Robust Resource Bounds with Static Analysis and Bayesian Inference

LONG PHAM, Carnegie Mellon University, USA
FERAS A. SAAD, Carnegie Mellon University, USA
JAN HOFFMANN, Carnegie Mellon University, USA

There are two approaches to automatically deriving symbolic worst-case resource bounds for programs: static analysis of the source code and data-driven analysis of cost measurements obtained by running the program. Static resource analysis is usually sound but incomplete. Data-driven analysis can always return a result, but its lack of robustness often leads to unsound results. This paper presents the design, implementation, and empirical evaluation of hybrid resource bound analyses that tightly integrate static analysis and data-driven analysis. The static analysis part builds on automatic amortized resource analysis (AARA), a state-of-the-art type-based resource analysis method that performs cost bound inference using linear optimization. The data-driven part is rooted in novel Bayesian modeling and inference techniques that improve upon previous data-driven analysis methods by reporting an entire probability distribution over likely resource cost bounds. A key innovation is a new type inference system called *Hybrid AARA* that coherently integrates Bayesian inference into conventional AARA, combining the strengths of both approaches. Hybrid AARA is proven to be statistically sound under standard assumptions on the runtime cost data. An experimental evaluation on a challenging set of benchmarks shows that Hybrid AARA (i) effectively mitigates the incompleteness of purely static resource analysis; and (ii) is more accurate and robust than purely data-driven resource analysis.

CCS Concepts: • **Theory of computation → Type theory**; **Program analysis**; • **Mathematics of computing →** *Probabilistic inference problems*; *Markov-chain Monte Carlo methods.*

Additional Key Words and Phrases: resource analysis, worst-case costs, static analysis, data-driven analysis, hybrid analysis, Bayesian inference

ACM Reference Format:
Long Pham, Feras A. Saad, and Jan Hoffmann. 2024. Robust Resource Bounds with Static Analysis and Bayesian Inference. *Proc. ACM Program. Lang.* 8, PLDI, Article 150 (June 2024), 26 pages. https://doi.org/10.1145/3656380

1 INTRODUCTION

Static resource analysis. The prevailing goal of resource analysis—and the goal of this article—is to derive a symbolic expression that bounds the worst-case execution cost of a program as a function of the inputs. Building on pioneering resource analysis works such as Wegbreit's metric system [88], modern tools use various static analysis techniques: type systems [5, 21, 24, 26, 38, 40, 59, 86], recurrence relations [2, 25, 35, 55, 56, 88], term rewriting [6, 7, 49, 72], ranking functions [12, 20, 31, 82], and abstract interpretation [3, 37, 91]. They can automatically and accurately analyze complex functions such as operations on splay trees [61] and software product code [37]. Automatic resource analysis is now used on a daily basis by developers in tools such as Meta's static analyzer Infer [22].

Authors' addresses: Long Pham, Carnegie Mellon University, Pittsburgh, USA, longp@andrew.cmu.edu; Feras A. Saad, Carnegie Mellon University, Pittsburgh, USA, fsaad@cmu.edu; Jan Hoffmann, Carnegie Mellon University, Pittsburgh, USA, jhoffmann@cmu.edu.

ACM 2475-1421/2024/6-ART150
https://doi.org/10.1145/3656380

However, since static resource analysis is undecidable for Turing-complete languages, even the most sophisticated tools fail to infer cost bounds for some programs. In such cases, users must resort to rewriting their code or using manual techniques for resource analysis [19, 59, 70, 74, 75]. Both of these workarounds require expertise in programming languages and resource analysis, which is a fundamental barrier to leveraging these methods in a wider spectrum of applications.

Data-driven resource analysis. An alternative approach to static resource analysis is *data-driven resource analysis*, which infers resource bounds from a set of cost measurements obtained by running the program [23, 30, 33, 50, 77, 80, 90]. Data-driven resource analysis is not as well established as static resource analysis. A main advantage of data-driven approaches is that a program can be treated as a black box, greatly expanding the scope of programs that can be analyzed.

Data-driven resource analysis comes with its own set of challenges, particularly for deriving worst-case bounds. First, the analysis is sensitive to the given dataset and data collection can be difficult. For example, inputs generated uniformly at random are unlikely to trigger worst-case behaviors in many programs. Second, commonly used statistical techniques for inferring a bound from the cost dataset lack *accuracy* (i.e., how close the inference result is to a sound worst-case cost bound) and *robustness* (i.e., the inference result has a positive probability of being a sound worst-case cost bound even if the dataset does not contain worst-case inputs). The aforementioned works use optimization, and as we demonstrate with experiments (Section 7), bounds derived in this way are prone to being unsound. Prior works also do not quantify any notion of uncertainty in the inferred bounds.

Bayesian resource analysis. The first contribution of this work is the design and implementation of Bayesian resource analysis for worst-case bounds. In general, Bayesian resource analyses are more customizable than optimization-based ones, as users can express domain knowledge in the form of probabilistic models. Additionally, Bayesian resource analyses return whole posterior distributions of inferred cost bounds, providing richer information about the uncertainty in inference results. This article presents two new Bayesian resource analyses: BayesWC and BayesPC. In BayesWC, for each input size present in the runtime cost data, we conduct Bayesian inference to infer likely values of worst-case costs that are no less than the observed costs. By treating these two costs separately, we account for the possibility that worst-case costs have not been observed in the runtime data. The inferred worst-case costs from BayesWC produce optimization problems that can be solved to obtain cost bounds. In BayesPC, on the other hand, we conduct Bayesian inference to directly infer cost bounds, bypassing optimization altogether. Compared to BayesWC, BayesPC allows users to construct more holistic probabilistic models for specifying how observed cost measurements are probabilistically generated. In our experiments, cost bounds inferred by BayesWC and BayesPC are shown to be more sound and robust than those inferred by an optimization-based data-driven baseline. However, because they ignore the source code and exclusively rely on data-driven analysis, BayesWC and BayesPC can still fail to infer sound cost bounds in several important benchmarks.

Hybrid resource analysis. **The main contribution of this work is the design, implementation, and evaluation of a new hybrid resource analysis method that combines static and data-driven resource analyses.** To the best of our knowledge, this article is the first to develop such a hybrid analysis. Hybrid resource analysis aims to mitigate the incompleteness of static resource analysis and improve the accuracy and robustness of data-driven resource analysis. A main research challenge is designing a principled interface between static and data-driven resource analyses that enables a tight integration to combine the respective strengths of both approaches.

For the static part of the hybrid resource analysis, we rely on automatic amortized resource analysis (AARA) [45, 47], a type-based state-of-the-art technique that is implemented for OCaml

programs in Resource-Aware ML (RaML) [41]. AARA supports advanced language features such as recursive types [36], side effects [62], and higher-order functions [53]. Two distinguishing features of AARA are the ability to handle non-monotone resources such as memory and the ability to account for amortization effects. As a type-based technique, AARA is naturally compositional, and cost bound inference can be reduced to off-the-shelf linear programming (LP) solving, even if the derived bounds are nonlinear. We dub our hybrid resource analysis method *Hybrid AARA*.

For the data-driven part of Hybrid AARA, we present a new type inference system that combines the two data-driven Bayesian resource analysis methods (BayesPC and BayesWC) with conventional AARA. To invoke Hybrid AARA, a user annotates part of the code for data-driven analysis using a syntactic form that has no runtime effect. Our integration of Bayesian resource analysis and AARA rests on two key technical innovations. For Hybrid AARA with BayesWC, we combine the optimization problems produced by the data-driven Bayesian inference with the linear constraints derived using conventional AARA type inference to obtain and solve a joint linear program. For Hybrid AARA with BayesPC, we integrate constraints from conventional AARA into the probabilistic cost bound models of BayesPC. Bayesian inference within this model leverages recent innovations from the sampling algorithm literature that allow Hamiltonian Monte Carlo (HMC) sampling to be restricted to a convex polytope defined by AARA's linear constraints [18, 69]. To establish the soundness of Hybrid AARA, we first prove that its inferred bounds are sound with respect to runtime cost data. Additionally, we prove the statistical soundness that the inferred bounds converge to a sound bound if the analysis is repeated with a successively growing set of runtime cost data that contains worst-case inputs with nonzero probability.

Applications. Hybrid AARA (and its special case of purely data-driven analysis) using Bayesian inference returns a collection of cost bounds that approximate the posterior distribution. Even if the proportion of sound cost bounds in the posterior distribution is less than 100%, Hybrid AARA using Bayesian inference is useful for applications that can tolerate occasional underestimates of the worst-case cost. One such application is scheduling of jobs in cloud computing, where the cloud-service provider would like to have a reasonably accurate (but not necessarily sound at all times) estimate of the resources required to run the job. The job's cost use may respect the tighter bounds in the posterior sample, but if it happens to run out of computational resources, then the cloud-service provider can rerun the job with more resources. Other applications of Hybrid AARA with Bayesian inference include auto-grading of students' programming assignments and annotating software libraries to help users of the library understand its performance characteristics.

Evaluation. In an experimental evaluation with a prototype built on RaML [41], we compare fully data-driven analyses and Hybrid AARA on a curated set of benchmarks that pose challenges to static, data-driven, and hybrid analyses. We find that Hybrid AARA outperforms fully data-driven analyses both when considering soundness and tightness of the bounds.

Contributions. In summary, this article makes the following contributions.

- We present novel Bayesian data-driven resource analyses (BayesWC in Section 5.2; BayesPC in Section 5.3) to infer posterior probability distributions over program cost bounds.
- We present Hybrid AARA: a novel type inference system that combines BayesWC and BayesPC with conventional AARA (Sections 6.1 and 6.2).
- We formulate and prove two notions of soundness for Hybrid AARA (Theorems 6.1 and 6.2)
- We implement a prototype of Hybrid AARA by extending Resource-Aware ML (RaML) [41].
- We evaluate Hybrid AARA and fully data-driven resource analyses on a challenging benchmark set, showing examples of improvements in accuracy and robustness (Section 7).

```
let rec partition pivot xs =
  match xs with
  | [] → ([], [])
  | hd :: tl →
    let lower, upper = partition pivot tl in
    if complex_compare hd pivot then
      (hd :: lower, upper)
    else (lower, hd :: upper)
```

```
let rec quicksort xs =
  match xs with
  | [] → []
  | hd :: tl →
    let lower, upper = Raml.stat (partition hd tl) in
    let lower_sorted = quicksort lower in
    let upper_sorted = quicksort upper in
    append lower_sorted (hd :: upper_sorted)
```

Lst. 1. Quicksort in OCaml. The function `complex_compare` in line 6 is intractable to static resource analysis. The annotation `Raml.stat` in line 5 indicates data-driven resource analysis on `partition`.

2 OVERVIEW

In this section, we review static and data-driven resource analyses, identify their shortcomings, and outline our main contribution of hybrid resource analysis. We use the implementation of *quicksort* in OCaml given in Listing 1 as a running example. Our goal is to automatically derive a symbolic worst-case cost bound for the function `quicksort`. For simplicity, we use the resource metric that records the time cost of executing the comparisons `complex_compare hd pivot` in the function partition. We consider different versions of `complex_compare`. For example, if the cost of evaluating `complex_compare hd pivot` is bounded by 1, then the worst-case cost of `quicksort xs` is $n(n-1)/2$, where n is the length of the list `xs`.

Static resource analysis. The predominant method for automatically deriving symbolic worst-case bounds is *static resource analysis*. This article builds on automatic amortized resource analysis (AARA) [39–42] (Section 4), a compositional type-based static analysis technique.

Resource-Aware ML (RaML) is an implementation of AARA that derives polynomial bounds for a subset of OCaml. The compositionality of AARA ensures that RaML can derive a bound for `quicksort` if it can derive a bound for `complex_compare`. Assuming each comparison has cost 1, RaML correctly infers the tight bound $n(n-1)/2$ for `quicksort` in less than 0.1 seconds. Similarly, assume that the argument of `quicksort` is a list of lists and that `complex_compare` is a lexicographical comparison whose worst-case is k, where k is the length of the first argument list. Then RaML infers the tight bound $mn(n-1)/2$ for `quicksort` in less than 0.2 seconds, where m is the maximum length of the inner lists. This analysis is nontrivial because of the nonstructural recursion: to derive a cost bound, it is not enough to separately analyze the cost of the partition function and the number of recursive steps. We must also analyze how the partition function changes input sizes, relaying the size-change information to the next recursive call of `quicksort`.

Because resource analysis is undecidable, however, even the most sophisticated static resource analyses are incomplete, and there remain programs that cannot be analyzed automatically. If the comparison function `complex_compare` in `partition` is, for example, OCaml's built-in polymorphic comparator, which examines the low-level memory representations of input values, RaML is not able to derive a bound because the comparison code is not available to the analysis. Indeed, there are many other comparison functions that RaML cannot analyze, even if the cost of the comparison is bounded by a constant. AARA fails, for instance, if the control flow depends on mutable data or complex loop conditions. A concrete example of a constant-time function that cannot be statically analyzed is `compare_dist`, an implementation of `complex_compare` that compares two vectors by computing their distance to a fixed vector that is stored in a reference cell. Such limitations are not specific to RaML: every static resource analysis has unsupported language features or iteration patterns that make the analysis feel brittle for non-experts users.

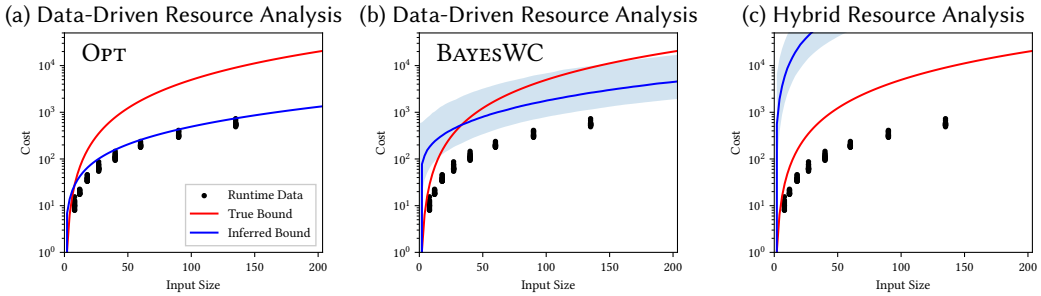


Fig. 1. Hybrid resource analysis on quicksort infers more accurate bounds than purely data-driven analyses.

Data-driven resource analysis. One way to overcome the incompleteness of static analysis is data-driven analysis. Existing data-driven techniques first collect execution costs for different inputs and then solve an optimization problem to fit a cost function on the data [23, 30, 33, 50, 77, 80, 90]. Although these optimization-based approaches are fast, they can infer unsound cost bounds and do not quantify the uncertainty over the unknown cost bounds.

To showcase such an approach, suppose that `quicksort` uses the aforementioned comparator `compare_dist` and that the cost varies between 0.5 and 1.0 for different inputs. Figure 1a shows the inferred quadratic cost bound (blue line) of data-driven optimization-based resource analysis adapted from [23, 33, 90], given a randomly generated dataset of measured costs of `quicksort` (black dots). Any worst-case cost bound must lie above all the measured costs and minimize the total L_1 distance between the curve and runtime data. This optimization problem can be framed as a linear program (Opt; Section 5.1). Optimization fails to infer the correct worst-case cost bound (red line), because randomly generated data sets rarely contain worst-case inputs.

To mitigate the shortcomings of such greedy optimizations, this paper introduces data-driven Bayesian resource analyses (BayesWC and BayesPC; Sections 5.2 and 5.3). Bayesian resource analysis enables users to express their domain knowledge in the form of models that specify how observed runtime samples are probabilistically generated. It returns an entire probability distribution over the corresponding cost bounds given observed samples. In particular, we condition the probabilistic model on observed cost data and compute the posterior distribution of cost bounds by running sampling-based probabilistic inference algorithms. For `quicksort`, Fig. 1b shows the posterior distribution over cost bounds from Bayesian resource analysis. The blue line indicates the median cost bound and the light-blue shade is the 10–90$^{\text{th}}$ percentile range. Most of the cost bounds in the posterior distribution are closer to the true worst-case bound (red line) as compared to optimization-based approach in Fig. 1a. In fact, 28/1000 bounds drawn from the posterior distribution are sound. Although this fraction is small, it is already a substantial improvement over optimization.

Hybrid resource analysis. A central contribution of this work is the design, implementation, and evaluation of Hybrid AARA (Section 6), which integrates our two novel data-driven methods for Bayesian resource analysis (BayesWC and BayesPC) and a simple optimization-based data-driven baseline (Opt) into conventional AARA. Hybrid AARA is modular and lets users combine data-driven and static analyses in different parts of a program. Therefore, it covers a spectrum ranging from fully data-driven to fully static analyses.

Consider again the example of `quicksort` where the comparator is `compare_dist`, which intractable for static analysis and has a cost varying between 0.5 and 1.0. If we perform data-driven analysis on the comparator and static analysis on the rest of the code, Hybrid BayesWC amounts

to (i) inferring the worst-case constant cost of the comparator by data-driven analysis; and (ii) incorporating the inferred cost into conventional AARA. All 1000/1000 cost bounds drawn from the posterior distribution are sound (the analysis time in our Hybrid AARA prototype is 66.2 seconds).

In the previous example, the interface between the statistical analysis and the static analysis is particularly simple since the cost of evaluating `quicksort` does not depend on the result of the comparison function. Many examples fall into this category, for which Hybrid AARA enables the analysis of code that is intractable for purely static methods while clearly outperforming purely data-driven methods. However, our experiment focuses on a different category of benchmarks for which the integration of data-driven methods is technically challenging and the benefits of a hybrid analysis are less clear.

Challenges for hybrid resource analysis. Revisit the `quicksort` example, using the `compare_dist` comparator. But this time let us assume that the user marks the call to `partition` for data-driven analysis instead of the call to `complex_compare`. A data-driven analysis could be able to derive a linear cost bound for `partition`, but how can use this information to derive a bound for `quicksort` using AARA? We would need additional information on the size of the result of partition. However, it is not clear what exact information we need. For example, it is insufficient for analyzing `quicksort` to statistically bound the size of each component in the result of `partition`.

A main technical innovation of our work is the design of a principled interface between data-driven and static analyses that can handle such challenging cases. BayesWC and Opt are integrated into AARA by developing a type inference system that combines the underlying LP problems. Integrating BayesPC into AARA poses a significant challenge because the former relies on sampling algorithms to approximate posterior distributions of resource coefficients instead of LP problems as in AARA. To overcome this challenge, we leverage an innovative sampling algorithm that allow Hamiltonian Monte Carlo (HMC) sampling to be restricted to a convex polytope [18, 69].

Figure 1c shows the posterior distribution over bounds from Hybrid BayesWC on `quicksort` with the data-driven analysis of `partition`. The $10–90^{\text{th}}$ percentile range (blue shade) is situated above the true worst-case bound (red line) for all input sizes between 0 and 200. In fact, 471/1000 samples drawn from the posterior distribution in Hybrid BayesWC are theoretically sound bounds for all input sizes, in contrast to the 28/1000 bounds for purely data-driven analysis with BayesWC (Fig. 1b) and 0/1000 bounds for data-driven resource analysis with Opt (Fig. 1a).

We have empirically evaluated hybrid resource analysis on challenging and realistic micro benchmarks where data-driven analysis is applied to non-trivial code that stresses the interface between data-driven and static analysis (Section 7). The benchmarks demonstrate that (i) Bayesian resource analysis returns more accurate cost bounds than optimization-based analysis and (ii) hybrid resource analysis returns more accurate cost bounds than fully data-driven analysis. Notable among our benchmarks is the median-of-medians-based linear-time selection algorithm [11]. Conventional AARA cannot statically analyze this program, as it is challenging to reason about how the median of medians influences the partition function. Fully data-driven analysis does not infer a sound cost bound, either, as the worst-case behavior of partition rarely occurs in all recursive calls. By exploiting hybrid resource analysis, we successfully infer a sound worst-case cost bound that neither fully data-driven nor fully static analyses can.

3 SYNTAX AND COST SEMANTICS

We set the stage for Hybrid AARA by introducing a functional programming language. While AARA can handle higher-order functions, we focus on the first-order setting to simplify the presentation.

$\tau ::= \texttt{unit} \mid \texttt{int} \mid \tau_1 + \tau_2 \mid \tau_1 \times \tau_2 \mid L(\tau)$	polynomial and list types
$e ::= \langle\,\rangle \mid n \mid x$	unit, integer, and variable; $x \in \mathcal{X}, n \in \mathbb{Z}$
$\mid \text{left} \cdot x \mid \text{right} \cdot x \mid \text{case}\ x\ \{\text{left} \cdot x_1 \hookrightarrow e_1 \mid \text{right} \cdot x_2 \hookrightarrow e_2\}$	sum constructor and destructor
$\mid \langle x_1, x_2\rangle \mid \text{case}\ x\ \{\langle x_1, x_2\rangle \hookrightarrow e\}$	product constructor and destructor
$\mid [\,] \mid x_1 :: x_2 \mid \text{case}\ x\ \{[\,] \hookrightarrow e_1 \mid (x_1 :: x_2) \hookrightarrow e_2\}$	list constructor and destructor
$\mid f\ x$	function application; $f \in \mathcal{F}$
$\mid \text{let}\ x = e_1\ \text{in}\ e_2 \mid \text{share}\ x\ \text{as}\ x_1, x_2\ \text{in}\ e$	let-binding and variable sharing
$\mid \text{tick}\ q$	resource consumption; $q \in \mathbb{Q}$
$\mid \text{stat}_\ell\ e$	statistical analysis; $\ell \in \mathcal{L}$

Lst. 2. Datatypes τ and expressions e in a first-order AARA functional programming language.

3.1 Syntax

Datatypes τ and expressions e are formed by the grammar shown in Listing 2, where $\mathcal{X}$, $\mathcal{F}$, and $\mathcal{L}$ are countable sets of identifiers for variable names, function names, and label names, respectively. A program $\mathcal{P}$ contains a finite set $F \subset \mathcal{F}$ of function identifiers and mutually-recursive definitions

$$f(x) = e \qquad \text{for every } f \in F. \tag{3.1}$$

Function types have the form $\tau_1 \to \tau_2$. The function body e can reference all functions $g \in F$.

As in AARA [39, 40, 42], we adopt the share-let normal[1] form where (i) variables are affine (i.e., they are used at most once) and (ii) constructors and destructors are applied to variables, but not general expressions. The first restriction ensures that the potential stored in variables is consumed at most once. To use variable x twice, we write $\text{share}\ x\ \text{as}\ x_1, x_2\ \text{in}\ e$, where fresh variables x_1 and x_2 have the same value as x. The second restriction simplifies proofs of AARA. To apply constructors and destructors to non-variable expressions, we first bind variables to them by let-bindings.

