COMET: Neural Cost Model Explanation Framework

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

[1]University of Illinois Urbana-Champaign [2]MIT CSAIL [3]VMWare Research

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 compilers. 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. We obtain an empirical inverse correlation between the prediction errors of Ithemal and uiCA and the granularity of basic block features in COMET's explanations for them, indicating potential reasons for Ithemal's higher error 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 [27, 10] and superoptimization [34]. They can be simulation-based, static-analysis-based, or learned models. Simulation-based cost models, such as uiCA [2] and LLVM-MCA [12], 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. As these models are traditional programs, domain experts can intuitively understand and debug them, and hence they are commonly deployed. Static-analysis-based cost models, such as IACA [16] and OSACA [21] use a model of the target CPU and static analysis methods to predict the cost of a given basic block. The above types of cost models require significant engineering effort to construct and must be manually re-engineered for different CPU microarchitectures.

Alternatively, machine learning can be used to learn a cost model [26, 17, 4, 37]. Developing ML-based cost models requires the one-time effort of collecting a dataset of representative programs with their execution costs on the target CPU and training a selected type of ML model. While simple ML models could be used for constructing cost models, prior work [26, 37, 17, 4] has used neural networks as cost predictors to precisely approximate the complex function mapping basic blocks to their costs. An example is Ithemal [26], an LSTM model trained on the BHive [7] 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 [7]. It 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 [28].

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. Neural cost models could have arbitrary architectures [26, 37], requiring custom

[*] Correspondence to: Isha Chaudhary <isha4@illinois.edu>

XAI in Action: Past, Present, and Future Applications @ NeurIPS 2023.

explanation 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 goal, we develop our explanation framework to generate explanations that (i) assume just query-access to the cost model, (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 [8]. 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 [30, 31] in domains such as Vision or NLP have used locally-perturbed inputs to efficiently generate explanations with only query access to the model. In the discrete domain of basic blocks, there is no well-defined concept of locality. Hence, we need custom perturbation algorithms to handle this domain-specific challenge and closely approximate the complex behavior of a given cost model in a reasonable number of queries.

Our approach. 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, which we observe to be intractable. Hence, we relax our requirements and develop COMET, a perturbation-based explanation framework based on the design of (i) novel primitives for explanations including both coarse-grained (e.g. instruction count) 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 to gauge the complex behaviors of cost models.

Contributions. We make the following contributions:

1. We formalize the ideal, query-based, block-specific neural cost model explanations with desirable properties as an optimization problem that is Instruction Set Architecture (ISA) agnostic.
2. We relax the problem to practically solve it. Building on our relaxation, we present COMET (COst Model ExplanaTion framework), a novel and efficient explanation framework for neural cost models. As COMET is ISA-dependent, 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 [7]. 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 Formalizing cost model explanations

In this section, we formalize query-based block-specific explanations for neural cost models and discuss their desirable properties. An explanation is a minimal set of features of the input basic block whose presence is sufficient for the cost model's prediction to nearly be the original prediction.

We denote the cost model as a function $\mathcal{M}$ that maps valid basic blocks in a given Instruction Set Architecture (ISA) to real-valued costs. The smallest 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). The rest of this paper describes our approach for generating explanations for cost model $\mathcal{M}$'s prediction for a basic block β. To simplify notation, unless mentioned otherwise, we will omit $\mathcal{M}$ and β from symbols, e.g., $\mathcal{P}_\beta$ will be written as $\mathcal{P}$.

2.1 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 $\mathcal{M}$. The desirable properties of $\mathcal{F}^*$ are that it should be *faithful* to the cost model's behavior, *generalizable* across multiple basic blocks, and be *simple* to comprehend [8], 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 (possibly none) perturbations to the features in $\mathcal{P} \setminus \mathcal{F}$ in β. 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(iv) is an element in the output set of basic blocks, as it perturbs some features not in the input set of features.

Faithfulness. Let $\mathcal{T}$ be an $\epsilon-$ball around $\mathcal{M}(\beta)$, where $\epsilon > 0$ is a small constant. 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. their predictions are in $\mathcal{T}$. (1) captures faithfulness as a logical statement $\varphi(\mathcal{F})$ that is satisfied by faithful explanation $\mathcal{F}$.

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

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 target cost model's behavior.

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.

For higher interpretability, ideal explanation $\mathcal{F}^*$ should be simple. A common metric for sets of features used as explanations is their cardinality [33, 28]. Hence, for simple, faithful, and generalizable explanations $\mathcal{F}^*$, we solve the optimization problem (3), where $\lambda > 0$ is a regularization parameter.

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

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

2.2 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, $\Pi(\mathcal{F})$. Appendix D contains examples of cardinality estimates of $\Pi(\mathcal{F})$. Second, the computation in (2) requires computing faithful explanations for all basic blocks in $\Pi(\mathcal{F})$. Hence, to practically solve (3), we relax it as follows.

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 [31], and hence we adopt this terminology. Thus, probabilistic faithful explanations $\mathcal{F}$ must satisfy the condition $\hat{\varphi}(\mathcal{F})$, given by (5), where $0 \leq \delta \leq 1$. 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 $\Pi(\mathcal{F})$.

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

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

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 C). 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 [31], 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. Thus, our optimization problem for practical and desirable explanation $\hat{\mathcal{F}}^*$ for $\mathcal{M}(\beta)$ becomes (7).

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

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

3 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). As COMET leverages the features of the underlying Instruction Set Architecture (ISA), we develop it for the popular ISA —x86. We note, however, that COMET can be extended to other ISAs without significant conceptual modifications and hence leave it to future work. An overview of COMET's algorithm on an example x86 basic block is shown in Figure 1. 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 3.1], to reduce the possible sets of features to evaluate in the optimization problem in (7). To generate explanations from the obtained block features, COMET adapts the Anchors explanation algorithm [31], which has a similar optimization objective and solves (7) [Section 3.2].

