COMET: NEURAL COST MODEL EXPLANATION FRAMEWORK

Isha Chaudhary [1]   **Alex Renda** [2]   **Charith Mendis** [1]   **Gagandeep Singh** [1] [3]

ABSTRACT

Cost models predict the cost of executing given assembly code basic blocks on a specific microarchitecture. Recently, neural cost models have been shown to be fairly accurate and easy to construct. They can replace heavily engineered analytical cost models used in mainstream compiler workflows. However, their black-box nature discourages their adoption. In this work, we develop the first framework, COMET, for generating faithful, generalizable, and intuitive explanations for neural cost models. We generate and compare COMET's explanations for the popular neural cost model, Ithemal against those for an accurate CPU simulation-based cost model, uiCA. Our empirical findings show an inverse correlation between the prediction errors of Ithemal and uiCA and the granularity of basic block features in COMET's explanations for them, thus indicating potential reasons for the higher error of Ithemal with respect to uiCA.

1 INTRODUCTION

Cost models predict the cost (memory, time, energy, etc) that an assembly code basic block, a sequence of assembly instructions with no jumps or loops, takes while executing on a specific microarchitecture. They are used to guide compiler optimization (Mendis et al., 2019b; Cummins et al., 2017) and superoptimization (Schkufza et al., 2013). They can be traditional or learned models.

Traditional cost models are typically simulation and static-analysis based models. Simulation-based cost models generate their predictions by simulating program execution for a given CPU. They are hand-engineered using released documentation and micro-benchmarking the CPU under study.

Examples of these cost models are uiCA (Abel and Reineke, 2022) and LLVM-MCA (Di Biagio and Davis, 2018). As these models are traditional programs, domain experts can intuitively understand and debug them, and hence they are commonly deployed in practical settings. However, they require significant engineering effort to construct and must be manually re-engineered to reflect changes across CPU microarchitectures. Static-analysis based cost models, such as IACA (Intel, 2017) and OSACA (Laukemann et al., 2018) predict the cost of executing a program on a given microarchitecture by developing a model of the target CPU and deploying static analysis methods to predict the cost of a given basic block with the model. These cost models often have a high error in their predictions (Abel and Reineke, 2022; Chen et al., 2019).

Alternatively, machine learning techniques can be used to learn a cost model (Mendis et al., 2019a; Kaufman et al., 2021; Baghdadi et al., 2021; Sýkora et al., 2022). Development of ML-based cost models requires the one-time effort of collecting a dataset of representative programs, collecting the end-to-end cost for the execution of those programs on the CPU under study, and training a selected type of ML model. While simple and interpretable ML models could be used for constructing cost models, prior work (Mendis et al., 2019a; Sýkora et al., 2022; Kaufman et al., 2021; Baghdadi et al., 2021) has used neural networks as cost predictors to precisely approximate the complex function mapping basic blocks to their costs. An instance of such neural cost models is Ithemal (Mendis et al., 2019a), which is an LSTM model trained on the BHive (Chen et al., 2019) dataset of x86 basic blocks to predict basic block throughput (average number of CPU clock cycles to execute the block when looped in steady state). Ithemal is more accurate on the BHive dataset than most throughput models (Chen et al., 2019). Ithemal needs less manual effort to construct than any simulation-based or static-analysis-based cost model. However neural models generally have the downside that they are uninterpretable (Molnar, 2022).

This work. Our goal is to bring interpretability to inherently black-box but accurate neural cost models, by developing a general framework that can generate trustworthy and intuitive explanations of their predictions. These neural cost models could have arbitrary architectures (Mendis et al., 2019a; Sýkora et al., 2022), requiring custom explanation

---

[1]Department of Computer Science, University of Illinois Urbana-Champaign, Illinois, USA [2]MIT CSAIL, Massachusetts, USA [3]VMWare Research, USA. Correspondence to: Isha Chaudhary <isha4@illinois.edu>.

methods, and could also be proprietary. To avoid engineering custom explanation methods for each model, we develop a common explanation framework that is agnostic to the type or structure of the model. Apart from saving manual engineering effort, a common framework would facilitate a comparison between neural and other types of cost models with respect to the explanations of their predictions. To achieve our goals, we develop our explanation framework to generate explanations that (i) assume just query-access to the cost model, which may be available for some proprietary cost models too, (ii) faithfully reflect the cost model's behavior, (iii) generalize across multiple basic blocks, and (iv) are simple and interpretable for domain experts.

Key challenges. For building trustworthy explanations, we need to formalize the desirable properties of faithfulness, generalizability, and simplicity (Subhash et al., 2022). There is a tradeoff between the degree to which a given explanation satisfies the above desirable properties and its computational cost. Therefore, we need to design efficient algorithms that can balance this tradeoff. Prior works (Ribeiro et al., 2016; 2018) in domains such as Vision or NLP have used perturbed inputs to efficiently generate explanations with only query-access to the model. However, their perturbation algorithms heavily utilize local neighborhoods of their inputs while creating their explanations. In the discrete domain of basic blocks, there is no well-defined concept of locality. Hence, we need specialized perturbation algorithms to handle this domain-specific challenge and derive close approximations to the complex behavior of a given cost model in a reasonable number of queries.

Our approach. We identify that global explanations with desirable properties may be computationally intractable or even may not exist for complex cost models. Hence, we focus on explaining a given model's prediction for a target basic block. We first formalize the ideal, query-based, block-specific explanations with desirable properties as an optimization problem. We observe that generating such ideal explanations is intractable. To practically generate explanations, we relax our requirements and develop COMET, a perturbation-based explanation framework based on the design of (i) novel primitives for explanations that capture both coarse-grained (e.g. number of instructions) and fine-grained (e.g., instructions and data dependencies) features of the basic block, and (ii) new custom perturbation algorithms for generating a diverse set of basic blocks that help gauge the complex behaviors of cost models.

Contributions. We make the following contributions:

1. We formalize the ideal query-based explanations having desirable properties for cost models for any target basic block as an optimization problem that is agnostic to any particular Instruction Set Architecture (ISA).

2. We relax the problem to make it practically solvable. Building on our relaxation, we present COMET (COst Model ExplanaTion framework), a novel and efficient explanation framework for neural cost models. As COMET depends on the ISA, we have implemented it for the popular x86 ISA, and it can be extended to other ISAs with non-trivial engineering effort. We open-source our implementation at `https://github.com/uiuc-focal-lab/COMET`. COMET's explanations identify the features of a target basic block that are important for a given cost model's prediction.

3. We systematically analyze COMET's accuracy and use it to gain insights into the working of common cost models. We explain basic blocks in the popular BHive dataset (Chen et al., 2019). We empirically observe that COMET's explanations for the neural cost model Ithemal more often consist of coarser-grained features of the basic block, such as the block's number of instructions, as compared to the explanations for the lowest error simulation-based cost model uiCA, indicating potential sources of the relatively higher error in Ithemal's predictions with respect to uiCA.

COMET aims to help our stakeholders, i.e. compiler and performance engineers, develop an intuition about and debug neural cost models in a simple yet precise way. We anticipate this work to go a long way in developing better neural cost models and making them trustworthy.

2 RELATED WORK

Explanation techniques. Explanations for ML models consist of either building inherently interpretable ML models (Lakkaraju et al., 2016) or creating post-hoc explanations for the models (Ribeiro et al., 2016; 2018; Lakkaraju et al., 2019; Martens and Provost, 2014). Post-hoc explanations are preferred as accurate cost modeling for the CPU's pipelined architecture makes complex models more suitable. These can either describe a model globally (Lundberg and Lee, 2017) or for specific inputs (Ribeiro et al., 2016; 2018). Explanation techniques can also be broadly classified as black-box (Ribeiro et al., 2016; 2018; Lundberg and Lee, 2017) and white-box techniques (Simonyan et al., 2014; Seo et al., 2018). Further classifications of explanation techniques can be as perturbation/example-based (Bui et al., 2019; Liu et al., 2018; Zeiler and Fergus, 2014) and symbolic explanation techniques (Boumazouza et al., 2021; Marques-Silva, 2023; Ignatiev et al., 2019; Arenas et al., 2021). While symbolic methods give formal guarantees on the explanations, they do not scale to complex models yet. (Ritter and Hack, 2022) is a differential-testing tool to analyze the inconsistencies between multiple cost models. This tool, unlike COMET, is not meant to explain a particular prediction of a cost model to enable case analysis.

