

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

Md Shahriar Iqbal
University of South Carolina

Ziyuan Zhong
Columbia University

Iftakhar Ahmad
University of South Carolina

Baishakhi Ray
Columbia University

Pooyan Jamshidi
University of South Carolina

Abstract

Modern computer systems are highly configurable, with hundreds of configuration options that interact, resulting in an enormous configuration space. As a result, optimizing performance goals (e.g., latency) in such systems is challenging due to frequent uncertainties in their environments (e.g., workload fluctuations). Recently, transfer learning has been applied to address this problem by reusing knowledge from configuration measurements from the source environments, where it is cheaper to intervene than the target environment, where any intervention is costly or impossible. Recent empirical research showed that statistical models can perform poorly when the deployment environment changes because the behavior of certain variables in the models can change dramatically from source to target. To address this issue, we propose CAMEO—a method that identifies invariant causal predictors under environmental changes, allowing the optimization process to operate in a reduced search space, leading to faster optimization of system performance. We demonstrate significant performance improvements over state-of-the-art optimization methods in MLPerf deep learning systems, a video analytics pipeline, and a database system.

CCS Concepts

- Machine Learning for Systems → Edge Computing.

Keywords

Highly Configurable Systems, Performance Optimization, Causal Transfer Learning, Resource Constraints

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SoCC '23, October 30–November 1, 2023, Santa Cruz, CA, USA

© 2023 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-XXXX-X/18/06.

<https://doi.org/10.1145/1122445.1122456>

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 *ACM Symposium on Cloud Computing (SoCC '23), October 30–November 1, 2023, Santa Cruz, CA, USA*. ACM, New York, NY, USA, 22 pages. <https://doi.org/10.1145/1122445.1122456>

1 Introduction

Modern computer systems are continuously deployed in heterogeneous environments (e.g., cloud, FPGA, SoC) 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 [15, 61]. 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 [6, 10, 65].

Finding an optimal configuration in a highly configurable system, however, is challenging [2, 8, 21, 29, 60, 62]: (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

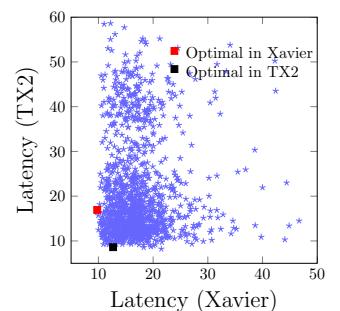


Figure 1: The optimal configuration for MLPerf Object Detection pipeline deployed on TX2 is not optimal in Xavier.

user requirements and reduce service management costs, underlying systems often undergo environmental changes, that is, hardware updates, changes in deployment topology, etc. [14]. Therefore, optimizing the performance of these evolving systems becomes even more challenging since there is no guarantee that the optimal configurations found in one environment will remain optimal in a different environment [29, 30, 32]¹ as shown in Figure 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. 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 target production settings for downstream performance optimizations or debugging. However, in most cases, the staging environment result is completely different from the production result, resulting in a misleading or even wrong indication about the configurations that produce optimal performance. These differences in the results occur mainly due to the hardware gap or workload differences between the development environment and the production environment. 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) [4, 25, 29, 32, 44, 64, 66], 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 may be 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

¹we 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	RESUME-W/O-ML	RESUME	CAMEO
Detects Spurious Features	✗	✗	✓	✗	✗	✓
Handles Distribution Shift	✗	✗	✗	✗	✓	✓
Suitable for Benchmarks	✓	✓	✓	✗	✓	✓
Knowledge Reuse	✗	✗	✗	✗	✓	✓
Constrained Optimization	✗	✓	✗	✓	✓	✓

performance evaluations of different configurations from a staging environment. Exploiting this additional information using transfer learning can result in efficient optimization, as demonstrated by recent work [26, 32, 33, 39, 40, 43]. For example, searching for optimized performance in the target setting can use the summary statistics of the models built using the performance of the source [67]. However, each environmental change can potentially cause a distribution shift. The ML models used in these transfer learning methods are vulnerable to spurious correlations, which do not hold between distribution shifts and result in inferior performance [27, 45, 68] (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 [16, 27, 53] to build a causal performance model² that captures the dependencies 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 the learning and optimization of the performance model.

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 needs 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 that by 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 observational data in

²A 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) between configuration options, system events, and performance objectives.

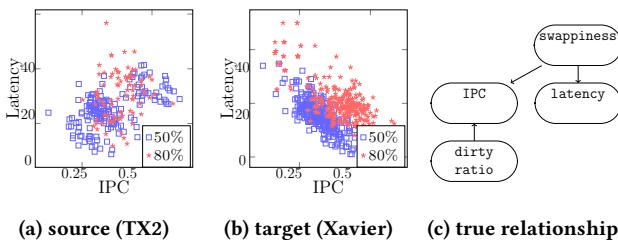


Figure 2: (a)-(b) The relationship between IPC and latency reverse from source (TX2) to target (XAVIER) while the relationship between swappiness (the values are denoted as colors) and latency stays invariant. (c) The true causal relationship among the relevant variables.

the production environment. Therefore, we will reduce the cost of optimization tasks without compromising accuracy.

To this end, we propose CAMEO (**C**ausal **M**ulti **E**nvironment **O**ptimization), a causal transfer-based optimization algorithm aimed at overcoming the limitation of prior approaches. Our approach is built on top of two previous works, JUMBO (a multitask 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 the configuration options, system events, and performance objectives for each environment using the previous performance measurements of the source environment and a considerably smaller number of measurements of 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 the source and target. This way of combining individual acquisition functions of both CGPs allows one to rely only 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.

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

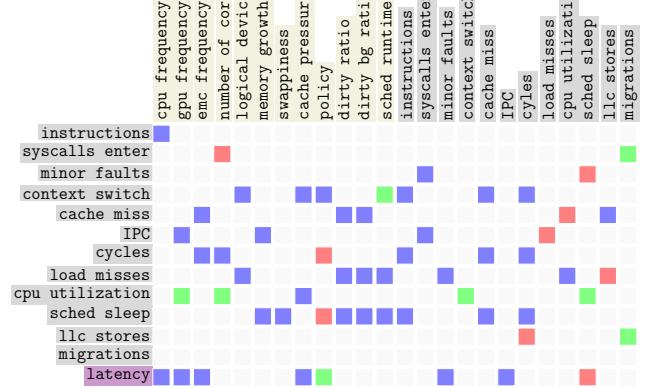


Figure 3: There is a significant overlap between the causal structures (the common edges are represented as blue squares) developed in different environments (e.g., Jetson TX2 and Xavier). Some edges unique to the source (green squares) or target (red squares) also exist.