3.1 Extracting block features

COMET casts the input basic block into a multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Figure 1(ii) shows the multigraph for the example block. $\mathcal{V}$ consists of the vertices of $\mathcal{G}$ corresponding to the instructions annotated with their positions in the block. $\mathcal{E}$ consists of directed edges between instructions that have data dependencies, labeled by the dependency type. Please refer to Appendix E for different types of data dependencies in assembly basic blocks. Figure 1(ii) shows the RAW data dependency in the example block. We constitute $\hat{\mathcal{P}}$ with the instructions, data dependencies, and number of instructions of the block. These features are important for the algorithms of common cost models [2, 26] and hence are interpretable for our stakeholders. Figure 1(iii) shows $\hat{\mathcal{P}}$ for the example basic block.

3.2 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 designed *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. As discussed in Section 2.2, 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 valid values. Figure 1(iv) shows an example perturbation of β created by Γ. While we ideally want $\mathcal{D}_\mathcal{F}$ to be a uniform distribution, its underlying sample space of perturbed basic blocks which preserve features

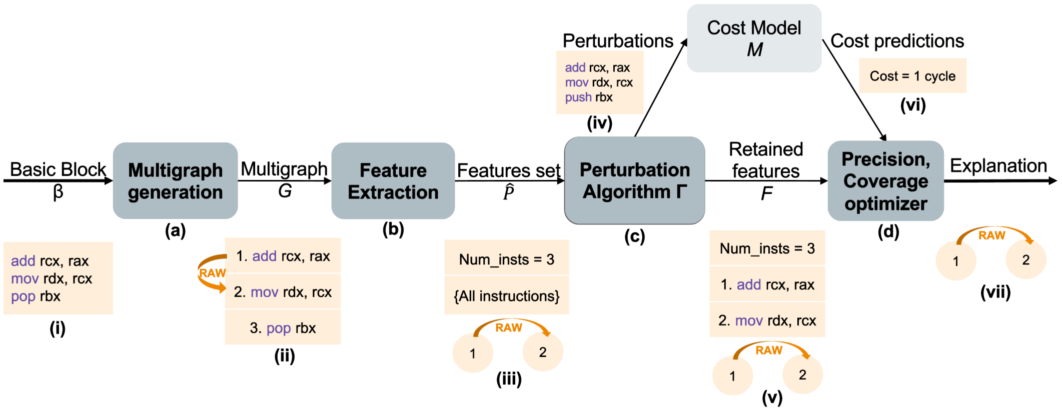


Figure 1: COMET is input a cost model $\mathcal{M}$ and a basic block β. It first converts β to a multigraph $\mathcal{G}$ in (a). $\mathcal{G}$ has the instructions and data dependencies of β as its vertices and edges respectively. The features $\hat{\mathcal{P}}$ of β, are then extracted from $\mathcal{G}$ 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 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 $\mathcal{M}(\beta)$.

in $\mathcal{F}$ is large (Appendix D) 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 F.2 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 restricts the possible perturbed blocks and hence disproportionately increases the probabilities of some perturbations. Γ 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. Γ perturbs only the opcode of the corresponding instruction of a vertex for vertex perturbation and only the operands of the instructions connected by an edge for edge perturbation, for independence. Γ preserves the vertices corresponding to every data dependency edge in $\mathcal{F}$ but can perturb other data dependency edges between those vertices. Γ perturbs individual vertices by either deleting or replacing them with other valid vertices and individual data dependency edges by deleting the corresponding dependency. The details of the perturbations are in Appendix F.1.

Computing explanations. $Prec(\mathcal{F})$ and $Cov(\mathcal{F})$ are empirically estimated for a given set of features $\mathcal{F} \subseteq \hat{\mathcal{P}}$ with Γ. Similar to the Anchors' construction [31], COMET iteratively builds its explanation feature set starting with an empty feature set, such that the maximum (estimated) precision feature sets at each level are expanded to all possible and distinct feature sets made by adding a block feature not present in them. Maximum precision feature sets are identified using the KL-LUCB [18] algorithm. More details are provided in Appendix F.4. Among the feature sets that have precision $> (1-\delta)$, the maximum coverage feature set is COMET's explanation for $\mathcal{M}(\beta)$.

4 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?

Table 1: Accuracy of COMET's explanations.

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

Experimental setup. Our experimental setup and hyperparameters are detailed in Appendix G.1. We have developed and tested COMET for the x86 microarchitecture. We use x86 basic blocks from the popular BHive dataset [7]. To analyze COMET's explanations, we randomly pick 200 basic blocks having 4 to 10 instructions from BHive, to make our *explanation test set*. 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 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 the *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 having 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 $\mathcal{C}$. $\mathcal{C}$ computes its cost predictions as the maximum cost of a feature over all the block features in the basic block, β. Our rationale behind $\mathcal{C}$ is derived from a throughput prediction baseline analytical model in [2]. Thus, $\mathcal{C}$ serves as a realistic, interpretable cost model, to measure the accuracy of COMET's explanations. The exact, microarchitecture-dependent forms of the 3 cost functions used in our experiments are given in Appendix I.

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

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.

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

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 call an explanation accurate if and only if it is a non-empty subset of $GT(\beta)$. We are not aware of any other competent cost model explanation methods to compare COMET's accuracy against, hence we compare COMET's explanations against those from two natural baseline explanation algorithms: *random* and *fixed*, described in Appendix G.2.

4.1 Accuracy-based evaluation of COMET

Table 1 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.