Input perturbation algorithms. For domains wherein the input is a sequence of discrete entities such as NLP and code, the prior perturbation-based explanation algorithm by Ribeiro et al. (2018) has used generative models (Devlin et al., 2019; Feng et al., 2020) to obtain input perturbations. These perturbations might not be syntactically correct and can result in erroneous explanations (Cito et al., 2022). Hence, we have not used such unconstrained perturbation techniques in our explanation framework. Moreover, as mentioned above, there is no well-defined concept of locality in this domain. Thus, we can not use the perturbation algorithms from prior work in other domains which generally perturb the input in some local regions. Stoke (Schkufza et al., 2013) is a stochastic superoptimizer that perturbs input x86 assembly programs to optimize them. While Stoke does not operate on embedding spaces, it can generate syntactically incorrect perturbations.

3 OVERVIEW AND MOTIVATING EXAMPLE

Consider the x86 basic block in Listing 1(a), represented in the Intel assembly syntax (Bartlett, 2021) (default syntax throughout the paper). The throughput cost of this basic block predicted by the neural cost model Ithemal for the Haswell microarchitecture is 1.3 cycles. We use COMET to explain the prediction made by Ithemal. An *explanation* is a small set of features of the input basic block whose presence is *sufficient* to get a prediction from the cost model close to its original prediction. COMET identifies the feature corresponding to Read-after-Write (RAW) data dependency between instructions 1 and 2 as an explanation for Ithemal's throughput prediction. Please refer to Appendix B for information on the types of common data dependencies in assembly basic blocks.

```
1  add rcx, rax    1  add rcx, rax
2  mov rdx, rcx    2  mov rdx, rcx
3  pop rbx         3  push rbx
```

(a) Input basic block   (b) Perturbed basic block

Listing 1. Motivating Example

Figure 1 illustrates COMET's working on the above example. COMET first extracts the candidate features, i.e., individual instructions, their count, and data dependencies between instruction pairs, as shown in Figure 1(iii). It searches over all possible combination sets of the candidate features for an explanation that is simple, faithful to the cost model's behavior, and extends to other basic blocks as well. COMET evaluates any feature set through perturbations of the input basic block that retain the features in the set [Figure 1(c)]. For example, for the singleton set consisting of just the RAW dependency in the block, a possible perturbation is shown in Listing 1(b), which preserves the

Table 1. Notation

Notation	Meaning
$\mathcal{M}$	Cost model
β	Basic block
$\mathcal{P}$	Set of all block features
$\hat{\mathcal{P}}$	Block features used to form explanations
$\mathcal{F}$	Given set of features
$Prec(\mathcal{F})$	Precision of $\mathcal{F}$
$1\text{-}\delta$	Precision threshold
$Cov(\mathcal{F})$	Coverage of $\mathcal{F}$
$\Pi(\mathcal{F})$	All perturbations of β retaining $\mathcal{F}$
$\mathcal{D}_{\mathcal{F}}$	Distribution over blocks in $\Pi(\mathcal{F})$
$\mathcal{G}$	Multigraph of β
Γ	COMET's perturbation algorithm

dependency. COMET obtains the predictions of the given cost model for each perturbed basic block and uses them for the estimation of the faithfulness (precision) and generalizability (coverage) of a set of features. The set with precision higher than a threshold and maximum coverage is output as COMET's explanation for the given cost model and basic block [Figure 1(d)].

As per Abel and Reineke (2019), for Haswell, the canonical forms of the individual instructions in the block have similar throughputs. The RAW data dependency will disable the independent, parallel execution of instructions 1 and 2, and hence intuitively be a bottleneck for the block's execution. As COMET's explanation matches our intuition, we can be confident that Ithemal predicts throughput focusing on the correct features for this and similar blocks.

4 FORMALIZING COST MODEL EXPLANATIONS

In this section, we formalize our notion of explanations for cost models and discuss the desirable properties of the explanations. Our objective is to develop an explanation framework that only queries a given cost model (query-access only) to explain its behavior. Table 1 presents the important notation used when discussing the following formalism.

We first formalize the cost model as a function $\mathcal{M}$ that maps valid basic blocks in a given Instruction Set Architecture (ISA) to real-valued costs. Let $\mathcal{T}$ be a set of $\mathcal{M}$'s predictions that we want to explain over the set of valid basic blocks (global explanation).

An explanation for the behavior of $\mathcal{M}$ over $\mathcal{T}$ is the common features of basic blocks having cost prediction in $\mathcal{T}$, that are not present in other basic blocks.

For example, consider a hypothetical, crude throughput-predicting cost model $\mathcal{M}_1$ that assigns a throughput of 2
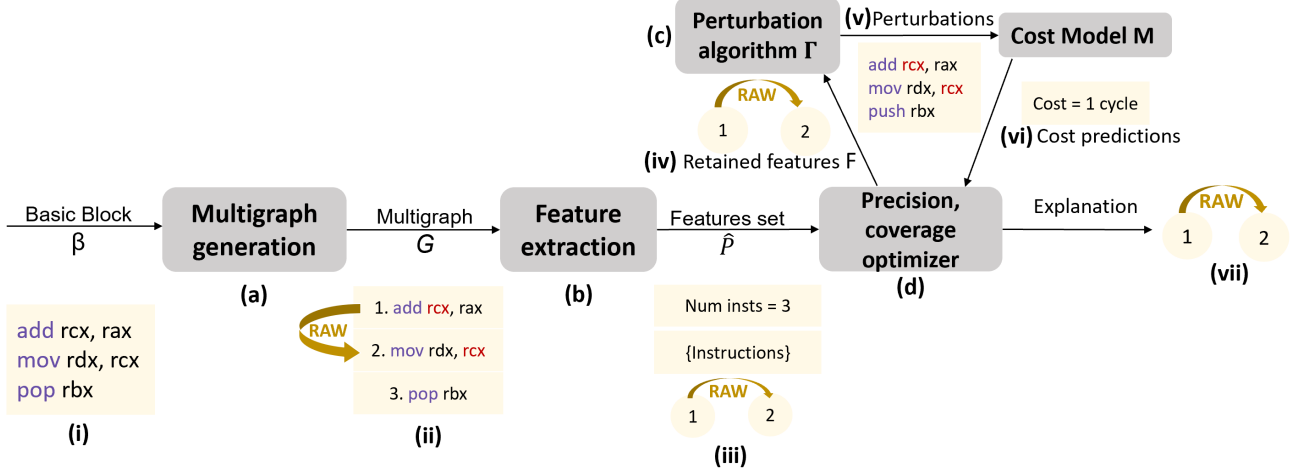
Figure 1. COMET is given a cost model $\mathcal{M}$ and a basic block β as input. COMET identifies the features of β that explain the prediction $\mathcal{M}(\beta)$. COMET first converts β to a multigraph $\mathcal{G}$ in (a). $\mathcal{G}$ has the instructions and data dependencies of β as its vertices and edges respectively. The features $\hat{\mathcal{P}}$ of β, are then extracted from $\mathcal{G}$ in (b). Sets of these features are fed into COMET's perturbation algorithm Γ (c) that generates several perturbations that preserve the features in the corresponding retained input feature set $\mathcal{F}$. COMET obtains the predictions of cost model $\mathcal{M}$ for each perturbed basic block, which are then used for the estimation of the precision and coverage of a feature set $\mathcal{F}$. $\mathcal{F}$ having precision higher than $(1 - \delta)$ with maximum coverage is identified by the precision and coverage optimizer in (d) and is output as COMET's explanation for $\mathcal{M}(\beta)$.

cycles if and only if a basic block has 8 instructions. If $\mathcal{T} = \{2\}$, then all the blocks having prediction in $\mathcal{T}$ will have 8 instructions while the rest will not. Hence the distinguishing factor and therefore a correct global explanation of the cost model's behavior in $\mathcal{T}$ will be the number of instructions equal to 8.