The construct $\text{tick}\ q$ increments a cost counter by $q \in \mathbb{Q}$, which is possibly negative, and returns the unit element $\langle\,\rangle$. To specify a particular resource metric, users (manually or automatically) insert $\text{tick}\ q$ throughout their code. As the tick metric lets us consider arbitrary resource metrics, it is a standard practice in the literature of static resource analysis [25, 42, 67, 75, 86].

If all q's in $\text{tick}\ q$ are non-negative, such resource metrics (e.g., running time) are said to be *monotone*. Conversely, if we have $q < 0$, it means resources can be freed up, and such resource metrics (e.g., memory) are *non-monotone*. This article focuses on monotone resource metrics.

The construct $\text{stat}_\ell\ e$ does not have a runtime effect, but is used in the hybrid resource analysis. It indicates that expression e should be analyzed using data-driven analysis instead of static analysis. The label $\ell \in \mathcal{L}$ is used to uniquely identify sites of data-driven analysis in source code.

Our prototype implementation is based on Resource-Aware ML (RaML) [41], which uses OCaml extended with annotations `Raml.tick` and `Raml.stat` as the input language. By way of example, Listing 1 displays an implementation of `quicksort`.

The annotation stat can be either manually inserted by the user or automatically inserted by walking over the program's source code bottom-up to identify functions (or more fine-grained code fragments) that cannot be analyzed statically by conventional AARA. Concretely, we first look at the leaves of the program call graph (i.e., functions which do not call other functions), check if we can analyze them using conventional AARA, and then recurse up the call graph to identify other problematic functions. We then insert the annotations at all the required points.

[1]The share-let normal form does not affect the expressive power of the language. The implementation of AARA for OCaml [41, 42] only uses the share-let normal form internally—users are allowed to write any OCaml programs.

3.2 Cost Semantics

Given a program $\mathcal{P}$, the cost semantics of expression e is given by the judgment

$$V \vdash_{\mathcal{P}} e \Downarrow^{c} v, \tag{3.2}$$

where V is an environment (i.e., a mapping from variables to values), v is the value that e evaluates to, and $c \in \mathbb{Q}_{\geq 0}$ denotes the cost of evaluating e. Values v are defined by the following grammar:

$$v ::= \langle\,\rangle \mid n \mid \mathsf{left} \cdot v \mid \mathsf{right} \cdot v \mid \langle v_1, v_2 \rangle \mid [\,] \mid v_1 :: v_2.$$

A well-formed value can be assigned a type τ and we write $v : \tau$.

Under non-monotone resource metrics, we would have two notions of costs: peak costs and net costs [39, 40, 42]. However, as this article focuses on monotone resource metrics, (3.2) only contains a single cost $c \in \mathbb{Q}_{\geq 0}$. Additionally, for simplicity, this article assumes that (i) q in tick q is known with absolute certainty and (ii) costs are measured perfectly without measurement errors. Measurement errors can be factored into probabilistic models of Bayesian resource analysis by encoding them in the input program.

3.3 Data Collection

In fully data-driven analysis (Section 5), we collect a dataset of runtime cost measurements for a program $\mathcal{P} = \{f(x) = e\}$ with a single function f. Sweeping through a list of environments $V_i = \{x : v_i\}$ $(i = 1, \ldots, N)$ that each includes bindings for argument x, we evaluate the cost semantics $V_i \vdash_{\mathcal{P}} e \Downarrow^{c_i} \tilde{v}_i$. We then define the dataset as $\mathcal{D} = \{(V_i, \tilde{v}_i, c_i)\}_{i=1}^{N}$.

The data collection for hybrid resource analysis (Section 6) is more involved. It is formalized with a judgment that extends the cost semantics (3.2). Letting $N > 0$ be a positive integer, the expression

$$(V_i \vdash_{\mathcal{P}} e \Downarrow^{c_i} v_i)_{i=1}^{N} \mid \mathcal{D} \tag{3.3}$$

is defined as follows. We again perform independent executions of the program sweeping through environments $(V_1, \ldots, V_N)$. Let $L' \subset \mathcal{L}$ denote the labels of all stat_ℓ subexpressions in program e. We collect in $\mathcal{D}_\ell$ all measurements (V, v, c) associated with expression $\mathsf{stat}_\ell\ e_\ell$, for each $\ell \in L'$. The overall dataset is $\mathcal{D} = \{(\ell, V, v, c) \mid \ell \in L', (V, v, c) \in \mathcal{D}_\ell\}$.

4 BACKGROUND ON STATIC RESOURCE ANALYSIS

This work builds on automatic amortized resource analysis (AARA) [21, 39, 40, 42, 47, 48, 60], a type-based resource analysis technique for functional programs. It aims to automatically infer polynomial cost bounds. It adopts the potential method of amortized analysis [85], where data structures are equipped with *potential functions* mapping concrete values to non-negative numbers.

4.1 Linear AARA

To illustrate how AARA works, consider the function `partition` in Listing 1. Our goal is to derive a worst-case bound on the number of comparisons during an evaluation of `partition`, namely n, where n is the input list length. In AARA, we type expression `partition (p,x)`, where `p` is a pivot and `x` is an input list, as follows:

$$\{\mathtt{p} : \mathtt{int}, \mathtt{x} : L^1(\mathtt{int})\}; 0 \vdash \mathtt{partition\ (p,x)} : \langle L^0(\mathtt{int}) \times L^0(\mathtt{int}), 0\rangle.$$

The type $L^1(\mathtt{int})$ assigns the potential function $\Phi(v : L^1(\mathtt{int})) = 1 \cdot |v|$ to an input list v, representing the tight worst-case bound. The annotation 0 in the typing context indicates that 0 additional constant potential is stored in the context.

AARA is naturally compositional because potential functions explicitly track size changes of data structures. Assume we have two nested calls to `partition` as in the following function `f`.

```
f(x) = let (x1,x2) = partition (8128,x) in partition (1,x1)
```

In the call `partition (1,x1)`, we can use the previous type for partition. However, for the call `partition (8128,x)`, we use the typing judgment

$$\{\mathtt{p} : \mathtt{int}, \mathtt{x} : L^2(\mathtt{int})\}; 0 \vdash \mathtt{partition\ (p,x)} : \langle L^1(\mathtt{int}) \times L^1(\mathtt{int}), 0\rangle.$$

It assigns type $L^1(\mathtt{int})$ to the output lists. Let v, v_1, v_2 be values of variables x, x_1, x_2, respectively. The intuition is that the input potential $\Phi(v : L^2(\mathtt{int})) = 2 \cdot |v|$ is used to cover both the cost $(1 \cdot |v|)$ and the potential of the result $(1 \cdot |v_1| + 1 \cdot |v_2|)$. It relies on the fact $|v_1| + |v_2| = |v|$. The remaining potential $\Phi(v_1 : L^1(\mathtt{int})) = 1 \cdot |v_1|$ covers the cost of the second function call `partition (1,x1)`. In general, the type of `partition (p,x)` can be expressed with linear constraints:

$$\{\mathtt{p} : \mathtt{int}, \mathtt{x} : L^{p_1}(\mathtt{int})\}; p_0 \vdash \mathtt{partition\ (p,x)} : \langle L^{q_1}(\mathtt{int}) \times L^{q_2}(\mathtt{int}), q_0\rangle$$
$$\text{subject to } p_1 \geq 1 + q',\ q' \geq q_1,\ q' \geq q_2,\ p_0 \geq q_0.$$

Similar constraints are emitted by the local type rules during the type inference. The constraints are then solved with an off-the-shelf linear programming solver to derive a bound for the program.

4.2 Polynomial AARA

AARA can be extended to polynomial potential functions and therefore polynomial cost bounds while retaining compositionality and type inference with linear constraint solving [39, 43]. Consider `quicksort` from Listing 1. In terms of the number of comparisons, the worst-case cost of `quicksort` is $n(n-1)/2$, where n is the input list length. This cost bound is expressed by the following typing judgment in univariate polynomial AARA:

$$\{\mathtt{x} : L^{(0,1)}(\mathtt{int})\}; 0 \vdash \mathtt{quicksort\ x} : \langle L^{(0,0)}(\mathtt{int}), 0\rangle. \tag{4.1}$$

Here, the input type $L^{(0,1)}(\mathtt{int})$ assigns the potential function $\Phi(v : L^{(0,1)}(\mathtt{int})) = 0 \cdot \binom{|v|}{1} + 1 \cdot \binom{|v|}{2}$, and the constant 0 after $L^{(0,1)}(\mathtt{int})$ indicates the constant zero potential stored in the input.

Univariate polynomial AARA extends linear AARA with univariate polynomial potential functions. For instance, given two values v_1 and v_2, univariate AARA can express a potential function $|v_1|^2 + |v_2|^2$. Resource-annotated datatypes in univariate AARA are formed by the grammar

$$a ::= \mathtt{unit} \mid \mathtt{int} \mid \langle a_1, q_1\rangle + \langle a_2, q_2\rangle \mid a_1 \times a_2 \mid L^{\vec{q}}(a).$$

Here, $q_1, q_2 \in \mathbb{Q}_{\geq 0}$ denote constant potential, and a tuple $\vec{q}$ over $\mathbb{Q}_{\geq 0}$ records coefficients of polynomial potential functions, except their constant (i.e., degree-zero) potential. To indicate constant potential $q \in \mathbb{Q}_{\geq 0}$ in addition to higher-degree coefficients, we write $\langle a, q\rangle$.

The amount Φ of potential stored in value v according to a resource-annotated datatype a is

$$\Phi(v : \mathtt{unit}) = \Phi(v : \mathtt{int}) \coloneqq 0 \qquad \Phi(\langle v_1, v_2\rangle : a_1 \times a_2) \coloneqq \Phi(v_1 : a_1) + \Phi(v_2 : a_2)$$
$$\Phi(\mathsf{left} \cdot v : \langle a_1, q_1\rangle + \langle a_2, q_2\rangle) \coloneqq q_1 + \Phi(v : a_1) \qquad \Phi([\,] : L^{\vec{q}}(a)) \coloneqq 0$$
$$\Phi(\mathsf{right} \cdot v : \langle a_1, q_1\rangle + \langle a_2, q_2\rangle) \coloneqq q_2 + \Phi(v : a_2) \qquad \Phi(v_1 :: v_2 : L^{\vec{q}}(a)) \coloneqq q_1 + \Phi(v_1 : a) + \Phi(v_2 : L^{\triangleleft(\vec{q})}(a)),$$

where $\vec{q} = (q_1, \ldots, q_d)$ and the shift operator $\triangleleft$ on tuples is $\triangleleft(q_1, \ldots, q_d) \coloneqq (q_1 + q_2, \ldots, q_{d-1} + q_d, q_d)$. Equivalently, if $\vec{q} = (q_1, \ldots, q_d)$, then a size-$n$ list $[v_1, \ldots, v_n]$ of type $L^{\vec{q}}(a)$ has potential

$$\Phi([v_1, \ldots, v_n] : L^{\vec{q}}(a)) = \textstyle\sum_{i=1}^{d} q_i \binom{n}{i} + \sum_{i=1}^{n} \Phi(v_i : a). \tag{4.2}$$

Given an evaluation context $V = \{x_1 : v_1, \ldots, x_n : v_n\}$, its potential according to a resource-annotated typing context $\Gamma = \{x_1 : a_1, \ldots, x_n : a_n\}$ is $\Phi(V : \Gamma) \coloneqq \sum_{i=1}^{n} \Phi(v_i : a_i)$. The notation of potential function Φ is used for both value $v : a$ and evaluation context $V : \Gamma$.

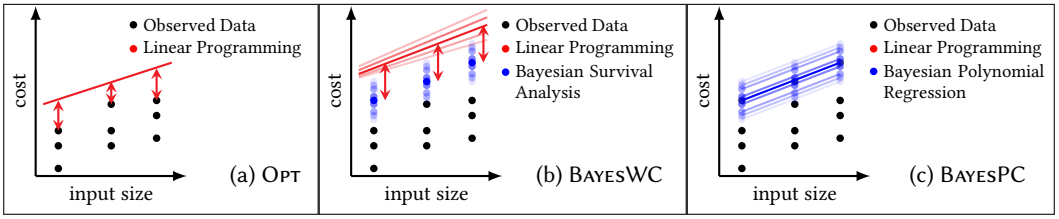


Fig. 2. Three approaches to data-driven resource analysis. (a) Opt uses linear programming to fit a polynomial curve that lies above the runtime data while minimizing the distance to the observed worst-case cost at each input size. (b) BayesWC uses a two-step approach: first, Bayesian survival analysis is used to infer a posterior distribution over the worst-case cost at each input size; second, linear programming is used to fit polynomial curves with respect to samples from the inferred distribution of worst-case costs. (c) BayesPC uses Bayesian polynomial regression to infer the coefficients of polynomial curves that lie above the observed runtime data.

A resource-annotated typing judgment of univariate AARA is

$$\Gamma; p \vdash e : \langle a, q \rangle, \tag{4.3}$$

where Γ is a resource-annotated typing context (i.e., a mapping from variables to resource-annotated types), $p \in \mathbb{Q}_{\geq 0}$ is constant potential of the context, and a is a resource-annotated type of the output with constant potential $q \in \mathbb{Q}_{\geq 0}$. The judgment (4.3) means, given an environment V that carries potential $p + \Phi(V : \Gamma)$, if e evaluates to value v, then v carries $q + \Phi(v : a)$ much potential.

The soundness of univariate AARA is formally stated in Theorem 4.1 [44].

Theorem 4.1 (Soundness of univariate AARA). *Under a monotone resource metric, suppose $\Gamma; p \vdash e : \langle a, q \rangle$. If we have $V \vdash_{\mathcal{P}} e \Downarrow^c v$, then $\Phi(V : \Gamma) + p - \Phi(v : a) - q \geq c$ holds.*

In our prototype of hybrid resource analysis, we use multivariate polynomial AARA [40, 42] that can express multivariate polynomial potential functions and hence is strictly more expressive than univariate polynomial AARA. Multivariate AARA preserves compositionality and linear-programming-based type inference from univariate AARA. However, for the ease of presentation, we use the notation of univariate AARA throughout this article.

5 DATA-DRIVEN BAYESIAN RESOURCE ANALYSIS

This section presents two novel data-driven resource analysis methods called BayesWC and BayesPC that use Bayesian inference to learn probability distributions over cost bounds of programs.

Recall from Section 3.3 that fully data-driven resource analysis begins with a program $\mathcal{P} = \{f(x) = e\}$ that contains a single function f and runtime dataset $\mathcal{D} = \{(V_i, \tilde{v}_i, c_i)\}_{i=1}^N$. To simplify the presentation, we assume in this section that f takes as input a length-n integer list, returns a length $\xi(n)$-integer list, and contains no free variables. Since $V_i \equiv \{x : v_i\}$ holds for each $i, \ldots, N$, we denote the measurements more concisely as $(v_i, \tilde{v}_i, c_i)$. Following Eq. (4.3), a cost bound of f is

$$\{x : L^{\vec{p}}(\texttt{int})\}, p_0 \vdash f\, x : \langle L^{\vec{q}}(\texttt{int}), q_0 \rangle. \tag{5.1}$$

This typing judgment is sound if, for all lists $v : L(\texttt{int})$, the relation $\{x : v\} \vdash_{\mathcal{P}} f\, x \Downarrow^c \tilde{v}$ holds, i.e.,

$$[\Phi(v : L^{\vec{p}}(\texttt{int})) + p_0] - [\Phi(\tilde{v} : L^{\vec{q}}(\texttt{int})) + q_0] \equiv \left[\Psi(|v|; p_0, \vec{p}) - \Psi(|\tilde{v}|; q_0, \vec{q})\right] \geq c, \tag{5.2}$$

where we have introduced the function $\Psi(n; p_0, \vec{p}) \coloneqq \sum_{i=1}^{|\vec{p}|} p_i \binom{n}{i} + p_0 (n \in \mathbb{N})$. Unlike conventional AARA, which derives (5.1) by static analysis of e and linear programming, in data-driven resource analysis we will infer the parameters $(p_0, \vec{p})$ and $(q_0, \vec{q})$ using the dataset $\mathcal{D}$.

5.1 Optimization

Before presenting Bayesian inference, we consider a simple optimization-based baseline (adapted from [23, 33, 90]) to ensure that (5.2) is satisfied with respect to the runtime data $\mathcal{D}$, i.e.,

$$\forall i = 1, \ldots, N.\, \Psi(|v_i|; p_0, \vec{p}) \geq c_i + \Psi(|\tilde{v}_i|; q_0, \vec{q}). \tag{5.3}$$

We seek the tightest bound among all $p_0, \vec{p}, q_0, \vec{q}$ that minimizes the nonnegative cost gaps between the predicted and observed costs in the dataset $\mathcal{D}$. Letting

$$N_{\mathcal{D}} \coloneqq \{|v_i|; i = 1, \ldots, N\} \qquad \text{set of unique input sizes appearing in } \mathcal{D} \tag{5.4}$$

$$\hat{c}_n^{\max} \coloneqq \max\{c_i \mid i = 1, \ldots, N; |v_i| = n\} \qquad \text{max. observed cost for input size } n \in N_{\mathcal{D}} \tag{5.5}$$

$$c_n^{\max} \coloneqq \max\{\mathrm{cost}(f\, v) \mid v : L(\mathtt{int}), |v| = n\} \qquad \text{true worst-case cost for input size } n \in N_{\mathcal{D}}, \tag{5.6}$$

we define the following linear program:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} \left[\Psi(|v_i|; p_0, \vec{p}) - \Psi(|\tilde{v}_i|; q_0, \vec{q})\right] - \hat{c}_{|v_i|}^{\max} \\ \text{subject to} & \Psi(|v_i|; p_0, \vec{p}) \geq \Psi(|\tilde{v}_i|; q_0, \vec{q}) + \hat{c}_{|v_i|}^{\max}\ (i = 1, \ldots, N); \quad p_0, p_1, \ldots, p_{|\vec{p}|}, q_0, q_1, \ldots, q_{|\vec{q}|} \geq 0. \end{array} \tag{Opt-LP}$$

An example of this approach, which we call Opt, is shown in Fig. 2a. While any solution $\hat{p}_0, \hat{\vec{p}}, \hat{r}_0, \hat{\vec{q}}$ to (Opt-LP) is guaranteed to satisfy (5.3), even a conservative estimate $\Psi(n; \hat{p}_0, \hat{\vec{p}})$ of the worst-case cost may lie below the true value $c_n^{\max}$ in Eq. (5.6) (which we assume is finite), as shown in Fig. 1a. This shortcoming occurs because Opt uses the point estimate $\hat{c}_n^{\max}$ given in Eq. (5.5) as a proxy for $c_n^{\max}$, which is not robust in cases where the data $\mathcal{D}$ is such that $\hat{c}_n^{\max} < c_n^{\max}$ for some $n \in N_{\mathcal{D}}$.

5.2 Bayesian Inference on Worst-Case Costs

Overview. Our first approach to addressing the aforementioned limitation of Opt is *Bayesian inference on worst-case costs* (BayesWC). Whereas Opt uses the data $\mathcal{D}$ to form a point estimate $\hat{c}_n^{\max}$ of the worst-case cost $c_n^{\max}$ for each input size $n \in N_{\mathcal{D}}$ in the linear program, BayesWC instead leverages $\mathcal{D}$ to learn an entire probability distribution μ_n that characterizes our uncertainty about $c_n^{\max}$. We identify two requirements that the inferred worst-case cost distributions μ_n must satisfy:

$$\mu_n([\hat{c}_n^{\max}, \infty)) = 1, \qquad \forall \epsilon > 0, w > \hat{c}_n^{\max}.\, \mu_n([w - \epsilon, w + \epsilon]) > 0. \tag{5.7}$$

The left expression guarantees soundness (5.3) with respect to $\mathcal{D}$ and the right expression ensures robustness with respect to the true worst-case cost $c_n^{\max}$. The latter property is not satisfied by Opt.

If we have access to probability distributions μ_n ($n \in N_{\mathcal{D}}$) over worst-case costs, we can use them to robustly estimate bounds by generating $|N_{\mathcal{D}}|$ batches of $M > 0$ i.i.d. samples

$$(c'_{n,1}, \ldots c'_{n,M}) \sim \mu_n \qquad (n \in N_{\mathcal{D}}). \tag{5.8}$$

Reorganizing these $|N_{\mathcal{D}}| \times M$ samples into M lists $\mathbf{c}'_j \coloneqq (c'_{n,j}; n \in N_{\mathcal{D}})$ $(j = 1, \ldots, M)$ each of length $N_{\mathcal{D}}$, we obtain posterior samples of coefficients $p_0, \vec{p}, q_0, \vec{q}$ by solving M linear programs parametrized the random samples $\mathbf{c}'_j$:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} \left[\Psi(|v_i|; p_0, \vec{p}) - \Psi(|\tilde{v}_i|; q_0, \vec{q})\right] - c'_{|v_i|,j} \\ \text{subject to} & \Psi(|v_i|; p_0, \vec{p}) \geq \Psi(|\tilde{v}_i|; q_0, \vec{q}) + c'_{|v_i|,j}\ (i = 1, \ldots, N); \quad p_0, p_1, \ldots, p_{|\vec{p}|}, q_0, q_1, \ldots, q_{|\vec{q}|} \geq 0. \end{array} \tag{BayesWC-LP}$$

Figure 2b shows an example of BayesWC, where the blue dots above a given input size n represents the samples $c'_{n,j}$ from the worst-case cost distribution μ_n. The solutions of the corresponding linear programs (BayesWC-LP) are shown in red. Whereas Opt delivers a single bound using from one LP, BayesWC delivers a posterior samples of bounds using multiple randomly generated LPs.

Sampling worst-case costs via Bayesian inference. A central design challenge in BayesWC is obtaining the distributions μ_n over worst-case costs that satisfy Eq. (5.7). Our approach uses Bayesian inference in a probabilistic generative model over observed costs of the target function f.

In particular, let C_n be a random variable representing the cost of f applied to a size-n input v (the randomness is taken over the (unknown) distribution of v). Letting $\mathbf{v} \coloneqq (v_1, \ldots, v_N)$ denote the observed inputs in $\mathcal{D}$ and $\mathbf{c} \coloneqq (c_1, \ldots, c_N)$ the corresponding costs, we infer the distribution of C_n ($n \in N_{\mathcal{D}}$) by designing a (yet-to-be-specified) Bayesian model that is indexed by $\mathbf{v}$ and defines a probability distribution $\pi_{\mathbf{v}}(\theta, \mathbf{c})$ over latent parameters θ and observable costs $\mathbf{c}$. Conditioned on the realization of $\mathbf{c}$ in the dataset $\mathcal{D}$, the posterior distribution of θ is given by the Bayes rule as

$$\pi_{\mathbf{v}}(\theta|\mathbf{c}) \coloneqq \frac{\pi_{\mathbf{v}}(\theta, \mathbf{c})}{\int_{\theta'} \pi_{\mathbf{v}}(\theta', \mathbf{c}) \mathrm{d}\theta'} \propto h(\theta) \prod_{i=1}^{N} g(c_i; \theta, |v_i|) \tag{5.9}$$

where the joint probability factorizes into *prior distribution* $h(\theta)$ over parameters θ and a product of *likelihood terms* $g(c_i; \theta, |v_i|)$ for the conditionally independent observed costs c_i. Here, g denotes a distribution over costs $\mathbb{R}_{>0}$ parameterized by an input size n and latent parameter θ.

