knowledge, this is the first approach that addresses the performance optimization of configurable systems using causal transfer learning.
- We conduct a comprehensive evaluation of Cameo by comparing it with state-of-the-art optimization methods on five real-world highly configurable systems under a range of different environmental changes and studied the effectiveness in design explorations with different varieties and severity of environmental changes and show the scalability of our approach to colossal configuration spaces. The artifacts and supplementary materials can be found at https://github.com/softsys4ai/CAMEO.

2 Motivation and Insights

In this section, we motivate our approach by illustrating why causal reasoning can contribute to more effective (faster and less costly) optimization of system performance. In particular, we focus on how the properties of the causal performance models can be leveraged across environments. For this purpose, we used the Mlperf Object Detection [50] pipeline as a part of MLPerf Inference Benchmark[3] by following the benchmark rules[4], with the following setup: Model: Resnet50-v1.5; Test Scenario: Offline; Metric: inference latency; Workload: 5000 ImageNet samples; workload generator: Mlperf Load Generator; Source Hardware: Jetson TX2; Target Hardware: Jetson Xavier and TX1. For better control, we limit the configuration space to 28 options across the stack—4 hardware options (e.g., cpu cores), 22 OS options (e.g., dirty ratio), and 2 compiler options (e.g., allow memory). We sampled 2000 random configurations and measured inference latency in each environment. We also collected performance counters and system events statistics using *Linux perf profiler*[5].

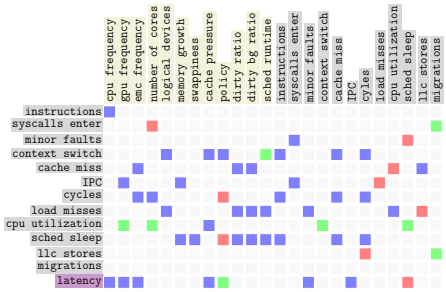


Figure 3: There is a significant overlap between the causal structures (the common edges are represented as blue squares) developed in different environments (e.g., Jetson TX2 and Xavier). Some edges unique in the source (green squares) or target (red squares) also exist.

2.1 Why performance optimization using causal reasoning is more effective?

In order to deploy a configurable computer system such as Mlperf Object Detection in a new environment with low latency and energy consumption, the dominant approach is to train a performance model using a limited number of samples and use the model for predicting performance for unmeasured configurations and select the configuration with the optimal performance. To show how spurious features could mislead performance optimization, we investigate the impact of confounders and how they make it difficult for an ML model to determine the accurate relationship between configuration options and performance objectives. We perform a sandbox experiment where we carefully tune swappiness [6] and dirty ratio [7] both in source and target, while leaving all other options at their default values. Here, the observational data collected from the experiment indicates that as IPC [8] (one of the system events) increases, latency increases, which is a spurious proportional relationship. Relying on spurious features (IPC in this example) can lead to poor performance predictions (as one might try to reduce IPC and expect lower latency but end up getting higher latency) when the environment changes as they are susceptible to *correlation shifts*—i.e., the direction of correlation may change across environments. As shown in Figure 2(a)-(b), a correlation shift happens in this sandbox experiment as IPC is positively correlated with latency in the source but negatively correlated in the target.

To investigate the reason behind the correlation shift, we group the data based on their swappiness (50% and 80%, respectively) and observe that the correlation between swappiness and latency

[3] https://mlcommons.org/en/inference-edge-30/
[4] https://github.com/mlcommons/inference_policies/blob/master/inference_rules.adoc
[5] https://perf.wiki.kernel.org/
[6] swappiness is the rate at which the kernel moves pages into and out of the physical memory. The higher the value, the more aggressive the kernel will be in moving the pages out of physical memory to the swap memory.
[7] dirty ratio is the value that represents the percentage of physical memory that can consume dirty pages before all processes must write dirty buffers back to the disk.
[8] IPC represents instruction per cycle, which is the average number of instructions executed for each clock cycle.

Table 2: ML-based regressors (GPR, RFR) have higher generalization error compared to causal-based regressor (CGPR).

Source	Target	KL Div.	Prediction Error (%)		
			GPR	RFR	CGPR
TX2	Xavier	476	22.4	25.6	11.2
TX2	TX1	519	27.6	23.2	11.4

remains the same (larger swappiness implies higher latency in both environments) whereas the correlation between swappiness and IPC reverses (from proportional to inverse proportional) as shown in Figure 2(a)-(b). Figure 2(c) shows the causal structure where swappiness is a *common cause* of both IPC and latency. swappiness should be considered for latency since it remains invariant across environments. In contrast, the relationship between IPC and latency is environment dependent, and their correlation can change when another confounder variable, dirty ratio, is different in source and target. In our example, since the source has 4× lower physical memory than the target, the allocated memory for the dirty pages becomes filled sooner and must be returned to the disk. As a result, the source will have higher IPC for a lower value of swappiness as the dirty pages will be flushed before the limit for swappiness is reached. However, the application is not making any forward progress here, resulting in increased latency. In the target (due to larger memory), the dirty pages might never become full, and only swappiness would cause the IPC to be positively correlated to latency. The example in Figure 2 shows that the casual model can capture the *data generation process* better as it only relies on the invariant causal mechanisms (swappiness for latency) and can remove spurious correlations (IPC for latency) that are specific to a particular environment. Therefore, causal models may suffice to predict the consequences of interventions (*what if* scenarios) on variables to particular values for effective search during optimization and allow for better explorations in limited budget scenarios.

To show the benefits of correctly identifying the invariant features, we train different ML-based regressors, e.g., Gaussian Process Regressor (GPR) and Random Forest Regressor (RFR), using data collected for the sandbox system deployed in TX2 and determined their prediction error in TX1 and Xavier (shown in Table 2). Here, we observe that the ML-based regressors have considerably higher errors in the target environment despite low source errors. The prediction error increases further as the distributions become more dissimilar (indicated by a higher KL-divergence value). In contrast, the causal approach, Causal Gaussian Process Regressor (CGPR), has a considerably lower error and remains stable as the degree of distribution shift increases.

Takeaway 1 *Causal models generalize better in performance prediction tasks across environments by distinguishing invariant from spurious features.*

2.2 Learning from Causal Structural Properties in Various Environments

As we have established that a causal model can be reliably used for performance predictions in new environments, we next study the properties of the causal graph that can be exploited for faster optimization. We build a causal graph using a causal structure discovery algorithm [56] in source and target, respectively, and compare them. As shown in Figure 3, both causal graphs are sparse (the white squares indicate no dependency relationship exists) and share a significant overlap (the blue squares indicate the edges present in both). Therefore, a causal model developed in one environment can be leveraged in another as prior knowledge. However, reusing the causal graph entirely might induce some wrong biases as the causal graphs in the two environments are not identical (the green and red squares indicate the edges present uniquely in the source and target, respectively). We must discover the target's new causal connections (indicated by the red squares) based on the observation. Since the number of edges that must be discovered is small, this can be easily done with a small number of observational samples from the target environment.

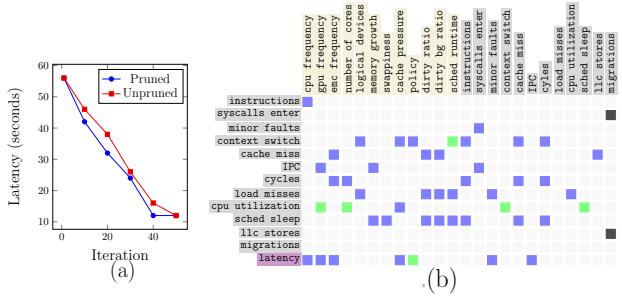


Figure 4: (a) Pruning edges with a Markov blanket identifies the optimal configuration faster. (b) Combining the top K nodes' Markov blankets eliminates the wrong biases (shown as black squares).

Takeaway 2 *A performance optimization approach should locate high-quality observational samples in the target, simultaneously leveraging the source knowledge to guide the search.*

2.3 Learning to Intervene based on Causal Structure

We need to remove the edges unique to the source. The removal operation can be accomplished by performing interventions that estimate the effects of deliberate actions. For example, we measure how the distribution of an outcome (e.g., latency $\mathcal{Y}$) would change if we intervened during the data gathering process by forcing the variable cpu frequency O_i to a certain value o_i while retaining the other variables as is. We can estimate the outcome of the intervention by modifying the CPM to reflect our intervention and applying Pearl's *do-calculus* [49], which is denoted by $Pr(\mathcal{Y} \mid do(O_i = o_i))$. However, since many configurations need to be measured, it is not feasible to perform interventions to estimate the existence of every edge. Instead, we can significantly reduce the number of configurations by avoiding the interventions on nodes with limited causal effects on the performance objective. For this purpose, we rank the causal effects of all the existing nodes on latency and observe that only one source-specific edge (policy) is among the top 10 most influential nodes. Thus, we can select the *top K* nodes with

the highest causal effects and combine the *Markov blanket* [9] of them, which would eliminate all the nodes that have lower causal effects. In our example, if we select K=6 with Markov blankets then the wrong biases, migrations->syscalls enter and migrations->llc stores (the nodes marked by black in Figure 4(b)), are eliminated. Figure 4(a) shows that pruning the edges helps to reach the optimal value 19% faster. Therefore, *we require an approach that relies on intervening only on the top K nodes based on source knowledge in the target environment.*

Takeaway 3 *Employing rich knowledge in a causal performance model, we can intervene on specific configurations to learn the most about the underlying causal structure and be able to gather the most relevant data in a limited budget scheme.*

3 CAMEO Design

In this section, we present CAMEO—a framework for performance optimization of highly configurable computer systems.

3.1 Problem Formulation

Let us consider a highly configurable system of interest with configuration space O, system events and performance counters space C, and a performance objective $\mathcal{Y}$. Denote O_i to be the i^{th} configuration option of a system, which can be set to a range of different values (e.g., categorical, Boolean, and numerical). The configuration space is a Cartesian product of all hardware, software, and application-specific options: $O = Domain(O_1) \times ... \times Domain(O_d)$, where d is the number of options. Configuration options and system events are jointly represented as a vector $\mathcal{X} = (O, C)$. We assume that in each environment $e \in \mathcal{E}$ (a combination of hardware, workload, software, and deployment topology), the variables $(\mathcal{X}_e, \mathcal{Y}_e)$ have a joint distribution $\mathcal{P}_e$. In the source environment e_s, there are n independent and identically distributed (i.i.d) observations. In the target environment e_t, m ($\ll n$) observations can be collected within a given budget $\mathcal{B}$. The task is to find a near-optimal configuration, o^*, with a fixed measurement budget, β, in the target environment, e_t, that results in Pareto-optimal performance:

$$o^* = \text{argmin}_{o \in O} \mathcal{Y}_{e_t}(o), \tag{1}$$

where O represents the configuration space, $\mathcal{Y}$ is a set of performance metrics measured in the target environment e_t.