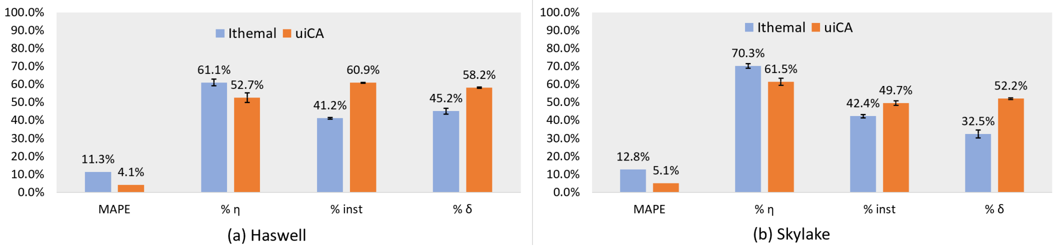
4.2 Precision and coverage evaluation

Next, we study the average precision and coverage of COMET's explanations for state-of-the-art throughput-predicting cost models: neural model Ithemal [26], and simulation-based model uiCA [2]

Table 2: Average Precision, Coverage, and Time for COMET's explanations

Model	Av. Precision	Av. Coverage	Av. Time (s)
Ithemal (HSW)	0.79 ± 0.005	0.19 ± 0.007	62.23 ± 2.0
Ithemal (SKL)	0.81 ± 0.004	0.19 ± 0.014	95.17 ± 38.7
uiCA (HSW)	0.78 ± 0.006	0.18 ± 0.012	96.88 ± 32.4
uiCA (SKL)	0.79 ± 0.006	0.18 ± 0.012	100.12 ± 23.6

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 which facilitate 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 [31]. We also analyze the average time taken to explain a block for each model. Table 2 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 faithfulness) and can be computed in a reasonable amount of time. The coverage values (generalizability) obtained are similar to the coverage of explanations of NLP models [31]. 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 show how COMET can become a useful analysis tool for our stakeholders.



(a) Haswell

(b) Skylake

Figure 2: Variation of Mean Absolute Percentage Error (MAPE) in Ithemal and uiCA with the % of explanations consisting: number of instructions η, specific instructions $inst$ and data dependencies δ.

4.3 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 study 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 block features 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 finer-grained features when compared to the feature corresponding to the block's number of instructions. We use COMET's explanations to identify the block features on which the model's prediction depends.

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. Such insights can be used by cost model developers to enhance the performance of their models. We discuss similar insights obtained for blocks derived from different partitions of the BHive dataset in Appendix K.1. We present case studies illustrating COMET's utility in explaining the predictions of both cost models on individual basic blocks in Appendix K.2.

References

[1] 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. URL `http://doi.acm.org/10.1145/3297858.3304062`.

[2] 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. URL `https://dl.acm.org/doi/pdf/10.1145/3524059.3532396`.

[3] 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. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/60cb558c40e4f18479664069d9642d5a-Paper.pdf`.

[4] Riyadh Baghdadi, Massinissa Merouani, Mohamed-Hicham Leghettas, Kamel Abdous, Taha Arbaoui, Karima Benatchba, and Saman Amarasinghe. A deep learning based cost model for automatic code optimization. 2021.

[5] Ryma Boumazouza, Fahima Cheikh-Alili, Bertrand Mazure, and Karim Tabia. ASTERYX. In *Proceedings of the 30th ACM International Conference on Information &amp Knowledge Management*. ACM, oct 2021. doi: 10.1145/3459637.3482321. URL `https://doi.org/10.1145%2F3459637.3482321`.

[6] 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.

[7] 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.

[8] Zixi Chen, Varshini Subhash, Marton Havasi, Weiwei Pan, and Finale Doshi-Velez. What makes a good explanation?: A harmonized view of properties of explanations, 2022.

[9] Jürgen Cito, Isil Dillig, Vijayaraghavan Murali, and Satish Chandra. Counterfactual explanations for models of code, 2021.

[10] 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.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

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

[13] 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, 2020.

[14] 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.

[15] 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.

[16] Intel. Intel architecture code analyzer, 2017. URL `https://software.intel.com/en-us/articles/intel-architecture-code-analyzer`.

[17] Samuel J. Kaufman, Phitchaya Mangpo Phothilimthana, Yanqi Zhou, Charith Mendis, Sudip Roy, Amit Sabne, and Mike Burrows. A learned performance model for tensor processing units, 2020.

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

[19] 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. URL `https://doi.org/10.1145/2939672.2939874`.

[20] 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. URL `https://doi.org/10.1145/3306618.3314229`.

[21] 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.

[22] 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.

[23] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017.

[24] Joao Marques-Silva. Logic-based explainability in machine learning, 2023.

[25] David Martens and Foster Provost. Explaining data-driven document classifications. *MIS Q.*, 38(1):73–100, mar 2014. ISSN 0276-7783. doi: 10.25300/MISQ/2014/38.1.04. URL `https://doi.org/10.25300/MISQ/2014/38.1.04`.

[26] Charith Mendis, Alex Renda, Saman Amarasinghe, and Michael Carbin. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. 2018.

[27] Charith Mendis, Cambridge Yang, Yewen Pu, Saman Amarasinghe, and Michael Carbin. *Compiler Auto-Vectorization with Imitation Learning*. Curran Associates Inc., Red Hook, NY, USA, 2019.

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

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

[30] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

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

[32] Fabian Ritter and Sebastian Hack. Anica: Analyzing inconsistencies in microarchitectural code analyzers, 2022.

[33] 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. URL `https://doi.org/10.1007/978-3-319-90403-0_9`.

[34] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization, 2012.

[35] 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, 2018.

[36] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013.

[37] Ondrej Sykora, Phitchaya Mangpo Phothilimthana, Charith Mendis, and Amir Yazdanbakhsh. Granite: A graph neural network model for basic block throughput estimation, 2022.

[38] 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.

A Related Work

Explanation techniques. Explanations for ML models consist of either building inherently interpretable ML models [19] or creating post-hoc explanations for the models [30, 31, 20, 25]. 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 [23] or for specific inputs [30, 31]. Explanation techniques can also be broadly classified as black-box [30, 31, 23] and white-box techniques [36, 35]. Further classifications of explanation techniques can be as perturbation/example-based [6, 22, 38] and symbolic explanation techniques [5, 24, 15, 3]. While symbolic methods give formal guarantees on the explanations, they do not scale to complex models. Hence, we have developed a scalable perturbation-based explanation method for high-quality explanations for cost models. [32] 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. [31] has used generative models [11, 13] to obtain input perturbations. These perturbations might not be syntactically correct and can result in erroneous explanations [9]. 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 [34] 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.