Suppose that, given $\mathcal{D}$, we are able to infer the posterior $\pi_{\mathbf{v}}(\theta|\mathbf{c})$ as defined in Eq. (5.9). We generate samples $(c'_{n,1}, \ldots, c'_{n,M})$ in Eq. (5.8) from the worst-case cost distribution μ_n by sampling:

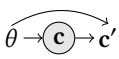
$$\theta_j \sim \pi_{\mathbf{v}}(\theta|\mathbf{c}), \qquad c'_{n,j} \sim \widetilde{g}(\cdot; n, \theta_j, [\hat{c}_n^{\max}, \infty)) \qquad (j = 1, \ldots, M; n \in N_{\mathcal{D}}), \tag{5.10}$$

where the distribution $\widetilde{g}$ is defined as the restriction of g to an interval $U \subset \mathbb{R}$, that is,

$$\widetilde{g}(x; n, \theta, U) \coloneqq \frac{g(x; n, \theta)\mathbf{I}[x \in U]}{\int_{x' \in U} g(x'; n, \theta) \mathrm{d}x'} \qquad (x \in \mathbb{R}). \tag{5.11}$$

Proposition 5.1. *If the likelihood $g(c; \theta, n)$ has full support over $[0, \infty)$, then the inferred worst-case distribution μ_n defined by Eqs. (5.10) and (5.11) satisfies the soundness and robustness properties (5.7).*

Remark 5.2. The reader may be concerned that the distributions of the M simulated worst-cost bounds $\mathbf{c}' \coloneqq (c'_n, n \in N_{\mathcal{D}})$ in Eq. (5.10) are defined in terms of $\hat{c}_n^{\max}$, which are observation-specific quantities. However, because the prior $\pi_{\mathbf{v}}$ is a probability distribution indexed by a *fixed* vector of input instances $\mathbf{v}$ that uniquely define the sizes $N_{\mathcal{D}}$, the random variable c'_n has a well-defined prior distribution, as demonstrated by the factorization and graphical representation:



$$\pi_{\mathbf{v}}(\theta, \mathbf{c}, \mathbf{c}') = h(\theta) \textstyle\prod_{i=1}^{N} g(c_i; \theta, |v_i|) \prod_{n \in N_{\mathcal{D}}} \widetilde{g}(c'_n; n, \theta, [\max_{i \in [N]; |v_i| = n} c_i, \infty)).$$

The conditional independence structure in the model is now obvious, and the M replicates of $\tilde{\mathbf{c}}$ drawn in Eq. (5.11) are valid posterior inferences conditioned on the observed costs $\mathbf{c}$ in $\mathcal{D}$. «

Survival analysis for worst-case costs. To obtain the distribution $\pi_{\mathbf{v}}(\theta, \mathbf{c})$ (5.9) we design a domain-general probability model grounded in survival analysis [4] for predicting "time-to-occurrence" data beyond an observed horizon. We have three parameters $\theta = \{\beta_0, \beta_1, \sigma\}$ with i.i.d. normal prior h; a hyperparameter γ_0; a likelihood model g over observable costs c that is defined implicitly through a variable transformation; and a noise distribution g_{noise}:

$$\beta_0, \beta_1, \sigma \overset{\text{iid}}{\sim} \text{Normal}(0, \gamma_0), \quad \epsilon_i \sim g_{\text{noise}}(0, 1), \quad y_i = \beta_0 + \beta_1 |v_i| + |\sigma| \epsilon_i, \quad c_i = \exp(y_i). \tag{5.12}$$

for each $i = 1, \ldots, N$ (i.i.d.). Possible choices for the noise distribution g_{noise} include the standard normal, logistic, or Gumbel distributions, which in turn imply that the likelihood model g is a log-normal, log-logistic, or Weibull distribution each with scale parameter $\exp(\beta_0 + \beta_1 |v_i|)$ and shape parameters $|\sigma|$, $|\sigma|^{-1}$, and $|\sigma|^{-1}$, respectively. Our reference implementation sets g_{noise} to be a Gumbel distribution, for its relatively heavier tails as compared to other choices.

5.3 Bayesian Inference on Polynomial Coefficients

Overview. Whereas BayesWC performs Bayesian inference on worst-case costs and composes the results with (BayesWC-LP) to deliver bounds, we develop another approach that bypasses LP solving and directly performs inference over the unknown coefficients $p_0, \vec{p}, q_0, \vec{q}$ in the judgment (5.1).

In this approach, which we call BayesPC, our Bayesian model is again indexed by the input instances $\mathbf{v}$ and defines a probability distribution $\pi_{\mathbf{v}}(\theta, p_0, \vec{p}, q_0, \vec{q}, \mathbf{c})$ over a set of auxiliary latent parameters θ, resource coefficients $p_0, \vec{p}, q_0, \vec{q}$, and observable costs $\mathbf{c}$. Conditioned on $\mathbf{c}$, we sample

$$p_0', \vec{p}', q_0', \vec{q}' \sim \pi_{\mathbf{v}}(p_0, \vec{p}, q_0, \vec{q} | \mathbf{c}), \tag{5.13}$$

which define the posterior bound $\lambda n.\ \Psi(n; p_0', \vec{p}') - \Psi(\xi(n); q_0', \vec{q}')$. Figure 2c illustrates this idea: the blue curves represent posterior samples of cost bounds and the blue dots show samples c_n' of inferred worst-case costs that estimate the true value $c_n^{\max}$ at each input size $n \in N_{\mathcal{D}}$. As in BayesWC, BayesPC delivers posterior samples of both worst-case costs and cost bounds, but it rests on a different modeling and inference approach that bypasses linear programming entirely.

Bayesian polynomial regression for inferring coefficients. The generative model $\pi_{\mathbf{v}}(\theta, p_0, \vec{p}, q_0, \vec{q}, \mathbf{c})$ in BayesPC must be carefully designed to ensure that the posterior distribution assigns probability one to the set of coefficients $(p_0, \vec{p}, q_0, \vec{q})$ that satisfy Eq. (5.3), for *any* dataset $\mathcal{D}$. As in Remark 5.2, to obtain a valid Bayesian model, we must define a prior distribution independently of the observed maximum costs $\hat{c}_n^{\max}$. We thus model the observable costs $(c_1, \ldots, c_N)$ as random variables where each c_i takes values in a bounded interval $[0, c'_{|v_i|}]$, $i = 1, \ldots, N$. For each $n \in N_{\mathcal{D}}$, the endpoints c_n' of these intervals are themselves random variables defined as polynomial regression outputs that capture uncertainty in the true worst-case cost $c_n^{\max}$. The generative model in BayesPC is given by:

$$(p_j)_{j=0}^{|\vec{p}|}, (q_j)_{j=0}^{|\vec{q}|} \overset{\text{iid}}{\sim} \text{Normal}_{\geq 0}(0, \gamma_0), \quad \theta \sim h_{\text{noise}}(\gamma_1) \tag{5.14}$$

$$c_n' \coloneqq \Psi(n; p_0, \vec{p}) - \Psi(\xi(n); q_0, \vec{q}) \qquad (n \in N_{\mathcal{D}}) \tag{5.15}$$

$$\epsilon_i \sim \widetilde{g}_{\text{noise}}(\cdot; \theta, [0, c'_{|v_i|}]), \quad c_i \coloneqq c'_{|v_i|} - \epsilon_i \qquad (i = 1, \ldots, N; \text{i.i.d.}), \tag{5.16}$$

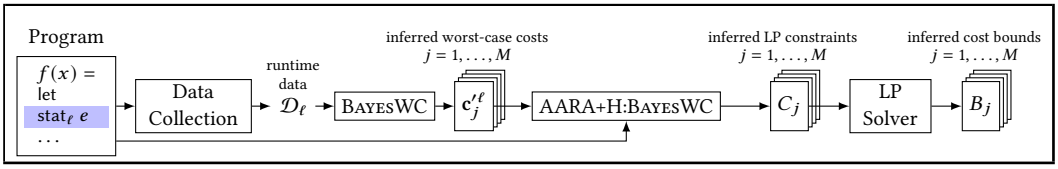
where γ_0, γ_1 are hyperparameters. The term $\widetilde{g}_{\text{noise}}(\cdot; \theta, [0, c_n'])$ is the truncation of an underlying noise distribution $g_{\text{noise}}(\cdot; \theta)$ that has full support over $[0, \infty)$ (c.f., Eq. (5.11)). Its parameter θ has prior h_{noise}. We take g_{noise} to be a Weibull distribution with scale and shape parameters $\theta \coloneqq (\theta_0, \theta_1)$.

Remark 5.3. The main challenge to posterior inference in BayesPC is the fact that the polynomial coefficients $p_0, \vec{p}, q_0, \vec{q}$ are constrained by $\widetilde{g}_{\text{noise}}$ to the linear regions $c_i \leq \Psi(|v_i|; p_0, \vec{p}) - \Psi(\xi(|v_i|); q_0, \vec{q})$ for $i = 1, \ldots, N$ (in addition to further constraints discussed in Section 6.2). Coefficients outside this region have zero posterior probability density because they require $\epsilon_i < 0$ for some i, which has zero prior probability. Whereas traditional Markov chain Monte Carlo algorithms struggle in this setting, we leverage "reflective" Hamiltonian Monte Carlo sampling [15, 18, 58, 69] for posterior inference in BayesPC, where simulated trajectories reflect at the boundaries of the convex polytopes. A high-quality implementation is available in the C++ library Volesti [17]. «

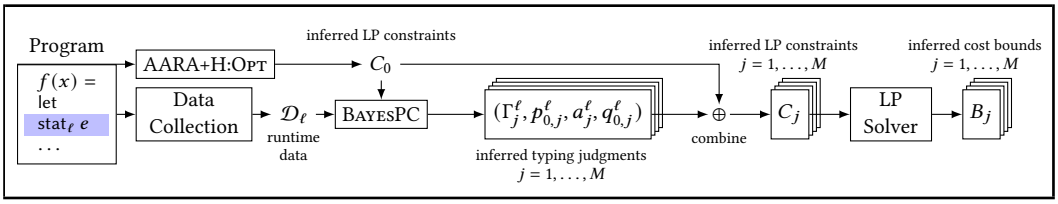
5.4 Generalizations

We briefly describe generalizations of BayesWC and BayesPC that relax the simplifying assumptions made at the beginning of this section and are needed to describe hybrid resource analysis in Section 6.

General variable environments. Suppose we perform data-driven analysis on a function $f(x) = e$ where the measurements $(V_i, \tilde{v}_i, c_i)$ in $\mathcal{D}$ have general variable environments $V_i = \{x_1 : v_1, x_2 : v_2, \ldots\}$. In this case, we define a projection function $\varphi(V, v)$ that maps an environment and value to integer tuples in the set $\cup_{d=0}^{\infty} \mathbb{N}^d$. For example, if $v_1 : L(\texttt{int})$, $v_2 : L(\texttt{int})$, $\tilde{v} : L(\texttt{int})$ are three integer lists



(a) Hybrid BayesWC



(b) Hybrid BayesPC

Fig. 3. Two hybrid resource analysis techniques for composing static and data-driven resource analysis. Subexpression e in program f has a cannot be analyzed using AARA: it is analyzed using Bayesian inference.

and $V = \{x_1 : v_1, x_2 : v_2\}$, we may define $\varphi(V, v) = (|v_1|, |v_2|, |v_1| + |v_2|, |\tilde{v}|) \in \mathbb{N}^4$. The unique input sizes (5.4) are $N_{\mathcal{D}} \coloneqq \{\varphi(V_i, \tilde{v}_i) \mid i = 1, \ldots, N\}$. The models $\pi_{\mathbf{v}}$ in BayesWC and BayesPC are now indexed by $\mathbf{v} \coloneqq ((V_i, \tilde{v}_i), i = 1, \ldots, n)$. Remaining details generalize with minor modifications.

General type judgments. When generalizing the resource-annotated typing judgment (5.1) to the form $\{x_1 : a_1, x_2 : a_2, \ldots, x_m : a_m\}\,; p_0 \vdash e : \langle a, q_0 \rangle$ in Eq. (4.3), each resource-annotated type a_i is associated with symbolic coefficients $\vec{p}_i$ $(i = 1, \ldots, m)$, and the output type a with coefficients $\vec{q}$. The constant potentials p_0 and q_0 are unchanged. The linear programs (Opt-LP) and (BayesWC-LP) and the BayesPC generative model (5.14)–(5.16) are then defined over this expanded collection of coefficients. Simulations in Eq. (5.13) produced from BayesPC, for example, may be written as equivalently $(p_0', \{\vec{p}_i'\}_{i=1}^m, \vec{q}, q_0') \sim \pi_{\mathbf{v}}(\cdot|\mathbf{c})$ or $(p_0', \Gamma', a', q_0') \sim \pi_{\mathbf{v}}(\cdot|\mathbf{c})$. In the latter case, the sampled typing environment Γ' is obtained by using the sampled coefficient $\vec{p}_i'$ in place of the symbolic coefficient within each resource-annotated type a_i $(i = 1, \ldots, m)$, and similarly for a' and $\vec{q}'$.

6 HYBRID RESOURCE ANALYSIS

Having described two novel methods for Bayesian data-driven resource analysis, we next present *hybrid resource analysis* (Hybrid AARA), which integrates data-driven analysis for parts of the program tagged by stat_ℓ with static AARA analysis (Section 4) on remaining parts. Hybrid AARA is based on a formal typing system that extends AARA with a new type judgment:

$$\Gamma; p_0 \vdash_{\mathcal{D}} \mathrm{stat}_\ell\ e : \langle a, q_0 \rangle. \tag{6.1}$$

Eq. (6.1) extends (4.3) by including a dataset $\mathcal{D}$ of runtime measurements that are collected using the procedure described in Section 3.3. We also describe novel type inference algorithms and key technical challenges in the design of the interface between data-driven resource analysis using Bayesian inference and conventional AARA using static inference and linear programming.

6.1 Hybrid BayesWC and Opt

Typing rules. Opt and BayesWC are integrated into the AARA type system (described in Appendix A) by adding the following rules for stat_ℓ subexpressions:

$$\frac{p_0 + \Phi(V_i^\ell : \Gamma) \geq q_0 + \Phi(v_i^\ell : a) + c_i^\ell \quad (i = 1, \ldots, |\mathcal{D}_\ell|)}{\Gamma; p_0 \vdash_{\mathcal{D}} \text{stat}_\ell\ e : \langle a, q_0 \rangle}\ \text{H:Opt} \qquad \frac{p_0 + \Phi(V_i^\ell : \Gamma) \geq q_0 + \Phi(v_i^\ell : a) + c'^\ell_{\varphi(V,v)} \quad c'^\ell_n \sim \pi^\ell_{\mathbf{v}^\ell}(\cdot|\mathbf{c}^\ell) \quad (i = 1, \ldots, |\mathcal{D}_\ell|; n \in N_{\mathcal{D}_\ell})}{\Gamma; p_0 \vdash_{\mathcal{D}} \text{stat}_\ell\ e : \langle a, q_0 \rangle}\ \text{H:BayesWC} \tag{6.2}$$

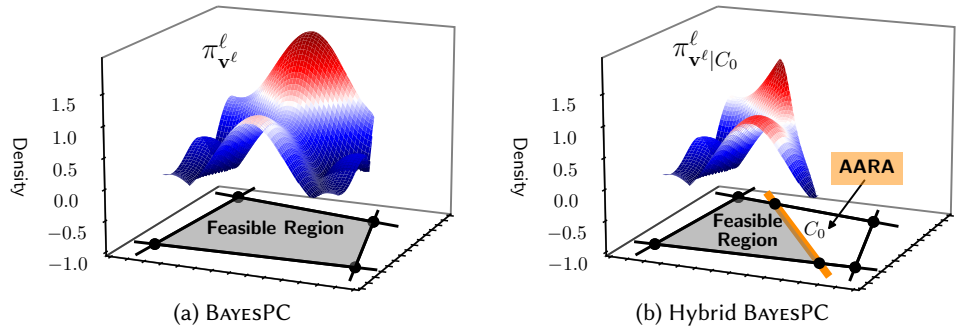
H:Opt states that the consequent holds whenever the input potential $p_0 + \Phi(V : \Gamma)$ is large enough to cover both cost c and leftover potential $q_0 + \Phi(v : a)$ for every measurement $(V, v, c) \in \mathcal{D}_\ell$. In H:BayesWC, we use labels ℓ to disambiguate data-driven inferences related to different labeled stat_ℓ sites in the program. That is, we have $\mathcal{D}_\ell = \left\{(V_i^\ell, v_i^\ell, c_i^\ell) \mid i = 1, \ldots, M_\ell\right\}$, $\mathbf{v}^\ell := ((V_i^\ell, v_i^\ell), i = 1, \ldots, M_\ell)$, and $\mathbf{c}^\ell := (c_i^\ell, i = 1, \ldots, M_\ell)$. The corresponding probabilistic model (e.g., Eq. (5.12)) used within stat_ℓ is denoted π^ℓ. H:BayesWC is similar to H:Opt, except that each observed cost c within a measurement (V, v, c) is replaced with a posterior sample $c'^\ell_{\varphi(V,v)}$ from BayesWC (5.8) that captures inferential uncertainty about the true worst-case cost $c^{\max}_{\varphi(V,v)}$.

Type inference. Because the premises of H:Opt and H:BayesWC are linear constraints over the resource coefficients in e, type inference operates similarly to conventional AARA. Figure 3a shows the type inference workflow for BayesWC. Given the runtime data $\mathcal{D}$ we first perform data-driven BayesWC inference to produce M batches of posterior samples of $\mathbf{c}'^\ell_j := (c'^\ell_{n,j}, n \in N_{\mathcal{D}_\ell})$ for $j = 1, \ldots, M$ and each label ℓ, which define M versions of H:BayesWC for each stat_ℓ subexpression. Next, for each $j = 1, \ldots, M$ we perform a static pass, denoted AARA+H:BayesWC in Fig. 3a, that constructs a template typing tree according to the conventional AARA type system for traditional expressions and uses (the posterior sample of) H:BayesWC for stat_ℓ subexpressions. This process produces M systems of linear constraints C_j $(j = 1, \ldots, M)$, where the linear constraints within each C_j are derived from two provenances: those from the conventional AARA type system and those from the H:BayesWC type rule. Each C_j is provided to an LP solver to provide a typing judgment J_j for the root node's typing context, which translates to an inferred cost bound.

Linear Programming Objective. When solving the overall linear program in AARA+H:BayesWC or AARA+H:Opt, our solver first minimizes the sum of cost gaps form the data-driven components (i.e., (Opt-LP) or (BayesWC-LP)) and then performs minimization of input coefficients at the root. As with conventional AARA [42], it is possible to either minimize the sum of coefficients at the root or minimize higher-degree coefficients with higher priorities. Our prototype of hybrid resource analysis allows users to make either choice.

6.2 Hybrid BayesPC

Key challenge. Integrating Bayesian data-driven resource analysis with BayesPC into conventional AARA is fundamentally more difficult as compared to integrating Opt and BayesWC. This difficulty arises because resource coefficients in the typing judgment $\Gamma; p_0 \vdash_{\mathcal{D}} \text{stat}_\ell\ e : \langle a, q_0 \rangle$ are sampled using Bayesian inference in BayesPC, while they are optimized using LP solvers in both Opt and BayesWC. AARA poses a challenge because we do not know in advance how much potential should be stored in $\langle a, q_0 \rangle$. Unlike in fully data-driven resource analysis, in hybrid resource analysis the output of $\text{stat}_\ell\ e$ may be used in a subsequent computation that also consumes potential. Both Γ and p_0 should store enough potential to pay for both its own cost and the cost of subsequent computation. Naïvely sampling resource annotations (Γ, p_0, a, q_0) for the stat_ℓ subexpressions using BayesPC and providing them to a conventional AARA pass over the remaining program will



(a) BayesPC (b) Hybrid BayesPC

Fig. 4. Posterior distributions over resource coefficients restricted to convex polytopes using BayesPC.

likely produce a system with no solution. Hence, we need an interface between sampling-based probabilistic inference over some coefficients and linear programming over other coefficients.

Type inference. We address this challenge by adding linear constraints to the BayesPC probabilistic models that encode feasible regions of linear programs computed by conventional AARA. This approach guarantees that judgments from BayesPC cannot impose new constraints that cause the linear programs to fail to have a solution.

Figure 3b shows our type inference approach. We start by performing a static analysis pass through the program using the conventional AARA type system to obtain a set of linear constraints C_0, treating any stat_ℓ using the rule H:Opt to ensure consistency with runtime data $\mathcal{D}_\ell$. Next, for each subexpression stat_ℓ encountered in the first pass, we apply a variant of BayesPC that combines both the runtime data $\mathcal{D}_\ell$ and constraints C_0 from the first pass to infer a resource-annotated typing judgment of the form $\Gamma; p_0 \vdash_{\mathcal{D}} \mathrm{stat}_\ell\ e_\ell : \langle a, q_0 \rangle$.

Let $\pi^{\ell}_{\mathbf{v}^{\ell}}(\theta, \Gamma, p_0, q_0, a, \mathbf{c})$ denote the BayesPC probabilistic model for stat_ℓ (e.g., Eqs. (5.14)–(5.16)). The constraints C_0 are used to construct a modified probabilistic model $\pi^{\ell}_{\mathbf{v}^{\ell}|C_0}(\varepsilon, \theta, \Gamma, p_0, q_0, a, \mathbf{c})$, where ε is a collection of nuisance parameters that contains all the resource coefficients appearing in the AARA constraints C_0. Letting $h(\varepsilon)$ denote an uninformative prior (e.g., uniform if C_0 is bounded), the modified probabilistic model is restricted to the convex polytope defined by C_0:

$$\pi^{\ell}_{\mathbf{v}^{\ell}|C_0}(\varepsilon, \theta, \Gamma, p_0, q_0, a, \mathbf{c}) \propto h(\varepsilon)\pi^{\ell}_{\mathbf{v}^{\ell}}(\varepsilon, \theta, \Gamma, p_0, q_0, a, \mathbf{c})\mathbf{I}[(\varepsilon, \dots) \in C_0], \tag{6.3}$$

where $\mathbf{I}[(\varepsilon, \dots) \in C_0]$ is 1 if the resource coefficients in the arguments to $\pi^{\ell}_{\mathbf{v}^{\ell}|C_0}$ satisfy constraints C_0 and 0 otherwise. Figure 4 shows an example: the original "Feasible Region" (grey, Fig. 4a) in BayesPC induced by runtime data is further constrained by the constraints from the first AARA pass (orange, Fig. 4b), with the distribution re-normalized appropriately.