3.2 CAMEO Overview

CAMEO is a causal transfer learning optimization algorithm that enables developers and users of highly configurable computer systems to optimize performance objectives such as latency, energy, and throughput when the deployment environment changes. Figure 5 illustrates the overall design of our approach. CAMEO works in two phases: (i) *knowledge extraction phase*, and (ii) *knowledge update phase*. In the knowledge extraction phase, CAMEO first determines the user requirements using a query engine. Then, it learns a causal performance model $\mathcal{G}_s$ using the cheaper offline performance measurements $\mathcal{D}_s$ from the source environment e_s, which is later reused to obtain meaningful information that is shared with the target environment e_t for faster optimization. As performance evaluations in the target are expensive, this way of warm-starting the optimization process by reusing the causal performance model $\mathcal{G}_s$ enables us to navigate the configuration space more effectively with less number of interventions in the target. However, relying solely on the source's information is insufficient to effectively optimize performance in the target due to the differences across environments (as shown in Section 2.2). Therefore, in the knowledge update phase, CAMEO employs an active learning mechanism combining the source causal performance model $\mathcal{G}_s$ with a new causal performance model $\mathcal{G}_t$ collected from a small number of samples, $\mathcal{D}_t$, from the target environment.

Once the two causal performance models are constructed, we simultaneously train two causal Gaussian processes (CGPs) as the surrogate models—CGP_{warm} and CGP_{cold}—to model performance objective $\mathcal{Y}$ from $\mathcal{G}_s$ and $\mathcal{G}_t$, respectively. The two CGPs operate on different input spaces. CGP_{warm} works on a reduced configuration space that is derived from $\mathcal{G}_s$. In contrast, to ensure that any information omitted in the source is not left undiscovered in the target, CGP_{cold} works on the entire configuration space. We integrate the posterior estimates from both CGP_{warm} and CGP_{cold} to develop an acquisition function α that can regulate the information from two CGPs through a controlling variable λ. The larger λ is, we rely more on the information in CGP_{warm}. Next, we evaluate our acquisition function α for different configurations and select the one for which the α value is maximum for observation or intervention. The choice of observation and intervention for performance evaluation is guided by an exploration coefficient ϵ. Finally, we use the newly evaluated configurations to update the causal performance and surrogate models. We continue the active learning loop until the stopping criterion is met (i.e., maximum budget β is exhausted or convergence is attained). The pseudocode for our approach is provided in Algorithm 1.

3.3 Knowledge Extraction Phase

We next describe the offline knowledge extraction phase.

User Query Translation A developer can use CAMEO to find the optimal configurations optimizing a system's performance objectives in a target environment within a limited experimentation budget β. The developer can start the optimization process by querying CAMEO with requests like *"How to improve latency within 1 hour or 50 samples"* or *"I want to find the configuration with minimum energy for which latency is less than 20 seconds within 45 minutes?"*. The query engine initially translates the user requests to determine the allowable budget β, constraints ψ, and the performance goal $\mathcal{Y}$ to optimize. In the first query, the budget is 1 hour or 50 samples, the performance objective is latency, and no constraints exist. In the second query, the budget is 45 minutes, the performance objective is energy, and the constraint is a latency of less than 20 seconds.

Learning Causal Performance Model We begin by building two causal performance models: $\mathcal{G}_s$ and $\mathcal{G}_t$ using the offline performance evaluation dataset $\mathcal{D}_s$ from the source with n configurations and the performance dataset $\mathcal{D}_t$ from the target with randomly sampled m initial configurations, respectively (line 1). We use an existing structure discovery algorithm *fast causal inference* (FCI) to learn $\mathcal{G}_s$ and $\mathcal{G}_t$ that describes the causal relations among configuration options O_i, system events and performance counters C_i, and

[9] A Markov blanket of a node includes all its parents, children, and children's parents.

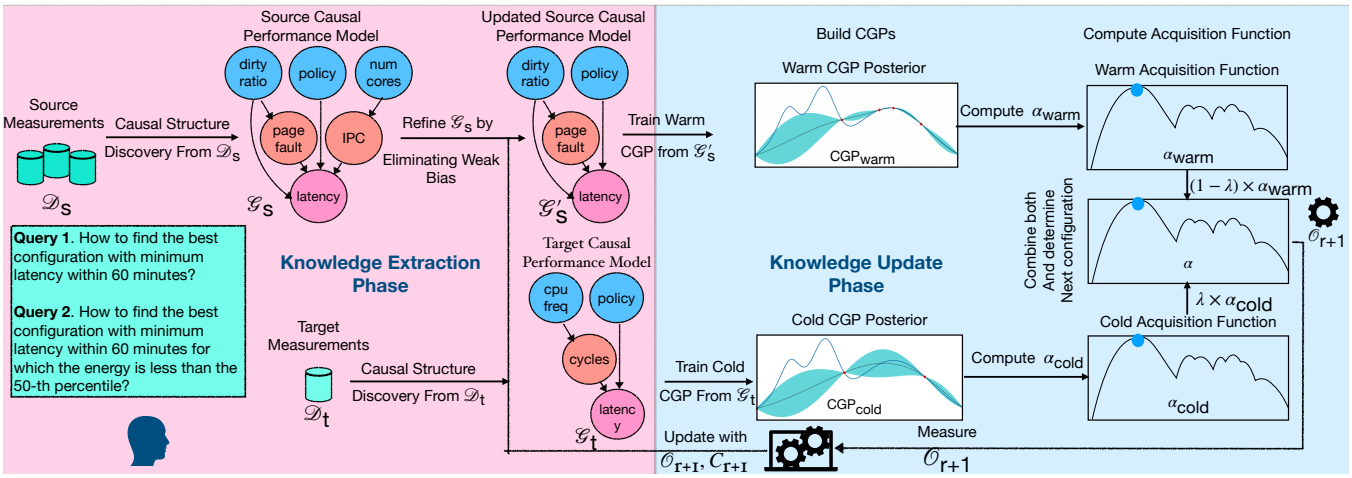


Figure 5: Overview of CAMEO

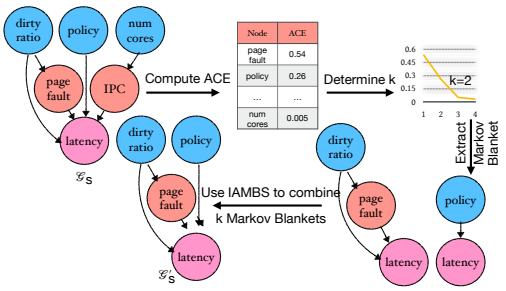


Figure 6: Refining the causal performance model from the source to eliminate unwanted information.

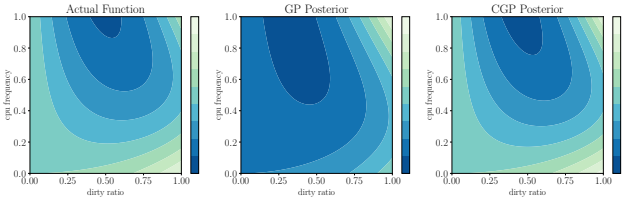


Figure 7: The posterior of CGP relying on interventional distribution can capture the target function behavior better than GP.

performance objectives $\mathcal{Y}$. We select FCI as the causal structure discovery algorithm because (i) it accommodates variables that belong to various data types such as nominal, ordinal, and categorical data common across the system stack, and (ii) it accommodates for the existence of unobserved confounders [56, 46, 18]. This is crucial because we do not assume absolute knowledge of configuration space, so there may be configurations we cannot intervene in or system events we have not observed. FCI operates in three stages. First, we construct a fully connected undirected graph where each variable is connected to every other variable. Second, we use statistical independence tests (Fisher z-test for continuous variables and mutual information for discrete variables) to prune away edges between independent variables. Finally, we orient undirected edges using prescribed edge orientation rules [56, 46, 18, 12, 11] to produce a *partial ancestral graph* (or PAG). In addition to both directed and undirected edges, a PAG also contains partially directed edges that need to be resolved to generate an acyclic-directed mixed graph (ADMG), i.e., we must fully orient partially directed edges with the correct edge orientation. This work uses an information theoretic approach to automatically orient partially directed edges using the *LatentSearch* algorithm [38] by entropic causal discovery.

Refining Causal Performance Model Now that we have constructed the causal performance models which rely on the invariant features, we may be tempted to directly reuse source model $\mathcal{G}_s$ in the target to warm start the optimization process. However, since some edges are specific to the source (as discussed in Section 2.2), directly reusing $\mathcal{G}_s$ will bias the optimization in the target. To avoid wasting the budget allocated for the online optimization procedure, we attempt to minimize those biases as much as possible in this offline phase. To do so, we transfer the Markov blanket (Mb) of the top k nodes ranked based on their causal effects on the performance objective to eliminate unwanted information. This is an important step as we need to rely on the optimal core features that remain invariant when a performance distribution shift happens to reason better in the new environment. Theoretically, a node's Mb is the best solution to the feature selection problem for that node [34]. The variables in the Mb can be confidently employed as causally informative features in the target because it provides a thorough picture of the local causal structure around the variable. Initially, we determine k using the method proposed in [22] (line 2). Then, we extract the Mb of the k nodes to determine the final $\mathcal{G}_s$ that will be reused in the subsequent phase (line 4) using the IAMBS algorithm presented in [41]. The IAMBS algorithm is focused on constructing a Mb for multiple variables (top k nodes). It operates by determining whether the additivity property holds for Mb of k variables, further, how to proceed if the additivity property is violated by selectively performing conditional independence tests using a growing and a shrinking phase [41].

3.4 Knowledge Update Phase

In this phase, we exploit the knowledge gained from the earlier phase to guide the search strategy for optimization using the three components described below.