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 \times and energy by 5.6 \times on average than the best baseline optimization approach, RESTUNE.

Contributions. Our contributions are as follows:

- We propose CAMEO, a novel causal transfer-based approach that allows faster optimization of software systems when the environment changes. CAMEO is one of the first approaches to use causal transfer learning for the optimization of the performance of configurable systems.
- We conducted a comprehensive evaluation of CAMEO by comparing it with state-of-the-art optimization methods in five highly configurable systems in the real world under a range of different environmental changes and studied the effectiveness of design explorations with different varieties and severity of environmental changes and showed 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 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 part of the MLPerf Inference Benchmark³ following the benchmark

³<https://mlcommons.org/en/inference-edge-30/>

rules⁴, 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 2,000 random configurations and measured the inference latency in each environment. We also collected performance counters and system events statistics using *Linux perf profiler*⁵.

2.1 Why performance optimization using causal reasoning is more effective?

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 to predict 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 swappiness⁶ and dirty ratio⁷ 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 IPC⁸ (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 because 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 occurs 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%,

⁴https://github.com/mlcommons/inference_policies/blob/master/inference_rules.adoc

⁵<https://perf.wiki.kernel.org/>

⁶swappiness 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.

⁷dirty 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.

⁸IPC 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

respectively) and observe that the correlation between swappiness and latency 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. On the contrary, 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 causal model can better capture the *data generation process* as it only relies on 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 better explorations in limited budget scenarios.

To show the benefits of correctly identifying the invariant features, we train different ML-based regressors, e.g., the Gaussian Process Regressor (GPR) and the 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.

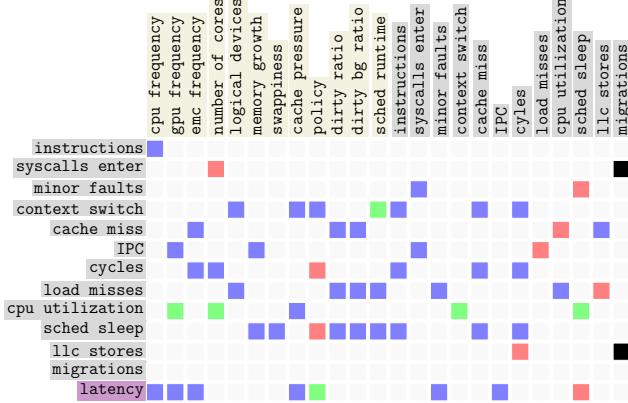


Figure 4: Combining the top K nodes' Markov blankets eliminates the wrong biases (shown as black squares).

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 the 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 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.

To eliminate biases, we need to remove unique edges of the source. Removal operations 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 collection process by forcing the variable $\text{cpu frequency } O_i$ to a certain value o_i while retaining the other variables as is. We can estimate the outcome of the

intervention by modifying the causal performance model 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 interventions on nodes with limited causal effects on the performance objective. For this purpose, we rank the causal effects of all existing nodes on latency

and observe that only one source-specific edge (policy) is among the top 10 most influential nodes. Therefore, we can select the 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 5(b)), are eliminated. Figure 5(a) shows that pruning the edges helps to reach the optimal value 19% faster. Therefore, *we require an approach that relies on intervening only in the top K nodes based on the source knowledge in the target environment*.

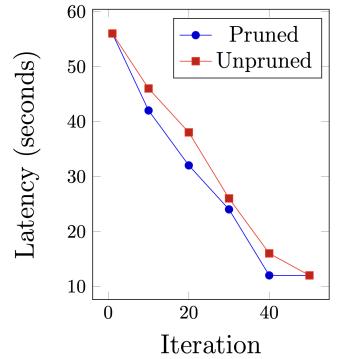


Figure 5: Pruning edges with a Markov blanket identifies the optimal configuration faster.

Takeaway 2 Employing rich knowledge in a causal performance model, we can intervene in specific configurations to learn the most about the underlying causal structure and be able to gather the most relevant data under a limited budget.

3 CAMEO Design

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

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

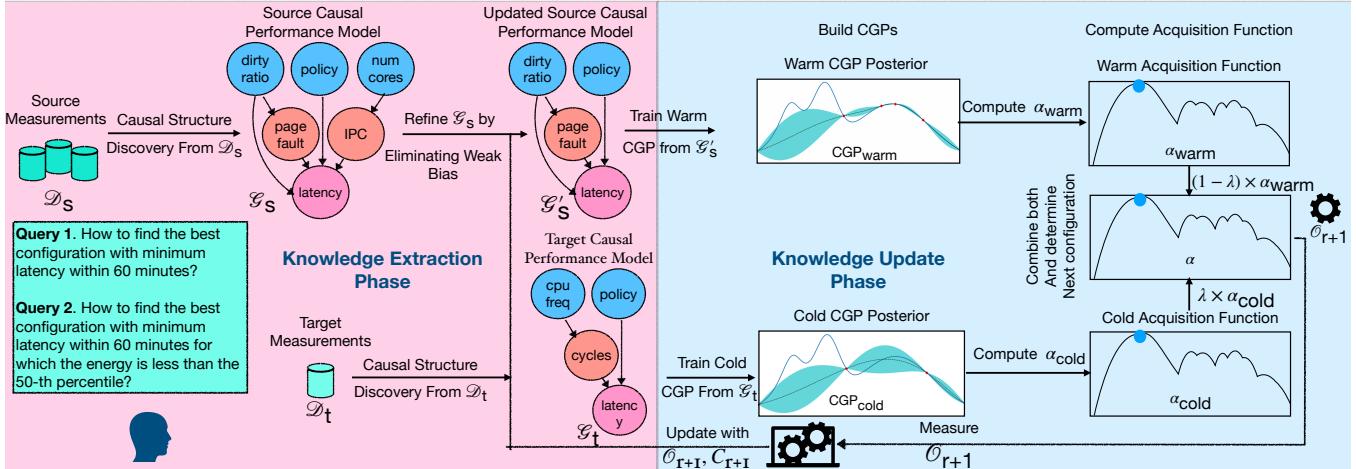


Figure 6: Overview of CAMEO.

and numerical). The configuration space is a Cartesian product of all hardware, software, and application-specific options: $\mathcal{O} = \text{Domain}(\mathcal{O}_1) \times \dots \times \text{Domain}(\mathcal{O}_d)$, where d is the number of options. Configuration options and system events are jointly represented as a vector $\mathcal{X} = (\mathcal{O}, \mathcal{C})$. 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. The task is to find a near-optimal configuration, \mathcal{O}^* , with a fixed measurement budget, β , in the target environment, e_t , that results in Pareto-optimal performance:

$$\mathcal{O}^* = \operatorname{argmin}_{\mathcal{O} \in \mathcal{O}} \mathcal{Y}_{e_t}(\mathcal{O}) \text{ within } \beta, \quad (1)$$

where \mathcal{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 6 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 G_s using cheaper offline performance measurements 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 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 G_s with a new causal performance model G_t collected from a small number of samples, 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 G_s and G_t , respectively. The two CGPs operate on different input spaces. CGP_{warm} works on a reduced configuration space that is derived from 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 , the more we rely 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., the maximum budget β is exhausted or convergence is achieved). The pseudocode for our approach is provided in Algorithm 1.