B Discussions and future work

We demonstrated how COMET's explanations can be used to gain both high-level [Section 4.3] and case-specific [Appendix K.2] 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 ISAs by replacing the x86 grammar with that of the given ISA.

C Monotonicity of perturbation function

Theorem 1. Π *is a monotonically decreasing function.*

Proof. Let $F_1, F_2 \in \wp(\mathcal{P})$ such that $F_1 \subseteq F_2$.

$$\begin{aligned}
\Pi(F_1) =& \{\beta' \mid \beta' \in B, F_1 \subseteq \mathcal{P}_{\beta'}, \mathcal{P}_{\beta'} \setminus F_1 \text{ are obtained from } \mathcal{P} \setminus F_1\} \\
=& \{\beta' \mid \beta' \in B, F_2 \subseteq \mathcal{P}_{\beta'}, \mathcal{P}_{\beta'} \setminus F_2 \text{ are obtained from } \mathcal{P} \setminus F_2\} \\
& \cup \{\beta' \mid \beta' \in B, F_1 \subseteq \mathcal{P}_{\beta'}, F_2 \not\subseteq \mathcal{P}_{\beta'}, \mathcal{P}_{\beta'} \setminus F_1 \text{ are obtained from } \mathcal{P} \setminus F_1\} \\
=& \Pi(F_2) \cup \{\beta' \mid \beta' \in B, F_1 \subseteq \mathcal{P}_{\beta'}, F_2 \not\subseteq \mathcal{P}_{\beta'}, \mathcal{P}_{\beta'} \setminus F_1 \text{ are obtained from } \mathcal{P} \setminus F_1\}
\end{aligned}$$

Hence, $\Pi(F_2) \subseteq \Pi(F_1)$ □

Π being a monotonically decreasing set function implies that the minimum cardinality input feature set $\mathcal{F}_{min}$ would have maximum cardinality output basic block set from Π, $\Pi(\mathcal{F}_{min})$.

Note that in the above proof, features in feature sets such as $\mathcal{P}_{\beta'} \setminus F_1$ are obtained by either retaining or perturbing the features in $\mathcal{P} \setminus F_1$.

A similar proof can be used to prove the monotonicity of $\hat{\Pi}$ as well.

D Perturbation function output sizes

The perturbation function, $\Pi_\beta : \wp(\mathcal{P}_\beta) \to \wp(\mathcal{B})$ maps a given set of basic block features $\mathcal{F}$ to the set of basic blocks $\mathcal{B}_\mathcal{F}$ that have $\mathcal{F}$ and where the other features are obtained from perturbations to the features in $\mathcal{P}_\beta \setminus \mathcal{F}$. In this section, we provide estimates of cardinalities of $\mathcal{B}_\mathcal{F}$ for some basic blocks β and feature sets $\mathcal{F}$. With this analysis, we allude to the practical intractability of generating ideal black-box explanations for cost models.

Note that, as $\mathcal{P}_\beta$ is the set of all features (all basic features and all of their functions) of β, it can be an infinite set itself. $\hat{\mathcal{P}}_\beta \subset \mathcal{P}_\beta$, hence for $\mathcal{F} \subseteq \hat{\mathcal{P}}_\beta$, $\hat{\Pi}_\beta(\mathcal{F}) \subseteq \Pi_\beta(\mathcal{F})$. Hence, $|\hat{\Pi}_\beta(\mathcal{F})| \leq |\Pi_\beta(\mathcal{F})|$. Thus, we provide estimates for $|\Pi_\beta(\mathcal{F})|$ by reporting the rough values for $|\hat{\Pi}_\beta(\mathcal{F})|$.

First, consider the basic block β_1 in Listing -1, for $\mathcal{F} = \emptyset$. $|\hat{\Pi}_{\beta_1}(\emptyset)| \approx 1.94 \times 10^{38}$. As we add more elements to $\mathcal{F}$, the size of $|\hat{\Pi}_{\beta_1}(\mathcal{F})|$ will reduce due to the constraints introduced to the perturbations.

```
1  vdivss xmm0, xmm0, xmm6
2  vmulss xmm7, xmm0, xmm0
3  vxorps xmm0, xmm0, xmm5
4  vaddss xmm7, xmm7, xmm3
5  vmulss xmm6, xmm6, xmm7
6  vdivss xmm6, xmm3, xmm6
7  vmulss xmm0, xmm6, xmm0
```

Listing -1: Basic block β_1 for perturbation function size estimation

Next, for $\mathcal{F} = \{inst_1\}$ i.e. with no perturbations to instruction 1 in β_1, $|\hat{\Pi}_{\beta_1}(\mathcal{F})| \approx 6.58 \times 10^{29}$.

Similarly, consider the basic block β_2 in Listing 0, for $\mathcal{F} = \emptyset$. $|\hat{\Pi}_{\beta_2}(\emptyset)| \approx 1.63 \times 10^{32}$. For $\mathcal{F} = \{inst_2\}$ i.e. with no perturbations to instruction 2 in β_2, $|\hat{\Pi}_{\beta_2}(\mathcal{F})| \approx 2.77 \times 10^{28}$.

```
 1  shl eax, 3
 2  imul rax, r15
 3  xor edx, edx
 4  add rax, 7
 5  shr rax, 3
 6  lea rax, [rbp + rax - 1]
 7  div rbp
 8  imul rax, rbp
 9  mov rbp, qword ptr [rsp + 8]
10  sub rbp, rax
```

Listing 0: Basic block β_2 for perturbation function size estimation

Thus, we find that the perturbation function's output set can have very high cardinality, posing a challenge for generating desirable explanations.