Eq. (6.3) is then used to sample judgments

$$(\Gamma^{\ell}_j, p^{\ell}_{0,j}, a^{\ell}_j, q^{\ell}_{0,j}) \sim \pi^{\ell}_{\mathbf{v}^{\ell}|C_0}(\Gamma, p_0, q_0, a|\mathbf{c}^{\ell}) \qquad (j = 1, \dots, M; \ell \in L'), \tag{6.4}$$

which are shown as the output of BayesPC in Fig. 3a. Each sampled typing judgment in Eq. (6.4) corresponds to a concrete realization of symbolic LP variables in C_0 created by the corresponding H:Opt rule at label ℓ during the first pass. We can then obtain M new constraints

$$C_j \coloneqq C_0 \oplus \left\{(\Gamma^{\ell}_j, p^{\ell}_{0,j}, a^{\ell}_j, q^{\ell}_{0,j}) \mid \ell \in L'\right\} \qquad (j = 1, \dots, M) \tag{6.5}$$

by syntactically replacing the symbolic LP variables in C_0 with concrete variables at all labels $\ell \in L'$. Each C_j is then fed to an LP solver to obtain M posterior samples $(B_1, \dots, B_M)$ of cost bounds.

6.3 Soundness

We formulate and prove two soundness theorems for Hybrid AARA.

Theorem 6.1 establishes that inferred cost bounds from Hybrid AARA are sound with respect to all N measurements in the runtime data (collected using the procedure described in Section 3.3)

Theorem 6.1. *Given a program $\mathcal{P}$, expression e and $\mathcal{D}$ be the runtime data such that $(V_i \vdash_{\mathcal{P}} e \Downarrow^{c_i} v_i)_{i=1}^N \mid \mathcal{D}$. The following property holds with probability one: If $\Gamma; p_0 \vdash_{\mathcal{D}} e : \langle a, q_0\rangle$ holds in the type system of hybrid resource analysis, then $\Phi(V_i : \Gamma) + p_0 - \Phi(v_i : a) - q_0 \geq c_i$ for any $i = 1, \ldots, N$.*

Proof. The proof proceeds by nested induction on $(V_i \vdash e \Downarrow_{c_i} v_i)_{i=1}^N \mid \mathcal{D}$ (outer induction) and $\Gamma; p_0 \vdash_{\mathcal{D}} e : \langle a, q_0\rangle$ (inner induction), following the same structure as the soundness proof of conventional AARA [39, 40]. The only necessary modification is proving the base case where $e \equiv \mathrm{stat}_\ell\ e'$ for some expression e', for the new inference rules H:Opt, H:BayesWC, and H:BayesPC.

We argue that for each typing rule, the resource-annotated judgment $\Gamma; P \vdash_{\mathcal{D}} \mathrm{stat}_\ell\ e' : \langle a, q_0\rangle$ is sound with respect to any measurement $(V, v, c) \in \mathcal{D}_\ell$, meaning that the typing rule ensures $\Phi(V : \Gamma) + p_0 - \Phi(v : a) - q_0 \geq c$ holds with probability one. For H:OPT, the property follows immediately from the premise of the typing rule. For H:BayesWC, Proposition 5.1 establishes a condition that guarantees, for any runtime sample $(V, v, c) \in \mathcal{D}_\ell$, the probabilistic model π_{v} used in BayesWC has zero probability of simulating a cost $c'^{\ell}_{\varphi(V,v)} < c$. Our choice of the likelihood function g in Eq. (5.12) satisfies this requirement. Using this fact, the premise of H:BayesWC implies

$$\Phi(V : \Gamma) + p_0 - \Phi(v : a) - q_0 \geq c'^{\ell}_{\varphi(V,v)} \geq \hat{c}^{\max,\ell}_{\varphi(V,v)} \geq c = 1 \qquad \text{almost surely,} \tag{6.6}$$

and the conclusion follows. For H:BayesPC, Remark 5.3 establishes that the π^{ℓ}_{v} assigns zero posterior probability density to any (Γ, p_0, a, q_0) that satisfies $\Phi(V : \Gamma) + p_0 - \Phi(v : a) - q_0 < c$ for some runtime sample $(V, v, c) \in \mathcal{D}_\ell$. As constraining π^{ℓ}_{v} to the convex polytope C_0 in Eq. (6.3), cannot possibly increase its support, the conclusion follows for H:BayesPC. □

The next result establishes the probability (over the randomness of the collected runtime data) that inferred cost bounds from Hybrid AARA are sound up to a given input-size limit converges to one as the dataset size N tends to infinity.

Theorem 6.2. *Let $\mathcal{D}$ be runtime data such that $(V_i \vdash_{\mathcal{P}} e \Downarrow^{c_i} v_i)_{i=1}^N \mid \mathcal{D}$, where $(V_i)_{i=1}^N$ are N i.i.d. samples from an (unknown) input distribution and e is an expression in program $\mathcal{P}$. Assume there exists an integer m and finite set $\mathcal{V}_m$ of well-typed environments such that for all $V \in \mathcal{V}_m$: $\varphi(V) < m$ and V has nonzero probability of appearing in $\mathcal{D}$. Then for any typing judgment $\Gamma; p_0 \vdash_{\mathcal{D}} e : \langle a, q_0\rangle$ inferred from Hybrid AARA, the probability that it satisfies $\Phi(V : \Gamma) + p_0 - \Phi(v : a) - q_0 \geq c$ for all V such that $V \vdash_{\mathcal{P}} e \Downarrow^c v$ and $\varphi(V) \leq m$ converges to one as $N \to \infty$.*

Proof. For each $n = 1, \ldots, m$, there exists a worst-case environment

$$V_n^{\max} := \arg\max_{V \in \mathcal{V}_m} \{c \mid \phi(V) = n, V \vdash_{\mathcal{P}} e \Downarrow^c \tilde{v}\} \tag{6.7}$$

with a maximal execution cost. Let $p_n^{\max} > 0$ be the probability $V_n^{\max}$ is selected. Let $W_{n,N}$ be the event that $V_n^{\max}$ is sampled within N i.i.d. draws. The probability that all worst-case inputs $V_1^{\max}, \ldots, V_m^{\max}$ occur in runtime dataset $\mathcal{D}$ of size N is

$$\mathbb{P}\left(\bigcap_{n=0}^m W_{n,N}\right) = 1 - \mathbb{P}\left(\bigcup_{n=0}^m \overline{W}_{n,N}\right) \geq 1 - \sum_{n=1}^m \mathbb{P}(\overline{W}_{n,N}) = 1 - \sum_{n=1}^m (1 - p_n^{\max})^N \tag{6.8}$$

where $\overline{W}_{n,N}$ denotes the complement of event $W_{n,N}$. The inequality follows from the union bound. The last expression converges to one as $N \to \infty$ because $p_n^{\max} > 0$ for each n. The conclusion follows from Theorem 6.1, which establishes that resource-annotated typing judgements from Hybrid AARA are sound with respect to the environments used to generate runtime data $\mathcal{D}$. □

7 EVALUATION

Implementation. We implemented a prototype of Hybrid AARA [76] that integrates data-driven (optimization-based and Bayesian) resource analysis into Resource-Aware ML (RaML) [42] and evaluated it on challenging benchmarks that purely static analysis cannot solve. Each analysis run using the prototype requires: (i) an OCaml program annotated with `Raml.tick` and `Raml.stat`; (ii) a list of inputs for runtime cost data generation; and (iii) a configuration file, which includes e.g., a polynomial degree, a data-driven analysis technique, a probabilistic model, and hyperparameters. Appendix B describes an empirical Bayesian procedure to automatically determine hyperparameters.

Benchmarks. We built a benchmark suite of 10 functional programs (source code in Appendix C).

- `MapAppend`: Given two lists, x and y, for each element of x, run some (statically unanalyzable) function f and append its output to the cumulative result (whose initial value is y).
- `Concat`: Given a nested list, recursively append inner lists to the cumulative result.
- `InsertionSort2`: Sequentially run insertion sort twice on a list. We focus on the cost of comparisons in the second insertion sort.
- `QuickSort`: Run deterministic quicksort on lists with their heads as pivots for partition.
- `QuickSelect`: Given an integer i and a list, run deterministic quickselect and return the i^{th} smallest element in the list. Like `Quicksort`, it uses the heads of lists as pivots.
- `MedianOfMedians`: Given an integer i and a list, recursively compute the i^{th} smallest element in the list by first computing the median of medians then using it to partition the list.
- `ZAlgorithm`: Given a list x, return a list y such that $y[i]$ stores the maximum integer ℓ such that $x[0, \ldots, \ell - 1] = x[i, \ldots, i + \ell - 1]$.
- `BubbleSort`: Run bubble sort where pairs of adjacent reverse-ordered elements are repeatedly swapped until no such pairs exist (i.e., saturation).
- `Round`: Given a natural number x (represented as a list), compute a natural number y such that y is the largest power of two below x. Once y is computed, we traverse y.
- `EvenOddTail`: Given a natural number x (represented as a list), first traverse the list and if x is even, we divide it by two; otherwise, we subtract one from it.

Conventional AARA cannot return a tight cost bound for any of these 10 programs. Specifically, it cannot analyze 7/10 programs at all as they contain statically unanalyzable code fragments. For `BubbleSort`, AARA cannot even reason about its termination due to bubble sort being saturation-based. For `MedianOfMedians`, AARA cannot reason about the mathematical properties of medians. For `Round`, to prove its linear complexity, AARA would need an infinitely tall resource-annotated typing tree. The four benchmarks `MapAppend`, `Concat`, `QuickSort`, and `QuickSelect` fail because they contain some complex function that conventional AARA cannot analyze (e.g., OCaml's built-in polymorphic comparator). For the remaining 3/10 programs, to infer some polynomial cost bound, conventional AARA requires a wrong polynomial degree that is too high.

Proportions of sound cost bounds. Table 1 shows the proportion of inferred cost bounds that are sound as well as analysis runtimes for all 10 benchmarks using data-driven analysis (Opt, BayesWC, BayesPC) and their hybrid counterparts, where applicable. Optimization-based resource analyses (data-driven and hybrid Opt) always return a single inferred cost bound, while Bayesian resource analyses return collections of samples of cost bounds drawn from the posterior probability distribution. In all benchmarks, for both data-driven and hybrid analyses, BayesWC produces strictly higher proportions of sound cost bounds than Opt. This finding demonstrates that cost bounds inferred by BayesWC are more *robust* than those inferred by Opt in 10/10 cases (data-driven) and 7/7 cases (hybrid); that is, BayesWC has a strictly positive probability of inferring a sound bound even though the runtime cost dataset does not contain worst-case inputs. In contrast,

Table 1. Percentage of inferred cost bounds that are sound and analysis runtime for 10 benchmark programs.

Benchmark Program	Conventional AARA	Analysis Method	Fraction of Sound Inferred Bounds		Analysis Runtime	
			Data-Driven	Hybrid	Data-Driven	Hybrid
MapAppend	*Cannot Analyze*	Opt	0%	0%	0.01 s	0.01 s
		BayesWC	68.5%	100%	1.87 s	12.44 s
		BayesPC	75.5%	100%	51.83 s	360.80 s
Concat	*Cannot Analyze*	Opt	0%	0%	0.00 s	0.01 s
		BayesWC	67.3%	96.7%	2.54 s	14.73 s
		BayesPC	96%	100%	113.53 s	125.28 s
InsertionSort2	*Wrong Degree*	Opt	0%	0%	0.01 s	0.02 s
		BayesWC	57.6%	100%	1.53 s	5.46 s
		BayesPC	21%	57.5%	10.68 s	220.66 s
QuickSort	*Cannot Analyze*	Opt	0%	0%	0.01 s	0.11 s
		BayesWC	4%	96%	2.20 s	144.88 s
		BayesPC	0%	100%	13.72 s	274.51 s
QuickSelect	*Cannot Analyze*	Opt	0%	0%	0.02 s	0.19 s
		BayesWC	0.2%	98.2%	1.83 s	222.47 s
		BayesPC	0%	100%	12.39 s	277.20 s
MedianOfMedians	*Cannot Analyze*	Opt	0%	0%	0.17 s	0.21 s
		BayesWC	11.5%	71.3%	2.36 s	93.89 s
		BayesPC	0%	100%	70.39 s	896.98 s
ZAlgorithm	*Wrong Degree*	Opt	0%	0%	0.09 s	0.13 s
		BayesWC	13.7%	95.9%	1.96 s	72.21 s
		BayesPC	28%	100%	11.11 s	509.29 s
BubbleSort	*Cannot Analyze*	Opt	0%	*Cannot Analyze*	0.01 s	∅
		BayesWC	40.1%	*Cannot Analyze*	2.69 s	∅
		BayesPC	31.5%	*Cannot Analyze*	11.70 s	∅
Round	*Cannot Analyze*	Opt	0%	*Cannot Analyze*	0.01 s	∅
		BayesWC	58.3%	*Cannot Analyze*	1.91 s	∅
		BayesPC	81%	*Cannot Analyze*	12.87 s	∅
EvenOddTail	*Wrong Degree*	Opt	0%	*Wrong Degree*	0.01 s	∅
		BayesWC	65.1%	*Wrong Degree*	1.98 s	∅
		BayesPC	70%	*Wrong Degree*	11.79 s	∅

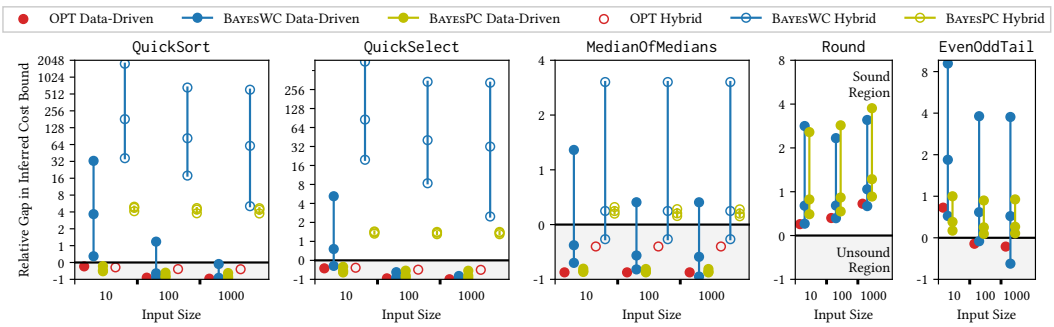


Fig. 5. Estimation gaps of inferred cost bounds with respect to ground-truth worst-case costs on 5 benchmarks.

in these benchmarks, Opt never returns a sound bound. Likewise, BayesPC returns more robust bounds than Opt in 7/10 (data-driven) and 7/7 cases (hybrid). These improvements highlight the benefits of probabilistic inference as compared to optimization.

Between fully data-driven Bayesian analysis and Hybrid AARA using BayesWC and BayesPC, the latter delivers a substantially larger fraction of sound cost bounds in all seven benchmarks. These results illustrate the benefits of integrating data-driven analysis with static type inference on the remaining parts of the program not tagged with stat expressions.

The right two columns of Table 1 show the analysis runtime. Opt uses an LP solver that is much faster than sampling algorithms: it returns answers in less than one second, whereas Bayesian resource analysis can take minutes. Between BayesWC and BayesPC, the former is faster in terms of analysis time per iteration. The main reason is that BayesWC uses HMC sampling without constraints, which is much simpler than reflective HMC sampling over convex polytopes (Fig. 4) needed for inference in BayesPC.

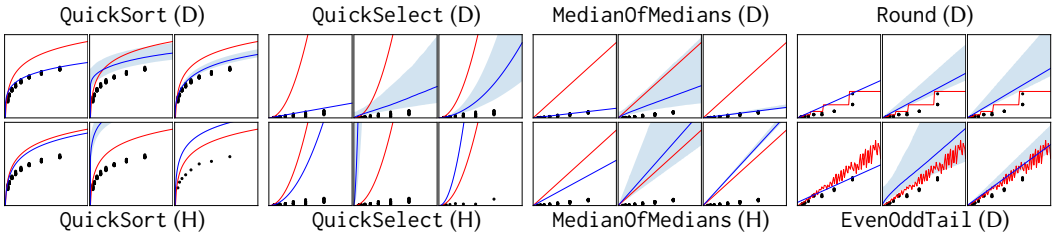
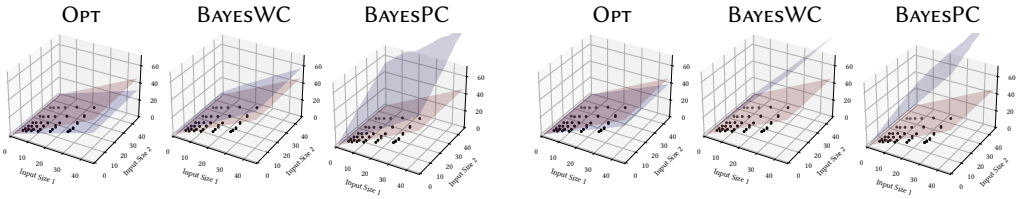


Fig. 6. Plots of inferred bounds for various benchmarks and resource analysis methods shown in Fig. 5 (D=Data Driven; H=Hybrid). Each benchmark has three plots (left-to-right): Opt, BayesWC, and BayesPC.



(a) Data-Driven Resource Analysis (b) Hybrid Resource Analysis

Fig. 7. Plots of inferred multivariate cost bounds for MapAppend using data-driven resource analysis and Hybrid AARA. Red planes show ground-truth tight worst-case bound and blue planes show inferred bounds.

Distributions of estimation gaps. Figure 5 shows relative estimation gaps of inferred cost bounds with respect to the ground truth worst case bound in five benchmarks (full results in Appendix C). In each benchmark, we fix three input sizes (10, 100, and 1000). For each size, we show the 5^{th}, 50^{th}, and 95^{th} percentiles of relative estimation gaps. Because Opt infers a single bound, it has the same estimation gap for all percentiles. Relative estimation gaps below 0 (resp. above 0) indicate an underestimate (resp. overestimate) of the true bound. A cost bound is sound if its estimation gap is at least 0. Figure 5 exhibits similar results as in Table 1. The `QuickSort` and `QuickSelect` panels in Fig. 5 show an interesting finding. At input size 10, the bounds from Data-Driven BayesWC and BayesPC are tighter than those from their Hybrid counterparts, but some of the former bounds are unsound. As the input size increases, the estimation gaps from Hybrid AARA shrink but remain in the "Sound Region," whereas those from data-driven Bayesian analysis become unsound.

Posterior distributions of cost bounds. Figure 6 shows posterior distributions of cost bounds of five benchmarks. Red curves are the true bounds and black dots show the runtime cost data. Blue curves are median cost bounds, and light-blue shades around them show the 10–90^{th} percentile ranges. In both data-driven and hybrid analyses, Bayesian resource analysis delivers a sizeable fraction of correct bounds. Certain datasets such as `MedianOfMedians` in Fig. 6 are particularly challenging for purely data-driven analysis, however. Between data-driven and hybrid analyses, the cost bounds from the latter are always more *accurate* (i.e., closer to sound worst-case cost bounds). Fig. 7 shows inferred cost bounds for `MapAppend`, which are multivariate. The median bounds for the Bayesian methods always lie about the ground-truth plane, whereas the single bounds for Data-Driven Opt and Hybrid Opt are both incorrect.

Summary. Our evaluation has several key takeaways. First, in the 7/10 programs that conventional AARA cannot solve (first 7 rows of Table 1), Hybrid AARA successfully returns accurate and robust bounds using at least one of Hybrid BayesWC or Hybrid BayesPC. In the 3/10 programs that Hybrid AARA cannot solve (last 3 rows of Table 1), purely data-driven analysis using BayesWC or

BayesPC gives good proportions of sound bounds. Second, Bayesian resource analysis delivers a substantially higher percentage of sound cost bounds as compared to optimization-based analysis in both the purely data-driven and hybrid case. Between BayesWC and BayesPC, the former is more conservative: it gives a larger fraction of sound bounds in the purely data-driven case, but its cost gaps are higher. Third, for Hybrid AARA, incorporating static analysis (when possible) gives a higher proportion of sound bounds than purely data-driven analysis with a similar runtime.

We have no clear winner between Hybrid BayesWC and Hybrid BayesPC: the latter is slower but more accurate than the former in 5/7 cases. Additionally, BayesPC enables richer probabilistic models than BayesWC since the probabilistic model of BayesPC models not only worst-case costs but also resource coefficients. In practice, depending on the application, one can take the looser or tighter result of the two methods.

8 RELATED WORK

Static resource analysis. Existing static resource techniques are based on type systems [5, 21, 24, 26, 38, 59, 86], recurrence relations [2, 25, 35, 55, 56, 88], term rewriting [6, 7, 49, 72], ranking functions [12, 20, 31, 82], and abstract interpretation [3, 37, 91]. Our works builds on automatic amortized resource analysis (AARA) [36, 39, 40, 42, 45, 48, 54, 57, 61] a type-based resource-analysis technique to analyze functional programs by automating the potential method of amortized analysis [83–85]. AARA also has been used to analyze imperative code [13], parallel programs [46], probabilistic programs [8, 73, 87], and session-typed concurrent programs [27, 28]. We are not aware of prior works that combine static resource analysis with data-driven resource analysis.

Experimental algorithmics. Inference of algorithmic complexity from experiment data has been extensively studied in the field of experimental algorithmics [29, 52, 65, 66, 71, 81]. These works focus on using experimental data for improving pen-and-paper analyses of asymptotic complexity of algorithms. Instead, this work focuses on using data-driven approaches for deriving non-asymptotic bounds for programs and combining data-driven analysis with static resource analysis.

Data-driven resource analysis. There exist tools for analyzing the complexity of programs from runtime cost samples [23, 30, 33, 50, 77, 80, 90]. Most of these works deliver average-case bounds instead of worst-case bounds. Dynaplex [51] first infers a recurrence relation of a recursive program from its execution traces by optimization and then solves the recurrence relation by the master theorem. Our work on data-driven Bayesian resource analysis differs from existing works in this field in several ways. First, we develop novel data-driven resource analyses based on Bayesian inference (Section 5), which give us posterior distributions over unknown bounds that are more robust than point estimates from optimization-based inference (e.g., Demontiê et al. [30] use polynomial interpolation and Goldsmith et al. [33] use least-square power regression). Second, we integrate data-driven Bayesian resource analysis into AARA to obtain Hybrid AARA (Section 6), which to our knowledge is the first such combination of static and data-driven analysis.

Worst-case execution time analysis. Data-driven techniques have been studied in the worst-case execution time (WCET) analysis of real-time embedded systems. This area focuses on deriving concrete (i.e., non-symbolic) cost bounds of programs running on real-world hardware. Typically, the maximum number of loop iterations in programs is assumed to be a constant independent of input sizes. Numerous works study the use of extreme-value theory in measurement-based, probabilistic WCET analysis [1, 16, 32, 34, 63, 64, 68, 78]. These methods all leverage frequentist (non-Bayesian) inference. The hybrid of static and measurement-based WCET analyses has been studied [9, 10, 89]. Unlike WCET analysis, our work focuses on symbolic cost bounds of programs parametric in input sizes.