3.3 Knowledge Extraction Phase

We next describe the offline knowledge extraction phase.

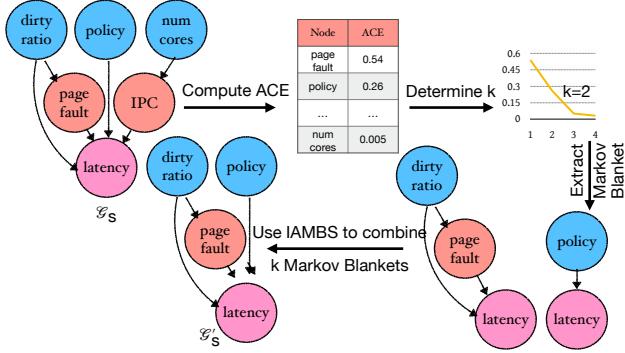


Figure 7: Refining the causal performance model from the source to eliminate unwanted information.

User query translation. A developer can use CAMEO to find the optimal configurations that optimize 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 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. The query translator extracts this information by directly accepting user inputs with some fixed guided keyword directives.

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. 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 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 the existence of unobserved confounders [18, 46, 56]. This is crucial because we do not assume absolute knowledge of configuration space, so there may be configurations in which we cannot intervene 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’s z test for continuous variables and mutual information for discrete variables) to remove edges between independent variables. Finally, we orient undirected edges using prescribed edge orientation rules [11, 12, 18, 46, 56] 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 based on the invariant features, we may be tempted to directly reuse the 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 try 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 according to their causal effects on the performance objective to eliminate unwanted information. Higher causal effects indicate a stronger influence of the configuration option on performance. When scaling option values within a constant context, options with higher causal effects become top features. 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, the Mb of a node 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 they provide a thorough picture of the local causal structure around the variable. Initially, we determine k using the method proposed in [22]. Then we extract the Mb of the k nodes to determine the final \mathcal{G}_s that will be reused in the subsequent phase using the IAMBS algorithm presented in [41]. The IAMBS algorithm is focused on constructing an Mb for multiple variables (top k nodes). It operates by determining whether the additivity property holds for Mb of k variables and, 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 use the knowledge gained from the earlier phase to guide the optimization search strategy 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}_s and \mathcal{G}_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, allowing the surrogate model to capture the behavior of the performance objective better than GPs (as shown in Figure 17), particularly in areas where observational data are 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}_s and \mathcal{G}_t as the following:

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

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

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

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}_s or \mathcal{D}_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 hyperparameter. 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}_s and \mathcal{G}_t , and compute the mean and uncertainty estimates for the configurations in the exploration set.

Compute acquisition function for sampling. Denote by $\alpha_{\text{warm}}^r(o)$ and $\alpha_{\text{cold}}^r(o)$ to be the single objective acquisition functions of the two CGPs. For CAMEO, we choose to use expected improvement (EI) as an acquisition function [63] since EI has been shown to perform well in the configuration search. EI selects the configuration that would have the highest expected improvement to the current best interventional setting separately from e_s and e_t in all configurations in the respective exploration set:

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

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 the $\alpha_{\text{warm}}^r(o)$ scores and then select the ones with the highest $\alpha_{\text{cold}}^r(o)$ score. Our acquisition function is defined as the following:

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

where λ^r is an interpolation coefficient 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 when 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:

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

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

In contrast, configurations that are further away from the optimal points of α_{warm} (configurations that do not satisfy $l_\alpha \leq 0.1$) will have a higher expected improvement value for α_{warm} . This indicates that 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.

Observation-intervention trade-offs. We find a configuration o^{r+1} for either observation or for intervention for which the α^r value is maximum. We employ the ϵ -greedy strategy used by the CBO to choose between observation and intervention. 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 requires intervention. The developer must identify the optimal combination of these operations to capitalize on observational data while intervening in regions with higher uncertainty. Following CBO, we define ϵ as :

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

where $D_v = \mathcal{D}_s \cup \mathcal{D}_t$, $Vol(H(\mathcal{D}_v))$ represents the volume of the convex hull for the observational data and $Vol(o_{o \in \mathcal{O}}(\mathcal{D}(O)))$ gives the volume of the interventional domain. N_{\max} represents the maximum number of observations the developer is willing to collect in a particular environment, and N is the current size of D_v . The interventional space is larger than the observational space when the volume of the observational data $Vol(H(\mathcal{D}_v))$ is smaller than 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 observational data $Vol(H(\mathcal{D}_v))$ is large in relation to N , we need to make observations. This is because we need to obtain consistent estimates of the causal effects, which can only be

Algorithm 1 CAMEO

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

Knowledge Extraction Phase

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

Knowledge Update Phase

- 4: Initialize CGP_{warm} and CGP_{cold} .
- 5: $\beta^r = 0$
- 6: **while** $\beta^r \leq \beta$ **do**
- 7: Compute the exploitation coefficient ϵ using Equation (8) and sample a random number $u \sim \mathcal{U}(0, 1)$
- 8: **if** $\epsilon > u$ **then**
- 9: make a new observation $(o^{r+1}, c^{r+1}, y^{r+1})$.
- 10: **else**
- 11: Set $\alpha_{\text{warm}^*}^r = \text{argmin}_{o \in \mathcal{O}} \alpha_{\text{warm}}^r(o)$
- 12: Set interpolation coefficient: $\lambda^r(o) = 1(\alpha_{\text{warm}^*}^r - \alpha_{\text{warm}}^r(o) \leq l_\alpha)$
- 13: Set the acquisition function: $\alpha^r(o) = \lambda^r(o)\alpha_{\text{cold}}^r(o) + (1 - \lambda^r(o))\alpha_{\text{warm}}^r(o)$
- 14: Pick a new configuration: $o^{r+1} = \text{argmin}_{o \in \mathcal{O}} \alpha^r(o)$
- 15: Intervene on the system to obtain an interventional measurement $(o^{r+1}, c^{r+1}, y^{r+1})$.
- 16: **end if**
- 17: **if** y^{r+1} does not satisfy Ψ **then**
- 18: $y^{r+1} = \infty$
- 19: **end if**
- 20: Update \mathcal{G}_s , CGP_{warm} , D_s , \mathcal{G}_t , CGP_{cold} , D_t , and β^r .
- 21: **end while**
- 22: **return** the configuration with the best performance objective.

achieved with more observations. We update the convex hull incrementally for computation purposes.