E Types of data dependencies in basic blocks

While instructions are decomposed into microoperations and processed in parallel by the different components of the CPU, an instruction $inst_j$'s execution can get stalled due to the requirement for a previous instruction $inst_i$ to be completed, creating a *data dependency* [29]. A Read-After-Write (RAW) data dependency arises when $inst_j$ reads the value in an operand that is written by $inst_i$. $inst_j$ can not get executed until $inst_i$ ends to ensure correct execution. A Write-After-Read (WAR) dependency occurs when $inst_j$ writes to an operand that is read by $inst_i$. A Write-After-Write (WAW) dependency arises when $inst_j$ writes to an operand that is written to by $inst_i$. There can be multiple data dependencies, possibly of different kinds, between a given pair of instructions.

F Specific details of COMET

F.1 Perturbations to individual components of $\mathcal{G}$

Perturbing vertices of $\mathcal{G}$. The basic block perturbation algorithm 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 F.3.

F.2 Basic block perturbation algorithm

Algorithm 1 presents our stochastic perturbation algorithm Γ to conditionally perturb a given basic block β to β'. The perturbation algorithm creates the graph $\mathcal{G}'$ of β' while preserving a set of instructions/their corresponding vertices $\overline{\mathcal{V}}$, a set of data dependencies/their corresponding edges $\overline{\mathcal{E}}$ and possibly the number of instructions/the number of vertices, denoted by the boolean $preserve_\eta$ which is set to true when the number of instructions η is to be kept constant. If the number of vertices is to be kept constant, then the vertex/instruction deletion operation is forbidden [lines 1-1]. The vertices at the ends of the edges in $\overline{\mathcal{E}}$ are preserved as well [line 1] by adding them to $\overline{\mathcal{V}}$. Then each vertex of $\mathcal{G}$ is perturbed with a probability of $(1 - p_{I,ret})$ if it is not required to be retained [lines 1-1]. If the deletion perturbation operation is in vertexPerturbationOps, then a vertex is deleted or replaced with probabilities of p_{del} and $(1 - p_{del})$ respectively. Otherwise, it is replaced with a valid vertex. The replacement of a vertex/corresponding instruction involves changing its opcode to another opcode that can take the original operands and still constitute valid x86 syntax according to the x86 Instruction Set Architecture. Similarly, each data-dependency edge is perturbed with a probability of $(1 - p_{D,ret})$ if it is not required to be retained [lines 1-1], to form $\mathcal{G}'$ [line 1]. The only perturbation of any data dependency is its deletion, which is conducted by the perturbation of the operands involved in the data dependency.

F.3 Case specificity of perturbation probabilities

COMET's perturbation algorithm Γ consists of primarily 3 probability terms: $p_{I,ret}$, $p_{D,ret}$, and p_{del} as described in Appendix F.2. $p_{I,ret}$ and $p_{D,ret}$ are the probabilities of retention of a given instruction and a given data dependency respectively, in the perturbed basic block. p_{del} is the probability of deletion of an instruction when the deletion perturbation operation is allowed for instructions. The

Algorithm 1 Basic Block Perturbation Algorithm

```
Input: basic block graph $\mathcal{G}$, vertices to preserve $\overline{\mathcal{V}}$, data-dependency edges to preserve $\overline{\mathcal{E}}$, $preserve_\eta$, $p_{I,ret}$, $p_{D,ret}$, $p_{del}$
Output: perturbed basic block graph, $\mathcal{G}'$
vertexPerturbationOps = {replacement, deletion}
if $preserve_\eta$ then
    vertexPerturbationOps.remove({deletion})
end if
$\overline{\mathcal{V}} \leftarrow addVerticesForPreservedDeps(\overline{\mathcal{V}}, \overline{\mathcal{E}})$
for $v \in GetVertices(\beta)$ do
    if $v \notin \overline{\mathcal{V}}$ and $rand([0,1]) > p_{I,ret}$ then
        $v \leftarrow PerturbVertex(\mathcal{G}, v, vertexPerturbationOps, p_{del})$
    end if
end for
for $\varepsilon \in GetDepEdges(\beta)$ do
    if $\varepsilon \notin \overline{\mathcal{E}}$ and $rand([0,1]) > p_{D,ret}$ then
        $\varepsilon \leftarrow PerturbEdge(\mathcal{G}, \varepsilon)$
    end if
end for
$\mathcal{G}' \leftarrow \mathcal{G}$
```

deletion perturbation operation will not be allowed for instructions when the number of instructions is to be kept constant.

Γ perturbs a basic block β by essentially perturbing every instruction while preserving certain tokens of the instruction from getting perturbed. These preserved tokens correspond to the features that are required to be preserved by Γ and also the features that Γ voluntarily does not attempt to perturb. Γ has voluntary retention of randomly selected basic block features to output perturbed basic blocks β' that are very similar to the original basic block β. Γ attempts to perturb the other tokens of β to obtain β'.

Γ can delete an instruction in case none of its tokens are required to be preserved. Otherwise, Γ replaces a token with another token that can form a basic block with valid x86 syntax alongside the other tokens. Thus, every token has a set of potential replacements. Perturbations to opcode tokens are counted as changes to the instruction features, while perturbations to the operand tokens are considered as changes to any data dependency features. As the perturbation space consists of only valid basic blocks, the overall probabilities of the primitive perturbation operations (instruction deletion, instruction replacement, and data dependency deletion) vary with the target basic block.

Following is an example of this variation. Several tokens of x86 assembly have no possible replacements resulting in no probability of replacement, such as the opcode lea. This is a special opcode that loads the effective memory address of its source operand into the destination register. There is no other x86 opcode that shows similar behavior. Hence, the lea can not be replaced with any other opcode. Such failed attempts at opcode replacement lead to the retention of the instruction, thus leading to an increase in the probability of retention of specific features of the basic block. This change in probabilities is specific to the basic blocks having the lea opcode in its instructions.