9 CONCLUSION

This article has introduced Hybrid AARA, which to the best of our knowledge is the first resource analysis method that combines data-driven and static resource analysis. We have presented two novel data-driven Bayesian resource analysis methods (BayesWC and BayesPC) for inferring principled posterior distributions over cost bounds. The data-driven Bayesian resource analyses introduced in this work deliver more accurate and robust bounds as compared to previous methods that use greedy optimization, which only gives a single point estimate. Our main contribution—Hybrid AARA—integrates our Bayesian data-driven resource analyses into conventional AARA using a novel type inference system. Two notions of soundness for Hybrid AARA are identified and proven. We implemented a prototype of Hybrid AARA by extending Resource-Aware ML (RaML) and evaluated it on a set of 10 challenging benchmarks that conventional AARA cannot solve. The results on this benchmark set indicate that Hybrid AARA achieves several of its design goals. Our system demonstrates the benefits of integrating Bayesian inference with traditional type inference for improving the coverage of static analysis on the one hand and the accuracy and robustness of data-driven analysis on the other hand.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous referees for their valuable comments and helpful suggestions. We also would like to thank Richard Peng and Daniel Sleator at Carnegie Mellon University for their helpful suggestions for the benchmark suite of Hybrid AARA. This material is based upon work supported by the National Science Foundation under Grant Nos. 2311983, 2007784, and 1845514. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Jaume Abella, Maria Padilla, Joan Del Castillo, and Francisco J. Cazorla. 2017. Measurement-Based Worst-Case Execution Time Estimation Using the Coefficient of Variation. *ACM Transactions on Design Automation of Electronic Systems* 22, 4, Article 72 (June 2017), 29 pages. https://doi.org/10.1145/3065924

[2] Elvira Albert, Puri Arenas, Samir Genaim, German Puebla, and Damiano Zanardini. 2007. Cost Analysis of Java Bytecode. In *Programming Languages and Systems (Lecture Notes in Computer Science, Vol. 4421)*. Springer, Berlin, 157–172. https://doi.org/10.1007/978-3-540-71316-6_12

[3] Elvira Albert, Puri Arenas, Samir Genaim, German Puebla, and Damiano Zanardini. 2008. COSTA: Design and Implementation of a Cost and Termination Analyzer for Java Bytecode. In *Formal Methods for Components and Objects (Lecture Notes in Computer Science, Vol. 5382)*. Springer, Berlin, 113–132. https://doi.org/10.1007/978-3-540-92188-2_5

[4] Per K. Andersen and Michael Vaeth. 2015. Survival Analysis. In *Wiley StatsRef: Statistics Reference Online*. John Wiley & Sons, Ltd, Hoboken, NJ, USA. https://doi.org/10.1002/9781118445112.stat02177.pub2

[5] Martin Avanzini and Ugo Dal Lago. 2017. Automating Sized-Type Inference for Complexity Analysis. *Proc. ACM Program. Lang.* 1, ICFP, Article 43 (Aug. 2017), 29 pages. https://doi.org/10.1145/3110287

[6] Martin Avanzini, Ugo Dal Lago, and Georg Moser. 2015. Analysing the Complexity of Functional Programs: Higher-order Meets First-Order. In *Proc. 20th ACM SIGPLAN International Conference on Functional Programming*. ACM, New York, NY, USA, 152–164. https://doi.org/10.1145/2784731.2784753

[7] Martin Avanzini and Georg Moser. 2016. A Combination Framework for Complexity. *Information and Computation* 248 (2016), 22–55. https://doi.org/10.1016/j.ic.2015.12.007

[8] Martin Avanzini, Georg Moser, and Michael Schaper. 2020. A Modular Cost Analysis for Probabilistic Programs. *Proc. ACM Program. Lang.* 4, OOPSLA (Nov. 2020), 1–30. https://doi.org/10.1145/3428240

[9] Guillem Bernat, Antoine Colin, and Stefan Petters. 2003. *pWCET: A Tool for Probabilistic Worst-Case Execution Time Analysis of Real-Time Systems.* Technical Report YCS-2003-353. University of York.

[10] Adam Betts, Nicholas Merriam, and Guillem Bernat. 2010. Hybrid Measurement-Based WCET Analysis at the Source Level Using Object-Level Traces. In *10th International Workshop on Worst-Case Execution Time Analysis (OpenAccess Series in Informatics (OASIcs), Vol. 15)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 54–63. https://doi.org/10.4230/OASIcs.WCET.2010.54

[11] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. 1973. Time Bounds for Selection. *J. Comput. System Sci.* 7, 4 (Aug. 1973), 448–461. https://doi.org/10.1016/S0022-0000(73)80033-9

[12] Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl. 2014. Alternating Runtime and Size Complexity Analysis of Integer Programs. In *Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2014) (Lecture Notes in Computer Science, Vol. 8413)*. Springer, Berlin, 140–155. https://doi.org/10.1007/978-3-642-54862-8_10

[13] Quentin Carbonneaux, Jan Hoffmann, and Zhong Shao. 2015. Compositional Certified Resource Bounds. In *Proc. 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, USA, 467–478. https://doi.org/10.1145/2737924.2737955

[14] George Casella. 1985. An Introduction to Empirical Bayes Data Analysis. *The American Statistician* 39, 2 (1985), 83–87. https://doi.org/10.1080/00031305.1985.10479400

[15] Frederic Cazals, Augustin Chevallier, and Sylvain Pion. 2022. Improved Polytope Volume Calculations Based on Hamiltonian Monte Carlo with Boundary Reflections and Sweet Arithmetics. *Journal of Computational Geometry* 13, 1 (April 2022), 52–88. https://doi.org/10.20382/jocg.v13i1a3

[16] Francisco J. Cazorla, Tullio Vardanega, Eduardo Quiñones, and Jaume Abella. 2013. Upper-Bounding Program Execution Time with Extreme Value Theory. In *13th International Workshop on Worst-Case Execution Time Analysis (Open Access Series in Informatics, Vol. 30)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Wadern, 64–76. https://doi.org/10.4230/OASIcs.WCET.2013.64

[17] Apostolos Chalkis and Vissarion Fisikopoulos. 2021. Volesti: Volume Approximation and Sampling for Convex Polytopes in R. *The R Journal* 13, 2 (2021), 642–660. https://doi.org/10.32614/RJ-2021-077

[18] Apostolos Chalkis, Vissarion Fisikopoulos, Marios Papachristou, and Elias Tsigaridas. 2023. Truncated Log-concave Sampling for Convex Bodies with Reflective Hamiltonian Monte Carlo. *ACM Trans. Math. Software* 49, 2, Article 16 (June 2023), 25 pages. https://doi.org/10.1145/3589505

[19] Arthur Charguéraud and François Pottier. 2019. Verifying the Correctness and Amortized Complexity of a Union-Find Implementation in Separation Logic with Time Credits. *Journal of Automated Reasoning* 62, 3 (March 2019), 331–365. https://doi.org/10.1007/s10817-017-9431-7

[20] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2019. Non-Polynomial Worst-Case Analysis of Recursive Programs. *ACM Transactions on Programming Languages and Systems* 41, 4, Article 20 (Oct. 2019), 52 pages. https://doi.org/10.1145/3339984

[21] Ezgi Çiçek, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Jan Hoffmann. 2017. Relational Cost Analysis. In *Proc. 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 316–329. https://doi.org/10.1145/3009837.3009858

[22] Ezgi Çiçek, Mehdi Bouaziz, Sungkeun Cho, and Dino Distefano. 2021. Static Resource Analysis at Scale (Extended Abstract). In *Proc. 27th International Static Analysis Symposium (Lecture Notes in Computer Science, Vol. 12389)*. Springer, Cham, 3–6. https://doi.org/10.1007/978-3-030-65474-0_1

[23] Emilio Coppa, Camil Demetrescu, and Irene Finocchi. 2012. Input-Sensitive Profiling. In *Proc. 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, USA, 89–98. https://doi.org/10.1145/2254064.2254076

[24] Karl Crary and Stephnie Weirich. 2000. Resource Bound Certification. In *Proc. 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 184–198. https://doi.org/10.1145/325694.325716

[25] Joseph W. Cutler, Daniel R. Licata, and Norman Danner. 2020. Denotational Recurrence Extraction for Amortized Analysis. *Proc. ACM Program. Lang.* 4, ICFP, Article 97 (Aug. 2020), 29 pages. https://doi.org/10.1145/3408979

[26] Nils Anders Danielsson. 2008. Lightweight Semiformal Time Complexity Analysis for Purely Functional Data Structures. In *Proc. 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 133–144. https://doi.org/10.1145/1328438.1328457

[27] Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, and Ishani Santurkar. 2019. Resource-Aware Session Types for Digital Contracts. arXiv:1902.06056 [cs]

[28] Ankush Das, Jan Hoffmann, and Frank Pfenning. 2018. Work Analysis with Resource-Aware Session Types. In *Proc. 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, New York, NY, USA, 305–314. https://doi.org/10.1145/3209108.3209146

[29] Camil Demetrescu, Irene Finocchi, and Giuseppe F. Italiano. 2003. Algorithm Engineering, Algorithmics Column. *Bull. EATCS* 79 (2003), 48–63.

[30] Francisco Demontiê, Junio Cezar, Mariza Bigonha, Frederico Campos, and Fernando Magno Quintão Pereira. 2015. Automatic Inference of Loop Complexity Through Polynomial Interpolation. In *Proc. 19th Brazilian Symposium on Programming Languages (Lecture Notes in Computer Science, Vol. 9325)*, Alberto Pardo and S. Doaitse Swierstra (Eds.). Springer, Cham, 1–15. https://doi.org/10.1007/978-3-319-24012-1_1

[31] Jürgen Giesl, Nils Lommen, Marcel Hark, and Fabian Meyer. 2022. Improving Automatic Complexity Analysis of Integer Programs. In *The Logic of Software. A Tasting Menu of Formal Methods: Essays Dedicated to Reiner Hähnle on the Occasion of His 60th Birthday*, Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, and Einar Broch Johnsen (Eds.). Lecture Notes in Computer Science, Vol. 13360. Springer, Cham, 193–228. https://doi.org/10.1007/978-3-031-08166-8_10

[32] Samuel Jiménez Gil, Iain Bate, George Lima, Luca Santinelli, Adriana Gogonel, and Liliana Cucu-Grosjean. 2017. Open Challenges for Probabilistic Measurement-Based Worst-Case Execution Time. *IEEE Embedded Systems Letters* 9, 3 (2017), 69–72.

[33] Simon F. Goldsmith, Alex S. Aiken, and Daniel S. Wilkerson. 2007. Measuring Empirical Computational Complexity. In *Proc. the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*. ACM, New York, NY, USA, 395–404. https://doi.org/10.1145/1287624.1287681

[34] David Griffin and Alan Burns. 2010. Realism in Statistical Analysis of Worst Case Execution Times. In *10th International Workshop on Worst-Case Execution Time Analysis (OpenAccess Series in Informatics (OASIcs), Vol. 15)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Wadern, 44–53. https://doi.org/10.4230/OASIcs.WCET.2010.44

[35] Bernd Grobauer. 2001. Cost Recurrences for DML Programs. In *Proc. 6th ACM SIGPLAN International Conference on Functional Programming*. ACM, New York, NY, USA, 253–264. https://doi.org/10.1145/507635.507666

[36] Jessie Grosen, David M. Kahn, and Jan Hoffmann. 2023. Automatic Amortized Resource Analysis with Regular Recursive Types. In *Proc. 38th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, New York, NY, USA, 1–14. https://doi.org/10.1109/LICS56636.2023.10175720

[37] Sumit Gulwani, Krishna K. Mehra, and Trishul Chilimbi. 2009. SPEED: Precise and Efficient Static Estimation of Program Computational Complexity. In *Proc. 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 127–139. https://doi.org/10.1145/1480881.1480898

[38] Martin A. T. Handley, Niki Vazou, and Graham Hutton. 2019. Liquidate Your Assets: Reasoning about Resource Usage in Liquid Haskell. *Proc. ACM Program. Lang.* 4, POPL, Article 24 (Dec. 2019), 27 pages. https://doi.org/10.1145/3371092

[39] Jan Hoffmann. 2011. *Types with Potential: Polynomial Resource Bounds via Automatic Amortized Analysis*. Ph. D. Dissertation. Ludwig-Maximilians-Universität München. https://edoc.ub.uni-muenchen.de/13955/

[40] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. 2011. Multivariate Amortized Resource Analysis. In *Proc. 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 357–370. https://doi.org/10.1145/1926385.1926427

[41] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. 2012. Resource Aware ML. In *Proc. 24 International Conference on Computer Aided Verification (Lecture Notes in Computer Science, Vol. 7358)*, P. Madhusudan and Sanjit A. Seshia (Eds.). Springer, Berlin, 781–786. https://doi.org/10.1007/978-3-642-31424-7_64

[42] Jan Hoffmann, Ankush Das, and Shu-Chun Weng. 2017. Towards Automatic Resource Bound Analysis for OCaml. In *Proc. 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 359–373. https://doi.org/10.1145/3009837.3009842

[43] Jan Hoffmann and Martin Hofmann. 2010. Amortized Resource Analysis with Polymorphic Recursion and Partial Big-Step Operational Semantics. In *Proc. 8th Asian Symposium on Programming Languages and Systems (Lecture Notes in Computer Science, Vol. 6461)*, Kazunori Ueda (Ed.). Springer, Berlin, 172–187. https://doi.org/10.1007/978-3-642-17164-2_13

[44] Jan Hoffmann and Martin Hofmann. 2010. Amortized Resource Analysis with Polynomial Potential. In *Proc. 19th European Symposium on Programming Languages and Systems (Lecture Notes in Computer Science, Vol. 6012)*, Andrew D. Gordon (Ed.). Springer, Berlin, 287–306. https://doi.org/10.1007/978-3-642-11957-6_16

[45] Jan Hoffmann and Steffen Jost. 2022. Two Decades of Automatic Amortized Resource Analysis. *Mathematical Structures in Computer Science* 32, 6 (June 2022), 729–759. https://doi.org/10.1017/S0960129521000487

[46] Jan Hoffmann and Zhong Shao. 2015. Automatic Static Cost Analysis for Parallel Programs. In *Proc. 24th European Symposium on Programming Languages and Systems (Lecture Notes in Computer Science, Vol. 9032)*, Jan Vitek (Ed.). Springer, Berlin, 132–157. https://doi.org/10.1007/978-3-662-46669-8_6

[47] Martin Hofmann and Steffen Jost. 2003. Static Prediction of Heap Space Usage for First-Order Functional Programs. In *Proc. 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 185–197. https://doi.org/10.1145/604131.604148

[48] Martin Hofmann, Lorenz Leutgeb, David Obwaller, Georg Moser, and Florian Zuleger. 2022. Type-Based Analysis of Logarithmic Amortised Complexity. *Mathematical Structures in Computer Science* 32, 6 (June 2022), 794–826. https://doi.org/10.1017/S0960129521000232

[49] Martin Hofmann and Georg Moser. 2014. Amortised Resource Analysis and Typed Polynomial Interpretations. In *Rewriting and Typed Lambda Calculi (Lecture Notes in Computer Science, Vol. 8560)*. Springer, Heidelberg, 272–286. https://doi.org/10.1007/978-3-319-08918-8_19

[50] Ling Huang, Jinzhu Jia, Bin Yu, Byung-Gon Chun, Petros Maniatis, and Mayur Naik. 2010. Predicting Execution Time of Computer Programs Using Sparse Polynomial Regression. In *Advances in Neural Information Processing Systems*,

Vol. 23. Curran Associates Inc., Red Hook, NY, USA, 883–891.

[51] Didier Ishimwe, KimHao Nguyen, and ThanhVu Nguyen. 2021. Dynaplex: Analyzing Program Complexity Using Dynamically Inferred Recurrence Relations. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 138 (Oct. 2021), 23 pages. https://doi.org/10.1145/3485515

[52] David S. Johnson. 1999. A Theoretician's Guide to the Experimental Analysis of Algorithms. In *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 59)*, Michael H. Goldwasser, David S. Johnson, and Catherine C. McGeoch (Eds.). DIMACS/AMS, Piscataway, NJ, USA, 215–250. https://doi.org/10.1090/dimacs/059/11

[53] Steffen Jost, Kevin Hammond, Hans-Wolfgang Loidl, and Martin Hofmann. 2010. Static Determination of Quantitative Resource Usage for Higher-Order Programs. *Proc. 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* 45, 1 (Jan. 2010), 223–236. https://doi.org/10.1145/1706299.1706327

[54] David M. Kahn and Jan Hoffmann. 2020. Exponential Automatic Amortized Resource Analysis. In *Proc. 23rd International Conference on Foundations of Software Science and Computation Structures (Lecture Notes in Computer Science, Vol. 12077)*, Jean Goubault-Larrecq and Barbara König (Eds.). Springer, Cham, 359–380. https://doi.org/10.1007/978-3-030-45231-5_19

[55] G. A. Kavvos, Edward Morehouse, Daniel R. Licata, and Norman Danner. 2019. Recurrence Extraction for Functional Programs through Call-by-Push-Value. *Proc. ACM Program. Lang.* 4, POPL, Article 15 (Dec. 2019), 31 pages. https://doi.org/10.1145/3371083

[56] Zachary Kincaid, John Cyphert, Jason Breck, and Thomas Reps. 2017. Non-Linear Reasoning for Invariant Synthesis. *Proc. ACM Program. Lang.* 2, POPL, Article 54 (Dec. 2017), 33 pages. https://doi.org/10.1145/3158142

[57] Tristan Knoth, Di Wang, Adam Reynolds, Jan Hoffmann, and Nadia Polikarpova. 2020. Liquid Resource Types. *Proc. ACM Program. Lang.* 4, ICFP, Article 106 (Aug. 2020), 29 pages. https://doi.org/10.1145/3408988

[58] Yunbum Kook, YinTat Lee, Ruoqi Shen, and Santosh Vempala. 2022. Sampling with Riemannian Hamiltonian Monte Carlo in a Constrained Space. In *Advances in Neural Information Processing Systems*, Vol. 35. Curran Associates Inc., Red Hook, NY, USA, 13 pages.

[59] Ugo Dal Lago and Marco Gaboardi. 2011. Linear Dependent Types and Relative Completeness. In *Proc. 26th IEEE Symposium on Logic in Computer Science*. IEEE, New York, NY, USA, 133–142. https://doi.org/10.1109/LICS.2011.22

[60] Lorenz Leutgeb, Georg Moser, and Florian Zuleger. 2021. ATLAS: Automated Amortised Complexity Analysis of Self-adjusting Data Structures. In *Proc. 33rd International Conference on Computer Aided Verification (Lecture Notes in Computer Science, Vol. 12760)*. Springer-Verlag, Berlin, 99–122. https://doi.org/10.1007/978-3-030-81688-9_5

[61] Lorenz Leutgeb, Georg Moser, and Florian Zuleger. 2022. Automated Expected Amortised Cost Analysis of Probabilistic Data Structures. In *Proc. 34th International Conference on Computer Aided Verification (Lecture Notes in Computer Science, Vol. 13372)*, Sharon Shoham and Yakir Vizel (Eds.). Springer, Cham, 70–91. https://doi.org/10.1007/978-3-031-13188-2_4

[62] Benjamin Lichtman and Jan Hoffmann. 2017. Arrays and References in Resource Aware ML. In *Proc. 2nd International Conference on Formal Structures for Computation and Deduction (Leibniz International Proceedings in Informatics, Vol. 84)*, Dale Miller (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 20 pages. https://doi.org/10.4230/LIPIcs.FSCD.2017.26

[63] George Lima, Dario Dias, and Edna Barros. 2016. Extreme Value Theory for Estimating Task Execution Time Bounds: A Careful Look. In *Proc. 28th Euromicro Conference on Real-Time Systems*. IEEE, New York, NY, USA, 200–211. https://doi.org/10.1109/ECRTS.2016.20

[64] Yue Lu, Thomas Nolte, Iain Bate, and Liliana Cucu-Grosjean. 2011. A New Way About Using Statistical Analysis of Worst-Case Execution Times. *ACM SIGBED Review* 8, 3 (2011), 11–14. https://doi.org/10.1145/2038617.2038619

[65] Catherine McGeoch. 1987. *Experimental Analysis of Algorithms.* Ph. D. Dissertation. Carnegie Mellon University.

[66] Catherine McGeoch, Peter Sanders, Rudolf Fleischer, Paul R. Cohen, and Doina Precup. 2002. Using Finite Experiments to Study Asymptotic Performance. In *Experimental Algorithmics: From Algorithm Design to Robust and Efficient Software*, Rudolf Fleischer, Bernard Moret, and Erik Meineche Schmidt (Eds.). Springer, Berlin, 93–126. https://doi.org/10.1007/3-540-36383-1_5

[67] Glen Mével, Jacques-Henri Jourdan, and François Pottier. 2019. Time Credits and Time Receipts in Iris. In *Proc. 28th European Symposium on Programming (Lecture Notes in Computer Science, Vol. 11423)*, Luís Caires (Ed.). Springer, Cham, 3–29. https://doi.org/10.1007/978-3-030-17184-1_1

[68] Suzana Milutinovic, Enrico Mezzetti, Jaume Abella, Tullio Vardanega, and Francisco J. Cazorla. 2017. On Uses of Extreme Value Theory Fit for Industrial-Quality WCET Analysis. In *Proc. 12th IEEE International Symposium on Industrial Embedded Systems*. IEEE, New York, NY, USA, 1–6. https://doi.org/10.1109/SIES.2017.7993402

[69] Hadi Mohasel Afshar and Justin Domke. 2015. Reflection, Refraction, and Hamiltonian Monte Carlo. In *Advances in Neural Information Processing Systems*, Vol. 28. MIT Press, Cambridge, MA, USA, 3007–3015.