Evaluate selected configuration and belief update. We measure the selected configuration o^{r+1} and check whether it satisfies the constraints. 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. We update the causal performance and surrogate models using the new measurement. 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 optimal.

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. As opposed to prior works (e.g., [58, 59]) that only support binary options due to scalability issues, we additionally included discrete and continuous options. We use the recommended values and ranges from system documents for both of these categories of 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 cloud resources 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 to determine the ground truth. We also empirically justify our selection of ground truth in Figure 18 in the appendix A. 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 with 3 different random seeds for reliability. We follow the standard tuning and report parameter values for SMAC, UNICORN, RESTUNE-w/o-ML, RESTUNE, and CELLO. More details about experimental choices (Table 7-13), implementation (Figure 19-23), and hyperparameters (Table 14-15) are in appendix A.

Baselines. We compare CAMEO against the following:

- **SMAC** [25]: A sequential model-based configuration optimization algorithm.
- **UNICORN** [27]: A method that can be used for optimization by transferring the source causal model in the target and later updating it using an active learning strategy.
- **CELLO** [15]: An optimization framework that augments Bayesian optimization with predictive early termination.
- **RESTUNE** [67]: An 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.

Table 3: Summarized results averaged over all environmental changes.

	Latency	Energy
	RE(%)	RE(%)
SMAC	88.2	268.9
CELLO	46.2	182.5
RES TUNE-W/O-ML	48.8	191.1
UNICORN	55.5	179.9
RES TUNE	29.2	81.2
CAMEO	7.8	14.4

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) to present the summarized results using $RE = \frac{|\mathcal{Y}_{\text{pred}} - \mathcal{Y}_{\text{opt}}|}{|\mathcal{Y}_{\text{opt}}|} \times 100\%$, where $\mathcal{Y}_{\text{pred}}$ is the best value achieved by each method, and \mathcal{Y}_{opt} is the optimal measured value from our observational dataset of 2000 samples. A method with a lower RE value is considered more effective.

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.

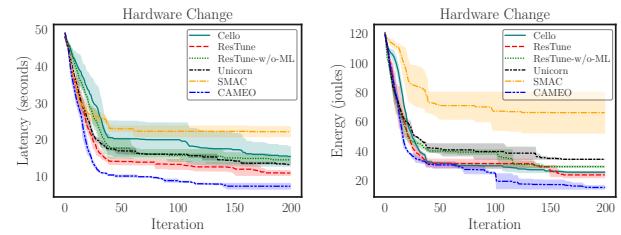
RQ2: How does the effectiveness of CAMEO change when the severity of environmental changes varies?

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 consider four types of environmental changes typically occurring when a system is deployed into production to evaluate the effectiveness of CAMEO in finding an optimal configuration compared to the state-of-the-art. Table 3 shows the summarized results for each approach averaged over different environmental changes considered in this paper. It indicates that CAMEO outperforms other optimization approaches for both latency and energy, e.g., CAMEO achieves $3.7\times$ and $5.6\times$ lower RE for latency and energy, respectively, compared to RES TUNE, the next best method after CAMEO. We describe the experimental setting and the results for the four environmental changes below.

Hardware change. We consider the MLPERF OBJECT DETECTION pipeline that uses ResNet-18 for inference of 5k images selected from the 100k test images of the ImageNet dataset [51]. We use TX2 as the source hardware and XAVIER as the target hardware. We examine these hardware changes since there are variable degrees of microarchitecture differences among this hardware separately. As shown in Figure 8,

**Figure 8: Compared to other state-of-the-art methods, CAMEO identifies the configurations with reduced latency (left) and energy (right) when hardware changes take place.**

CAMEO finds the configuration with the lowest values of latency (left) and energy (right). For example, CAMEO finds a configuration with $1.6\times$ lower latency than RES TUNE. 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 in our experiments. 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 a workload, we perform sentiment analysis on 1000 of the 25,000 reviews from the IMDB test dataset [42]. The results presented in Figure 9 demonstrate that the optimal configurations found by CAMEO have a $1.1\times$ lower latency and a $1.7\times$ lower energy value compared to RES TUNE.

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, RES TUNE outperforms CAMEO by finding a configuration with $1.02\times$ higher throughput. Upon further investigation, we found that the distributions between source and target were relatively similar, and the shared covariance learning in RES TUNE helped to find a better configuration. Additionally, the knowledge extraction module in RES TUNE is particularly developed to correctly capture workload behavior, making it more suitable for this workload change scenario. However, as the distribution difference increases, CAMEO outperforms RES TUNE, e.g., for UPDATE HEAVY workload CAMEO has $1.06\times$ higher throughput than RES TUNE.

Deployment topology change. To test the effectiveness of CAMEO across deployment topology change, we consider a video analytics pipeline: DEEPSTREAM that uses 4 camera

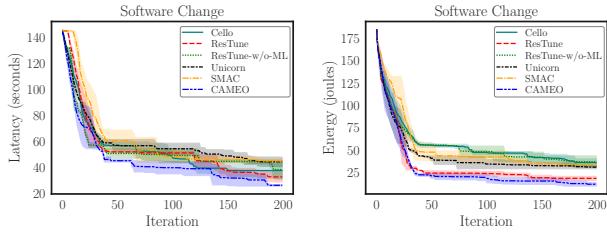


Figure 9: CAMEO finds the configurations with lowest latency (left) and energy (right) when software changes.

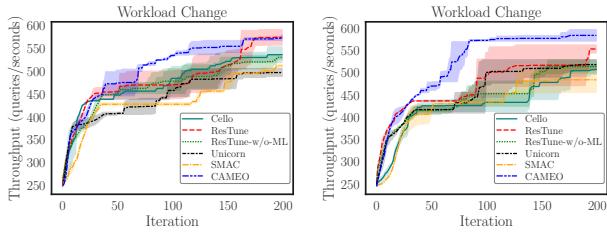


Figure 10: For read-only to balanced workload change, RESTUNE slightly outperforms CAMEO in finding configurations with higher throughput (left). For read-only to update-heavy workload change, CAMEO dominates other approaches in finding optimal configuration higher throughput (right).