Another example of basic-block-specific probability settings occurs due to data dependencies. The data dependencies in a basic block can be varied with changes in just the opcodes of the corresponding instructions. Thus, while we keep the perturbation probability of a data dependency $(1 - p_{D,ret})$ to be 0.5 in the general case, it can vary with the basic block. A basic block having all the potential replacements for the opcodes involved in a data dependency with similar behavior as the original opcodes will have 0.5 probability of perturbation of the data dependency, while the opcodes for which we have potential replacements show variable behaviors, the data dependency perturbation probability can be more than 0.5. (Opcodes add and sub have similar behavior as they read the value in the source operand and read-write the value in the destination operand. They have different behavior from mov that reads the source operand value and writes to the destination operand. All 3 opcodes could be potential replacements for each other in instructions having certain pairs of operands.)

F.4 Details on computing explanations

Similar to the Anchors' construction [31], COMET iteratively builds its explanation feature set starting with an empty feature set, such that the maximum (estimated) precision feature sets at each iteration are expanded to all possible and distinct feature sets made by adding a block feature not present in them. Maximum precision feature sets are identified using the KL-LUCB [18] algorithm. We consider the candidate feature sets in each iteration as arms in a pure exploration multi-arm bandit instance. Each pull of an arm is the same as preserving the corresponding set of features when creating input perturbations. If a perturbation preserving the features set has a similar cost prediction as $\mathcal{M}(\beta)$, then that counts towards the reward gained from using the arm. Note that the cumulative reward of an arm is the precision of the corresponding set of features. This is because precision can be equivalently written as in (10). Thus, precision becomes the expected reward of the arm corresponding to a given set of features $\mathcal{F}$.

$$Prec(\mathcal{F}) = \mathbb{E}_{\alpha \sim \mathcal{D}_{\mathcal{F}}}(\mathbb{I}_{\mathcal{M}(\alpha) \in \mathcal{T}}) \tag{10}$$

We use KL-LUCB to find the maximum reward arms and the corresponding maximum precision feature sets with a predefined confidence level. We select the same KL-LUCB hyperparameters as those in [31]. We continue expanding feature sets till the precision threshold is crossed or the feature set with all the block features is obtained (which has precision = $1 > 1 - \delta$). We can either output the obtained feature set as further expansion will lower the coverage ($Cov(\mathcal{F})$ is proportional to $|\Pi(\mathcal{F})|$ which is inversely proportional to $|\mathcal{F}|$ (Appendix C)) or if we are maintaining the top-k maximum precision feature sets at each iteration, we can expand other feature sets till we get precision more than the threshold for all, in which case we select the maximum coverage feature set from all the candidates. Among the considered feature sets that have precision $> (1 - \delta)$, the maximum coverage feature set is COMET's explanation for $\mathcal{M}(\beta)$.

G Experimental setup and baselines

G.1 Experimental setup

All our experiments were conducted on a 12th Gen 20-core Intel i9 processor. 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 H). We have used the default hyperparameters in the Anchor algorithm [31] for the beam-search-based iterative explanation construction method. We study the sensitivity of COMET to its hyperparameters in Appendix H.

G.2 Baseline explanation algorithms

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.

H Ablation and sensitivity studies

In this section, we study the variations in our results, with COMET's hyperparameters and design choices. We use our explanation accuracy-based evaluation scheme based on our crude but interpretable cost model that is presented in Section 4.1, to study the effects of the different hyperparameters and design choices. For this study, we have used the crude cost model for the Haswell microarchitecture. We have randomly selected 100 basic blocks from the BHive dataset [7] for which we generate COMET's explanations with different settings. We have dropped the error bars for clarity of the results, as we note from Table 1 that the standard deviations in our results are generally low.

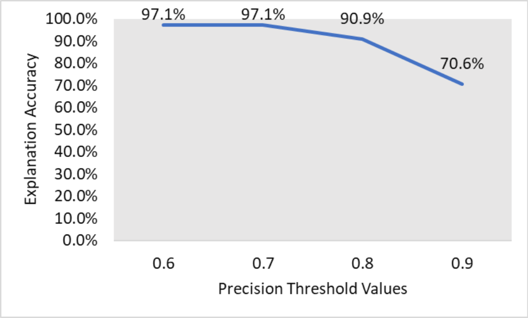


Figure 3: Variation in explanation accuracy with the precision threshold $(1 - \delta)$ setting in COMET

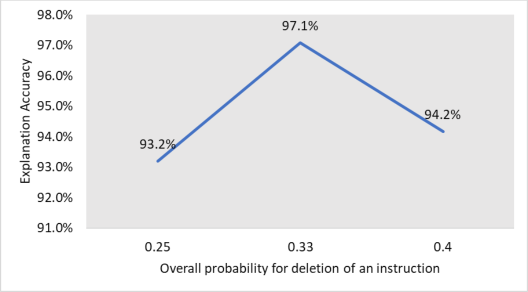


Figure 4: Variation in explanation accuracy with the probability of instruction deletion in Γ

H.1 Precision threshold

In this section, we study the variation in the explanations' accuracy with the precision threshold set in COMET, above which we consider the explanation feature set to be approximately faithful to the cost model's predictions. We want the precision threshold to be high such that the most precise and accurate explanations are given as output. Figure 3 presents the variation in the accuracy of COMET's explanations with various values for the precision threshold $(1 - \delta)$ in COMET. We observe that 0.7 is the highest precision threshold that gives the highest accuracy and hence we have set it as the precision threshold in our experiments.

H.2 Perturbation probabilities for instructions

Γ attempts to perturb a given instruction $inst$ in a basic block β only when it is not required to be preserved. Γ retains $inst$ with a probability of $p_{I,ret}$ and perturbs it otherwise. There are 2 potential operations for perturbing $inst$: Deletion and Replacement (with valid x86 instruction), each probabilities p_{del} and $(1 - p_{del})$ respectively. We have set $p_{del} = 0.33$ based on a sensitivity study that we conducted with respect to this hyperparameter, for all of our experiments. Figure 4 presents our findings. We find that our choice of $p_{del} = 0.33$ leads to the maximum accuracy among other candidates.