While it might be possible for simple cost models such as $\mathcal{M}_1$ to have global concepts for their predictions that can explain them in certain $\mathcal{T}$, generally cost models are complex and specialized to exploit features of the input basic block. Hence, as it may be infeasible to generate global explanations with desirable properties, we focus on generating block-specific explanations for cost model $\mathcal{M}$. To explain the cost prediction for the basic block β, we specialize $\mathcal{T}$ to be an $\epsilon-$ball around $\mathcal{M}(\beta)$, where $\epsilon > 0$ is a small constant. Our generated explanations are sets of features of the block whose presence is sufficient for the cost model to make its prediction.

The smallest meaningful units (basic features) of an assembly basic block are its tokens (opcodes and operands). Let $\mathcal{P}_\beta$ be the set of all basic features and all functions of basic features, which we cumulatively call features, of the basic block β. Some elements of $\mathcal{P}_\beta$ for the input basic block in Figure 1 are shown in Figure 1(iii). Note that, as $\mathcal{P}_\beta$ captures all the features of the basic block, there is a one-to-one correspondence between $\mathcal{P}_\beta$ and the block. The remaining discussion in this paper will describe our approach for generating explanations for the cost model $\mathcal{M}$ when pre-

dicting the cost for a basic block β. To simplify notation, unless mentioned otherwise, we will omit $\mathcal{M}$ and β from the subscripts of symbols, e.g., $\mathcal{P}_\beta$ will be written as $\mathcal{P}$.

4.1 Ideal query-only explanation

Let the set of features $\mathcal{F}^* \subseteq \mathcal{P}$ be the ideal explanation for $\mathcal{M}(\beta)$ based on queries to $\mathcal{M}$. Desirable properties of $\mathcal{F}^*$ are that it should be *faithful* to cost model's behavior, *generalizable* to multiple basic blocks, and *simple* (Subhash et al., 2022), which we formalize next.

Let Π be a perturbation function that is given a set of features $\mathcal{F}$ of basic block β as input, and returns a set of valid assembly basic blocks $B_\mathcal{F}$, where each basic block $\beta' \in B_\mathcal{F}$ differs from β only by some perturbations to the features in $\mathcal{P} \setminus \mathcal{F}$ in β. Output of $\Pi(\mathcal{F})$ includes β also. Consider the block in Figure 1(i). If the set {instruction 1: add rcx rax} is input into Π, then the basic block shown in Figure 1(v) is an element in the output set of basic blocks, as it perturbs some features not in the input set of features.

Faithfulness. A set of features $\mathcal{F} \subseteq \mathcal{P}$ will be a faithful explanation for the prediction of $\mathcal{M}(\beta)$ if perturbations of features of β that are not in $\mathcal{F}$ cannot change the cost prediction of $\mathcal{M}$ significantly, i.e., by more than ϵ. Otherwise, $\mathcal{F}$ does not completely capture the features used by $\mathcal{M}$ for its prediction for β, and hence is not faithful to $\mathcal{M}$'s behavior. (1) presents the above notion as a logical statement $\varphi(\mathcal{F})$

that must be satisfied by the ideal, faithful explanation $\mathcal{F}^*$.

$$\varphi(\mathcal{F}) \triangleq (\mathcal{F} \subseteq \mathcal{P}) \text{ and } (\forall \alpha \in \Pi(\mathcal{F}). \mathcal{M}(\alpha) \in \mathcal{T}) \quad (1)$$

A trivially faithful set of features is $\mathcal{P}$, as it would contain all the basic block features that are important for cost prediction. But this explanation is not useful, as P can faithfully explain β for any cost model but it does not precisely distinguish features according to the behavior of a target cost model.

Generalizability and simplicity. To overcome the above issue, we require that faithful explanations of basic block β should also explain other blocks that contain the features in the explanation and where the cost model $\mathcal{M}$ makes predictions close to $\mathcal{M}(\beta)$ (generalizable). Every set $\mathcal{F} \subseteq \mathcal{P}$ will have a corresponding set of basic blocks (potentially empty), $\Omega_\mathcal{F}$ (2) containing basic blocks with similar predictions as β and having $\mathcal{F}$ as faithful explanations. For faithful explanations with maximum generalizability, we need to maximize the cardinality of $\Omega_\mathcal{F}$ over the set of faithful explanations.

$$\Omega_\mathcal{F} \triangleq \{\alpha \in \Pi(\mathcal{F}) \text{ and } \mathcal{M}(\alpha) \in \mathcal{T} \text{ and } \varphi_\alpha(\mathcal{F})\} \quad (2)$$

For higher interpretability, ideal explanation $\mathcal{F}^*$ should be simple. While there are many metrics for simplicity of explanations, a common metric for sets of features used as explanations is their cardinality (Robnik-Šikonja and Bohanec, 2018; Molnar, 2022). Hence, for simple, faithful, and generalizable explanations $\mathcal{F}^*$, we solve the optimization problem (3), where $\lambda > 0$ is a regularization parameter.

$$\mathcal{F}^* \triangleq \underset{\mathcal{F} \text{ s.t. } \varphi(\mathcal{F})}{argmax}(|\Omega_\mathcal{F}| - \lambda.|\mathcal{F}|) \quad (3)$$

4.2 Practical query-only explanations

There are two levels of intractability in the above formulation of ideal explanations (3). First, the evaluation of the faithfulness condition (1) for a given set of features $\mathcal{F}$ requires querying $\mathcal{M}$ for the cost prediction of all the basic blocks in the large set, $\Pi(\mathcal{F})$. We refer the reader to Appendix F for examples of some estimates of the cardinality of $\Pi(\mathcal{F})$. Second, the computation in (2) requires predicting the cost and computing faithful explanations for all basic blocks in $\Pi(\mathcal{F})$. Hence, to practically solve the explanation problem, we relax it as described next.

Probabilistic faithfulness. To simplify the faithfulness condition in (1), we relax the requirement of the cost prediction for all perturbed basic blocks to be in $\mathcal{T}$ with the requirement that the probability of the cost of perturbed blocks being in $\mathcal{T}$ to be higher than a threshold. This threshold will denote the degree of faithfulness of our explanations. This probability can be represented as $Pr_{\alpha \sim \mathcal{D}_\mathcal{F}}(\mathcal{M}(\alpha) \in \mathcal{T})$, where $\mathcal{D}_\mathcal{F}$ is a distribution over all perturbed basic blocks that retain

the features in $\mathcal{F}$, $\Pi(\mathcal{F})$. We identify that the probability is analogous to *precision* (4) used in prior work (Ribeiro et al., 2018), and hence we adopt this terminology. Thus, probabilistic faithful explanations $\mathcal{F}$ must satisfy the condition $\hat{\varphi}(\mathcal{F})$, given by (5), where $0 \leq \delta \leq 1$.

$$Prec(\mathcal{F}) \triangleq Pr_{\alpha \sim \mathcal{D}_\mathcal{F}}(\mathcal{M}(\alpha) \in \mathcal{T}) \quad (4)$$

$$\hat{\varphi}(\mathcal{F}) \triangleq (\mathcal{F} \subseteq \mathcal{P}) \text{ and } (Prec(\mathcal{F}) \geq (1 - \delta)) \quad (5)$$

As the distribution over basic blocks, $\mathcal{D}_\mathcal{F}$ in (4) should be such that $\hat{\varphi}(\mathcal{F})$ closely approximates the ideal faithfulness condition (1) which has no prioritization over the perturbed basic blocks, it should ideally be a uniform distribution over its sample space $\Pi(\mathcal{F})$ and hence depend on $\mathcal{F}$.