Build Causal Gaussian Processes. At this stage, we train two surrogate models: CGP_{warm} and CGP_{cold} for the performance objective $\mathcal{Y}$ from $\mathcal{G}_{\text{s}}$ and $\mathcal{G}_{\text{t}}$, respectively. For this purpose, we use the mathematical formulation proposed in the CBO approach [3] to build a CGP. Unlike GPs, CGPs represent the mean using interventional estimates via do-calculus, which allows the surrogate model to capture the behavior of the performance objective better than GPs (as shown in Figure 7), particularly in areas where observational data is not available. Therefore, we fit a prior on $f(o) = E[\mathcal{Y}|do(O_i = o_i)]$ with mean and kernel function computed via do-calculus separately for each CGP obtained from $\mathcal{G}_{\text{s}}$ and $\mathcal{G}_{\text{t}}$ as the following:

$$f_e(o) \sim GP(\mu_e(o), k_{c_e}(o, o')) \tag{2}$$

$$\mu_e(o) = \hat{E}[\mathcal{Y}|do(O_i = o_i)] \tag{3}$$

$$k_{c_e}(o, o') = k_{RBF}(o, o') + \sigma_e(o)\sigma_e(o'), \tag{4}$$

where $\sigma_e(o) = \sqrt{\hat{V}_e(\mathcal{Y}|do(O_i = o_i))}$ with $\hat{V}_e$ representing the variance estimated from the configuration measurements ($\mathcal{D}_{\text{s}}$ or $\mathcal{D}_{\text{t}}$) for a particular environment. k_{RBF} is the radial basis function of the kernel defined as $k_{RBF}(o, o') = exp(-\frac{||o-o'||^2}{2l^2})$, where l is a hyper-parameter. As a result, the shape of the posterior variance enables a proper calculation of the uncertainties about the causal effects (enabling identification of influential configuration options and interactions). We extract the exploration set (ES) for each environment, guided by $\mathcal{G}_{\text{s}}$ and $\mathcal{G}_{\text{t}}$, and compute the mean and uncertainty estimates for configurations in the exploration set.

Compute Acquisition Function for Sampling. Denote $\alpha^r_{\text{warm}}(o)$ and $\alpha^r_{\text{cold}}(o)$ to be the single objective acquisition functions of the two CGPs. For Cameo, we choose to use the expected improvement (EI) as the acquisition function [63] since EI has been demonstrated to perform well for configuration search. EI selects the configuration that would have the highest expected improvement with respect to the current best interventional setting separately from e_{s} and e_{t} across all configurations in the respective exploration set:

$$EI_e(o) = E_{p(y)}[max(y - y^*, 0)], \tag{5}$$

where $y = E[\mathcal{Y}|do(O_i = o_i)]$ and y^* is the optimal value observed thus far. In our implementation, we rank the configurations based on $\alpha^r_{\text{warm}}(o)$ scores and then select the ones with the highest $\alpha^r_{\text{cold}}(o)$ score. The acquisition function (line 10) is:

$$\alpha^r(o) = \lambda^r(o)\alpha^r_{\text{cold}}(o) + (1 - \lambda^r(o))\alpha^r_{\text{warm}}(o), \tag{6}$$

where λ^r is an interpolation coefficient (line 8-9) that controls the proportion of knowledge used from source and target and is dependant on l_α and the expected improvement of a configuration. The above equation shows that λ is 1; it would use the contribution from α_{cold} and use α_{warm} when λ is 0. The interpolation coefficient λ^r is defined as the following:

$$\lambda^r(o) = \mathbb{1}(\alpha^r_{\text{warm}^*} - \alpha^r_{\text{warm}}(o) \leq l_\alpha), \tag{7}$$

where $\alpha^r_{\text{warm}^*}$ is the optimal acquisition value obtained from α^r_{warm} scores. The choice of l_α is critical since it balances the knowledge used from the source and target. We set l_α to be 0.1, which shows good empirical performance (as shown in Figure 15(b)). Intuitively, the acquisition function should operate in such a way so that it uses α_{cold} for the configurations that are near the optimal points. Here, the l_α is an acquisition threshold hyper-parameter used to define near optimal points w.r.t. α_{warm}. Therefore, configurations that are nearer to the optimal points of α_{warm} (configurations which satisfy $l_\alpha \leq 0.1$) will provide higher expected improvement value for α_{cold}.

On the contrary, configurations that are further away from the optimal points of α_{warm} (configurations which do not satisfy $l_\alpha \leq 0.1$) will have higher expected improvement value for α_{warm}. This indicates that either such configurations contain options that have some environment-specific behavior that is not captured or learned correctly by the source causal model and the source causal model needs to be updated.

Now, we find a configuration $o^{\text{r}+1}$ for which the α^r value is maximum for either observation or intervention (line 11). Observational data may be used to correctly predict the causal effects of configuration options on the performance objective. On the other hand, estimating consistent causal effects for values outside of the observable range necessitates intervention. The developer must identify the optimal combination of these operations to capitalize on observational data while intervening in regions with higher uncertainty. We adopt the $\epsilon-$greedy approach as in CBO to trade-off exploration and exploitation, which is defined as the following:

$$\epsilon = \frac{Vol(H(\mathcal{D}_v))}{Vol(o_{o \in O}(\mathcal{D}(O)))} \times \frac{N}{N_{\text{max}}}, \tag{8}$$

where $D_v = \mathcal{D}_{\text{s}} \cup \mathcal{D}_{\text{t}}$, $Vol(H(\mathcal{D}_v))$ represents the volume of the convex hull for the observational data and $Vol(o_{o \in O}(\mathcal{D}(O)))$ gives the volume of the interventional domain. N_{max} represents the maximum number of observations the developer is willing to collect on a particular environment, and N is the current size of D_v. The interventional space is bigger than the observational space when the volume of the observational data $Vol(H(\mathcal{D}_v))$ is smaller with respect to the number of observations N. Therefore, we must perform interventions to explore regions of the interventional space not covered by observational data. On the other hand, if the volume of the observational data $Vol(H(\mathcal{D}_v))$ is large with respect to N, we need to perform observations. This is because we need to obtain consistent estimates of the causal effects, which can only be achieved with more observations. We update the convex hull incrementally for computation purposes.

Evaluate Selected Configuration and Update Belief We measure the selected configuration $o^{\text{r}+1}$ (lines 13-17) and check whether the newly measured configuration satisfies the constraints (line 18). If not, we replace the performance objective value with an infinitely high value to force the optimizer to avoid searching in regions of the space where the constraints are not satisfied (line 19). We update the causal performance and surrogate models using the new measurement (line 21). We repeat the optimization loop until the maximum budget β is exhausted or convergence is reached and return the configuration with minimum $\mathcal{Y}$ as the optimal.

Algorithm 1 CAMEO

Require: Offline source dataset $\mathcal{D}_s$, Initial target datatset $\mathcal{D}_t$, Configuration space O, Total budget β, Threshold l_α, Performance Objective $\mathcal{Y}$, Constraint Ψ.

Knowledge Extraction Phase

Construct a causal performance model from $\mathcal{G}_s, \mathcal{G}_t$ using $\mathcal{D}_s, \mathcal{D}_t$, respectively.
Extract the top k nodes from $\mathcal{G}_s$ in terms of causal effect on performance objective.
Extract Markov Blanket of the top k nodes to construct a new updated $\mathcal{G}_s$.

Knowledge Update Phase

Initialize CGP_{warm} and CGP_{cold}.
$\beta^r = 0$
while $\beta^r \leq \beta$ **do**
  Set $\alpha^r_{\text{warm}^*} = argmin_{o \in O} \alpha^r_{\text{warm}}(o)$
  Set interpolation coefficient: $\lambda^r(o) = \mathbb{1}(\alpha^r_{\text{warm}^*} - \alpha^r_{\text{warm}}(o) \leq l_\alpha)$
  Set the acquisition function: $\alpha^r(o) = \lambda^r(o)\alpha^r_{\text{cold}}(o) + (1 - \lambda^r(o))\alpha^r_{\text{warm}}(o)$
  Pick a new configuration: $o^{r+1} = argmin_{o \in O}\ \alpha^r(o)$
  Compute the exploitation coefficient ϵ using Equation (8) and sample a random number $u \approx \mu(0, 1)$
  if $\epsilon < u$ **then**
    make a new observation $(o^{r+1}, c^{r+1}, y^{r+1})$.
  else
    Intervene on the system to obtain an interventional measurement $(o^{r+1}, c^{r+1}, y^{r+1})$.
  end if
  if y^{r+1} does not satisfy Ψ **then**
    $y^{r+1} = \infty$
  end if
  Update $\mathcal{G}_s$, CGP_{warm}, D_s, $\mathcal{G}_t$, CGP_{cold}, and D_t.
  Update β^r
end while
return the configuration with the best performance objective.

4 Evaluation

Subject systems and configurations. We selected five configurable computer systems, including a video analytics pipeline, a CASSANDRA database system, and three deep learning systems (for image, speech, and NLP, respectively). Following configuration guides and other related work [20, 27, 55], we used a wide range of configuration options and system events that impact scheduling, memory management, and execution behavior. The complete list of configuration options per system can be found in the supplementary materials on GitHub. As opposed to prior works (e.g., [59, 58]) that only support binary options due to scalability issues, we additionally included discrete options and continuous options. For discrete options, we exhaustively set each one to all permitted values. We choose the recommended range from system documents for continuous options.

We run each software with a set of popular workloads that are extensively used in benchmarks and prototypes (more details are provided in Section 5-7). We use various deployment platforms with distinct resources (e.g., computation power, memory) and microarchitectures to demonstrate our approach's versatility. We use NVIDIA Jetson TX2, TX1, AGX Xavier, and Xavier NX devices for edge deployment. To deploy a particular system on the cloud, we use Chameleon configurable cloud systems where each node is a dual-socket system running Ubuntu 20.04 (GNU/Linux 6.4) with 2 Intel(R) Xeon(R) processors, 64 GB of RAM, hyperthreading, and TurboBoost. Each socket has 12 cores/24 hyper-threads with multiple Nvidia Tesla P100 16GB GPU and K80 24GB GPU for deep learning inference.

Data collection We measure the system's latency/throughput and energy for each configuration. Following a common practice [14, 15], we randomly select 2000 configurations for each system for performance measurements. We repeat each measurement 5 times and record the median to reduce the effect of measurement noise and other variabilities [26].

Experimental parameters We use a budget of 200 iterations for each optimization method, similar to standard system optimization approaches [67]. We repeat each method's optimization process 3 with different random seeds for reliability. We follow the standard tuning and reported parameter values for SMAC, UNICORN, RESTUNE-W/O-ML, RESTUNE, and CELLO. More details about different experimental choices (Table 8-14), implementation (Figure 17-21), and hyperparameters (Table 15-16) can be found in the supplementary materials.