H.3 Perturbation probabilities for data dependencies

Similar to the case for instructions, Γ attempts to perturb a given data dependency δ in a basic block β with probability $(1 - p_{D,ret})$. As discussed in Section F.3, the exact probabilities of the retention/deletion of data dependencies are basic-block-specific. However, we vary these probabilities by varying the probability of explicit retention of a data dependency, i.e. the probability by which a data dependency will be retained for sure. This probability is a lower bound for $p_{D,ret}$ and higher values of this lower bound imply higher values for $p_{D,ret}$ for any given basic block. Figure 5 shows our findings. We have shown the variation in explanation precision as well, as we observe precision to have a trend different from explanation accuracy in this case. We find that a value of 0.1 for this probability parameter leads to optimum values for both explanation accuracy and precision. Thus, we have selected the explicit data dependency retention probability to be 0.1 in COMET.

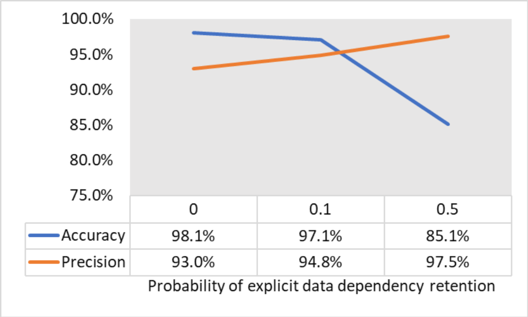


Figure 5: Variation in explanation accuracy and precision with the probability of explicit data dependency retention

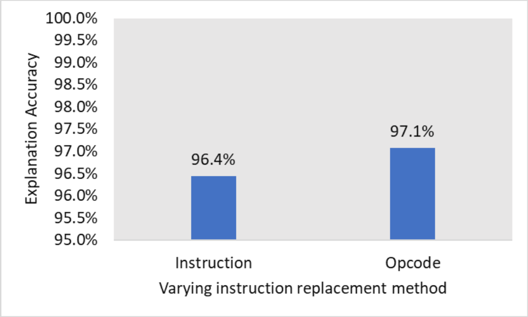


Figure 6: Variation in explanation accuracy with just opcode and whole instruction replacement schemes.

H.4 Replacement of instructions

Γ considers only the changes to an instruction's opcode as changes to the feature corresponding to the instruction. However, another possibility could be to consider operand changes (such that their types and sizes are preserved) as well as changes to the instruction feature. We analyze the effects of the two instruction changing/replacement schemes in Figure 6. We observe that the accuracy of the explanations is higher with just the opcode replacement method, justifying our choice of this instruction replacement scheme.

An important hyperparameter that we have set according to our intuitive understanding is the ϵ error, which marks the radius of the ball of acceptable cost predictions around the prediction of cost model $\mathcal{M}$ for basic block β ($\mathcal{M}(\beta)$). For our crude cost model $\mathcal{C}$, we have kept ϵ to be a quarter of one unit of its cost prediction, as the least change in its cost prediction can be a quarter unit ($\frac{\Delta n}{4} = 0.25$). For the practical cost models such as Ithemal and uiCA, we have set ϵ as 0.5 cycles of throughput prediction, as that is the least, significant change in practically-useful throughput values.

I Crude interpretable cost model details

We define $cost_{inst}(inst)$ as the throughput of the instruction $inst$ on actual hardware. We obtain the throughputs of instructions over actual hardware from `https://www.uops.info/table.html`. We define $cost_{dep}(\delta_{ij})$ as in (11). Our intuition behind keeping the costs of WAR and WAW type of dependencies to be 0 is that these dependencies are not true dependencies and can be generally resolved by the compiler by register renaming [29]. The RAW data dependency, on the other hand, is a true dependency. As the two instructions forming a RAW dependency will be executed sequentially on hardware, the addition of their individual costs would be a good proxy for the actual throughput cost brought in by the data dependency.

$$cost_{dep}(\delta_{ij}) = \begin{cases} 0, & \delta_{ij} = \text{WAR/WAW} \\ cost_{inst}(inst_i) + cost_{inst}(inst_j), & \delta_{ij} = \text{RAW} \end{cases} \tag{11}$$

We define the $cost_{\eta}(n) = \eta/4$ as the cost for having n number of instructions (denoted by η) in a given basic block β. We derive the expression for the cost of number of instructions from the simple baseline model presented in [2].

Our choice of $\mathcal{C}$ is microarchitecture-specific as the costs of individual instructions vary across microarchitectures. We have developed $\mathcal{C}$ models for the Haswell and Skylake microarchitectures, only for the purposes of evaluating COMET's explanations.

J Studied dataset and cost models

J.1 BHive dataset

BHive dataset[2] [7] is a benchmark suite of x86 basic blocks. It contains roughly 300,000 basic blocks annotated with their average throughput over multiple executions on actual hardware for 3 microarchitectures: Haswell, Skylake, and Ivy Bridge. We have generated explanations for basic blocks in this dataset.

The dataset can be partitioned by 2 criteria: by *source* and by *category* of its basic blocks. Partition by source annotates each block with the real-world code base from which it has been derived. Examples of BHive sources are Clang and OpenBLAS. Partition by category annotates each basic block by its type, characterized by the semantics of the instructions in the block. There are 6 types of blocks: Scalar, Vector, Scalar/Vector, Load, Store, and Load/Store.

J.2 Ithemal

Ithemal[3] [26] is an ML-based cost model, which predicts the throughput of input x86 basic blocks for a given microarchitecture. It is open-source and is currently trained for the Haswell, Skylake, and Ivy Bridge microarchitectures on the BHive dataset. A separate instance of Ithemal needs to be trained for every microarchitecture, due to the difference in the actual throughput values obtained over different hardware. Ithemal's throughput prediction is a floating point number, as it is trained on the BHive dataset.