Probabilistic generalizability and simplicity. To relax the computation in (2) we overapproximate it with the perturbed basic blocks' set, $\Pi(\mathcal{F})$. Thus, for higher generalizability, we maximize $|\Pi(\mathcal{F})|$. Note that Π is a monotonically decreasing function (proof in Appendix A). Thus, for simplicity of explanations too we can maximize $|\Pi(\mathcal{F})|$. We normalize $|\Pi(\mathcal{F})|$ with the number of all possible perturbations of the basic block, $|\Pi(\emptyset)|$ where $\emptyset$ denotes an empty set of features to preserve. $\Pi(\emptyset)$ is independent of $\mathcal{F}$ and hence the normalization will not affect the optimization problem's output. Intuitively, the resultant fraction in the optimization objective would denote the fraction of all possible perturbations that preserve the feature set $\mathcal{F}$. We relax this computation by replacing it with the probability of finding the features in $\mathcal{F}$ in a randomly selected valid perturbation of the basic block. We identify that this probability is analogous to *coverage* in prior work (Ribeiro et al., 2018), and hence we adopt this terminology. Coverage constitutes a probabilistic notion of generalizability and simplicity of explanations, and hence we maximize the coverage in our optimization objective. (6) defines the coverage of a set of features $\mathcal{F}$, where $\mathcal{D}$ is a distribution over all perturbations of the input basic block, $\Pi(\emptyset)$. To obtain an unbiased measure of coverage, $\mathcal{D}$ should ideally be a uniform distribution.

$$Cov(\mathcal{F}) \triangleq Pr_{\alpha \sim \mathcal{D}}(\mathcal{F} \subseteq \mathcal{P}_\alpha) \quad (6)$$

Overall practical optimization problem. Thus, our optimization problem to practically find the desirable explanation $\hat{\mathcal{F}}^*$ for $\mathcal{M}(\beta)$ becomes (7).

$$\hat{\mathcal{F}}^* \triangleq \underset{\mathcal{F} \text{ s.t. } \mathcal{F} \subseteq \mathcal{P}}{argmax}|Cov(\mathcal{F})| \text{ s.t. } Prec(\mathcal{F}) \geq (1 - \delta) \quad (7)$$

5 COMET: NEURAL COST MODEL EXPLANATION FRAMEWORK

This section presents COMET, our novel framework for efficiently generating desirable explanations for the predictions made by a given cost model for a target basic block.

The core operation of COMET is to efficiently solve the optimization problem in (7). While COMET is not conceptually limited to any particular Instruction Set Architecture (ISA), we develop the framework for one of the most popular ISA —x86 and leave the development for other ISAs to future work. An overview of COMET's algorithm on an example x86 basic block is shown in Figure 1. To create interpretable explanations, COMET first decomposes the input basic block β into its features. We restrict $\mathcal{P}$, which consists of all possible features of a basic block, to block features $\hat{\mathcal{P}} \subset \mathcal{P}$ [Section 5.1], to reduce the possible sets of features to evaluate in the optimization problem in (7). Next, we need to evaluate the precision of each subset of $\hat{\mathcal{P}}$ to identify a subset $\mathcal{F}$ that has $Prec(\mathcal{F}) \geq (1 - \delta)$ and has maximum $Cov(\mathcal{F})$. To efficiently solve this constrained optimization problem, COMET adapts the Anchors explanation algorithm (Ribeiro et al., 2018), which has a similar optimization objective [Section 5.2].

5.1   Extracting block features

COMET casts the input basic block into a multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which we describe next. Figure 1(ii) shows the multigraph for the example block. We define $\mathcal{V}$ as the set of vertices of the multigraph, corresponding to all the instructions, annotated with their positions in the block. $\mathcal{E}$ consists of directed edges between instructions that have data dependencies, labeled by the types of data dependency hazards between them. Please refer to Appendix B for information on the different types of data dependencies commonly found in assembly basic blocks. Figure 1(ii) shows the RAW type of hazard that is present in the example block. Separate edges for different dependencies help identify specific dependencies as bottlenecks in explanations. Distinguishing dependencies can be crucial to debugging cost models. If a cost model is found to base its predictions on dependencies that are optimized by the compiler, developers can debug the cost model to eradicate any spurious correlations.

We constitute $\hat{\mathcal{P}}$ with the instructions, data dependencies, and number of instructions of the block. Figure 1(iii) shows the features in $\hat{\mathcal{P}}$ for the example basic block. These features are used in the design of popular hand-engineered cost models (Abel and Reineke, 2022; Intel, 2017; Di Biagio and Davis, 2018), correspond better with the neural cost models (Mendis et al., 2019a), and hence are interpretable for our stakeholders. We restrict to these common features used in popular cost models to make COMET focus on the important set of features and generate explanations efficiently.

5.2   Efficiently computing explanations

To efficiently compute explanations, COMET empirically estimates $Prec(\mathcal{F})$ and $Cov(\mathcal{F})$ with samples from basic block distributions, $\mathcal{D}_\mathcal{F}$ and $\mathcal{D}$ respectively. We have de-

signed *basic block perturbation algorithms* to sample from $\mathcal{D}_\mathcal{F}$ and $\mathcal{D}$, which essentially perturb basic block β to obtain blocks β' according to the corresponding distribution from the underlying sample space. As discussed in Section 4.2, we want both $\mathcal{D}_\mathcal{F}$ and $\mathcal{D}$ to be uniform distributions over their respective sample spaces to compute unbiased approximations of the ideal desirable explanations. Observe that, $\mathcal{D}$ is hence a special case of $\mathcal{D}_\mathcal{F}$ with $\mathcal{F} = \emptyset$. Thus, a common perturbation algorithm can be used for both $\mathcal{D}_\mathcal{F}$ and $\mathcal{D}$.

Basic block perturbation algorithm. COMET's core basic block perturbation algorithm Γ takes a set of features $\mathcal{F} \subseteq \hat{\mathcal{P}}$ of basic block β as input and randomly perturbs β to obtain $\beta' \sim \mathcal{D}_\mathcal{F}$ such that β' retains the features in $\mathcal{F}$ and has some of the other features in $\hat{\mathcal{P}}$ perturbed to values valid according to the underlying ISA. Opcodes can be perturbed only to those that can accept the original set of operands, according to the ISA. Operands can be perturbed to only those having the same type and size. We elaborate on the perturbation algorithm further in this section. Figure 1(v) shows an example perturbation of β created by Γ when preserving the features in the retained features set [Figure 1(iv)]. While we ideally want $\mathcal{D}_\mathcal{F}$ to be a uniform distribution, its underlying sample space of perturbed basic blocks which preserve features in $\mathcal{F}$ is large (check Appendix F to get an idea of the magnitude of these sample spaces) and complex without a closed-form characterization and is also defined differently for individual $\mathcal{F}$. This makes designing an algorithm to generate uniform samples for each $\mathcal{F}$ hard. Hence we relax the requirement of sampling from a uniform distribution to the ability of Γ to produce diverse perturbed basic blocks so that the probability of obtaining a given basic block is small. Algorithm 1 in Appendix C presents the pseudocode of Γ to perturb a given basic block.