[70] Alexandre Moine, Arthur Charguéraud, and François Pottier. 2023. A High-Level Separation Logic for Heap Space under Garbage Collection. *Proc. ACM Program. Lang.* 7, POPL, Article 25 (Jan. 2023), 30 pages. https://doi.org/10.1145/3571218

[71] Bernard M. E. Moret. 1999. Towards a Discipline of Experimental Algorithmics. In *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 59)*, Michael H. Goldwasser, David S. Johnson, and Catherine C. McGeoch (Eds.). DIMACS/AMS, Piscataway, NJ, USA, 197–213. https://doi.org/10.1090/dimacs/059/10

[72] Georg Moser and Manuel Schneckenreither. 2020. Automated Amortised Resource Analysis for Term Rewrite Systems. *Science of Computer Programming* 185, Article 102306 (Jan. 2020), 18 pages. https://doi.org/10.1016/j.scico.2019.102306

[73] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded Expectations: Resource Analysis for Probabilistic Programs. In *Proc. 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, USA, 496–512. https://doi.org/10.1145/3192366.3192394

[74] Tobias Nipkow and Hauke Brinkop. 2019. Amortized Complexity Verified. *Journal of Automated Reasoning* 62, 3 (March 2019), 367–391. https://doi.org/10.1007/s10817-018-9459-3

[75] Yue Niu, Jonathan Sterling, Harrison Grodin, and Robert Harper. 2022. A Cost-Aware Logical Framework. *Proc. ACM Program. Lang.* 6, POPL (Jan. 2022), 1–31. https://doi.org/10.1145/3498670

[76] Long Pham, Feras Saad, and Jan Hoffmann. 2024. Hybrid Resource-Aware ML. https://doi.org/10.5281/zenodo.10937074

[77] Boqin Qin, Tengfei Tu, Ziheng Liu, Tingting Yu, and Linhai Song. 2022. Algorithmic Profiling for Real-World Complexity Problems. *IEEE Trans. Softw. Eng.* 48, 7 (July 2022), 2680–2694. https://doi.org/10.1109/TSE.2021.3067652

[78] Federico Reghenzani, Luca Santinelli, and William Fornaciari. 2020. Dealing with Uncertainty in pWCET Estimations. *ACM Transactions on Embedded Computing Systems* 19, 5, Article 33 (Sept. 2020), 23 pages. https://doi.org/10.1145/3396234

[79] Stefano Rizzelli, Judith Rousseau, and Sonia Petrone. 2024. Empirical Bayes in Bayesian Learning: Understanding a Common Practice. arXiv:2402.19036 [math]

[80] Daniele Rogora, Antonio Carzaniga, Amer Diwan, Matthias Hauswirth, and Robert Soulé. 2020. Analyzing System Performance with Probabilistic Performance Annotations. In *Proc. 15th European Conference on Computer Systems*. ACM, New York, NY, USA, 1–14. https://doi.org/10.1145/3342195.3387554

[81] Peter Sanders and Rudolf Fleischer. 2001. Asymptotic Complexity from Experiments? A Case Study for Randomized Algorithms. In *International Workshop on Algorithm Engineering (Lecture Notes in Computer Science, Vol. 1982)*, Stefan Näher and Dorothea Wagner (Eds.). Springer, Berlin, 135–146. https://doi.org/10.1007/3-540-44691-5_12

[82] Moritz Sinn, Florian Zuleger, and Helmut Veith. 2014. A Simple and Scalable Static Analysis for Bound Analysis and Amortized Complexity Analysis. In *Proc. 26th International Conference on Computer Aided Verification (Lecture Notes in Computer Science, Vol. 8559)*, Armin Biere and Roderick Bloem (Eds.). Springer, Cham, 745–761. https://doi.org/10.1007/978-3-319-08867-9_50

[83] Daniel D. Sleator and Robert E. Tarjan. 1983. Self-Adjusting Binary Trees. In *Proc. 15th Annual ACM Symposium on Theory of Computing*. ACM, New York, NY, USA, 235–245. https://doi.org/10.1145/800061.808752

[84] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM* 28, 2 (Feb. 1985), 202–208. https://doi.org/10.1145/2786.2793

[85] Robert E. Tarjan. 1985. Amortized Computational Complexity. *SIAM J. Matrix Anal. Appl.* 6, 2 (April 1985), 306–13. https://doi.org/10.1137/0606031

[86] Pedro B. Vasconcelos. 2008. *Space Cost Analysis Using Sized Types*. Ph. D. Dissertation. University of St. Andrews.

[87] Di Wang, David M. Kahn, and Jan Hoffmann. 2020. Raising Expectations: Automating Expected Cost Analysis with Types. *Proc. ACM Program. Lang.* 4, ICFP (Aug. 2020), 31 pages. https://doi.org/10.1145/3408992

[88] Ben Wegbreit. 1975. Mechanical Program Analysis. *Commun. ACM* 18, 9 (Sept. 1975), 528–539. https://doi.org/10.1145/361002.361016

[89] Reinhard Wilhelm et al. 2008. The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems* 7, 3, Article 36 (May 2008), 53 pages. https://doi.org/10.1145/1347375.1347389

[90] Dmitrijs Zaparanuks and Matthias Hauswirth. 2012. Algorithmic Profiling. In *Proc. 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, USA, 67–76. https://doi.org/10.1145/2254064.2254074

[91] Florian Zuleger, Sumit Gulwani, Moritz Sinn, and Helmut Veith. 2011. Bound Analysis of Imperative Programs with the Size-Change Abstraction. In *Proc. 18th International Static Analysis Symposium (Lecture Notes in Computer Science, Vol. 6887)*, Eran Yahav (Ed.). Springer, Berlin, 280–297. https://doi.org/10.1007/978-3-642-23702-7_22

A SUPPLEMENTARY MATERIAL OF UNIVARIATE AARA

Listing 3 presents the syntax-directed rules of univariate AARA and Listing 4 presents the structural rules. In a program $\mathcal{P}$, for each function symbol $f \in F$ with function definition $f(x)$=e, we assume that $\mathcal{T}_f$ is the set of valid resource-annotated arrow types $\langle a_1, p_1\rangle \to \langle a_1, p_1\rangle$ such that $x : a_1; p_1 \vdash e : \langle a_2, p_2\rangle$ holds. Listing 5 defines the sharing relation for resource annotations.

$$\text{U:Var}\ \frac{}{x : a; 0 \vdash x : \langle a, 0\rangle} \qquad \text{U:Unit}\ \frac{}{\cdot; 0 \vdash \langle\,\rangle : \langle \mathtt{unit}, 0\rangle} \qquad \text{U:SumL}\ \frac{\Gamma; 0 \vdash x : \langle a_1, q_1\rangle}{\Gamma; 0 \vdash \mathsf{left} \cdot x : \langle \langle a_1, q_1\rangle + \langle a_2, q_2\rangle, 0\rangle}$$

$$\text{U:SumR}\ \frac{\Gamma; 0 \vdash x : \langle a_2, q_2\rangle}{\Gamma; 0 \vdash \mathsf{right} \cdot x : \langle \langle a_1, q_1\rangle + \langle a_2, q_2\rangle, 0\rangle} \qquad \text{U:Prod}\ \frac{}{x_1 : a_1, x_2 : a_2; 0 \vdash \langle x_1, x_2\rangle : \langle a_1 \times a_2, 0\rangle}$$

$$\text{U:Nil}\ \frac{}{\cdot; 0 \vdash [\,] : \langle L^{\vec{q}}(a), 0\rangle} \qquad \text{U:Cons}\ \frac{\vec{q} = (q_1, \ldots, q_d)}{x_1 : a, x_2 : L^{\triangleleft(\vec{q})}(a); q_1 \vdash x_1 :: x_2 : \langle L^{\vec{q}}(a), 0\rangle} \qquad \text{U:App}\ \frac{(\langle a_1, p_1\rangle \to \langle a_2, p_2\rangle) \in \mathcal{T}}{f : \mathcal{T}_f, x : a_1; p_1 \vdash f\ x : \langle a_2, p_2\rangle}$$

$$\text{U:CaseSum}\ \frac{\Gamma, x_1 : a_1; p_1 \vdash e_1 : \langle a, q\rangle \qquad \Gamma, x_2 : a_2; p_2 \vdash e_2 : \langle a, q\rangle}{x : \langle a_1, p_1\rangle + \langle a_2, p_2\rangle, \Gamma; 0 \vdash \mathsf{case}\ x\ \{\mathsf{left} \cdot x_1 \hookrightarrow e_1 \mid \mathsf{right} \cdot x_2 \hookrightarrow e_2\} : \langle a, q\rangle}$$

$$\text{U:CaseProd}\ \frac{\Gamma, x_1 : a_1, x_2 : a_2; p \vdash e : \langle a, q\rangle}{x : a_1 \times a_2, \Gamma; p \vdash \mathsf{case}\ x\ \{\langle x_1, x_2\rangle \hookrightarrow e\} : \langle a, q\rangle}$$

$$\text{U:CaseList}\ \frac{\Gamma; q \vdash e_1 : \langle a, q\rangle \qquad \Gamma, x_1 : a, x_2 : L^{\triangleleft(\vec{p})}(a); q + p_1 \vdash e_2 : \langle a, q\rangle}{x : L^{\vec{p}}(a), \Gamma; q \vdash \mathsf{case}\ x\ \{[\,] \hookrightarrow e_1 \mid (x_1 :: x_2) \hookrightarrow e_2\} : \langle a, q\rangle}$$

$$\text{U:Let}\ \frac{\Gamma_1; p_1 \vdash e_1 : \langle a_1, p_2\rangle \qquad \Gamma_2, x : a_1; p_2 \vdash e_2 : \langle a_2, p_3\rangle}{\Gamma_1, \Gamma_2; p_1 \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \langle a_2, p_3\rangle} \qquad \text{U:Share}\ \frac{a \curlyvee (a_1, a_2) \qquad \Gamma, x_1 : a_1, x_2 : a_2; q \vdash e : \langle a, q\rangle}{\Gamma, x : a; q \vdash \mathsf{share}\ x\ \mathsf{as}\ x_1, x_2\ \mathsf{in}\ e : \langle a, q\rangle}$$

Lst. 3. Syntax-directed rules of univariate AARA.

$$\text{U:Sub}\ \frac{\Gamma; q \vdash e : \langle a, p\rangle \qquad a <: a'}{\Gamma; q \vdash e : \langle a', p\rangle} \qquad \text{U:Sup}\ \frac{\Gamma, x : a; q \vdash e : B \qquad a' <: a}{\Gamma, x : a'; q \vdash e : B} \qquad \text{U:Weak}\ \frac{\Gamma_1; q \vdash e : B}{\Gamma_1, \Gamma_2; q \vdash e : B}$$

$$\text{U:Re;ax}\ \frac{\Gamma; p_1 \vdash e : \langle a, p_2\rangle \qquad q_1 \geq p_1 \qquad q_1 - q_2 \geq p_1 - p_2}{\Gamma; q_1 \vdash e : \langle a, q_2\rangle}$$

Lst. 4. Structural rules of univariate AARA. B denotes $\langle a, p\rangle$.

$$\mathtt{unit} \curlyvee (\mathtt{unit}, \mathtt{unit}) \qquad \frac{a_1 \curlyvee (a_{1,1}, a_{1,2}) \qquad a_2 \curlyvee (a_{2,1}, a_{2,2})}{(a_1 + a_2) \curlyvee (a_{1,1} + a_{2,1}, a_{1,2} + a_{2,2})} \qquad \frac{a_1 \curlyvee (a_{1,1}, a_{1,2}) \qquad a_2 \curlyvee (a_{2,1}, a_{2,2})}{(a_1 \times a_2) \curlyvee (a_{1,1} \times a_{2,1}, a_{1,2} \times a_{2,2})}$$

$$\frac{\vec{p} = \vec{p_1} + \vec{p_2} \qquad a \curlyvee (a_1, a_2)}{L^{\vec{p}}(a) \curlyvee (L^{\vec{p_1}}(a_1), L^{\vec{p_2}}(a_2))} \qquad \frac{(B_1 \to B_2), (C_1 \to C_2), (D_1 \to D_2) \in \mathcal{T}_f}{(B_1 \to B_2) \curlyvee (C_1 \to C_1, D_1 \to D_2)}$$

Lst. 5. Definition of the sharing relation $a \curlyvee (a_1, a_2)$.

B MODEL HYPERPARAMETERS IN BAYESWC AND BAYESPC

This section describes an empirical Bayes approach [14] for automatically determining the hyperparameters of the probabilistic models in BayesWC (Section 5.2) and BayesPC (Section 5.3). Empirical Bayes is a widely used approach in Bayesian data analysis in which the observed data is used to infer plausible values of model hyperparameters: see Rizzelli et al. [79] for a recent survey. This approach applies directly to most benchmarks: only one benchmark uses a hyperparameter that deviates from the automatic procedure we have developed.

B.1 BayesWC

Recall from Eq. (5.12) that BayesWC uses a probabilistic model

$$\beta_0, \beta_1, \sigma \overset{\text{iid}}{\sim} \text{Normal}(0, \gamma_0) \qquad \epsilon_i \sim \text{Gumbel}(0, 1) \tag{B.1}$$

$$y_i = \beta_0 + \beta_1 |v_i| + |\sigma| \epsilon_i \qquad c_i = \exp(y_i), \tag{B.2}$$

where g_{noise} in Eq. (5.12) has been set to a standard Gumbel. The only hyperparameter in this model is γ_0, which is set to $\gamma_0 \coloneqq 5.0$ for all benchmarks. Coupled with the exp component in the likelihood model for the costs c_i, this choice determines a broad prior distribution over observed datasets.

B.2 BayesPC

Recall from Eq. (5.14) that, in the probabilistic model for BayesPC, resource coefficients are drawn from a truncated normal distribution:

$$(p_j)_{j=0}^{|\vec{p}|}, (q_j)_{j=0}^{|\vec{q}|} \overset{\text{iid}}{\sim} \text{Normal}_{\geq 0}(0, \gamma_0), \tag{B.3}$$

where $\gamma_0 \in \mathbb{R}_{>0}$ is the scale hyperparameter (i.e., standard deviation).

In our prototype implementation, the hyperparameter γ_0 for each benchmark is determined from data as follows. We first perform (Data-Driven or Hybrid) Opt to infer a resource-annotated typing judgment for the entire program $f\ x$:

$$\Gamma, p_0 \vdash_{\mathcal{D}} f\ x : \langle a, q_0 \rangle, \tag{B.4}$$

where Γ is a resource-annotated typing context, $p_0 \in \mathbb{Q}_{\geq 0}$ is constant potential for the input, a is a resource-annotated output type, and $q_0 \in \mathbb{Q}_{\geq 0}$ is constant potential for the output. Let d be the user-specified maximum polynomial degree for polynomial cost bounds, and let $p_1, \ldots, p_D$ be the resource coefficients inside Γ that correspond to indices of degree d. In other words, $p_1, \ldots, p_D$ are the polynomial coefficients of the highest-degree terms inside the polynomial potential function. The degree-0 coefficient p_0 for constant potential in the typing context is never in the set $\{p_1, \ldots, p_D\}$, unless the user specifies the maximum polynomial degree of 0. For all benchmarks in our evaluation, we set the hyperparameter γ_0 of the prior distribution for resource coefficients in BayesPC as

$$\gamma_0 \coloneqq \frac{8}{15} \max\{p_1, \ldots, p_D\} + \frac{4}{5}. \tag{B.5}$$

According to Eq. (5.16), the cost gap ϵ_i follows a truncated Weibull distribution:

$$c'_{|v_i|} - c_i =: \epsilon_i \sim \text{Weibull}_{[0, c'_{|v_i|}]}(\theta_0, \theta_1), \tag{B.6}$$

where $\theta_0 \in \mathbb{R}_{\geq 0}$ is the shape hyperparameter, $\theta_1 \in \mathbb{R}_{>0}$ is the scale hyperparameter, and $[0, c'_{|v_i|}]$ is the interval of truncation.

The two hyperparameters θ_0 and θ_1 are determined as follows. The shape θ_0 ranges from 1.0 to 1.5 across benchmarks. Specifically, we set $\theta_0 \coloneqq 1.25$ in Data-Driven BayesPC of `MapAppend` and Hybrid BayesPC of `ZAlgorithm`. We set $\theta_0 \coloneqq 1.5$ in Data-Driven BayesPC of `BubbleSort`,

Data-driven and Hybrid BayesPC of `Concat`, and Data-Driven BayesPC of `ZAlgorithm`. In all the remaining cases of BayesPC, we set $\theta_0 \coloneqq 1.0$.

The scale hyperparameter θ_1 is set as follows. We first perform (Data-Driven or Hybrid) Opt to obtain a resource annotated typing judgment around an annotated code fragment $\mathrm{stat}_\ell\ e$:

$$\Gamma, p_0 \vdash_{\mathcal{D}} \mathrm{stat}_\ell\ e : \langle a, q_0 \rangle, \tag{B.7}$$

where Γ is a resource-annotated typing context, $p_0 \in \mathbb{Q}_{\geq 0}$ is constant potential for the input, a is a resource-annotated output type, and $q_0 \in \mathbb{Q}_{\geq 0}$ is constant potential for the output. We then calculate the cost gap ϵ_i of $\mathrm{stat}_\ell\ e$:

$$\epsilon_i \coloneqq p_0 + \Phi(V_i^\ell : \Gamma) - q_0 - \Phi(v_i^\ell : a) \qquad (i = 1, \ldots, |\mathcal{D}_\ell|). \tag{B.8}$$

Let ε_α be the $\alpha \coloneqq 90^{\text{th}}$ percentile of these cost gaps $\epsilon_1, \ldots, \epsilon_{|\mathcal{D}_\ell|}$. The scale θ_1 is then

$$\theta_1 \coloneqq \frac{1100}{188.7}\varepsilon_\alpha + 100. \tag{B.9}$$

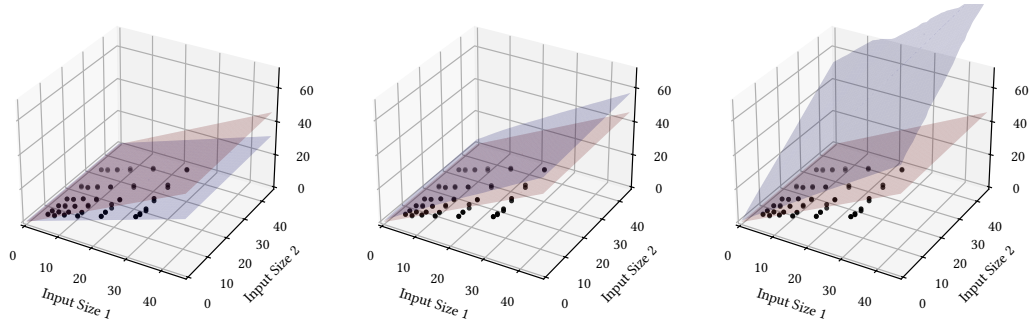
For Hybrid BayesPC of `MedianOfMedians`, the rule (B.9) suggests $\theta_1 \coloneqq 841.128$. However, this hyperparameter does not yield a posterior distribution whose median is close to the ground-truth cost bound of `MedianOfMedians` due to the inaccuracy of Opt. Therefore, Hybrid BayesPC analysis of `MedianOfMedians` is an exception, where we set $\theta_1 \coloneqq 41.128$ instead.

C FULL EXPERIMENT RESULTS

This appendix contains the full evaluation results on all 10 benchmark programs described in Section 7 of the main paper.

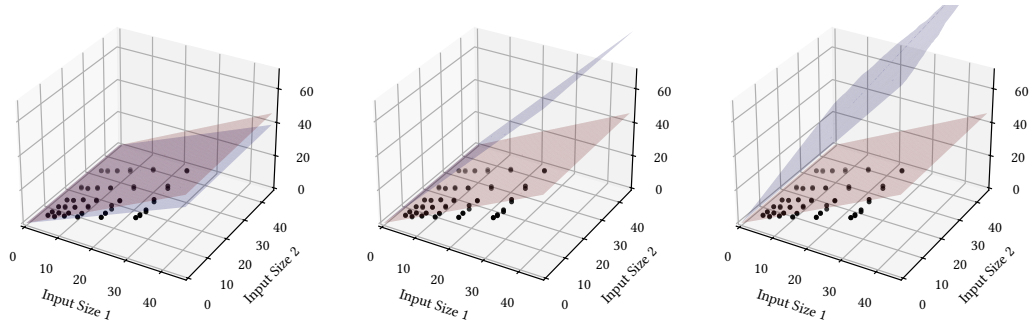
Table 2. Estimation gaps of inferred cost bounds for MapAppend benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
Input Size	Percentile **Method**	5^{th}	50^{th}	95^{th}	5^{th}	50^{th}	95^{th}
(10, 10)	Opt	-0.26	-0.26	-0.26	-0.15	-0.15	-0.15
	BayesWC	0.03	0.41	1.64	0.53	1.03	2.27
	BayesPC	0.85	1.62	2.61	1.18	1.92	2.91
(100, 100)	Opt	-0.32	-0.32	-0.32	-0.15	-0.15	-0.15
	BayesWC	-0.18	0.22	1.17	0.53	1.03	2.27
	BayesPC	0.75	1.54	2.53	1.12	1.89	2.89
(1000, 1000)	Opt	-0.32	-0.32	-0.32	-0.15	-0.15	-0.15
	BayesWC	-0.22	0.20	1.15	0.53	1.03	2.27
	BayesPC	0.74	1.54	2.52	1.11	1.88	2.89



(a) Opt (b) BayesWC (c) BayesPC

Fig. 8. MapAppend Data-Driven



(a) Opt (b) BayesWC (c) BayesPC

Fig. 9. MapAppend Hybrid

```
let incur_cost (hd : int) =
  let modulo = 5 in
  if (hd mod 100) = 0 then Raml.tick 1.0
  else (
    if (hd mod modulo) = 1 then
      Raml.tick 0.85
    else (
      if (hd mod modulo) = 2 then
        Raml.tick 0.65
      else Raml.tick 0.5))

let complex_function (hd : int) =
  let _ = incur_cost hd in
  if hd < 42 then hd / 2 else hd * 2

let rec map_append (xs : int list) (ys : int
     list) =
    match xs with
    | [] → ys
    | hd :: tl →
       let hd_new = complex_function hd in
       hd_new :: (map_append tl ys)

let map_append2 (xs : int list) (ys : int list)
     = Raml.stat (map_append xs ys)
```

(a) Fully data-driven resource analysis.

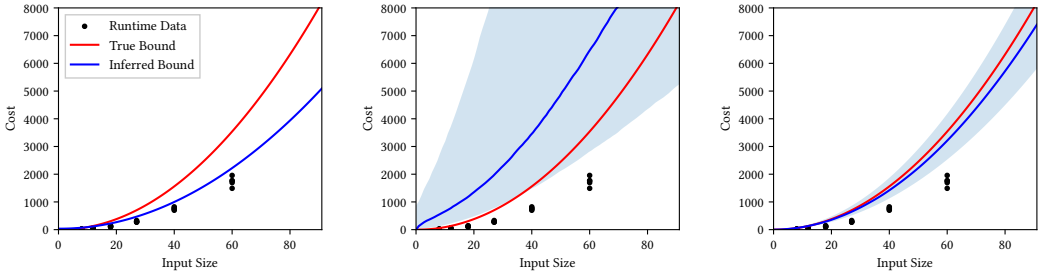
```
let step_function (x : int) (xs : int list) (ys
     : int list) =
  let x_new = complex_function x in (x_new, xs,
       ys)

let rec map_append (xs : int list)
  (ys : int list) =
  match xs with
  | [] → ys
  | hd :: tl →
    let hd_new, rec_xs, rec_ys = Raml.stat (
         step_function hd tl ys) in
    hd_new :: map_append rec_xs rec_ys
```

(b) Hybrid resource analysis.