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 discriminative learning algorithm online for tracking. As the source environment, we adopt a centralized deployment topology in which all four components run on XAVIER NX hardware. For the target, we use a distributed deployment topology with two XAVIER NX hardware, deploying the decoder and multiplexer in one and the detector and tracker in the other. We use APACHE KAFKA to send and receive the output of the multiplexer to the detector that uses a binary protocol over TCP. Our experimental results for the changes in the deployment environment (Figure 11) show that CAMEO significantly outperforms others in finding the optimal throughput and energy. For example, the optimal configuration discovered by CAMEO has an improvement of 1.3 \times and 1.5 \times (as) for throughput and energy, respectively, than the next-best method.

Summary of observations. Methods based on guided knowledge reuse (CAMEO and RESTUNE) consistently are the top performers over 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, in all environmental changes, methods that reuse

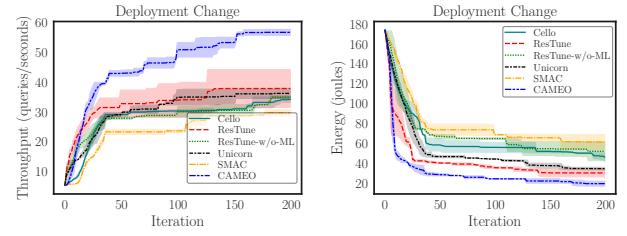


Figure 11: CAMEO has maximum effectiveness in finding configurations with the highest throughput (left) and energy (right) when deployment topology changes.

knowledge from the source outperform SMAC, RESTUNE-w/o-ML, and CELLO that do not rely on previous information and cannot achieve the optimal within the allowed budget. Unlike RESTUNE and CAMEO, UNICORN directly uses source information in the target, thereby introducing bias, which must be learned. This unlearning is not necessary for CAMEO due to its knowledge transfer strategy.

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

(i) The combined correctness of two causal performance models allows one to effectively identify the values of optimal options. Table 4 shows the optimal configuration discovered by different approaches. It is evident that CAMEO can correctly identify the maximum number of options values compared to other approaches (only misidentified `vm.dirty_bytes`). This is possible due to the usage of two causal models G_s and G_t as shown on the left of Figure 12. The right subfigure in Figure 12 shows the iterative changes in structural differences (by Hamming distance) with the causal model of the ground truth when using only G_s or G_t or when combining the two. Here, we find that the Hamming distance is significantly low when both G_s and G_t are

Table 4: Optimal configuration discovered by different baselines. 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
cpu_frequency	1.3	1.6	1.6	1.6	2.0	0.19	2.0
vm.dirty_ratio	20	5	20	5	5	0.13	5
vm.swappiness	60	60	60	60	60	0.11	60
gpu_frequency	1.3	1.3	1.3	1.3	1.3	0.08	1.3
num_cores	3	4	3	4	4	0.06	4
memory_growth	0.5	0.9	0.5	0.9	-1	0.04	-1
emc_frequency	1.1	1.3	1.3	1.1	1.3	0.009	1.3
drop_caches	0	0	0	0	0	0.008	0
scheduler_policy	NOOP	NOOP	CFP	NOOP	NOOP	0.001	NOOP
vm.vfs_cache_pressure	10	50	10	10	10	0.001	10
vm.dirty_bytes	30	60	60	30	60	0.0009	30
kernel.sched_rt_runtime_us	5x10 ⁶	5x10 ⁶	5x10 ⁶	5x10 ⁶	9.5x10 ⁶	0.0009	9.5x10 ⁶
logical_devices	1	1	0	1	1	0.0008	1
kernel.sched_child_runs_first				0	0	0.0006	
Latency	22s	15s	14s	13s	8s		8s

combined, indicating that the discovered causal performance model is nearly identical to the ground truth causal performance model in the target, as shown in Figure 12.

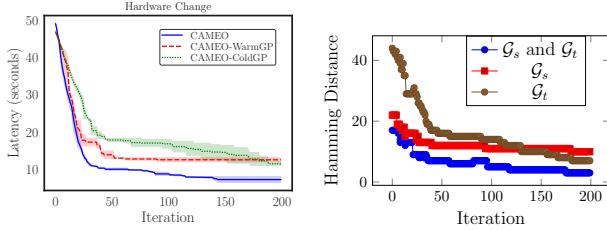


Figure 12: The causal performance models become more accurate with increasing iterations. The correctness of \mathcal{G}_s and \mathcal{G}_t when combined helps CAMEO in detecting the optimal configuration more effectively than others. A lower hamming distance value indicates a smaller difference with the ground truth causal performance model in the target.

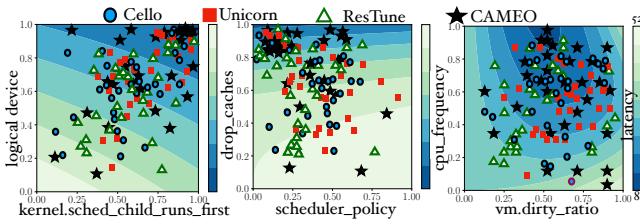


Figure 13: Contour plot with options of different causal effects. The color bar indicates the latency values, where lower values indicate better performance.

(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 13). The leftmost subfigure of Figure 13 contains options with lower ACE values, while the rightmost contains the options with high ACE values only. The middle subfigure of Figure 13 contains options that have ACE values near the median (the ACE values of the configuration options are provided in Table 4). The right-hand subfigure of Figure 13 shows that the response surfaces of the options with higher ACE values are more complex than those with lower ACE values. Table 4 demonstrates that CAMEO can accurately find the optimal values of options with higher ACE values, such as `cpu_frequency` and `dirty_ratio`, demonstrating a better understanding of such complex behavior. Figure 13 also shows how CAMEO has investigated more configurations by varying more options with higher ACE values than lower ones. By focusing on more sophisticated surfaces rather than wasting resources on less effective options, CAMEO can make the

best use of resources to better understand the performance behavior for navigating the search space.

(iii) CAMEO reaches better configurations by achieving better exploration-exploitation trade-offs. From Figure 13 (left and middle), we observe that for options with lower ACE values, CAMEO quickly reaches the region with configurations with lower latency within fewer explorations and then focuses on exploitation behavior to quickly determine the optimal configuration. In the rightmost subfigure of Figure 13, configurations evaluated by CAMEO cover the largest number of different regions (indicating a better exploration). Here, we also observe that CAMEO has evaluated a higher number of configurations near the optimal configuration (blue) regions of the response surface (indicating better exploitation). Therefore, CAMEO has a higher coverage of the configurations evaluated during the optimization procedure compared to other approaches for the core options with higher ACE values. The identification of such core features is central to achieving better exploration-exploitation trade-offs.