Baselines We compare CAMEO against the following:

- **SMAC** [25]: A sequential model-based configuration optimization algorithm.
- **UNICORN** [27]: An active learning approach that transfers knowledge via a causal model for optimization in the target.
- **CELLO** [15]: An optimization framework that augments Bayesian optimization with predictive early termination.
- **RESTUNE** [67]: A constrained optimization approach that uses multiple models (ensemble) to represent prior knowledge.
- **RESTUNE-W/O-ML** [67]: RESTUNE without meta-learning, i.e., it only learns from scratch in the target.

Evaluation Metrics. When running them for the same time limit, we compare the best performance objectives (e.g., latency, throughput, energy, etc.) achieved by each method. We also compare their relative error (RE) as follows:

$$RE = \frac{|\mathcal{Y}_{\text{pred}} - \mathcal{Y}_{\text{opt}}|}{|\mathcal{Y}_{\text{opt}}|} \times 100\%, \tag{9}$$

where $\mathcal{Y}_{\text{pred}}$ is the best value achieved by each method, and $\mathcal{Y}_{\text{opt}}$ is the optimal measured value from our observational dataset of 2000 samples. A method is considered more effective if it recommends a configuration achieving a lower error.

Research questions. We evaluate CAMEO by answering three research questions (RQs).

RQ1: How effective is CAMEO in comparison to the state-of-the-art approaches when the following environmental changes happen? (i) hardware change, (ii) workload change, (iii) software change, and (iv) deployment topology change.

RQ2: How the effectiveness of CAMEO changes when the severity of environmental changes varies?

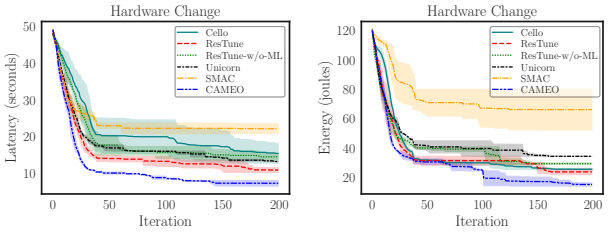


Figure 8: Effectiveness of Cameo when hardware changes happen in the deployment environment.

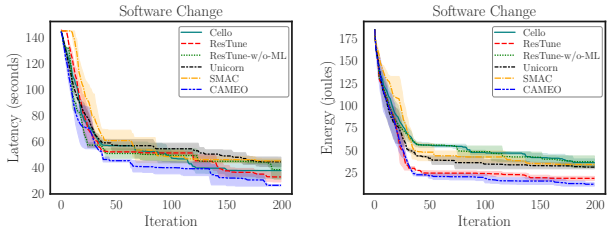


Figure 9: Effectiveness of Cameo when software changes happen in the deployment environment.

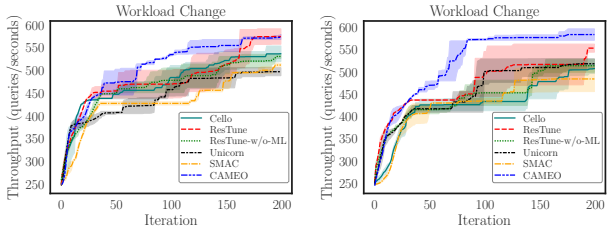


Figure 10: Effectiveness of Cameo in workload change scenarios: Read-Only to Balanced (left) and Update-Heavy (right).

RQ3: How sensitive is Cameo when (i) the number of samples in the source environment varies? (ii) the value of l_α varies? and (iii) the size of the configuration space increases?

5 RQ1: Effectiveness in Design Explorations

We evaluate the effectiveness of Cameo in finding an optimal configuration compared to the state-of-the-art. We consider four types of environmental changes typically occurring when a system is deployed into production. Table 3 shows the summarized results for each approach averaged over different environmental changes considered in this paper. Cameo finds the configuration with the lowest latency and energy than other approaches, e.g., Cameo achieves 3.7× and 5.6× lower RE for latency and energy, respectively, when compared to ResTune, the next best method after Cameo. We describe the experimental setting and results for all four environmental changes below.

Hardware change We consider the Mlperf object detection pipeline that uses ResNet-18 for inference over 5k images selected from the 100k test images of the ImageNet dataset [51]. We use the TX2 as the source hardware and Xavier and TX1 as the target hardware. We examine these hardware changes since there are variable degrees of micro-architecture differences among this hardware separately. We only show results for Xavier (for the other TX1, we refer to the appendix). As shown in Figure 8, Cameo finds the configuration with the lowest values. For example, Cameo finds configuration with 1.6× lower latency than ResTune. We also observe a similar trend for energy.

Software change We consider variants of a natural language processing (NLP) model—BERT [13] and TinyBERT [35]—deployed on Xavier for the experimental setup. We set up a software change by changing the model architecture across environments, where we use TinyBERT with 3 million parameters as the source and BERT-Base with 109M parameters as the target. As workload, we perform sentiment analysis on 1000 out of the 25000 reviews from IMDB test dataset [42]. We present the results for software change in Figure 9

Table 3: Summarized results averaged over all environment changes.

	Latency	Energy
	RE(%)	RE(%)
Smac	88.2	268.9
cello	46.2	182.5
ResTune-w/o-ML	48.8	191.1
Unicorn	55.5	179.9
ResTune	29.2	81.2
Cameo	7.8	14.4

for latency and energy optimization. The optimal configurations found by Cameo have 1.1× lower latency and 1.7× lower energy value compared to those discovered by ResTune, respectively.

Workload change We consider Cassandra database deployed on Chameleon cloud instance (see Section 4) while varying different workloads to create different source and target environments using the TPC-C benchmark [1]. We use a YCSB workload generator to generate 3 workloads: (i) read only- 100% read, (ii) balanced - 50% read and 50% update, and (iii) update heavy - 95% update and 5% read. To optimize throughput, we use a read only workload as the source and the remaining two workloads as the target separately. Results for workload changes are presented in Figure 10. When the workload changes from read only to balanced, ResTune outperforms Cameo by finding a configuration with 1.02× higher throughput. Upon further investigation, we find that here the distributions between source and target were relatively similar, and the shared covariance learning in MTGP helped ResTune in finding a better configuration. Moreover, the knowledge extraction module in ResTune is particularly developed for correctly capturing workload behavior, making it more suitable for this domain adaptation scenario. However, as the distribution difference increases, Cameo outperforms ResTune, e.g., update heavy workload Cameo has 1.06× higher throughput than ResTune.

Deployment topology change To test the effectiveness of Cameo across deployment topology change, we consider a video analytics pipeline: DeepStream that uses 4 camera streams as the workload. Our DeepStream pipeline has four components: (i) an x264 decoder, (ii) a multiplexer, (iii) a TrafficCamNet model with ResNet-18 as the detector, and (iv) an NvDCF tracker, which uses a correlation filter-based online discriminative learning algorithm for tracking. As the source environment, we adopt a centralized deployment topology where all four components run on the same Xavier NX hardware. We employ a dispersed deployment topology with two Xavier NX

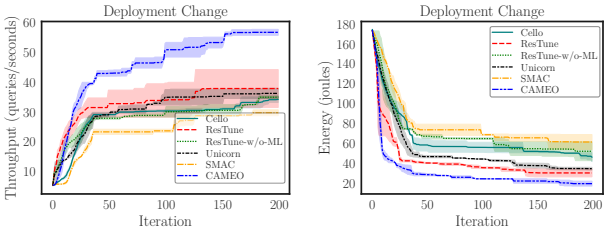


Figure 11: Effectiveness of CAMEO when deployment topology changes in the deployment environment.

hardware as the target, deploying the decoder and multiplexer in one and the detector and tracker in the other. We use APACHE KAFKA to send and receive output from the multiplexer to the detector that uses a binary protocol over TCP. Our experimental results for deployment environment change presented in Figure 11 show that CAMEO significantly outperforms others in finding optimal throughput and energy. For example, the optimal configuration discovered by CAMEO has as high as 1.3× and 1.5× (as) improvement for throughput and energy, respectively, than the next best method.

Constrained optimization For constrained optimization (optimizing latency with energy constraint or optimizing energy with latency constraint), we set the energy and latency constraints as [15, 30, 45, 60, 75, 90]-th percentiles of the corresponding distributions. Table 4 reports the summarized results compared with CELLO, as this is the only baseline that incorporates constraints. We observe that other than latency optimization under energy constraints for workload changes, CAMEO consistently outperforms CELLO for hardware, software, and deployment environment changes, e.g., under latency constraints, CAMEO finds configurations with 1.3× and 1.5× for software and deployment topology change, respectively.

Summary of observations From the above results, we also observe that the knowledge-reuse methods (CAMEO and RESTUNE) are consistently the top performers over the methods that do not reuse knowledge. The steep performance curves during the earlier iterations indicate that the optimization process's warm-starting helps quickly go to the region containing good configurations. As a result, across all environmental changes, methods that reuse knowledge from the source outperform SMAC, CELLO, and UNICORN, which do not rely on previous information and cannot reach the optimal within the allowed budget.

Why CAMEO works better? To further explain CAMEO's advantages over other methods, we conduct a case study using the setup for MLPERF OBJECT DETECTION pipeline deployed on TX2 as the source and XAVIER as the target mentioned in Section 2. We discuss our key findings below:

Table 4: Constrained optimization results for latency with energy and energy with latency constraints.

Environment Change	Latency w. Energy (RE%)		Energy w. Latency (RE%)	
	CELLO	CAMEO	CELLO	CAMEO
Hardware	16.8	9.7	14.1	13.9
Software	17.1	22.5	30.9	23.7
Workload	9.5	9.6	14.7	11.1
Deployment	14.3	11.4	16.7	11.3

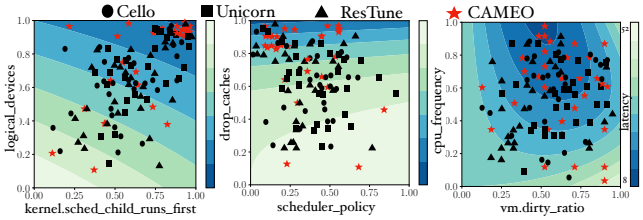


Figure 12: Contour plot with options with different causal effects. The color bar indicates latency values, whereas lower values (indicated by blue regions) indicate better performance.

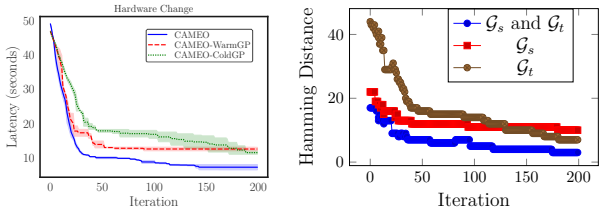


Figure 13: The causal performance models become more accurate with increasing iterations. The correctness of $\mathcal{G}_s$ and $\mathcal{G}_t$ when combined helps CAMEO in detecting the optimal configuration more effectively than other approaches. A lower hamming distance value indicates a smaller difference with the ground truth causal performance model in the target.