Ithemal consists of a hierarchical multiscale RNN structure. The first RNN layer takes embeddings of tokens of the input basic block and combines them to create embeddings for the instructions in the basic block. The second RNN layer takes the instruction embeddings as input and combines them to create an embedding for the basic block. The basic block embedding is passed through a linear regressor layer to compute the throughput prediction for the basic block.

Ithemal exhibits roughly 9% Mean Absolute Percentage Error for the Haswell microarchitecture on the BHive dataset. As Ithemal outputs only its throughput prediction and no insights into why the prediction was made, it can not be reliably deployed in mainstream compiler optimizations.

J.3 uiCA

uiCA[4] [2] is an analytical simulation-based cost model for several latest microarchitectures released by Intel over the last decade. uiCA's simulation model is hand-engineered to accurately match the model of each Intel microarchitecture and must be manually tuned to reflect new microarchitectures. It can output detailed insights into its process of computing its throughput prediction of input x86 basic blocks, such as where in the CPU's pipeline its simulator identified a bottleneck for the execution of the basic block.

[2] `https://github.com/ithemal/bhive`

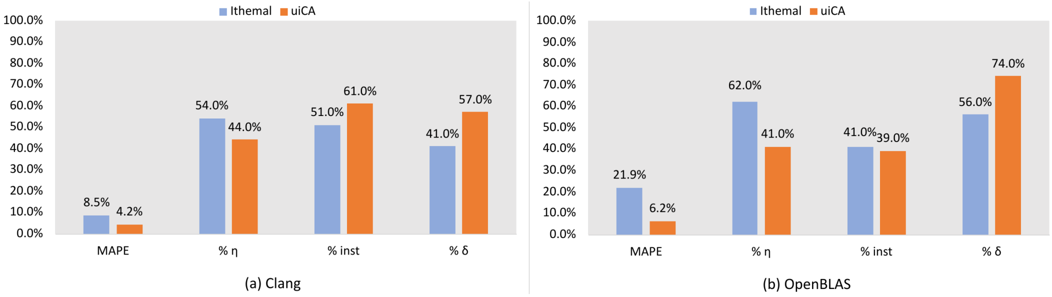
[3] `https://github.com/ithemal/Ithemal`

[4] `https://github.com/andreas-abel/uiCA`

K Additional experiments

K.1 BHive experiments on partitions

The experiments in this section continue to demonstrate the hypothesis in Section 4.3 on partitions of the BHive dataset, which were explained in Appendix J.1. We omit the error bars for clarity, as the standard deviations in our results are generally low [Figure 2].



(a) Clang

(b) OpenBLAS

Figure 7: 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

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 7 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 8 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 all types of features in COMET's explanations for both cost models. This observation further supports our hypothesis.

K.2 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 the Haswell microarchitecture on randomly picked blocks from the BHive dataset.

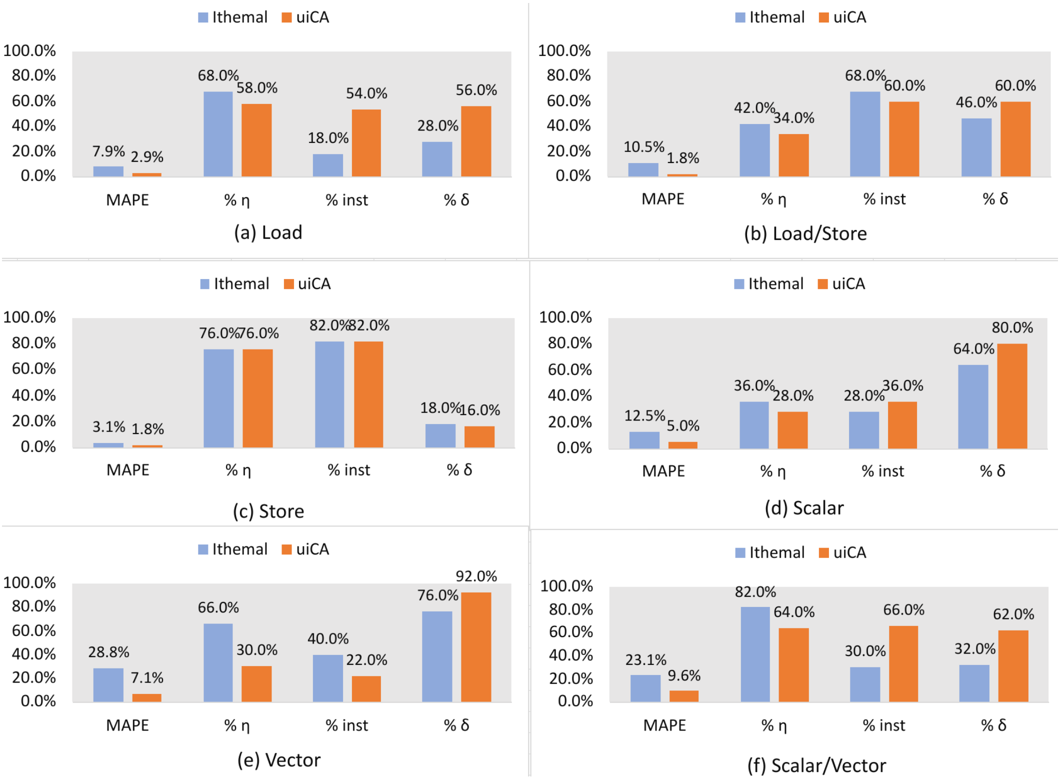
Case study 1. The block in Listing 1 is predicted to have a throughput of 2 cycles by both cost models which matches the throughput on actual hardware reported in the BHive dataset. Instructions 2 and 3 write to the memory and are thus the highest throughput instructions [1, 14]. Hence intuitively, for correct prediction, these instructions are important. COMET's explanations for both cost models match this intuition, thus suggesting 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 1: Case Study 1

Case study 2. The block in Listing 2 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



(a) Load

(b) Load/Store

(c) Store

(d) Scalar

(e) Vector

(f) Scalar/Vector

Figure 8: 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

instruction in general [1, 14]. 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 2: Case Study 2