6 RQ2: Severity of Environmental Changes

The effectiveness of CAMEO changes due to the amount of distribution shift 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 severity of change. 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 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 changed 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 a configuration with 1.3 \times , 1.5 \times , and 1.9 \times higher throughput than RESUME with low, medium, and high severity of changes, respectively. The KL divergence

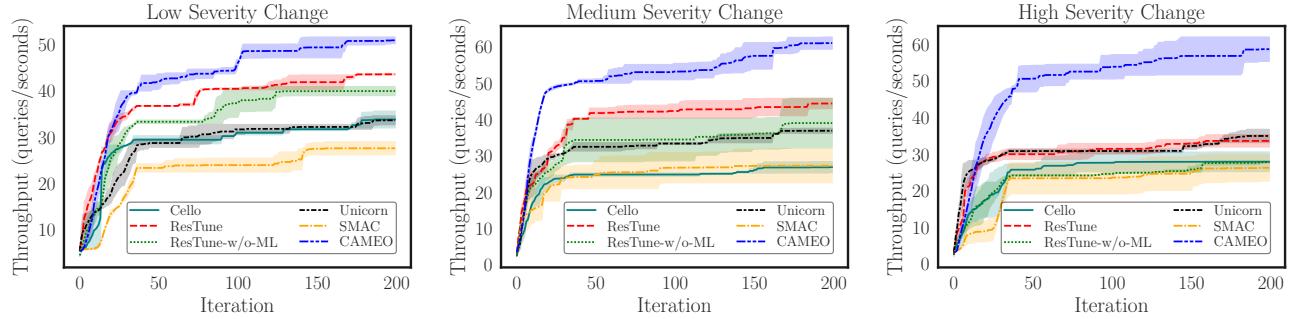


Figure 14: CAMEO achieves higher throughput when different severity of environmental changes take place.

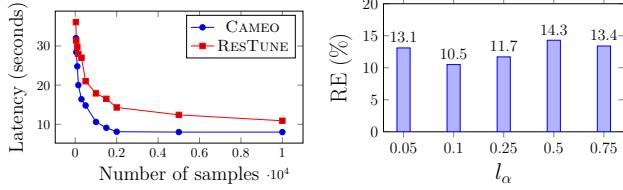


Figure 15: (left) Both approaches find better configurations with higher source samples. Compared to RESUME, CAMEO finds the optimal configuration with a lower minimum latency (right). CAMEO has a minimum RE when l_α is 0.1.

values between the distributions of the source and the low, medium, and high severity environmental changes setup are 418, 951, and 1329. Therefore, we conclude that CAMEO performs better than the baselines as 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 in TX2 as the source and the same pipeline in XAVIER as the target, varying the number of measurements in TX2 from 30 to 10000 for evaluation and comparison of their optimal values discovered by different approaches. As shown in Figure 15 (left), increasing the number of source measurements positively influences CAMEO’s as compared to RESUME. 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 can prevent those biases from being introduced into the target because more samples are used to extract knowledge from

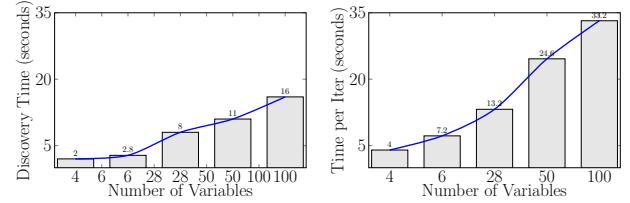


Figure 16: As the number of configuration options and system events increases, discovery time (left) and total time per iteration (right) increase sub-linearly.

the source. We also observe that CAMEO reaches a plateau (after 2000 samples) faster than RESUME, indicating that CAMEO can find better configurations with fewer source samples. Because of CAMEO’s ability to detect the core features, it can be reliably used across environments without much modification.

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 set of the COMMON VOICE dataset for 5 languages (English, Arabic, Chinese, German, and Spanish). We run inference on the Chameleon cloud instance with one P100 GPU for the source and one K80 GPU for the target. To evaluate the scalability of our approach to colossal configuration space [47], we increase the number of variables from 4 to 100 and determine the discovery time and

Table 5: Comparison of computation time in seconds per iteration for 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
RESUME-w/o-ML	8.3	9.2	61.3
RESUME	9.7	9.7	63.4
CAMEO	12.7	14.4	71.6

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 events. The scalability of CAMEO can be attributed to the sparsity of the causal graph, leading to a small exploration set 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 reduces safety constraint breaches by collecting 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. However, they 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. To accelerate optimization using transfer learning, it is essential to identify what knowledge is necessary to be reused. Jamshidi et al. [31] showed that when environmental changes are small, knowledge can be transferred to predict performance, while only knowledge can be transferred to efficient sampling when environmental 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 to approximate and transfer the Pareto frontiers of optimal configurations across different hardware environments. Ballesteros et al. [5] proposed a dynamic evolutionary transfer learning algorithm to generate effective quasi-optimal configurations at runtime. All these techniques incorporate transfer learning based on correlational statistics (ML-based). However, Section 2.1 shows that ML-based models tend to capture spurious correlations. In comparison, CAMEO uses causal 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 that intervenes through fault injection to pinpoint the root cause of intermittent failures. Johnson et al. [36] proposed Causal testing to analyze and fix software bugs by identifying a set of executions that contain important causal information. Dubslaff et al. [16] proposed a method to calculate feature causes effectively and used them to facilitate root

cause identification and estimation of feature effect/interaction. The causality analysis in these works is solely on one environment, whereas we focus on efficiently transferring the causal knowledge from one environment to another.

9 Limitations

Causal graph error. Causal discovery is an NP-hard problem [9]. Thus, the learned causal graphs might not be the ground-truth causal graphs and do not always reflect the true causal relationship. However, such causal graphs can still be leveraged to achieve better performance than ML-based approaches in system optimization and debugging tasks as they avoid capturing spurious correlations [16, 27].

Noisy Measurements. The system performance measurements are noisy and can affect the results. To mitigate this, we take each configuration’s median across 5 runs.

More model computational time. Due to the use of two CGPs, CAMEO takes more time than the baselines. For example, on average, CAMEO takes 27.1s per iteration versus 19.4s per iteration taken by RESTUNE (see Table 5). However, this time is usually small compared to the time required for each evaluation (44s on average in our experiments).

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 in the input domain trained on the target data and a warm-CGP that operates in the feature space of a causal graphical model pre-trained on the source data. The decomposition dynamically controls the reliability of information derived from the online and offline data and the use of CGPs helps avoid spurious correlations. Empirically, we demonstrate significant performance improvements of CAMEO over existing methods on real-world systems.