(i) The combined correctness of two causal performance models allows effectively identifying optimal options values. Table 5 shows the optimal configuration discovered by different approaches (highlighted when matched). CAMEO can correctly identify the maximum number of options values compared to other approaches with minimum latency value. This is possible due to the usage of two causal models $\mathcal{G}_s$ and $\mathcal{G}_t$ as when combined, they are nearly identical to the ground truth causal performance model in the target as shown in Figure 13.

(ii) CAMEO has utilized the budget more efficiently by carefully evaluating core configuration options. To better understand the optimization process, we visualize the response surfaces of three sets of options pairwise with different degrees of average causal effect (ACE) on latency (Figure 12). The leftmost subfigure of Figure 12 contains options with lower ACE values, whereas the rightmost subfigure of Figure 12 contains options with high ACE values only). The middle subfigure of Figure 12 contains options that have ACE values near the median (the ACE values of configuration options are provided in Table 5). We observe that the response surface of the options with higher ACE values is more complex than those with lower ACE values (rightmost subfigure of Figure 12). CAMEO is able to correctly determine the optimal values of cpu_frequency and dirty_ratio which indicates that CAMEO can understand such complex behavior better than others. Also, CAMEO has explored more configurations by varying core configurations options with higher ACE values than lower ones and better understands the response surface with effective resource utilization.

(iii) CAMEO reaches the better configuration by achieving better exploration-exploitation trade-offs. From Figure 12, we observe that CAMEO has higher coverage of the configurations

evaluated during the optimization procedure compared to other approaches. For example, in the rightmost subfigure of Figure 12, configurations evaluated by CAMEO cover the highest number of different regions (indicating better exploration). Here, we also observe that CAMEO has evaluated a higher number of configurations near the optimal (blue-colored) regions of the response surface (indicating better exploitation). The identification of core features has also enabled achieving better exploration-exploitation trade-offs. Therefore, CAMEO can learn about the regions with configuration options with lower causal effects within fewer explorations and focuses on exploitation behavior to quickly reach the optimum.

6 RQ2: Severity of Environmental Changes

The effectiveness of CAMEO changes due to the amount of distribution shift that can happen during environmental changes. Predicting how much the distribution will change when an environmental change occurs is impossible. Therefore, it is critical to understand how sensitive CAMEO is to different degrees of change severity. Following previous work [30], we consider various environmental changes of varying severity to answer this question. The scale and the number of changes that occur indicate the severity. For example, an environment change is more severe if both hardware and workload change, compared with only hardware changes.

We consider the centralized deployment of DEEPSTREAM used in RQ1 as the source and use the following as the targets: (i) *Low severity*: We only change one category, hardware (AGX XAVIER to XAVIER NX); (ii) *Medium severity*: We consider the change of two categories, hardware and deployment topology. In this setup, the target is deployed with DEEPSTREAM in a distributed fashion on two XAVIER NX devices with a decoder with four camera streams as the workload; and (iii) *High severity*: We consider a change of four categories, workload, deployment topology, hardware, and model. Our target has DEEPSTREAM distributedly deployed on two TX2s, with a workload of eight camera streams. We also change the detector from ResNet-18 to ResNet-50.

Results. As shown in Figure 14, CAMEO constantly outperforms the baselines by achieving maximum throughput for all severity of environmental changes. For example, CAMEO finds configuration with 1.3×, 1.5×, and 1.9× higher throughput than RESTUNE under low, medium, and high severity of changes, respectively. The KL divergence value between the distributions of the source and low, medium, and high severity environmental changes setup are 418, 951, and 1329. Therefore, we conclude that CAMEO performs better than baselines as the environmental changes become more severe.

7 RQ3: Sensitivity and Scalability

First, we investigate CAMEO's performance under different source measurements and how this affects the knowledge transferred from the source to the target and overall performance. Second, we determine how the value of l_α influences CAMEO's effectiveness. Finally, we investigate CAMEO's scalability in larger configuration space.

Sensitivity to the number of source measurements. We consider the MLPERF OBJECT DETECTION pipeline deployed on TX2 as the source and the same pipeline on XAVIER as the target, varying the number of measurements in TX2 from 30 to 10000 for evaluation and comparing their optimal values discovered by different approaches. As shown in Figure 15(a), increasing the number of source measurements positively influences CAMEO's as compared to RESTUNE. Including a greater number of source samples increases the danger of bias from the source environment, particularly when the distributions of two environments are extremely disparate. From this figure, we can infer that CAMEO is able to prevent those biases from getting introduced into the target as more samples are used for extracting knowledge from the source. We also observe that CAMEO reaches a plateau (after 2000 samples) faster than RESTUNE, indicating that CAMEO can find better configurations with fewer source samples. Since CAMEO can detect the core features which can be reliably used across environments without much modification.

Sensitivity to l_α value. One of the key parameters for CAMEO is λ that controls the amount of information from CGP_{warm} and CGP_{cold} for acquisition function calculation. The value of λ depends on l_α, which indicates the distance from the optimal configuration recommended by α_{warm}. A lower value of l_α can be interpreted as selecting configurations nearer to the optimal configuration, as it is expected to have similar behavior between source and target in the nearer regions. In this experiment, we vary the l_α value and record the RE value for each. The experimental results indicate that CAMEO achieves the minimum error when l_α is 1 (shown in Figure 15(b)).

Scalability to the number of configuration options. We consider a speech recognition pipeline that uses DEEPSPEECH [23] for inference. As workload, we use 2 hours of data extracted from 300 hours of test dataset of the COMMON VOICE dataset for 5 languages (English, Arabic, Chinese, German, and Spanish). We run inference on our CHAMELEON CLOUD INSTANCE with one P100 GPU for the source and one K80 GPU for the target. To evaluate the scalability of

Table 5: Optimal configuration discovered by different baselines. The configuration options are ranked in descending order based on their average causal effect (ACE) value on the performance objective, i.e., Latency.

Configuration Option	SMAC	UNICORN	RESTUNE-W/O-ML	RESTUNE	CAMEO	ACE	Optimal (Source)	Optimal (Target)	Default (Target)
cpu_frequency	1.3	1.6	1.6	1.6	2.0	0.19	1.4	2.0	1.1
vm.dirty_ratio	20	5	20	5	5	0.13	10	5	50
vm.swappiness	60	60	60	60	60	0.11	30	60	30
gpu_frequency	1.3	1.3	1.3	1.3	1.3	0.08	1.3	1.3	1.1
num_cores	3	4	3	4	4	0.06	3	4	2
memory_growth	0.5	0.9	0.5	0.9	-1	0.04	0.5	-1	0.5
emc_frequency	1.1	1.3	1.3	1.1	1.3	0.009	1.1	1.3	0.8
drop_caches	0	0	0	0	0	0.008	0	0	1
scheduler_policy	NOOP	NOOP	CFP	NOOP	NOOP	0.001	NOOP	NOOP	CFP
vm.vfs_cache_pressure	10	50	10	10	10	0.001	10	10	50
vm.dirty_bytes	30	60	60	30	60	0.0009	30	30	60
kernel.sched_rt_runtime_us	500000	500000	500000	500000	950000	0.0009	500000	9500000	500000
logical_devices	1	1	0	1	1	0.0008	1	1	0
kernel.sched_child_runs_first	0	0	0	0	0	0.0006	0	0	1
Latency	22s	15s	14s	13s	8s		7s	8s	48s

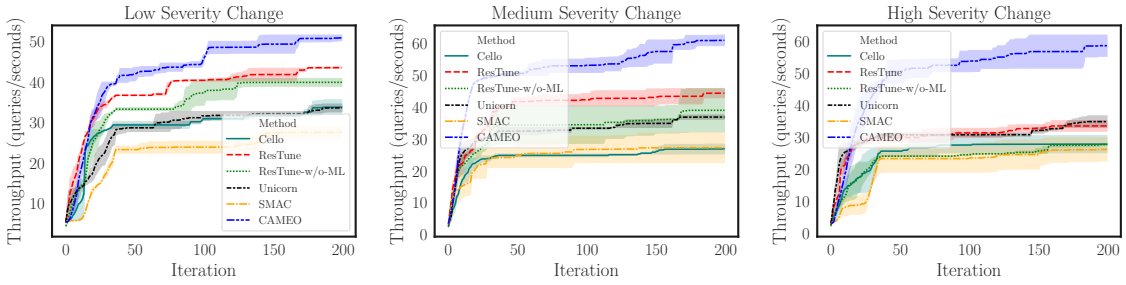


Figure 14: CAMEO achieves higher throughput when different severity of environmental changes happen in deployment.

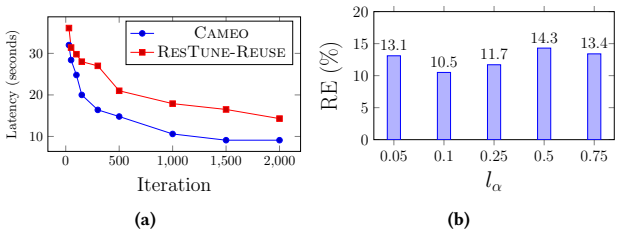


Figure 15: (a) Both approaches find better configurations when more samples are used in the source. Compared with RESTUNE, the optimal configuration found by CAMEO has lower minimum latency. (b) CAMEO has minimum RE when l_α is set to 0.1.

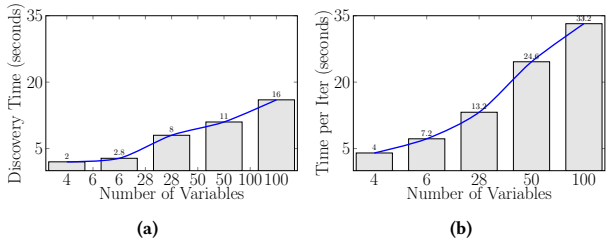


Figure 16: As the number of configuration options and system events increases, the causal graph discovery time (a) and total time per iteration (b) increase sub-linearly. Here, we do not include the time taken to measure a configuration in the optimization loop.

our approach to colossal configuration space [47]. In particular, we increase the number of variables from 4 to 100 and determine the discovery time and total time for each iteration using 300 samples in the target. Figure 16 indicates that the discovery time and time per iteration increase sub-linearly. Therefore, CAMEO is scalable to a large number of configuration options and system events. The scalability of CAMEO can be attributed to the sparsity of the causal graph, which leads to a small exploration set considered for the acquisition function.