Γ perturbs the multigraph corresponding to the basic block, $\mathcal{G}$ to obtain $\mathcal{G}'$, which uniquely corresponds to the perturbed basic block, such that the features in $\mathcal{F}$ are preserved. To obtain $\mathcal{G}'$, Γ attempts to perturb every feature that is allowed to be perturbed, independent of the others. This is because any dependence will restrict the possible choices for perturbed blocks and hence disproportionately increase the probabilities of some possible perturbations. To create independence between the basic block features, Γ perturbs vertices of the basic block graph $\mathcal{G}$ independent of each other. Γ also perturbs the data dependency edges that do not have any vertex in common, independent of each other. However, when two data dependency edges have at least one vertex in common if they are caused by a common operand in the instruction corresponding to the common vertex, then all perturbations to edges can not be made completely independent, otherwise they are perturbed independently. For independence between vertex and data dependency edge perturbations, Γ perturbs only the opcode of the corresponding instruction of the vertex to denote vertex perturbation and only the

operands of the instructions connected by edge to denote edge perturbations. Γ preserves the opcodes of the instructions corresponding to every data dependency in $\mathcal{F}$ but can perturb other data dependencies between the instructions by operand changes.

Perturbing vertices of $\mathcal{G}$. Γ perturbs vertices by either deleting or replacing them with other valid vertices. Deletion is permissible when the number of instructions is not required to be preserved. When a vertex is deleted, all incoming and outgoing edges of the vertex are removed from $\mathcal{G}$. To replace a vertex, the corresponding instruction's opcode is replaced with another opcode in the ISA that can produce a valid assembly basic block instruction (an instruction that does not contain certain opcodes such as call or jmp) with the operands of the original instruction. Overall, Γ independently perturbs or retains every vertex with equal probability, where a vertex is perturbed by either deleting or replacing it, again with equal probability.

Perturbing edges of $\mathcal{G}$. Γ perturbs data dependency edges by deleting the corresponding dependency. The dependency is deleted by perturbing some operands corresponding to the dependency to other operands of the same type and size. The type of an operand could be memory, register, or immediate/constant, while its size could be any power of 2 between $8 - 512$ bits. Hence, we change the operand registers/memory addresses to other registers/memory addresses to break the data dependencies. Overall, Γ either perturbs or retains a data dependency by similar probabilities. The exact probabilities of perturbation and retention will be basic block specific and are discussed in Appendix D.

Computing explanations. With the basic block perturbation algorithm, $Prec(\mathcal{F})$ is estimated using KL-divergence-based confidence intervals (Kaufmann and Kalyanakrishnan, 2013) and $Cov(\mathcal{F})$ is estimated by its empirical value, for a given set of features $\mathcal{F} \subseteq \hat{\mathcal{P}}$. Similar to the Anchors' construction (Ribeiro et al., 2018), COMET iteratively builds its explanation feature set using a beam search wherein the maximum (estimated) precision feature sets at each level are iteratively expanded to larger feature sets till the precision threshold of $(1 - \delta)$ is exceeded. The maximum coverage feature set with precision $> (1 - \delta)$ is COMET's explanation for $\mathcal{M}(\beta)$.

6 EVALUATION

We evaluate COMET to answer two main questions:

Correctness. Do COMET's explanations accurately reflect the given cost model's behavior?

Utility. Can COMET's explanations be used to understand the behavior of cost models?

Experimental setup. All our experiments were conducted on a 12th Gen 20-core Intel i9 processor (cache size: 24MB, RAM: 32GB, clock speed: 2.5GHz, with AVX support). We set the precision threshold $(1 - \delta)$ in (4) as 0.7. We have set the probabilities of retention and perturbation of every feature in a basic block as 0.5. For instruction-type features where there are two possible perturbations, deletion and replacement, we assign probabilities to the perturbation operations based on an extensive hyperparameter study (Appendix E). We have used the default hyperparameters in the Anchor algorithm (Ribeiro et al., 2018) for the beam-search-based iterative explanation construction method. We study the sensitivity of COMET to its hyperparameters in Appendix E. We have developed and tested COMET for the x86 microarchitecture. We use basic blocks from the popular BHive dataset (Chen et al., 2019). We randomly pick 200 basic blocks with number of instructions between 4 and 10 from BHive, to make our *explanation test set* for testing COMET's explanation. We run each experiment for 5 different seeds and report the average results, with their standard deviations.

Computing the accuracy of COMET's explanations. To evaluate the correctness of COMET's explanations, we have developed a crude, but non-trivial, interpretable, analytical cost model, $\mathcal{C}$. The advantage of such a model is that it gives us reliable *ground truth of explanations* with which we can compare COMET's explanations and compute their *accuracy*. We are not aware of any actual intricate analytical cost models that have a closed-form representation that could give us ground truth explanations to objectively compute COMET's accuracy, which is why we had to design $\mathcal{C}$ for COMET's evaluation. We define $cost_{inst}(inst)$, $cost_{dep}(\delta_{ij})$, and $cost_\eta(n)$ as the costs of the instruction $inst$, data dependency δ_{ij} between instructions i and j, and number of instructions $\eta = n$ respectively in a given basic block. (8) presents the functional form of $\mathcal{C}$. $\mathcal{C}$ identifies the features of the basic block β which have the maximum cost and hence are bottlenecks for its execution, and predicts their cost as β's cost. Our rationale behind $\mathcal{C}$ is derived from a throughput prediction baseline analytical model in (Abel and Reineke, 2022) whose throughput prediction is the maximum of the individual costs for the number of instructions, the number of memory reads, and the number of memory writes in the input basic blocks. In $\mathcal{C}$ we have instead picked up basic block features such as its instructions and data dependencies to make the cost predictions more specific to a given block. Thus, $\mathcal{C}$ serves as a realistic, interpretable cost model, to measure the accuracy of COMET's explanations. Custom $\mathcal{C}$ models can be developed for each microarchitecture when the individual cost functions vary with the microarchitecture. The exact, microarchitecture-dependent forms of the 3 cost functions used in our experiments are

Table 2. Accuracy of COMET's explanations.

Explanation	Acc.(%) over $\mathcal{C}_{HSW}$	Acc.(%) over $\mathcal{C}_{SKL}$
Random	26.56 ± 20.30	26.60 ± 20.34
Fixed	72.33	74.0
COMET	$\mathbf{96.90 \pm 0.92}$	$\mathbf{98.00 \pm 0.80}$

Table 3. Average Precision and Coverage for COMET's explanations for Ithemal (I) and uiCA (U) for Haswell and Skylake.

Model	Av. Precision	Av. Coverage
I (HSW)	0.79 ± 0.005	0.19 ± 0.007
I (SKL)	0.81 ± 0.004	0.19 ± 0.014
U (HSW)	0.78 ± 0.006	0.18 ± 0.012
U (SKL)	0.79 ± 0.006	0.18 ± 0.012

given in Appendix G.

$$\mathcal{C}(\beta) = max\{cost_\eta(n), \underset{i}{max}\{cost_{inst}(inst_i)\},$$
$$\underset{\delta_{ij}}{max}\{cost_{dep}(\delta_{ij})\}\} \quad (8)$$

The ground truth explanation for $\mathcal{C}(\beta)$ is given by $GT(\beta)$ (9), where $type(f)$ is the type of the feature f which would be one of $inst$, dep, and η. $GT(\beta)$ essentially is the set of basic block features that have the maximum cost among the costs for all the features.

$$GT(\beta) = \{f \mid f \in \hat{\mathcal{P}}, cost_{\langle type(f) \rangle}(f) = \mathcal{C}(\beta)\} \quad (9)$$

Note that $GT(\beta)$ may not be a singleton set, as there can be multiple features that are equally important and lead to the same $\mathcal{C}(\beta)$. We consider an explanation for $\mathcal{C}$'s prediction for β to be accurate if it identifies at least one feature from $GT(\beta)$ and nothing outside $GT(\beta)$. We are not aware of any other competent cost model explanation methods to compare COMET's accuracy against, hence we design two natural baseline explanation algorithms: *random* and *fixed*. The random explanation baseline includes features f of β based on the probability of occurrence of a feature of $type(f)$ in the set of all ground truth explanations of all basic blocks in the explanation test set. The fixed explanation baseline identifies the most frequent feature type in the set of ground truth explanations for all blocks in the explanation test set and assigns the first feature of that type in the block to be the fixed explanation.