Acknowledgements

This work has been supported, in part, by the National Science Foundation (Awards 2007202, 2107463, 2233873, 2107405, 1845893, and 2038080). 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 Lésoil, 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 Miquing 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] Awini 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 Jezequel. 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 *Conférence 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 Sanaei, 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 Grebahn, 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 \mathcal{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 $\mathcal{O} = \text{Dom}(O_1) \times \dots \times \text{Dom}(O_d)$, where d is the number of options. A configuration o is then a member of the configuration space \mathcal{O} in which all options are set to a given value within the range of values permitted 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 $\Pr(\mathcal{Y} | O_i = o_i)$.

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 $\Pr(\mathcal{Y} | \text{do}(O_i = o_i))$. 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 value of the objective functions $\hat{\mathcal{Y}} = f(o)$ using the predictive mean $\mu(o)$ and the uncertainty $\sigma(o)$ for the configurations $o \in \mathcal{O}$. A common practice is to use Gaussian processes (GPs) as surrogate models where the GP distribution over $f(o)$ is fully

Table 6: 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

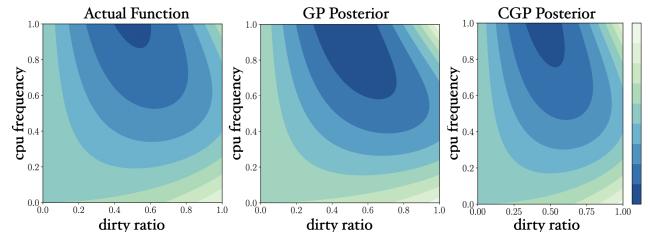


Figure 17: The posterior of CGP relying on interventional distribution can capture the target function better than GP, particularly near the optimal region.

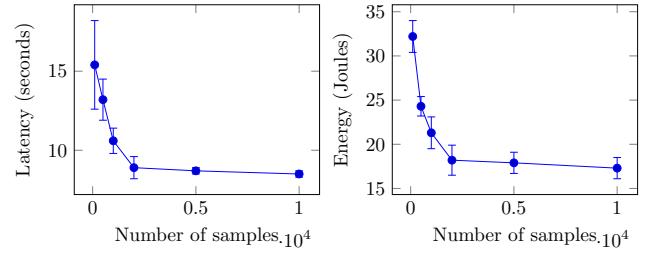


Figure 18: After 2000 configuration, the value of the optimal performance objective reaches a plateau as the number of configurations continues to rise.

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 marginal distributions $f(o)$ and $f(o')$. After the surrogate model outputs the 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 6 to 15 and Figures 20 to 23.

Table 7: 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)

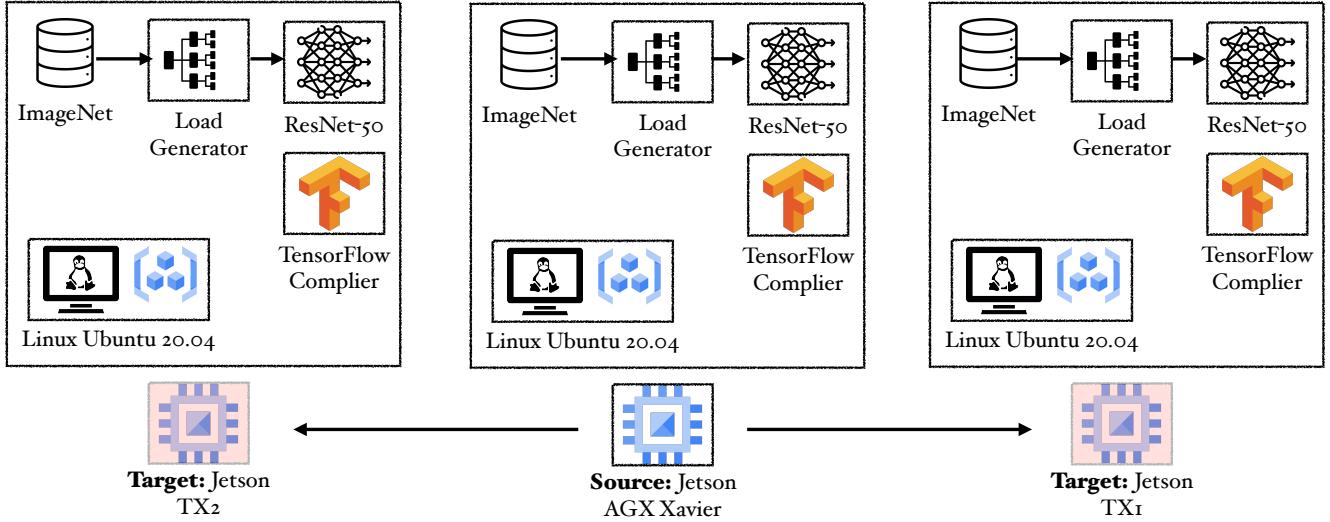


Figure 19: Experimental setup when hardware changes from XAVIER in the source to TX2 and TX1 in the target, separately, for MLPERF OBJECT DETECTION pipeline.

Empirical justification of using 2000 configurations to determine the ground truth We use the MLPERF OBJECT

DETECTION pipeline in XAVIER and compare the optimal performance values using different numbers of configurations ranging from 500 to 10000 to support our decision to use 2000 configurations to find the ground truth. We discover that the optimal values reach a plateau after 2000 configurations, as shown in Figure 18. Therefore, computing the RE value using the 2000 configuration as the ground truth for the evaluation can be reliably used for the evaluation.

A.3 RQ1 Additional Results

Constrained optimization For constrained optimization (optimizing latency with energy constraints or optimizing energy with latency constraints), we set the energy and latency constraints as [15, 30, 45, 60, 75, 90]-th percentiles of

Table 9: Configuration options in MLPERF OBJECT DETECTION, and SPEECH RECONGITION software system.

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 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: 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 13: 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 14: 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)
BERT	batch_size	32
	num_epochs	100
	dropout	0.3
	maximum_batch_size	16
	maximum_sequence_length	13
	learning_rate	1e ⁻⁴
DEEPSPEECH	weight_decay	0.3
	dropout	0.3
	maximum_batch_size	16
	maximum_sequence_length	32
	learning_rate	1e ⁻⁴
	num_epochs	10

Table 15: Hyperparameters for FCI used in CAMEO.