Lst. 6. Source code of MapAppend. The function `complex_function` is a computation that conventional AARA cannot statically analyze; e.g., if < is OCaml's built-in polymorphic comparison. Conventional AARA fails to infer any polynomial cost bounds for the `map_append` function. (a) Fully data-driven resource analysis. (b) Hybrid resource analysis. We perform data-driven analysis on `step_function`.

Table 3. Estimation gaps of inferred cost bounds for `BubbleSort` benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
Input Size	Percentile **Method**	5^{th}	50^{th}	95^{th}	5^{th}	50^{th}	95^{th}
10	Opt	0.01	0.01	0.01	∅	∅	∅
	BayesWC	0.44	6.29	60.73	∅	∅	∅
	BayesPC	-0.31	0.02	0.39	∅	∅	∅
100	Opt	-0.38	-0.38	-0.38	∅	∅	∅
	BayesWC	-0.48	0.41	8.34	∅	∅	∅
	BayesPC	-0.34	-0.10	0.17	∅	∅	∅
1000	Opt	-0.38	-0.38	-0.38	∅	∅	∅
	BayesWC	-0.93	-0.22	5.31	∅	∅	∅
	BayesPC	-0.35	-0.10	0.15	∅	∅	∅



(a) Opt (b) BayesWC (c) BayesPC

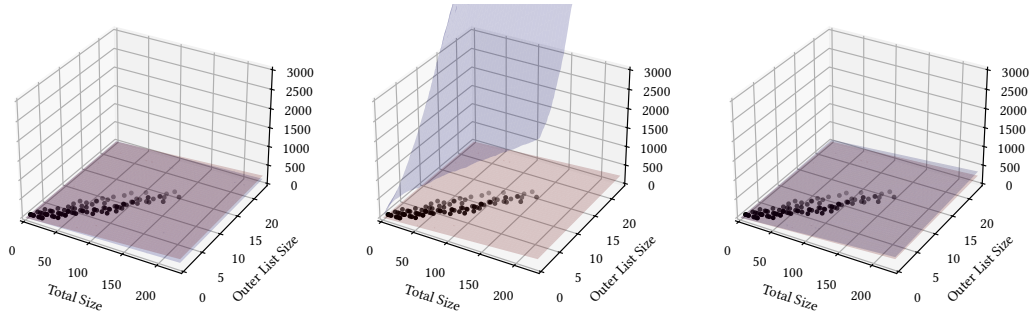
Fig. 10. `BubbleSort` Data-Driven

```
let incur_cost (hd : int) =
  if (hd mod 10) = 0 then Raml.tick 1.0 else Raml.tick 0.5

let rec scan_and_swap (xs : int list) =
  match xs with
  | [] → ([], false)
  | [ x ] → ([ x ], false)
  | x1 :: x2 :: tl →
      let _ = incur_cost x1 in
      if x1 <= x2 then
        let recursive_result, is_swapped = scan_and_swap (x2 :: tl) in
        (x1 :: recursive_result, is_swapped)
      else
        let recursive_result, _ = scan_and_swap (x1 :: tl) in
        (x2 :: recursive_result, true)

let rec bubble_sort (xs : int list) =
  let xs_scanned, is_swapped = scan_and_swap xs in
  if is_swapped then bubble_sort xs_scanned else xs_scanned

let bubble_sort2 (xs : int list) = Raml.stat (bubble_sort xs)
```

Lst. 7. Source code of BubbleSort for fully data-driven analysis. Conventional AARA cannot infer any polynomial cost bound as it fails to bound the number of recursive calls.

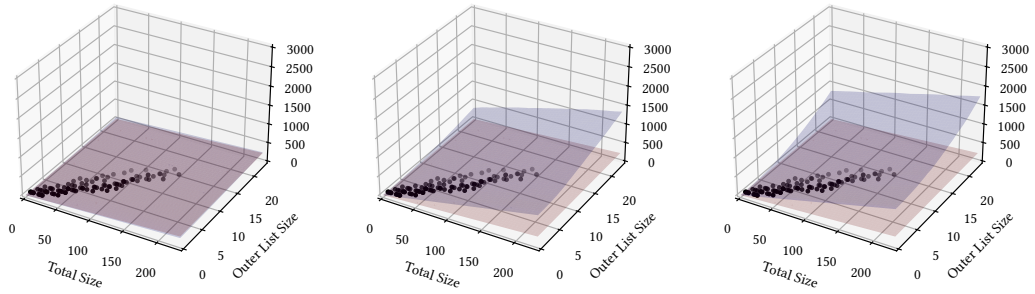
Table 4. Estimation gaps of inferred cost bounds for Concat benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
Input Size	Percentile **Method**	5^{th}	50^{th}	95^{th}	5^{th}	50^{th}	95^{th}
(50, 10)	Opt	-0.33	-0.33	-0.33	0.03	0.03	0.03
	BayesWC	14.05	66.64	744.65	1.74	4.80	19.86
	BayesPC	0.37	0.60	0.90	4.46	5.90	7.19
(500, 100)	Opt	-0.10	-0.10	-0.10	2.07	2.07	2.07
	BayesWC	12.54	183.95	3329.73	2.10	13.59	130.41
	BayesPC	0.50	1.25	4.50	16.22	32.27	47.71
(5000, 1000)	Opt	2.83	2.83	2.83	22.44	22.44	22.44
	BayesWC	11.04	931.52	32459.92	2.33	97.00	1309.28
	BayesPC	1.06	7.84	42.44	132.48	298.20	456.99



(a) Opt (b) BayesWC (c) BayesPC

Fig. 11. Concat Data-Driven



(a) Opt (b) BayesWC (c) BayesPC

Fig. 12. Concat Hybrid

```
let incur_cost (hd : int) =
  if (hd mod 5) = 0 then Raml.tick 1.0 else Raml.
       tick 0.5

let complex_function (hd : int) =
  let _ = incur_cost hd in
  if hd < 42 then hd / 2 else hd * 2

let rec map_append (xs : int list) (ys : int
     list) =
  match xs with
  | [] → ys
  | hd :: tl →
      let hd_new = complex_function hd in
      hd_new :: map_append tl ys

let rec concat (xss : int list list) =
  match xss with [] →[] | hd :: tl →map_append
       hd (concat tl)

let concat2 (xss : int list list) = Raml.stat (
     concat xss)
```

(a) Fully data-driven resource analysis.

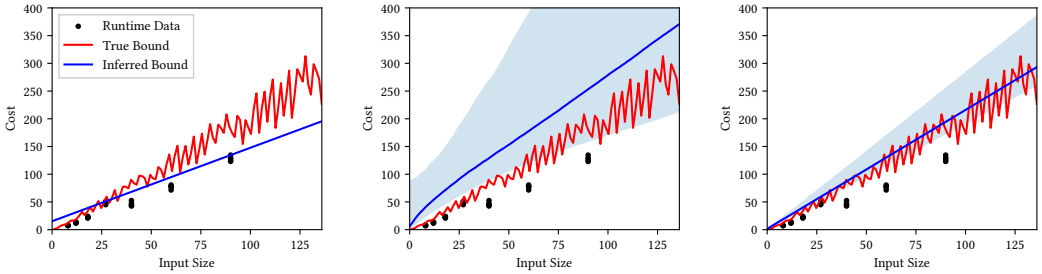
```
let rec concat (xss : int list list) =
  match xss with
  | [] → []
  | hd :: tl →
      let rec_tl = concat tl in
      Raml.stat (map_append hd rec_tl)
```

(b) Hybrid resource analysis.

Lst. 8. Source code of Concat. The function complex_function is a computation that conventional AARA cannot statically analyze; e.g., if < is OCaml's built-in polymorphic comparison. Conventional AARA fails to infer any polynomial cost bounds for the concat function. (a) Fully data-driven resource analysis. (b) Hybrid resource analysis. We perform data-driven analysis on map_append.

Table 5. Estimation gaps of inferred cost bounds for `EvenOddTail` benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
Input Size	Percentile **Method**	5th	50th	95th	5th	50th	95th
10	Opt	0.73	0.73	0.73	∅	∅	∅
	BayesWC	0.53	1.88	9.15	∅	∅	∅
	BayesPC	0.17	0.38	1.00	∅	∅	∅
100	Opt	-0.14	-0.14	-0.14	∅	∅	∅
	BayesWC	-0.08	0.62	3.80	∅	∅	∅
	BayesPC	0.10	0.25	0.90	∅	∅	∅
1000	Opt	-0.21	-0.21	-0.21	∅	∅	∅
	BayesWC	-0.62	0.52	3.75	∅	∅	∅
	BayesPC	0.11	0.27	0.92	∅	∅	∅



(a) Opt (b) BayesWC (c) BayesPC

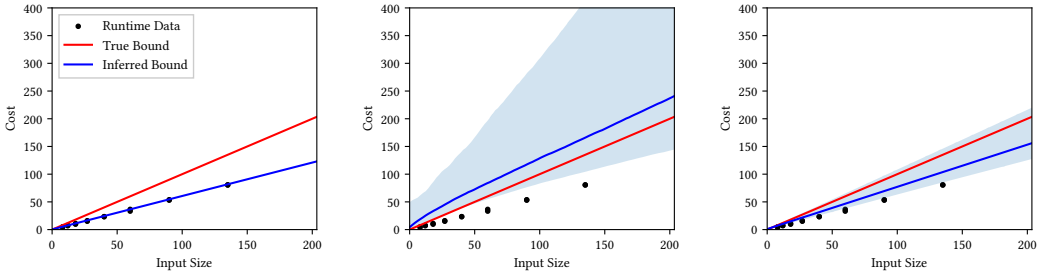
Fig. 13. `EvenOddTail` Data-Driven

```
exception Invalid_input

let incur_cost (hd : int) =
  if (hd mod 10) = 0 then Raml.tick 1.0 else Raml.tick 0.5

let rec linear_traversal (xs : int list) =
  match xs with
  | [] → []
  | hd :: tl →
      let _ = incur_cost hd in
      hd :: linear_traversal tl

let rec is_even (xs : int list) =
  match xs with [] →true | [ x ] →false | x1 :: x2 :: tl →is_even tl

let tail (xs : int list) =
  match xs with [] →raise Invalid_input | hd :: tl →tl

let rec split (xs : int list) =
  match xs with
  | [] → []
  | [ x ] → raise Invalid_input
  | x1 :: x2 :: tl →x1 :: split tl

let rec even_split_odd_tail (xs : int list) : int list =
  let xs_traversed = linear_traversal xs in
  match xs_traversed with
  | [] → []
  | hd :: tl →
      let xs_is_even = is_even xs_traversed in
      if xs_is_even then
        let split_result = split xs_traversed in
        even_split_odd_tail split_result
      else
        let tail_result = tail xs_traversed in
        even_split_odd_tail tail_result

let even_split_odd_tail2 (xs : int list) : int list =
  Raml.stat (even_split_odd_tail xs)
```

Lst. 9. Source code of `EvenOddTail` for fully data-driven resource analysis. Conventional AARA can only infer a quadratic cost bound for `EvenOddTail`, but not the true linear cost bound.

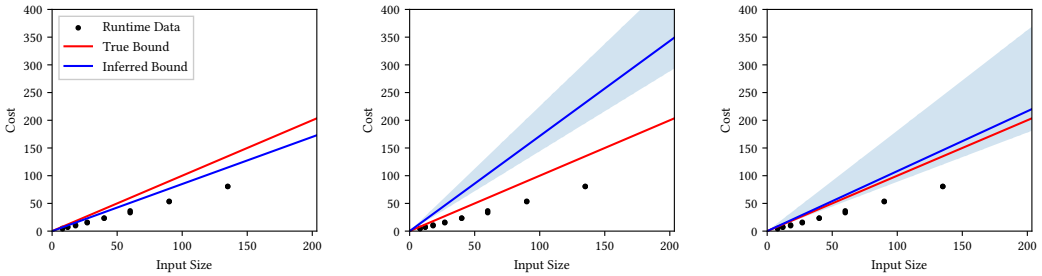
Table 6. Estimation gaps of inferred cost bounds for `InsertionSort2` benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
Input Size	Percentile **Method**	5^{th}	50^{th}	95^{th}	5^{th}	50^{th}	95^{th}
10	Opt	-0.37	-0.37	-0.37	-0.15	-0.15	-0.15
	BayesWC	0.05	1.17	8.68	0.39	0.72	1.47
	BayesPC	-0.33	-0.12	0.35	-0.14	0.08	0.84
100	Opt	-0.39	-0.39	-0.39	-0.15	-0.15	-0.15
	BayesWC	-0.23	0.29	3.58	0.39	0.72	1.47
	BayesPC	-0.39	-0.23	0.26	-0.14	0.08	0.84
1000	Opt	-0.40	-0.40	-0.40	-0.15	-0.15	-0.15
	BayesWC	-0.57	0.14	3.33	0.39	0.72	1.47
	BayesPC	-0.40	-0.24	0.25	-0.14	0.08	0.84



(a) Opt (b) BayesWC (c) BayesPC

Fig. 14. `InsertionSort2` Data-Driven



(a) Opt (b) BayesWC (c) BayesPC

Fig. 15. `InsertionSort2` Hybrid

```
let incur_cost (hd : int) =
  let modulo = 5 in
  if (hd mod 200) = 0 then Raml.tick 1.0
  else (if (hd mod modulo) = 1 then Raml.tick
       0.85
        else if (hd mod modulo) = 2 then Raml.
             tick 0.65 else Raml.tick 0.5)

let rec insert (x : int) (xs : int list) =
  match xs with
  | [] → [ x ]
  | hd :: tl →
      let _ = incur_cost hd in
      if x <= hd then x :: hd :: tl else hd ::
           insert x tl

let rec insertion_sort (xs : int list) =
  match xs with [] →[] | hd :: tl →insert hd (
       insertion_sort tl)

let rec insertion_sort_second_time (xs : int
     list) =
  match xs with
  | [] → []
  | hd :: tl → insert hd (
       insertion_sort_second_time tl)

let insertion_sort_second_time2 (xs : int list)
     =
  Raml.stat (insertion_sort_second_time xs)

let double_insertion_sort (xs : int list) =
  let sorted_xs = insertion_sort xs in
  Raml.stat (insertion_sort_second_time
       sorted_xs)
```

(a) Fully data-driven resource analysis.

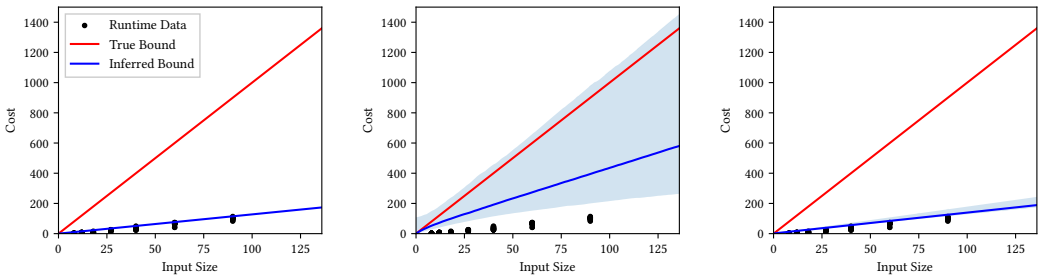
```
let rec insert (x : int) (xs : int list) =
  match xs with
  | [] → [ x ]
  | hd :: tl →
      let _ = incur_cost hd in
      if x <= hd then x :: hd :: tl else hd ::
           insert x tl

let rec insertion_sort (xs : int list) =
  match xs with [] →[] | hd :: tl →insert hd (
       insertion_sort tl)

let rec insert_second_time (x : int) (xs : int
     list) =
  match xs with
  | [] → [ x ]
  | hd :: tl →
      let _ = incur_cost hd in
      if x <= hd then x :: hd :: tl
      else hd :: (insert_second_time x tl)

let rec insertion_sort_second_time (xs : int
     list) =
  match xs with
  | [] → []
  | hd :: tl →
      let rec_result = insertion_sort_second_time
           tl in
      Raml.stat (insert_second_time hd rec_result
           )

let double_insertion_sort (xs : int list) =
  let sorted_xs = insertion_sort xs in
  insertion_sort_second_time sorted_xs
```

(b) Hybrid resource analysis.

Lst. 10. Source code of `InsertionSort2`. The `insertion_sort` procedure is called twice in a row. Our goal is to analyze the cost of the second call to insertion sort, which has a linear worst-case cost bound because inputs are already sorted. Conventional AARA can only infer a quadratic bound for the second call to insertion sort (just like the first call to insertion sort), but not a linear cost bound, because it cannot determine that the inputs to the second insertion sort are already sorted. (a) Fully data-driven resource analysis. (b) Hybrid resource analysis. We perform data-driven analysis on the function `insert_second_time`.

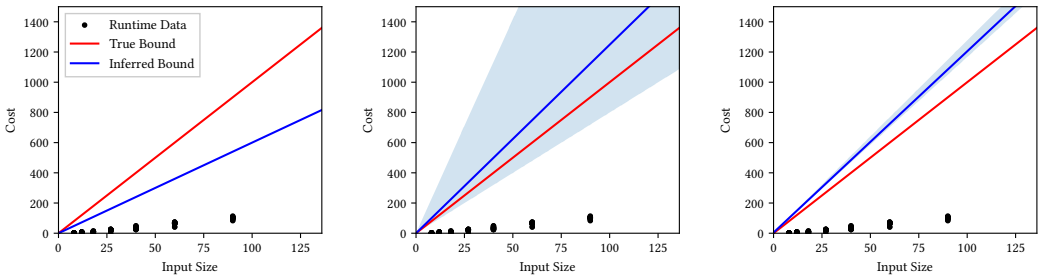
Table 7. Estimation gaps of inferred cost bounds for `MedianOfMedians` benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
Input Size	Percentile **Method**	5^{th}	50^{th}	95^{th}	5^{th}	50^{th}	95^{th}
10	Opt	-0.42	-0.42	-0.42	-0.39	-0.39	-0.39
	BayesWC	-0.29	0.60	5.20	19.69	85.53	709.77
	BayesPC	-0.64	-0.55	-0.34	1.41	1.48	1.52
100	Opt	-0.95	-0.95	-0.95	-0.49	-0.49	-0.49
	BayesWC	-0.95	-0.89	-0.62	8.35	40.30	339.77
	BayesPC	-0.91	-0.80	-0.54	1.38	1.45	1.50
1000	Opt	-0.99	-0.99	-0.99	-0.50	-0.50	-0.50
	BayesWC	-1.00	-0.99	-0.82	2.48	31.90	328.10
	BayesPC	-0.94	-0.81	-0.55	1.38	1.45	1.50



(a) Opt (b) BayesWC (c) BayesPC

Fig. 16. `MedianOfMedians` Data-Driven



(a) Opt (b) BayesWC (c) BayesPC

Fig. 17. `MedianOfMedians` Hybrid

```
exception Invalid_input

let incur_cost (hd : int) =
  if (hd mod 10) = 0 then Raml.tick 1.0 else
       Raml.tick 0.5

let rec append (xs : int list) (ys : int list) =
  match xs with [] →ys | hd :: tl →hd ::
       append tl ys

let rec insert (x : int) (list : int list) =
  match list with
  | [] → [ x ]
  | y :: ys → if x <= y then x :: y :: ys else y
        :: insert x ys

let rec insertion_sort (list : int list) =
  match list with [] →[] | x :: xs →insert x (
       insertion_sort xs)

let median_of_list_of_five (xs : int list) =
  let sorted_xs = insertion_sort xs in
  match sorted_xs with
  | [ x1; x2; x3; x4; x5 ] →(x3, [ x1; x2; x4;
       x5 ])
  | _ → raise Invalid_input

let rec partition_into_blocks (xs : int list) =
  match xs with
  | [] → ([], [])
  | x1 :: x2 :: x3 :: x4 :: x5 :: tl →
      let median, leftover =
           median_of_list_of_five [ x1; x2; x3;
           x4; x5 ] in
      let list_medians, list_leftover =
           partition_into_blocks tl in
      (median :: list_medians, append leftover
           list_leftover)
  | _ → raise Invalid_input

let rec partition (pivot : int) (xs : int list)
     =
  match xs with
  | [] → ([], [])
  | hd :: tl →
      let lower_list, upper_list = partition
           pivot tl in
      let _ = incur_cost hd in
      if hd <= pivot then (hd :: lower_list,
           upper_list)
      else (lower_list, hd :: upper_list)

let rec lower_list_length_after_partition (pivot
      : int) (xs : int list) =
  match xs with
  | [] → 0
  | hd :: tl →
      let lower_list_length =
           lower_list_length_after_partition
           pivot tl in
      if hd <= pivot then lower_list_length + 1
           else lower_list_length

let rec list_length (xs : int list) =
  match xs with [] →0 | hd :: tl →1 +
       list_length tl

let rec find_minimum_acc (acc : int list) (
     candidate : int) (xs : int list) =
  match xs with
  | [] → (candidate, acc)
  | hd :: tl →
      if hd < candidate then find_minimum_acc (
           candidate :: acc) hd tl
      else find_minimum_acc (hd :: acc) candidate
            tl

let find_minimum (xs : int list) =
  match xs with
  | [] → raise Invalid_input
  | hd :: tl → find_minimum_acc [] hd tl

let rec preprocess_list_acc (minima_acc : int
     list) (xs : int list) =
  let xs_length = list_length xs in
  if xs_length mod 5 = 0 then (minima_acc, xs)
  else
    let minimum, leftover = find_minimum xs in
    preprocess_list_acc (minimum :: minima_acc)
         leftover

let rec get_nth_element (index : int) (xs : int
     list) =
  match xs with
  | [] → raise Invalid_input
  | hd :: tl → if index = 0 then hd else
       get_nth_element (index - 1) tl
```

Lst. 11. Source code of helper functions used in `MedianOfMedians` (Listing 12).

```
let rec median_of_medians (index : int) (xs :
    int list) =
  match xs with
  | [] → raise Invalid_input
  | _ →
      let minima, xs_trimmed =
          preprocess_list_acc [] xs in
      let mod_five = list_length minima in
      if index < mod_five then get_nth_element (
          mod_five - index - 1) minima
      else
        let index_trimmed = index - mod_five in
        let list_medians, _ =
            partition_into_blocks xs_trimmed in
        let num_medians = list_length
            list_medians in
        let index_median = num_medians / 2 in
        let median_of_medians =
          Raml.stat (median_of_medians
              index_median list_medians)
        in
        let lower_list_length =
          lower_list_length_after_partition
              median_of_medians xs_trimmed
        in
        if index_trimmed = lower_list_length - 1
            then
          let _, _ = partition median_of_medians
              xs_trimmed in
          median_of_medians
        else if index_trimmed < lower_list_length
            - 1 then
          let lower_list, _ = partition
              median_of_medians xs_trimmed in
          Raml.stat (median_of_medians
              index_trimmed lower_list)
        else
          let new_index = index_trimmed -
              lower_list_length in
          let _, upper_list = partition
              median_of_medians xs_trimmed in
          Raml.stat (median_of_medians new_index
              upper_list)

let median_of_medians2 (index : int) (xs : int
    list) =
  Raml.stat (median_of_medians index xs)
```

(a) Fully data-driven resource analysis.