8 Additional Related Work

Performance optimization in configurable systems. BO-based optimization methods discover the best configuration suited for a particular application and platform [44] to streamline compiler autotuning [7]. SCOPE [37] improves system performance and lowers safety constraint breaches by gathering system activity and switching from resource to execution space for exploration. CELLO [15] uses prediction-based early termination of sample collection by censored regression. Siegmund et al. [54] proposed a performance-influence model for configurable systems to understand the influence of configuration options on system performance using machine learning and sampling heuristics. Nevertheless, these techniques are platform-specific and unsuitable when a distribution shift occurs due to environmental changes. In comparison, CAMEO tackles the shift by transferring causal knowledge.

Transfer learning for performance modeling. It is possible to expedite the optimization process by transferring performance behavior knowledge from one environment to another. However, it is essential to identify which knowledge is necessary to be transferred to reach this aim. Jamshidi et al. [31] showed that when the environment changes are small, knowledge for forecasting performance can be transferred, while only knowledge for efficient sampling can be transferred when the environment changes are severe. Krishna et al. [39]determined the most relevant source of historical data to optimize performance modeling. Valov et al. [57] proposed a novel method for approximating and transferring the Pareto frontiers of optimal configurations across different hardware environments. Ballesteros et al. [5] proposed a transfer learning dynamic evolutionary algorithm to generate effective run-time quasi-optimal configurations of Dynamic Software Product Lines. All these techniques incorporate transfer learning based on correlational statistics (ML-based). However, Section 2.1 shows that ML-based models are prone to capturing spurious correlations. In comparison, CAMEO makes use of causal-based models, which identify invariant features despite environmental fluctuations.

Usage of causal analysis in configurable systems. Causal analysis has been used for various debugging and optimization tasks in configurable systems. Fariha et al. [17] proposed AID which intervenes through fault injection to pinpoint the root cause of intermittent failures in the software. Johnson et al. [36] proposed Causal Testing to analyze and fix software bugs by identifying a set of executions containing important causal information. Dubslaff et al. [16] proposed a method to compute feature causes effectively and leveraged them to facilitate root cause identification and feature effect/interaction estimation. The causality analysis in these works is solely on one environment. In contrast, our studied problem involves two environments (source and target) efficiently transferring the causal knowledge from source to target.

Table 6: Comparison of computation time in seconds required per iteration for different baselines compared to Cameo. Lower is better.

Method	Model Update Time	Configuration Recommendation Time	Total Time
Smac	5.6	9.2	58.1
cello	8.1	9.2	60.3
Unicorn	11.5	11.3	65.4
ResTune-w/o-ML	8.3	9.2	61.3
ResTune	9.7	9.7	63.4
Cameo	12.7	14.4	71.6

9 Limitations

Causal graph error. Causal discovery is an NP-hard problem [9]. Thus, it is possible that the found causal graphs are not ground-truth causal graphs and do not always reflect the true causal relationship among variables. However, as shown in many previous works [27, 16], such causal graphs can still be leveraged to achieve better performance than ML-based approaches on system optimization and debugging tasks as they avoid capturing spurious correlations.
Noisy Measurements. The system performance measurements are noisy and can affect the results. To mitigate this, we take each configuration's median of 5 runs.
Longer model computational time. Due to using two CGPs, the computational time of Cameo is higher than the baseline methods. For example, on average, Cameo takes 27.1s per iteration versus 19.4s per iteration taken by ResTune (see Table 6 for detailed results). However, this time is usually negligible compared to evaluation time (44s in our experiments). Besides, when the modeling time is included, Cameo also outperforms the baseline methods.

10 Conclusion

The goal of performance optimization of software systems is to minimize the number of queries required to accurately optimize a target black-box function in the production, given access to offline performance evaluations from the source environment and a significantly small number of performance evaluations from the target environment. When the environment changes, existing ML-based optimization methods tend to be sub-optimal since they are vulnerable to spurious correlations between configuration variables and the optimization performance goals (e.g., latency and energy). In this work, we propose Cameo, an algorithm that overcomes this limitation of existing ML-based optimization methods by querying data based on a combination of acquisition signals derived from training two Causal Gaussian Processes (CGPs): a cold-CGP operating directly in the input domain trained using the target data and a warm-CGP that operates in the feature space of a causal graphical model pre-trained using the source data. Such a decomposition can dynamically control the reliability of information derived from the online and offline data and the use of CGPs helps avoid capturing spurious correlations. Empirically, we demonstrate significant performance improvements of Cameo over existing performance optimization on real-world systems.

Acknowledgements

This work has been supported, in part, by National Science Foundation (Awards 2007202, 2107463, and 2233873). We also thank Chameleon Cloud for providing cloud resources for the experiments.

References

[1] On-line transaction processing benchmark. https://www.tpc.org/tpcc/.
[2] Mathieu Acher, Hugo Martin, Juliana Pereira, Arnaud Blouin, Jean-Marc Jézéquel, Djamel Khelladi, Luc Lesoil, and Olivier Barais. Learning very large configuration spaces: What matters for linux kernel sizes. 2019.
[3] Virginia Aglietti, Xiaoyu Lu, Andrei Paleyes, and Javier González. Causal bayesian optimization. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3155–3164. PMLR, 26–28 Aug 2020.
[4] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. {CherryPick}: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 469–482, 2017.
[5] Joaquín Ballesteros and Lidia Fuentes. Transfer learning for multiobjective optimization algorithms supporting dynamic software product lines. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume B*, pages 51–59, 2021.
[6] Marcel Blöcher, Lin Wang, Patrick Eugster, and Max Schmidt. Switches for hire: resource scheduling for data center in-network computing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 268–285, 2021.
[7] Junjie Chen, Ningxin Xu, Peiqi Chen, and Hongyu Zhang. Efficient compiler autotuning via bayesian optimization. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1198–1209. IEEE, 2021.
[8] Tao Chen and Miqing Li. Do performance aspirations matter for guiding software configuration tuning? an empirical investigation under dual performance objectives. *ACM Transactions on Software Engineering and Methodology*, 32(3):1–41, 2023.
[9] David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-sample learning of bayesian networks is np-hard. *J. Mach. Learn. Res.*, 5:1287–1330, dec 2004.
[10] Alexei Colin, Emily Ruppel, and Brandon Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 767–781, 2018.
[11] Diego Colombo and Marloes H Maathuis. Order-independent constraint-based causal structure learning. *The Journal of Machine Learning Research*, 15(1):3741–3782, 2014.
[12] Diego Colombo, Marloes H Maathuis, Markus Kalisch, and Thomas S Richardson. Learning high-dimensional directed acyclic graphs with latent and selection variables. *The Annals of Statistics*, pages 294–321, 2012.
[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
[14] Yi Ding, Ahsan Pervaiz, Michael Carbin, and Henry Hoffmann. Generalizable and interpretable learning for configuration extrapolation. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 728–740, 2021.
[15] Yi Ding, Alex Renda, Ahsan Pervaiz, Michael Carbin, and Henry Hoffmann. Cello: Efficient computer systems optimization with predictive early termination and censored regression. *arXiv preprint arXiv:2204.04831*, 2022.
[16] Clemens Dubslaff, Kallistos Weis, Christel Baier, and Sven Apel. Causality in configurable software systems. *arXiv preprint arXiv:2201.07280*, 2022.
[17] Anna Fariha, Suman Nath, and Alexandra Meliou. Causality-guided adaptive interventional debugging. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 431–446, 2020.
[18] Clark Glymour, Kun Zhang, and Peter Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in genetics*, 10:524, 2019.
[19] Kourosh Hakhamaneshi, Pieter Abbeel, Vladimir Stojanovic, and Aditya Grover. Jumbo: Scalable multi-task bayesian optimization using offline data. *arXiv preprint arXiv:2106.00942*, 2021.
[20] Hassan Halawa, Hazem A. Abdelhafez, Andrew Boktor, and Matei Ripeanu. NVIDIA jetson platform characterization. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 10417 LNCS:92–105, 2017.
[21] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. Test them all, is it worth it? assessing configuration sampling on the jhipster web development stack. *Empirical Software Engineering*, 24(2):674–717, 2019.
[22] Greg Hamerly and Charles Elkan. Learning the k in k-means. *Advances in neural information processing systems*, 16, 2003.
[23] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[24] Chin-Jung Hsu, Vivek Nair, Tim Menzies, and Vincent W Freeh. Scout: An experienced guide to find the best cloud configuration. *arXiv preprint arXiv:1803.01296*, 2018.

[25] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[26] Md Shahriar Iqbal, Lars Kotthoff, and Pooyan Jamshidi. Transfer Learning for Performance Modeling of Deep Neural Network Systems. In *USENIX Conference on Operational Machine Learning*, Santa Clara, CA, 2019. USENIX Association.

[27] Md Shahriar Iqbal, Rahul Krishna, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. Unicorn: reasoning about configurable system performance through the lens of causality. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 199–217, 2022.

[28] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Autonomic resource provisioning for cloud-based software. In *Proceedings of the 9th international symposium on software engineering for adaptive and self-managing systems*, pages 95–104, 2014.

[29] Pooyan Jamshidi and Giuliano Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In *Proc. Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2016.

[30] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. Transfer learning for performance modeling of configurable systems: An exploratory analysis. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*. ACM, 2017.

[31] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. Transfer learning for performance modeling of configurable systems: An exploratory analysis. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 497–508. IEEE, 2017.

[32] Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems. In *Proc. Int'l Symp. Foundations of Software Engineering (FSE)*. ACM, 2018.

[33] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. Transfer learning for improving model predictions in highly configurable software. In *Proc. Int'l Symp. Soft. Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2017.

[34] Mohammad Ali Javidian, Om Pandey, and Pooyan Jamshidi. Scalable causal transfer learning. *arXiv preprint arXiv:2103.00139*, 2021.

[35] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[36] Brittany Johnson, Yuriy Brun, and Alexandra Meliou. Causal testing: Understanding defects' root causes. In *Proceedings of the 2020 International Conference on Software Engineering*, 2020.

[37] Hyunji Kim, Ahsan Pervaiz, Henry Hoffmann, Michael Carbin, and Yi Ding. Scope: Safe exploration for dynamic computer systems optimization. *arXiv preprint arXiv:2204.10451*, 2022.

[38] Murat Kocaoglu, Alexandros G. Dimakis, Sriram Vishwanath, and Babak Hassibi. Entropic causal inference. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, page 1156–1162, 2017.