6.1 Accuracy-based evaluation of COMET

Table 2 presents the explanation accuracy achieved by COMET and the explanation baselines over $\mathcal{C}$ for the Haswell (HSW) and Skylake (SKL) microarchitectures. The accuracy values indicate a significant improvement in the correctness of explanations given by COMET over the baselines and testify the correctness of COMET's explanations.

Note that as the fixed explanation baseline does not have any randomness, it does not have any uncertainty.

The high accuracy of COMET's explanations over $\mathcal{C}$, which makes its cost predictions using the same set of features as COMET, indicates that COMET can faithfully identify the set of features that lead to the prediction when they are within the set of features that it uses to compose explanations. Note that this high accuracy has been achieved with just query access to the cost model. However, for actual cost models, it may not be the case that COMET's explanation features are used directly for cost prediction. Generally, some complex functions of these basic features will be used to obtain the cost. Hence, we next estimate the precision of COMET's explanations for actual cost models.

6.2 Precision and coverage evaluation

Next, we study the average precision and coverage of COMET's explanations for state-of-the-art throughput-predicting cost models: neural model Ithemal (Mendis et al., 2019a), and simulation-based model uiCA (Abel and Reineke, 2022) over the basic blocks in the explanation test set. We selected Ithemal and uiCA as representative cost models due to their high prediction accuracy and popularity among our stakeholders. COMET is applicable to other models as well, as it requires just query access to them. The average precision and average coverage are metrics to indicate COMET's potential for generating faithful and generalizable explanations respectively of a target cost model for individual basic blocks in our explanation test set. As these cost models are not analytical, they do not have ground-truth explanations, and hence we use average precision and average coverage as proxies to evaluate the explanations, similar to (Ribeiro et al., 2018). The average time taken to explain a block for each model is roughly a minute. Table 3 presents our findings for Ithemal and uiCA developed for Haswell (HSW) and Skylake (SKL) microarchitectures.

We observe that the explanations for all the cost models have fairly high average precision (probability of being faithful). We estimate the coverage (generalizability) of each explanation over $10k$ perturbed basic blocks. Thus, an average coverage of 0.19 means that the explanation for 1 block generalizes to 1900 sampled blocks on average, making it significant for the complex domain of basic blocks. The coverage values obtained are similar to the coverage achieved while explaining models in NLP (Ribeiro et al., 2018). These results indicate that the high accuracy of COMET over our custom cost model $\mathcal{C}$ transfers to state-of-the-art cost models as well and COMET can be deployed to obtain high-quality explanations for common cost models. Next, we study how COMET can become an essential model analysis tool for our stakeholders.
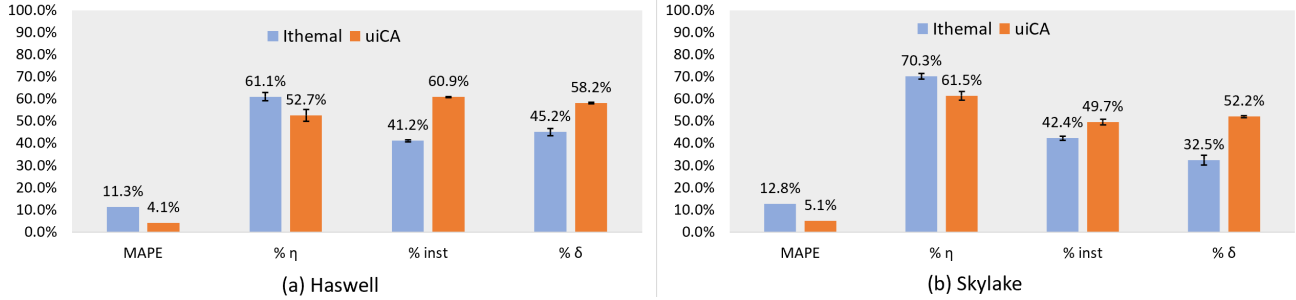
Figure 2. Variation of Mean Absolute Percentage Error (MAPE) in Ithemal and uiCA alongside variation in the % of explanations consisting: number of instructions η, specific instructions $inst$ and data dependencies δ. Figure (a): Haswell, (b): Skylake
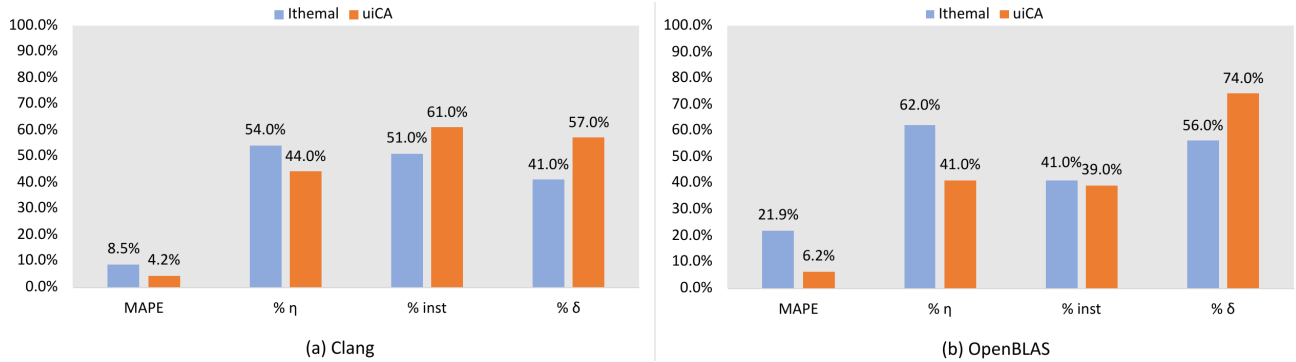

Figure 3. Variation of Mean Absolute Percentage Error (MAPE) in Ithemal and uiCA with the % of explanations consisting: number of instructions η, instructions $inst$ and data dependencies δ. BHive sources: (a) Clang, (b) OpenBLAS

6.3 Evaluating utility of COMET

We show a use case of COMET wherein we investigate the variation in the prediction errors of Ithemal and uiCA and empirically check its correlation with the dependence of the model's output on different types of block features. We hypothesize that as the error of the cost model decreases, its dependence on the finer-grained features of the block will increase. Out of the 3 types of features over which COMET composes its explanations, we identify the block's instructions and data dependencies as more specific, finer-grained features when compared to the feature corresponding to the number of instructions in the block. We use COMET's explanations to identify the block features on which the model's prediction depends. Note that our hypothesis is not obvious even for arbitrary traditional cost models (let alone the black-box neural cost models), as is evident from the performance of a baseline cost model in Abel and Reineke (2022), Table 1, that uses coarse-grained features of a basic block and achieves higher accuracy than LLVM-MCA (Di Biagio and Davis, 2018) which uses finer-grained features. Figure 2 shows the results of our investigation. It shows the variation of mean absolute percentage error of Ithemal and uiCA. Alongside the error, it shows the percentage of COMET's explanations over the entire explanation

test set that contain features corresponding to the number of instructions η, instructions $inst$, and data dependencies δ in the explained basic block.

The trends in Figure 2 for both Haswell and Skylake confirm our hypothesis. Interpret this insight as follows: as the cost model becomes more accurate, it focuses more on the finer-grained features of the basic block, as indicated by COMET's explanations. We discuss similar insights obtained for blocks derived from different partitions of the BHive dataset, as described in Appendix H.1 next. We omit the error bars for clarity, as the standard deviations in our results are generally low [Figure 2].