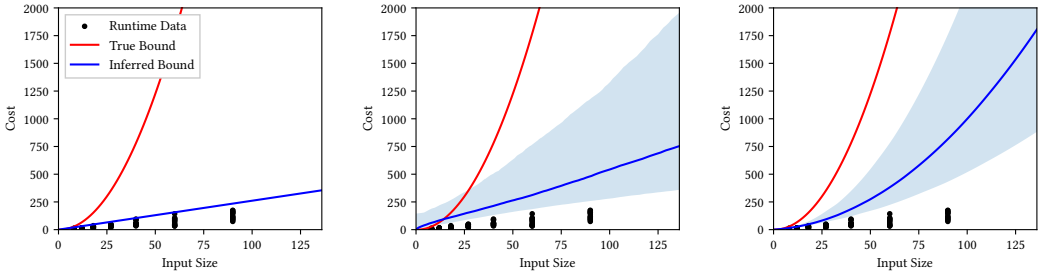
```
let rec median_of_medians (index : int) (xs :
    int list) =
  match xs with
  | [] → raise Invalid_input
  | _ →
      let minima, xs_trimmed =
          preprocess_list_acc [] xs in
      let mod_five = list_length minima in
      if index < mod_five then get_nth_element (
          mod_five - index - 1) minima
      else
        let index_trimmed = index - mod_five in
        let list_medians = partition_into_blocks
            xs_trimmed in
        let num_medians = list_length
            list_medians in
        let index_median = num_medians / 2 in
        let median_of_medians = median_of_medians
            index_median list_medians in
        let lower_list_length =
          lower_list_length_after_partition
              median_of_medians xs_trimmed
        in
        if index_trimmed = lower_list_length - 1
            then
          let _, _ = Raml.stat (partition
              median_of_medians xs_trimmed) in
          median_of_medians
        else if index_trimmed < lower_list_length
            - 1 then
          let lower_list, _ =
            Raml.stat (partition
                median_of_medians xs_trimmed)
          in
          median_of_medians index_trimmed
              lower_list
        else
          let new_index = index_trimmed -
              lower_list_length in
          let _, upper_list =
            Raml.stat (partition
                median_of_medians xs_trimmed)
          in
          median_of_medians new_index upper_list
```

(b) Hybrid resource analysis.

Lst. 12. Source code of MedianOfMedians. Conventional AARA cannot infer any polynomial bound for MedianOfMedians. (a) Fully data-driven resource analysis. (b) Hybrid resource analysis. We conduct data-driven analysis on the function partition. Although conventional AARA can derive a linear cost bound of the partition function, analyzing it using data-driven analysis gives a tighter cost bound. This tighter linear bound of the partition function is required for deriving an overall linear cost bound of MedianOfMedians.

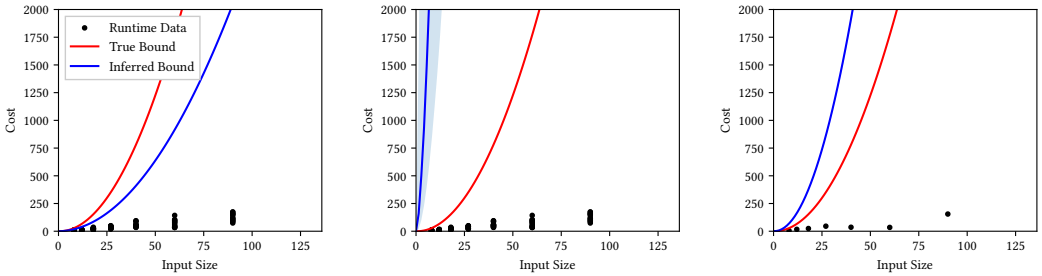
Table 8. Estimation gaps of inferred cost bounds for `QuickSelect` benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
Input Size	Percentile / **Method**	5^{th}	50^{th}	95^{th}	5^{th}	50^{th}	95^{th}
10	Opt	-0.42	-0.42	-0.42	-0.39	-0.39	-0.39
	BayesWC	-0.29	0.60	5.20	19.69	85.53	709.77
	BayesPC	-0.64	-0.55	-0.34	1.41	1.48	1.52
100	Opt	-0.95	-0.95	-0.95	-0.49	-0.49	-0.49
	BayesWC	-0.95	-0.89	-0.62	8.35	40.30	339.77
	BayesPC	-0.91	-0.80	-0.54	1.38	1.45	1.50
1000	Opt	-0.99	-0.99	-0.99	-0.50	-0.50	-0.50
	BayesWC	-1.00	-0.99	-0.82	2.48	31.90	328.10
	BayesPC	-0.94	-0.81	-0.55	1.38	1.45	1.50



(a) Opt (b) BayesWC (c) BayesPC

Fig. 18. `QuickSelect` Data-Driven



(a) Opt (b) BayesWC (c) BayesPC

Fig. 19. `QuickSelect` Hybrid

```
exception Invalid_input

let incur_cost (hd : int) =
  if (hd mod 10) = 0 then Raml.tick 1.0 else
       Raml.tick 0.5

let rec append (xs : int list) (ys : int list) =
  match xs with [] →ys | hd :: tl →hd ::
       append tl ys

let rec partition (pivot : int) (xs : int list)
     =
  match xs with
  | [] → ([], [])
  | hd :: tl →
     let lower_list, upper_list = partition
          pivot tl in
     let _ = incur_cost hd in
     if hd <= pivot then (hd :: lower_list,
          upper_list)
     else (lower_list, hd :: upper_list)

let rec list_length (xs : int list) =
  match xs with [] →0 | hd :: tl →1 +
       list_length tl

let rec quickselect (index : int) (xs : int list
     ) =
  match xs with
  | [] → raise Invalid_input
  | [ x ] → if index = 0 then x else raise
       Invalid_input
  | hd :: tl →
     let lower_list, upper_list = partition hd
          tl in
     let lower_list_length = list_length
          lower_list in
     if index < lower_list_length then
          quickselect index lower_list
     else if index = lower_list_length then hd
     else
       let new_index = index - lower_list_length
            - 1 in
       quickselect new_index upper_list

let quickselect2 (index : int) (xs : int list) =
  Raml.stat (quickselect index xs)
```

(a) Fully data-driven resource analysis.

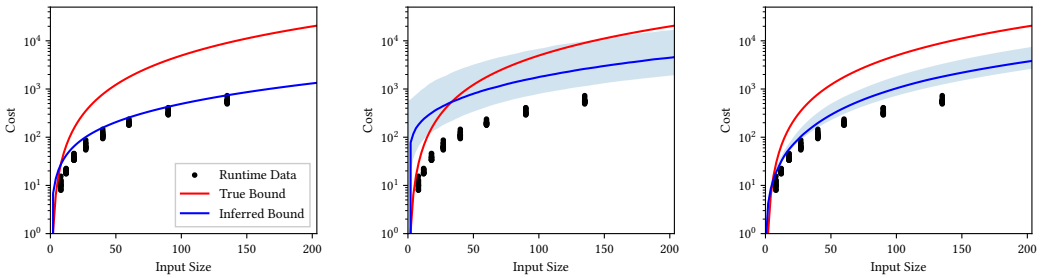
```
let rec quickselect (index : int) (xs : int list
     ) =
  match xs with
  | [] → raise Invalid_input
  | [ x ] → if index = 0 then x else raise
       Invalid_input
  | hd :: tl →
     (* This is a workaround for an issue with
          the let-normal form inside
     Raml.stat(...) in the implementation *)
     let tl = tl in
     let lower_list, _ = partition_cost_free hd
          tl in
     let lower_list_length = list_length
          lower_list in
     if index < lower_list_length then
       let lower_list, _ = Raml.stat (partition
            hd tl) in
       quickselect index lower_list
     else if index = lower_list_length then
       let _, _ = Raml.stat (partition hd tl) in
       hd
     else
       let _, upper_list = Raml.stat (partition
            hd tl) in
       quickselect (index - lower_list_length -
            1) upper_list
```

(b) Hybrid resource analysis.

Lst. 13. Source code of `QuickSelect`. Conventional AARA cannot analyze this source code if the comparison function used inside `QuickSelect` is complex (e.g., OCaml's built-in polymorphic comparison). (a) Fully data-driven resource analysis. (b) Hybrid resource analysis. We perform data-driven analysis on `partition`.

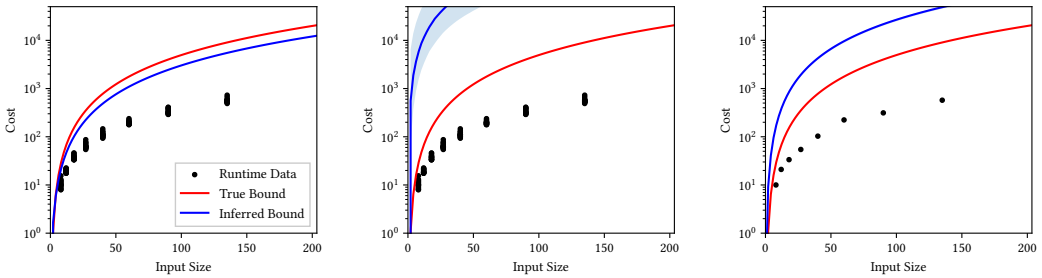
Table 9. Estimation gaps of inferred cost bounds for `QuickSort` benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
Input Size	Percentile **Method**	5^{th}	50^{th}	95^{th}	5^{th}	50^{th}	95^{th}
10	Opt	-0.23	-0.23	-0.23	-0.29	-0.29	-0.29
	BayesWC	0.37	3.66	32.71	36.48	181.96	1776.52
	BayesPC	-0.52	-0.47	-0.22	4.12	4.73	4.96
100	Opt	-0.90	-0.90	-0.90	-0.39	-0.39	-0.39
	BayesWC	-0.87	-0.64	1.24	17.83	82.90	667.39
	BayesPC	-0.88	-0.79	-0.61	3.78	4.41	4.69
1000	Opt	-0.96	-0.96	-0.96	-0.40	-0.40	-0.40
	BayesWC	-0.98	-0.91	-0.09	5.07	60.66	610.58
	BayesPC	-0.93	-0.83	-0.63	3.75	4.38	4.66



(a) Opt (b) BayesWC (c) BayesPC

Fig. 20. `QuickSort` Data-Driven



(a) Opt (b) BayesWC (c) BayesPC

Fig. 21. `QuickSort` Hybrid

```
let incur_cost (hd : int) =
  if (hd mod 5) = 0 then Raml.tick 1.0 else Raml.
       tick 0.5

let rec append (xs : int list) (ys : int list) =
  match xs with [] →ys | hd :: tl →hd ::
       append tl ys

let rec partition (pivot : int) (xs : int list)
     =
  match xs with
  | [] → ([], [])
  | hd :: tl →
     let lower_list, upper_list = partition
          pivot tl in
     let _ = incur_cost hd in
     if hd <= pivot then (hd :: lower_list,
          upper_list)
     else (lower_list, hd :: upper_list)

let rec quicksort (xs : int list) =
  match xs with
  | [] → []
  | hd :: tl →
     let lower_list, upper_list = partition hd
          tl in
     let lower_list_sorted = quicksort
          lower_list in
     let upper_list_sorted = quicksort
          upper_list in
     append lower_list_sorted (hd ::
          upper_list_sorted)

let quicksort2 (xs : int list) = Raml.stat (
     quicksort xs)
```

(a) Fully data-driven resource analysis.

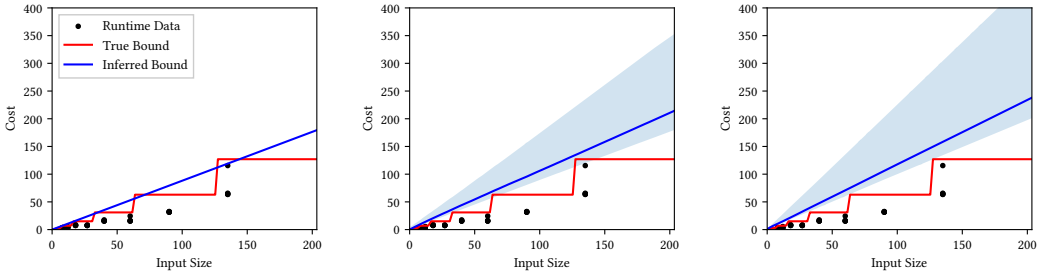
```
let rec quicksort (xs : int list) =
  match xs with
  | [] → []
  | hd :: tl →
     let lower_list, upper_list = Raml.stat (
          partition hd tl) in
     let lower_list_sorted = quicksort
          lower_list in
     let upper_list_sorted = quicksort
          upper_list in
     append lower_list_sorted (hd ::
          upper_list_sorted)
```

(b) Hybrid resource analysis.

Lst. 14. Source code of `QuickSort`. Conventional AARA cannot analyze this source code if the comparison function used inside `QuickSort` is complex (e.g., OCaml's built-in polymorphic comparison). (a) Fully data-driven resource analysis. (b) Hybrid resource analysis. We perform data-driven analysis on `partition`.

Table 10. Estimation gaps of inferred cost bounds for Round benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
	Percentile	5^{th}	50^{th}	95^{th}	5^{th}	50^{th}	95^{th}
Input Size	**Method**						
10	Opt	0.26	0.26	0.26	∅	∅	∅
	BayesWC	0.27	0.68	2.83	∅	∅	∅
	BayesPC	0.49	0.82	2.57	∅	∅	∅
100	Opt	0.40	0.40	0.40	∅	∅	∅
	BayesWC	0.40	0.68	2.33	∅	∅	∅
	BayesPC	0.55	0.87	2.86	∅	∅	∅
1000	Opt	0.73	0.73	0.73	∅	∅	∅
	BayesWC	0.67	1.06	3.11	∅	∅	∅
	BayesPC	0.89	1.29	3.75	∅	∅	∅



(a) Opt (b) BayesWC (c) BayesPC

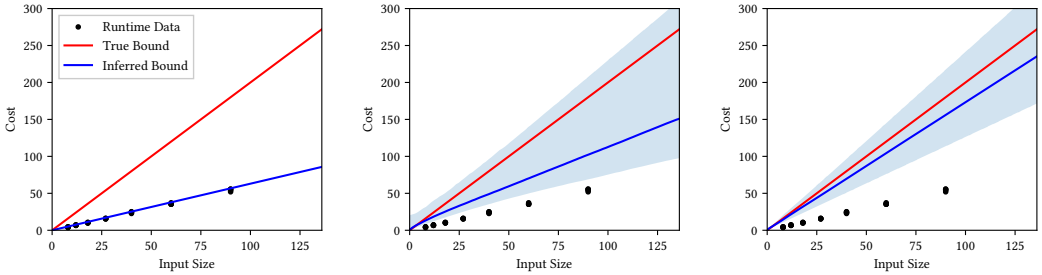
Fig. 22. Round Data-Driven

```
let incur_cost (hd : int) =
  if (hd mod 10) = 0 then Raml.tick 1.0 else Raml.tick 0.5

let rec double (xs : int list) =
  match xs with [] →[] | hd :: tl →hd :: hd :: double tl

let rec half (xs : int list) =
  match xs with [] →[] | [ x ] →[] | x1 :: x2 :: tl →x1 :: half tl

let rec round (xs : int list) =
  match xs with
  | [] → []
  | hd :: tl →
      let half_result = half tl in
      let recursive_result = round half_result in
      hd :: double recursive_result

let rec linear_traversal (xs : int list) =
  match xs with
  | [] → []
  | hd :: tl →
      let _ = incur_cost hd in
      hd :: linear_traversal tl

let round_followed_by_linear_traversal (xs : int list) =
  let round_result = round xs in
  linear_traversal round_result

let round2 (xs : int list) =
  Raml.stat (round_followed_by_linear_traversal xs)
```

Lst. 15. Source code of Round for fully data-driven resource analysis. Conventional AARA cannot infer any polynomial cost bounds for this code [39, Section 5.4.3].

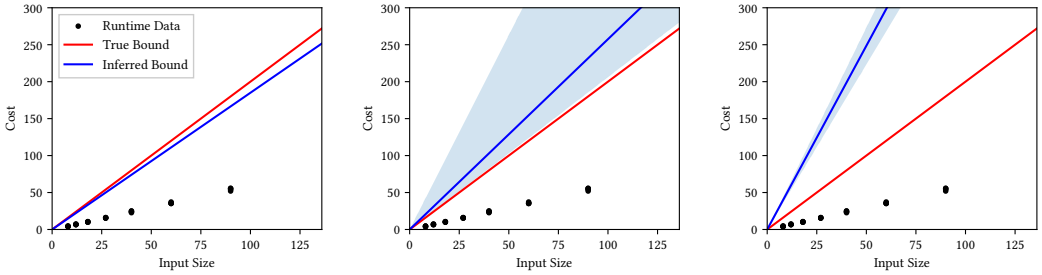
Table 11. Estimation gaps of inferred cost bounds for `ZAlgorithm` benchmark on various input sizes.

		Relative Gap in Inferred Cost Bound					
		Data-Driven			Hybrid		
Input Size	Percentile **Method**	5^{th}	50^{th}	95^{th}	5^{th}	50^{th}	95^{th}
10	Opt	-0.68	-0.68	-0.68	-0.08	-0.08	-0.08
	BayesWC	-0.53	-0.21	1.37	0.00	0.29	2.99
	BayesPC	-0.48	-0.10	0.33	1.18	1.49	1.78
100	Opt	-0.68	-0.68	-0.68	-0.08	-0.08	-0.08
	BayesWC	-0.65	-0.44	0.56	0.00	0.29	2.99
	BayesPC	-0.50	-0.13	0.23	1.18	1.49	1.78
1000	Opt	-0.68	-0.68	-0.68	-0.08	-0.08	-0.08
	BayesWC	-0.76	-0.47	0.56	0.00	0.29	2.99
	BayesPC	-0.50	-0.14	0.22	1.18	1.49	1.78



(a) Opt (b) BayesWC (c) BayesPC

Fig. 23. `ZAlgorithm` Data-Driven



(a) Opt (b) BayesWC (c) BayesPC

Fig. 24. `ZAlgorithm` Hybrid

Received 2023-11-16; accepted 2024-03-31

```
exception Invalid_input

let incur_cost (hd : int) =
  let modulo = 5 in
  if (hd mod 100) = 0 then Raml.tick 1.0
  else (if (hd mod modulo) = 1 then Raml.tick 0.85
        else if (hd mod modulo) = 2 then Raml.tick 0.65 else Raml.tick 0.5)

let rec list_length (xs : int list) =
  match xs with [] →0 | hd :: tl →1 + list_length tl

let hd_exn (xs : int list) =
  match xs with [] →raise Invalid_input | hd :: _ →hd

let min (x1 : int) (x2 : int) = if x1 < x2 then x1 else x2

let rec drop_n_elements (xs : int list) (n : int) =
  match xs with
  | [] → []
  | hd :: tl → if n = 0 then hd :: tl else drop_n_elements tl (n - 1)

let rec longest_common_prefix (xs1 : int list) (xs2 : int list) =
  match xs1 with
  | [] → 0
  | hd1 :: tl1 →(
      match xs2 with
      | [] → 0
      | hd2 :: tl2 →
          if hd1 = hd2 then
            let _ = incur_cost (hd1 + hd2) in
            1 + longest_common_prefix tl1 tl2
          else 0 )
```

Lst. 16. Source code of helper functions used in `ZAlgorithm` (Listing 17).

```
let rec z_algorithm_acc (acc : int list) (
     original_string : int list)
    (current_string : int list) (left : int) (
        right : int) =
  match current_string with
  | [] → acc
  | hd :: tl →
      let _ = incur_cost hd
      in let current_index = list_length acc
      in let old_result =
        if left = 0 then 0 else hd_exn (
            drop_n_elements acc (left - 1))
      in let current_result_initial =
        if current_index < right then min (right
            - current_index) old_result
        else 0
      in let first_sublist =
        drop_n_elements original_string
            current_result_initial
      in let second_sublist =
        drop_n_elements current_string
            current_result_initial
      in let common_prefix_size =
        longest_common_prefix first_sublist
            second_sublist
      in let current_result =
           current_result_initial +
           common_prefix_size
      in let cumulative_result_updated =
           current_result :: acc
      in if current_index + current_result >
           right then
        z_algorithm_acc cumulative_result_updated
            original_string tl
          current_index
          (current_index + current_result)
        else
          z_algorithm_acc
              cumulative_result_updated
              original_string tl left right

let rec reverse_acc (acc : int list) (xs : int
     list) =
  match xs with [] →acc | hd :: tl →
       reverse_acc (hd :: acc) tl

let z_algorithm (xs : int list) =
  match xs with
  | [] → []
  | hd :: tl →
    reverse_acc []
      (z_algorithm_acc [ 0 ] xs tl 0 0)

let z_algorithm2 (xs : int list) = Raml.stat (
     z_algorithm xs)
```

(a) Fully data-driven resource analysis.

```
let rec z_algorithm_acc (acc : int list) (
     original_string : int list)
    (current_string : int list) (left : int) (
        right : int) =
  match current_string with
  | [] → acc
  | hd :: tl →
      let _ = incur_cost hd in
      let current_index = list_length acc in
      let old_result =
        if left = 0 then 0 else hd_exn (
            drop_n_elements acc (left - 1))
      in let current_result_initial =
        if current_index < right then min (right
            - current_index) old_result
        else 0
      in let first_sublist =
        drop_n_elements original_string
            current_result_initial
      in let second_sublist =
        drop_n_elements current_string
            current_result_initial
      in let common_prefix_size =
        Raml.stat (longest_common_prefix
            first_sublist second_sublist)
      in let current_result =
           current_result_initial +
           common_prefix_size
      in let cumulative_result_updated =
           current_result :: acc
      in if current_index + current_result >
           right then
        z_algorithm_acc cumulative_result_updated
            original_string tl
          current_index
          (current_index + current_result)
        else
          z_algorithm_acc
              cumulative_result_updated
            original_string tl left right

let rec reverse_acc (acc : int list) (xs : int
     list) =
  match xs with [] →acc | hd :: tl →
       reverse_acc (hd :: acc) tl

let z_algorithm (xs : int list) =
  match xs with
  | [] → []
  | hd :: tl →
    reverse_acc []
      (z_algorithm_acc [ 0 ] xs tl 0 0)
```

(b) Hybrid resource analysis.

Lst. 17. Source code of `ZAlgorithm`. Conventional AARA can infer a quadratic cost bound, but not the true linear cost bound. (a) Fully data-driven resource analysis. (b) Hybrid resource analysis. We perform data-driven analysis on `longest_common_prefix`.