[39] Rahul Krishna, Vivek Nair, Pooyan Jamshidi, and Tim Menzies. Whence to learn? transferring knowledge in configurable systems using beetle. *IEEE Transactions on Software Engineering*, 2020.

[40] Luc Lesoil, Hugo Martin, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. Transferring performance between distinct configurable systems: A case study. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems*, pages 1–6, 2022.

[41] Xu-Qing Liu and Xin-Sheng Liu. Markov blanket and markov boundary of multiple variables. *The Journal of Machine Learning Research*, 19(1):1658–1707, 2018.

[42] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[43] Hugo Martin, Mathieu Acher, Luc Lesoil, Jean Marc Jezequel, Djamel Eddine Khelladi, and Juliana Alves Pereira. Transfer learning across variants and versions: The case of linux kernel size. *IEEE Transactions on Software Engineering*, 2021.

[44] Harshitha Menon, Abhinav Bhatele, and Todd Gamblin. Auto-tuning parameter choices in hpc applications using bayesian optimization. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 831–840. IEEE, 2020.

[45] Yifei Ming, Hang Yin, and Yixuan Li. On the impact of spurious correlation for out-of-distribution detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10051–10059, 2022.

[46] Juan Miguel Ogarrio, Peter Spirtes, and Joe Ramsey. A hybrid causal search algorithm for latent variable models. In *Conference on Probabilistic Graphical Models*, pages 368–379, 2016.

[47] JEHO OH, D Batory, and RUBÉN HERADIO. Finding near-optimal configurations in colossal spaces with statistical guarantees. 2022.

[48] Claus Pahl, Pooyan Jamshidi, and Olaf Zimmermann. Architectural principles for cloud software. *ACM Transactions on Internet Technology (TOIT)*, 18(2):1–23, 2018.

[49] Judea Pearl. *Causality*. Cambridge university press, 2009.

[50] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020.

[51] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015.

[52] Mehran Salmani, Saeid Ghafouri, Alireza Sanaee, Kamran Razavi, Max Mühlhäuser, Joseph Doyle, Pooyan Jamshidi, and Mohsen Sharifi. Reconciling high accuracy, cost-efficiency, and low latency of inference serving systems. In *Proceedings of the 3rd Workshop on Machine Learning and Systems*, pages 78–86, 2023.

[53] Norbert Siegmund, Johannes Dorn, Max Weber, Christian Kaltenecker, and Sven Apel. Green configuration: Can artificial intelligence help reduce energy consumption of configurable software systems? *Computer*, 55(3):74–81, 2022.

[54] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 284–294, 2015.

[55] Moisés Silva-Muñoz, Alberto Franzin, and Hugues Bersini. Automatic configuration of the cassandra database using irace. *PeerJ Computer Science*, 7:e634, 2021.

[56] Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. *Causation, prediction, and search*. MIT press, 2000.

[57] Pavel Valov, Jianmei Guo, and Krzysztof Czarnecki. Transferring pareto frontiers across heterogeneous hardware environments. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, pages 12–23, 2020.

[58] Miguel Velez, Pooyan Jamshidi, Florian Sattler, Norbert Siegmund, Sven Apel, and Christian Kästner. Configcrusher: Towards white-box performance analysis for configurable systems. *Automated Software Engineering*, 27:265–300, 2020.

[59] Miguel Velez, Pooyan Jamshidi, Norbert Siegmund, Sven Apel, and Christian Kästner. White-box analysis over machine learning: Modeling performance of configurable systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1072–1084. IEEE, 2021.

[60] Miguel Velez, Pooyan Jamshidi, Norbert Siegmund, Sven Apel, and Christian Kästner. On debugging the performance of configurable software systems: Developer needs and tailored tool support. In *2022 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2022.

[61] Luping Wang, Lingyun Yang, Yinghao Yu, Wei Wang, Bo Li, Xianchao Sun, Jian He, and Liping Zhang. Morphling: fast, near-optimal auto-configuration for cloud-native model serving. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 639–653, 2021.

[62] Shu Wang, Chi Li, Henry Hoffmann, Shan Lu, William Sentosa, and Achmad Imam Kistijantoro. Understanding and auto-adjusting performance-sensitive configurations. *ACM SIGPLAN Notices*, 53(2), 2018.

[63] James Wilson, Frank Hutter, and Marc Deisenroth. Maximizing acquisition functions for bayesian optimization. *Advances in neural information processing systems*, 31, 2018.

[64] Fan Wu, Westley Weimer, Mark Harman, Yue Jia, and Jens Krinke. Deep parameter optimisation. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1375–1382, 2015.

[65] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015.

[66] Nezih Yigitbasi, Theodore L Willke, Guangdeng Liao, and Dick Epema. Towards machine learning-based auto-tuning of mapreduce. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 11–20. IEEE, 2013.

[67] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. Restune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2102–2114, 2021.

[68] Chunting Zhou, Xuezhe Ma, Paul Michel, and Graham Neubig. Examining and combating spurious features under distribution shift. In *International Conference on Machine Learning*, pages 12857–12867. PMLR, 2021.

A Appendix.

A.1 Definitions and Background

Configuration Space O. Let O_i indicate the i^{th} configuration option of a system, which can be set to a range of different values (e.g., categorical, boolean, and numerical). The configuration space is a Cartesian product of all options $O = Dom(O_1) \times ... \times Dom(O_d)$, where d is the number of options. A configuration o is then a member of the configuration space O in which all options are set to a given value within the range of permitted values for that option.
Environment Space $\mathcal{E}$. We describe an environment e drawn from a given environment space $\mathcal{E}$, which consists of possible combinations of hardware, workload, software, and deployment topology.
Causal Performance Model $\mathcal{G}$. A causal performance model (CPM), denoted by $\mathcal{G}$, is an acyclic-directed mixed graph (ADMG) that provides the functional dependencies (e.g., how variations in one or multiple variables determine variations in other variables) between configuration options, system events, and performance objectives. While interpreting a CPM, we view the nodes as variables and the arrows as causal connections.
Observation. In the *observational* formulation, we measure the distribution of an outcome variable (e.g., latency $\mathcal{Y}$) given that we *observe* another variable (e.g., cpu frequency O_i for $1 \leq i \leq d$) taking a certain value o_i (e.g., $O_i = o_i$), denoted by $Pr(\mathcal{Y} \mid O_i = o_i)$.
Intervention. The *interventional* inference tackles a harder task of estimating the effects of deliberate actions. For example, we measure how the distribution of an outcome (e.g., latency $\mathcal{Y}$) would change if we (artificially) intervened during the data gathering process by forcing the variable cpu frequency O_i to a certain value o_i, but otherwise retain the other variables (e.g., dirty ratio) as is. We can estimate the outcome of the artificial intervention by modifying the CPM to reflect our intervention and applying Pearl's *do-calculus* [49], which is denoted by $Pr(\mathcal{Y} \mid do(O_i = o_i))$. Unlike observations, there is a structural change in CPM due to intervention that goes along with a change in a probability distribution over the variables.
Bayesian Optimization. Bayesian Optimization (BO) is an efficient framework to solve global optimization problems using black-box evaluations of expensive performance objectives $\mathcal{Y}$. A typical BO approach consists of two main elements: the surrogate model and the acquisition function. The surrogate models are trained with a small number of configuration measurements and are used to predict the objective functions value $\hat{\mathcal{Y}} = f(o)$ using predictive mean $\mu(o)$ and uncertainty $\sigma(o)$ for configurations $o \in O$. A common practice is to use Gaussian processes (GPs) as surrogate models where the GP distribution over $f(o)$ is fully specified by its mean function, its mean function $\mu(o)$, and its covariance function $k_c(o, o')$. The kernel or covariance function k_c captures the regularity in the form of the correlation of the marginal distributions $f(o)$ and $f(o')$. After the surrogate model outputs predictive mean and uncertainty for the unseen configurations, Cameo needs an acquisition function to select the best configuration to sample. A good acquisition function should balance the trade-offs between exploration and exploitation.

A.2 Additional Details for Evaluation

Tables 7 to 16 and Figures 18 to 21.

Table 7: Prediction errors in each environment.

Environment	Prediction Error (%)		
	GPR	RFR	CGPR
TX1	11.2	12.8	9.2
TX2	10.7	12.2	9.1
Xavier	13.2	12.4	8.8

Table 8: Hardware configuration options.

Configuration Options	Option Values/Range
num_cores	1 - 4
cpu_frequency	0.3 - 2.0 (GHz)
gpu_frequency	0.1 - 1.3 (GHz)
emc_frequency	0.1 - 1.8 (GHz)

Table 9: Linux OS/Kernel configuration options.

Configuration Options	Option Values/Range
vm.vfs_cache_pressure	1, 100, 500
vm.swappiness	10, 60, 90
vm.dirty_bytes	30, 60
vm.dirty_background_ratio	10, 80
vm.dirty_background_bytes	30, 60
vm.dirty_ratio	5, 10, 20, 50
vm.nr_hugepages	0, 1, 2
vm.overcommit_ratio	50, 80
vm.overcommit_memory	0, 2
vm.overcommit_hugepages	0, 1, 2
kernel.cpu_time_max_percent	10 - 100
kernel.max_pids	32768, 65536
kernel.numa_balancing	0, 1
kernel.sched_latency_ns	24000000, 48000000
kernel.sched_nr_migrate	32, 64, 128
kernel.sched_rt_period_us	1000000, 2000000
kernel.sched_rt_runtime_us	500000, 950000
kernel.sched_time_avg_ms	1000, 2000
kernel.sched_child_runs_first	0, 1
swap_memory	1, 2, 3, 4 (GB)
scheduler.policy	CFP, NOOP
drop_caches	0, 1, 2, 3

Table 10: Configuration options in Mlperf Object Detection, and Speech Recongition software system.

Configuration Options	Option Values/Range
memory_growth	-1, 0.5, 0.9
logical_devices	0, 1
inter_op_parallelism_threads	1, num cpus
intra_op_parallelism_threads	1, num cpus

A.3 RQ1 Additional Results

Figures 22 to 25.

A.4 RQ2 Additional Results

Figure 26.

Table 11: Configuration options in NLP software system.

Configuration Options	Option Values/Range
precision	8,16
distributed_backend	ddp, dp
num_workers	0, num gpus, 4× num gpus

Table 12: DEEPSTREAM software configuration options.