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

Table 11: 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
Nvinfer	nobuf_memory_type	0, 1, 2, 3
	net_scale_factor	0.01 - 10
	batch_size	1 - 60
	interval	1 - 20
	offset	0, 1
	process_mode	0, 1
Nvtracker	use_dla_core	0, 1
	enable_dla	0, 1
	enable_dbSCAN	0, 1
	secondary_reinfer_interval	0 - 20
	maintain_aspect_ratio	0, 1
	iou_threshold	0 - 60
Nvtracker	enable_batch_process	0, 1
	enable_past_frame	0, 1
	compute_hw	0, 1, 2, 3, 4

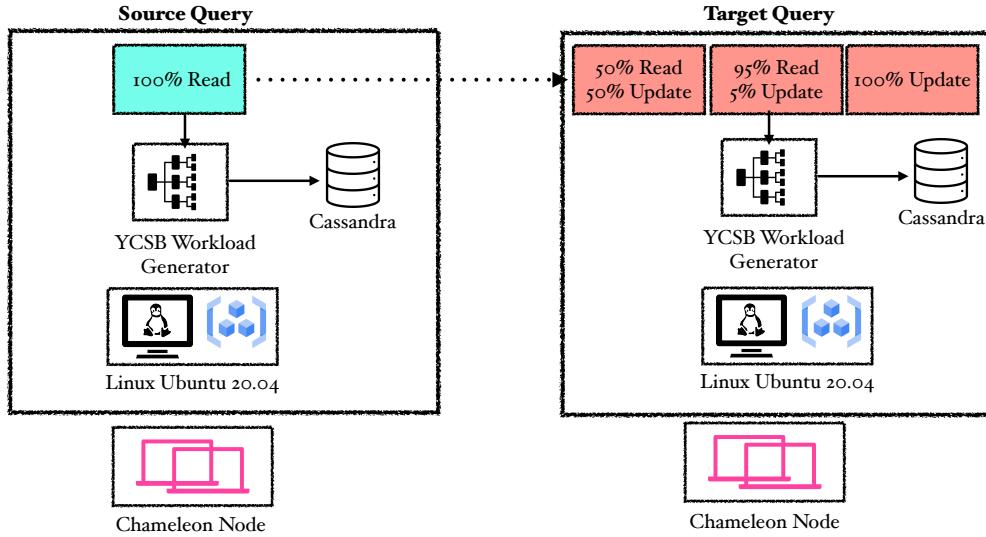


Figure 21: Experimental setup when the type of workload is different with a CASSANDRA database where the source uses a **READ ONLY** workload where the target uses a **BALANCED** and **UPDATE HEAVY** workload, separately.

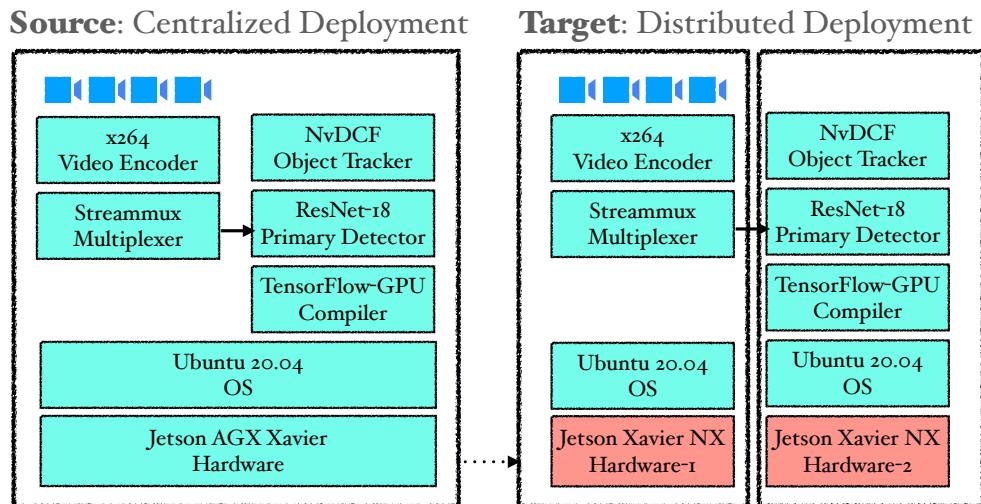


Figure 22: Experimental setup for our experiments when the deployment topology is changed from centralized to distributed in the target in the target using two XAVIER NX.

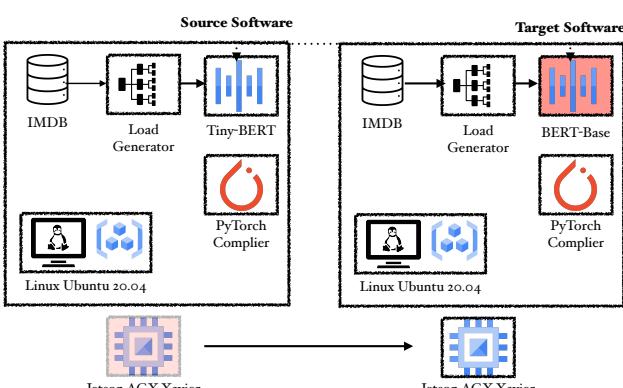


Figure 20: Experimental setup when a software change takes place from TinyBERT to BERT-Base in the target for a NLP system.

the corresponding distributions. Table 16 reports the summarized results compared to CELLO, as this is the only baseline that incorporates constraints. We observe that in addition to latency optimization under energy constraints for workload changes, CAMEO consistently outperforms CELLO for hardware, software, and deployment environment changes, for example, under latency constraints, CAMEO finds configurations with 1.3 \times and 1.5 \times for software and deployment topology changes, respectively.

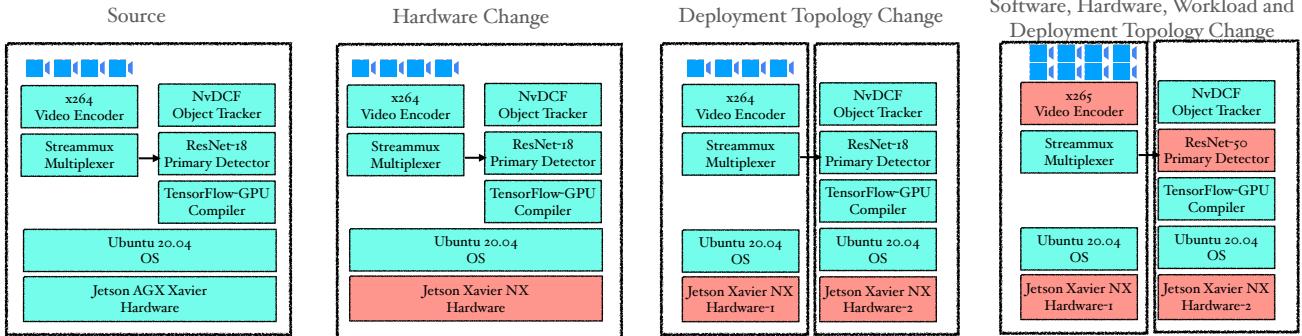


Figure 23: Experimental setup for different severity of environmental changes. Low severity change scenario when only hardware changes from XAVIER to XAVIER NX in the target (second figure). We change the hardware and deployment topology for the medium severity change scenario (third figure). For high-severity 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).

Table 16: 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