BHive partitions by source. We study the explanations for blocks in BHive derived from the Clang and OpenBLAS sources. We select 100 unique blocks from each source to separately analyze our hypothesis. Figure 3 presents our findings and confirms our hypothesis for both partitions.

BHive partitions by category. We conduct a similar study on 50 unique basic blocks corresponding to each category in the BHive dataset. Figure 4 presents our findings and confirms our hypothesis for all categories. Interestingly, for the *Store* category, as the error in throughput predictions of both cost models is similar, we observe similar prominence of
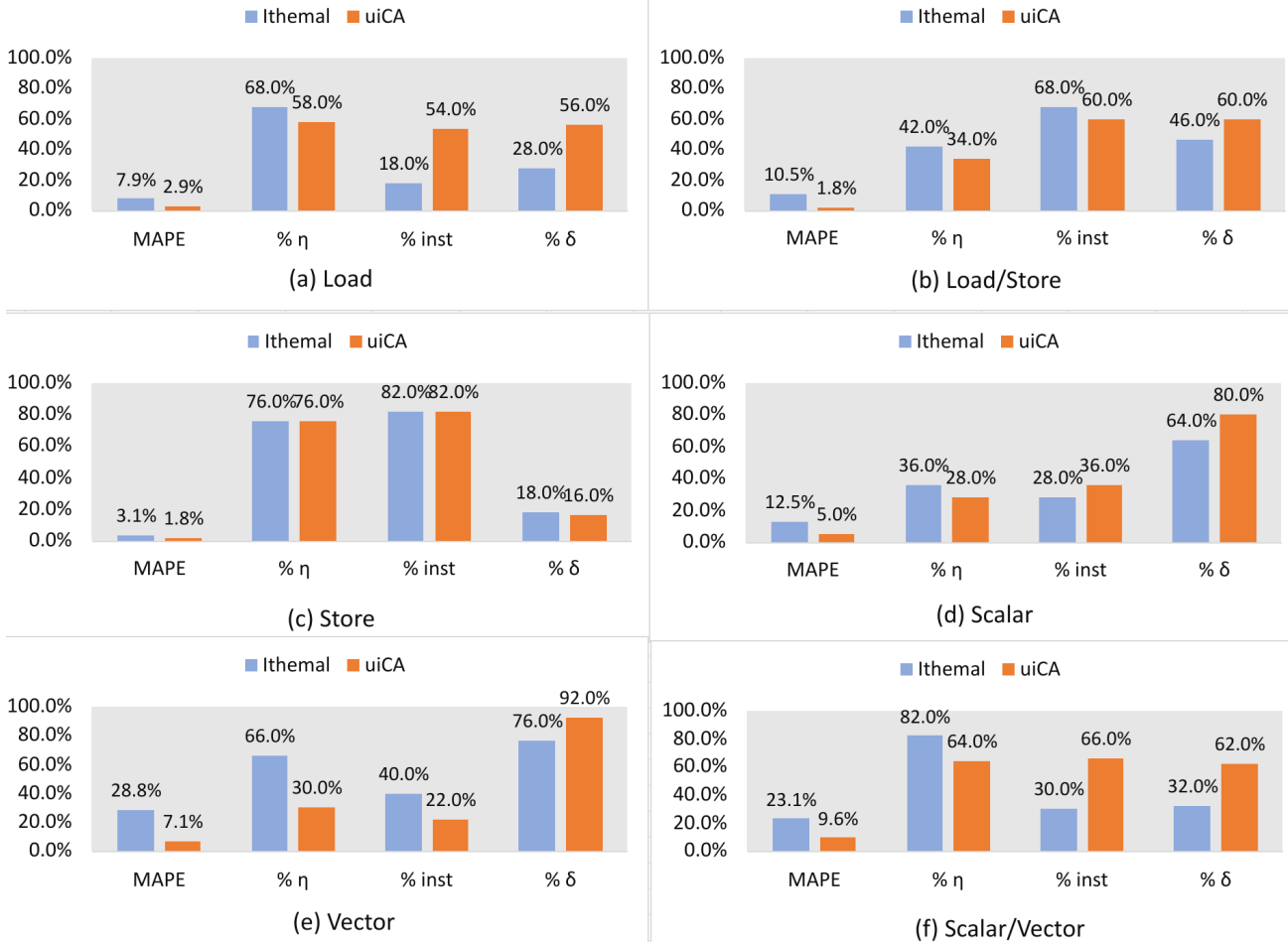
Figure 4. Mean Absolute Percentage Error (MAPE) in Ithemal and uiCA and the % of explanations having: number of instructions η, instructions $inst$ and data dependencies δ. BHive categories: (a) Load, (b) Load/Store, (c) Store, (d) Scalar, (e) Vector, (f) Scalar/Vector

all types of features in COMET's explanations for both cost models. This observation further supports our hypothesis.

6.4 Case studies

Next, we show another use case of COMET's explanations —to conduct analyses of cost prediction of individual basic blocks. Similar analyses can be useful to understand the cost model's behavior in corner cases. We discuss COMET's explanations for the predictions of Ithemal and uiCA for Haswell on randomly picked blocks from the BHive dataset.

Case study 1. The block in Listing 2 is predicted to have a throughput of 2 cycles by both cost models which matches the throughput on actual hardware reported in the BHive dataset. Instructions 2 and 3 write to the memory and are thus the highest throughput instructions (Abel and Reineke, 2019; Fog et al., 2011). Hence intuitively, for correct prediction, these instructions are important. COMET's explanations for both cost models match this intuition, thus suggest-

ing that both cost models actually consider the intuitive set of features to correctly predict throughput for this block.

```
1  lea  rdx , [ rax + 1]
2  mov  qword ptr [ rdi + 24] , rdx
3  mov  byte ptr [ rax ] , 80
4  mov  rsi , qword ptr [ r14 + 32]
5  mov  rdi , rbp
```

	Prediction	Explanation
Ithemal	2 cycles	$\{inst_2, inst_3\}$
uiCA	2 cycles	$\{inst_2, inst_3\}$

Listing 2. Case Study 1

Case study 2. The block in Listing 3 has a division instruction and many data dependencies such as a RAW data dependency between instructions 3 and 6 due to register rax and a WAR dependency between instructions 1 and 2 due to register edx. A div instruction is a very expensive instruction in general (Abel and Reineke, 2019; Fog et al.,

2011). The actual throughput of the basic block is 39 cycles. Thus, both cost models have made incorrect predictions, but the prediction of Ithemal is more erroneous as compared to uiCA. COMET's explanation for Ithemal consists of just the feature corresponding to the number of instructions in the basic block, while that for uiCA consists of the div instruction and a data dependency. These explanations suggest that Ithemal does not sufficiently prioritize costly instructions such as div and data dependencies, unlike the actual microarchitecture that Ithemal is trained to mimic, thus indicating potential sources of its throughput-prediction error.

```
1  mov ecx , edx
2  xor edx , edx
3  lea rax , [ rcx + rax − 1]
4  div rcx
5  mov rdx , rcx
6  imul rax , rcx
```

	Prediction	Explanations
Ithemal	23 cycles	$\{\eta(num\_insts)\}$
uiCA	36 cycles	$\{\delta_{RAW,3,6}, inst_4\}$

Listing 3. Case Study 2

7 DISCUSSION AND FUTURE WORK

We demonstrated how COMET's explanations can be used to gain both high-level [Section 6.3] and case-specific [Section 6.4] insights about cost models and compare their behaviors against other cost models. These insights can be useful for repairing high-error neural models with domain-specific insights and developing more generalizable models in the future. As indicated by these insights, neural architectures that explicitly utilize the finer-grained features of blocks can achieve better cost prediction performance. COMET's explanations can be used to select a model from a collection of similar performing neural models. COMET can be extended to run on GPUs to make it amenable to integration with cost model training and inference procedures, in the future. COMET's feedback can be leveraged to update the model parameters during training to have the predictions rely on finer-grained features. COMET can be augmented to existing cost models to guide compiler optimizations with information on what parts of the basic block need to be optimized for better performance.