Component	Configuration Options	Option Values/Range
Decoder	CRF	13, 18, 24, 30
	bitrate	1000, 2000, 2800, 5000
	buffer_size	6000, 8000, 20000
	presets	ultrafast, very fast, faster medium, slower
	maximum_rate	600k, 1000k
	refresh	OFF, ON
Stream Mux	batch_size	0 - 30
	batched_push_timeout	0 - 20
	num_surfaces_per_rame	1, 2, 3, 4
	enable_padding	0, 1
	buffer_pool_size	1 - 26
	sync_inputs	0, 1
	nvbuf_memory_type	0, 1, 2, 3
Nvinfer	net_scale_factor	0.01 - 10
	batch_size	1 - 60
	interval	1 - 20
	offset	0, 1
	process_mode	0, 1
	use_dla_core	0, 1
	enable_dla	0, 1
	enable_dbscan	0, 1
	secondary_reinfer_interval	0 - 20
	maintain_aspect_ratio	0, 1
Nvtracker	iou_threshold	0 - 60
	enable_batch_process	0, 1
	enable_past_frame	0, 1
	compute_hw	0, 1, 2, 3, 4

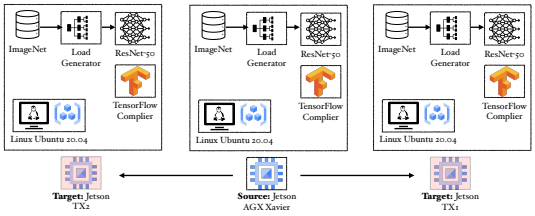


Figure 17: Experimental setup when hardware changes from XAVIER in the source to TX2 and TX1 in the target, separately, for MLPERF OBJECT DETECTION pipeline.

Table 13: CASSANDRA configuration options.

Configuration Options	Option Values/Range
concurrent_writes	32, 128, 512
file_cache_size	256, 512, 2048
memtable_cleanup	0.1, 0.3, 0.6
concurrent_compact	0.1, 0.3, 0.6
compaction_methods	SizeTiered, LeveledCompaction
num_tokens	256, 512, 1024
concurrent_reads	32, 64, 128
replication_factor	1, 2, 3
memtable_heap_space	256, 1024, 2048
memtable_allocation	heap, buffers
row_cache_size_in_mb	0, 1
sstable_open_interval	30, 50, 100
trickle_fsync	0, 1
inter_dc_stream	100, 200
key_cache_ssize	100, 200
stream_throughput	100, 200
row_cache_save	0, 1
column_index_size	16, 32, 64
compaction_throughput	16, 32, 64
memtable_offheap_space	256, 1024, 2048
commitlog_segment	32, 64, 256
mem_flush_writers	1, 2, 3
index_summary	100, 150

Table 14: Performance system events and tracepoints.

System Events
context_switches
major_faults
minor_faults
migrations
scheduler_wait_time
scheduler_sleep_time
cycles
instructions
number_of_syscall_enter
number_of_syscall_exit
l1_dcache_load_misses
l1_dcache_loads
l1_dcache_stores
branch_loads
branch_loads_misses
branch_misses
cache_references
cache_misses
emulation_faults
Tracepoint Subsystems
Block
Scheduler
IRQ
ext4

Table 15: Hyperparameters for DNNs used in CAMEO.

Architecture	Hyperparameters	Option Values
RESNET	num_filters_entry flow	32
	filter_size_entry_flow	(3×3)
	num_filters_middle_flow	64
	filter_size_middle_flow	(3×3)
	num_filters_exit_flow	728
	filter_size_exit_flow	(3×3)
	batch_size	32
	num_epochs	100
BERT	dropout	0.3
	maximum_batch_size	16
	maximum_sequence_length	13
	learning_rate	$1e^{-4}$
	weight_decay	0.3
DEEPSPEECH	dropout	0.3
	maximum_batch_size	16
	maximum_sequence_length	32
	learning_rate	$1e^{-4}$
	num_epochs	10

Table 16: Hyperparameters for FCI used in CAMEO.

Hyperparameters	Value
depth	-1
test_id	fisher-z-test
maximum_path_length	-1
complete_rule_set_used	False

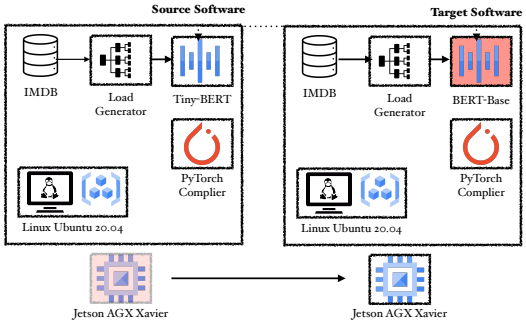


Figure 18: Experimental setup when a software change takes place from TinyBERT to BERT-Base in the target.

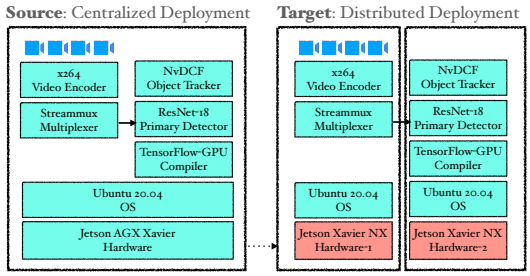


Figure 19: Experimental setup for our experiments when the deployment topology is changed from centralized to distributed in the target in the target using two XAVIER NX.

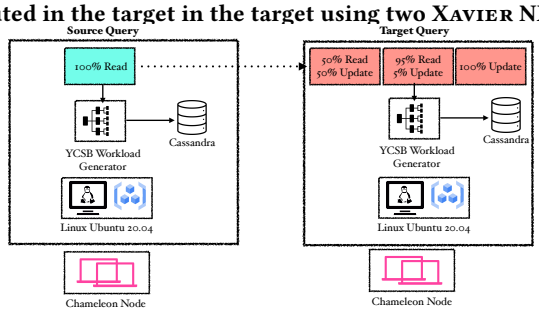


Figure 20: Experimental setup when the type of workload is different with a CASSANDRA database where the source uses a READ ONLY workload where the target uses a BALANCED and UPDATE HEAVY workload, separately.

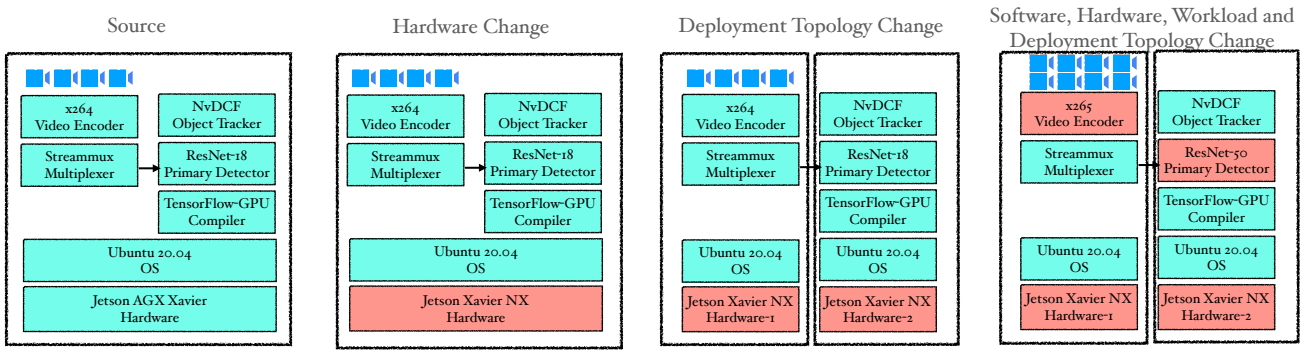


Figure 21: Experimental setup for different severity of environmental changes. Low severity change scenario when only hardware changes from Xavier to Xavier NX in the target (second figure). We change the hardware and deployment topology for the medium severity change scenario (third figure). For high-severity environmental changes experiments, the primary detector is changed from ResNet-18 to ResNet-50, the decoder is changed from x264 to x265 with a different deployment topology from the source distributed with two Xavier NX hardware that is different from the source as well (fourth figure).

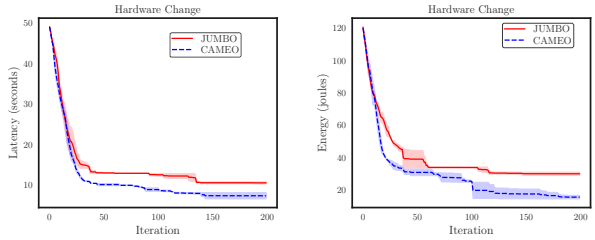


Figure 22: Effectiveness of Cameo with Jumbo when hardware changes in the deployment environment.

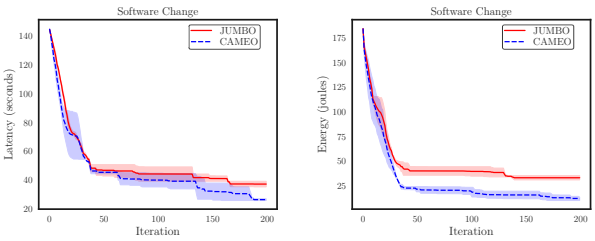


Figure 23: Effectiveness of Cameo with Jumbo when software changes in the deployment environment.

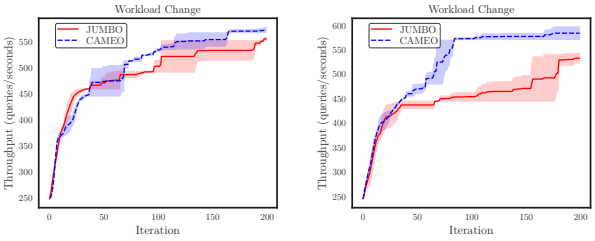


Figure 24: Effectiveness of Cameo with Jumbo when workload changes in the deployment environment.

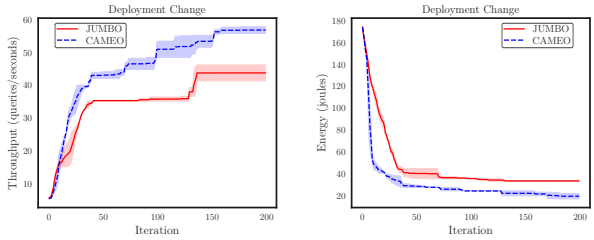


Figure 25: Effectiveness of Cameo with Jumbo when deployment topology changes in the deployment environment.

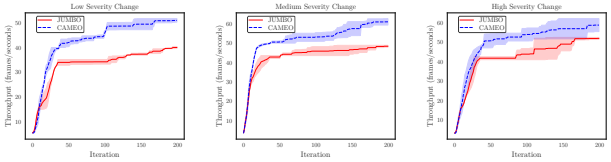


Figure 26: Effectiveness of Cameo when different severity of environmental changes happen in the deployment environment.