While explanation features employed by COMET currently capture the commonly used properties of a block, it will produce approximations for the most important factors behind a model's predictions when they cannot be captured by the current features. We will investigate expanding the explanation features in future work. Finally, COMET can be extended to other open-source ISAs, including those for GPUs and TPUs, by mapping the current perturbation algorithm to the new ISA. We need to define the opcodes (operands) that could replace each opcode (operand) to generate a valid perturbation. While the high-level formalism can be carried over, instance-specific challenges can arise.

8 CONCLUSION

We presented COMET, the first approach for efficiently generating faithful, generalizable, and interpretable explanations for neural cost models. Our results show that COMET can generate accurate and useful explanations that indicate potential sources of errors. We believe that COMET's explanations can be used to improve trust in the workings of neural cost models and accelerate their real-world adoption.

9 ACKNOWLEDGEMENT

REFERENCES

Andreas Abel and Jan Reineke. uops.info: Characterizing latency, throughput, and port usage of instructions on intel microarchitectures. In *ASPLOS*, ASPLOS '19, pages 673–686, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6240-5. doi: 10.1145/3297858.3304062.

Andreas Abel and Jan Reineke. uiCA: Accurate throughput prediction of basic blocks on recent Intel microarchitectures. In Lawrence Rauchwerger, Kirk Cameron, Dimitrios S. Nikolopoulos, and Dionisios Pnevmatikatos, editors, *ICS '22: 2022 International Conference on Supercomputing, Virtual Event, USA, June 27-30, 2022*, ICS '22, pages 1–12. ACM, June 2022.

Marcelo Arenas, Daniel Báez, Pablo Barceló, Jorge Pérez, and Bernardo Subercaseaux. Foundations of symbolic languages for model interpretability. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 11690–11701. Curran Associates, Inc., 2021.

Riyadh Baghdadi, Massinissa Merouani, Mohamed-Hicham Leghettas, Kamel Abdous, Taha Arbaoui, Karima Benatchba, et al. A deep learning based cost model for automatic code optimization. *Proceedings of Machine Learning and Systems*, 3:181–193, 2021.

Jonathan Bartlett. *Nasm (Intel) Assembly Language Syntax*, pages 271–273. Apress, Berkeley, CA, 2021. ISBN 978-1-4842-7437-8. doi: 10.1007/978-1-4842-7437-8_23.

Ryma Boumazouza, Fahima Cheikh-Alili, Bertrand Mazure, and Karim Tabia. ASTERYX. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. ACM, Oct 2021. doi: 10.1145/3459637.3482321.

Nghi DQ Bui, Yijun Yu, and Lingxiao Jiang. Autofocus: interpreting attention-based neural networks by code perturbation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 38–41. IEEE, 2019.

Yishen Chen, Ajay Brahmakshatriya, Charith Mendis, Alex Renda, Eric Atkinson, Ondrej Sykora, Saman Amarasinghe, and Michael Carbin. Bhive: A benchmark suite and measurement framework for validating x86-64 basic block performance models. In *2019 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2019.

Jürgen Cito, Isil Dillig, Vijayaraghavan Murali, and Satish Chandra. Counterfactual explanations for models of code. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '22, page 125–134, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392266. doi: 10.1145/3510457.3513081.

Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. End-to-end deep learning of optimization heuristics. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 219–232, 2017. doi: 10.1109/PACT.2017.24.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019.

A. Di Biagio and M. Davis. llvm-mca. 2018. URL https://lists.llvm.org/pipermail/llvm-dev/2018-March/121490.html.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.139.

Agner Fog et al. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for intel, amd and via cpus. *Copenhagen University College of Engineering*, 93:110, 2011.

Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:1511–1519, 07 2019. doi: 10.1609/aaai.v33i01.33011511.

Intel. Intel architecture code analyzer. 2017. URL https://software.intel.com/en-us/articles/intel-architecture-code-analyzer.

Sam Kaufman, Phitchaya Phothilimthana, Yanqi Zhou, Charith Mendis, Sudip Roy, Amit Sabne, and Mike Burrows. A learned performance model for tensor processing units. volume 3, pages 387–400, 2021.

E. Kaufmann and S. Kalyanakrishnan. Information complexity in bandit subset selection. *Journal of Machine Learning Research*, 30:228–251, 01 2013.

Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1675–1684, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939874.

Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Jure Leskovec. Faithful and customizable explanations of black box models. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '19, page 131–138, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450363242. doi: 10.1145/3306618.3314229.

J. Laukemann, J. Hammer, J. Hofmann, G. Hager, and G. Wellein. Automated instruction stream throughput prediction for intel and amd microarchitectures. In *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 121–131, 2018.

Shusen Liu, Zhimin Li, Tao Li, Vivek Srikumar, Valerio Pascucci, and Peer-Timo Bremer. Nlize: A perturbation-driven visual interrogation tool for analyzing and interpreting natural language inference models. *IEEE transactions on visualization and computer graphics*, 25(1):651–660, 2018.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Joao Marques-Silva. Logic-based explainability in machine learning. In Leopoldo Bertossi and Guohui Xiao, editors, *Reasoning Web. Causality, Explanations and Declarative Knowledge: 18th International Summer School 2022, Berlin, Germany, September 27–30, 2022, Tutorial Lectures*, pages 24–104, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-31414-8. doi: 10.1007/978-3-031-31414-8_2.

David Martens and Foster Provost. Explaining data-driven document classifications. volume 38, page 73–100, USA, mar 2014. Society for Information Management and The Management Information Systems Research Center. doi: 10.25300/MISQ/2014/38.1.04.

Charith Mendis, Alex Renda, Dr.Saman Amarasinghe, and Michael Carbin. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4505–4515. PMLR, 09–15 Jun 2019a.

Charith Mendis, Cambridge Yang, Yewen Pu, Saman Amarasinghe, and Michael Carbin. Compiler auto-vectorization with imitation learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019b. Curran Associates Inc.

Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022. URL https://christophm.github.io/interpretable-ml-book.

David A. Patterson and John L. Hennessy. *Computer Organization and Design*. Morgan Kaufmann Publishers, 2nd edition, 1998. ISBN 15-586-0428-6.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939778.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Fabian Ritter and Sebastian Hack. *AnICA: analyzing inconsistencies in microarchitectural code analyzers*, volume 6. Association for Computing Machinery, New York, NY, USA, oct 2022. doi: 10.1145/3563288.

Marko Robnik-Šikonja and Marko Bohanec. *Perturbation-Based Explanations of Prediction Models*, pages 159–175. Springer International Publishing, Cham, 2018. ISBN 978-3-319-90403-0. doi: 10.1007/978-3-319-90403-0_9.

Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, page 305–316, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318709. doi: 10.1145/2451116.2451150.

Junghoon Seo, Jeongyeol Choe, Jamyoung Koo, Seunghyeon Jeon, Beomsu Kim, and Taegyun Jeon. Noise-adding methods of saliency map as series of higher order partial derivative. In *ICML Workshop on Human Interpretability in Machine Learning (WHI 2018)*, 2018.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*, 2014.

Varshini Subhash, Zixi Chen, Marton Havasi, Weiwei Pan, and Finale Doshi-Velez. What makes a good explanation?: A harmonized view of properties of explanations. In *Progress and Challenges in Building Trustworthy Embodied AI at NeurIPS 2022*, 2022.

Ondřej Sýkora, Phitchaya Mangpo Phothilimthana, Charith Mendis, and Amir Yazdanbakhsh. Granite: A graph neural network model for basic block throughput estimation. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*, pages 14–26, Los Alamitos, CA, USA, nov 2022. IEEE Computer Society. doi: 10.1109/IISWC55918.2022.00012.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.