---

**Article:**
Baruah, Sanjoy, Burns, Alan and Davis, Robert Ian (2023) Optimal Synthesis of Robust IDK Classifier Cascades. ACM Transactions in Embedded Computing Systems. 150. pp. 1-26. ISSN: 1558-3465

https://doi.org/10.1145/3609129

---

# Optimal Synthesis of Robust IDK Classifier Cascades

SANJOY BARUAH, Washington University in St. Louis, USA
ALAN BURNS and ROBERT IAN DAVIS, The University of York, UK

An *IDK classifier* is a computing component that categorizes inputs into one of a number of classes, if it is able to do so with the required level of confidence, otherwise it returns "I Don't Know" (IDK). *IDK classifier cascades* have been proposed as a way of balancing the needs for fast response and high accuracy in classification-based machine perception. Efficient algorithms for the synthesis of IDK classifier cascades have been derived; however, the responsiveness of these cascades is highly dependent on the accuracy of predictions regarding the run-time behavior of the classifiers from which they are built. Accurate predictions of such run-time behavior is difficult to obtain for many of the classifiers used for perception. By applying the *algorithms using predictions* framework, we propose efficient algorithms for the synthesis of IDK classifier cascades that are *robust* to inaccurate predictions in the following sense: the IDK classifier cascades synthesized by our algorithms have short expected execution durations when the predictions are accurate, and these expected durations increase only within specified bounds when the predictions are inaccurate.

CCS Concepts: • **Computer systems organization → Embedded systems**; **Real-time systems**; • **Software and its engineering → Scheduling;**

Additional Key Words and Phrases: Algorithms using predictions, robust scheduling, classification, on-line scheduling

**ACM Reference format:**
Sanjoy Baruah, Alan Burns, and Robert Ian Davis. 2023. Optimal Synthesis of Robust IDK Classifier Cascades. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 150 (September 2023), 26 pages.
https://doi.org/10.1145/3609129



## 1 INTRODUCTION

Learning-Enabled Components (LECs) are increasingly used in safety-critical applications such as autonomous driving. Their use dramatically enhances the capabilities of autonomous Cyber-Physical Systems. For example, it is hard to conceive of self-driving cars that do not make extensive use of LECs, such as Deep Neural Networks, for perception. The use of LECs, however, requires that the developers of such systems address a new challenge: how to deal with the fact that most widely

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Embedded Software (EMSOFT), 2023.
This research was funded in part by Innovate UK HICLASS project (113213), and the US National Science Foundation (Grants CNS-2141256 and CPS-2229290). EPSRC Research Data Management: No new primary data was created during this study.
Authors' addresses: S. Baruah, Washington University in St. Louis, USA; email: baruah@wustl.edu; A. Burns and R. I. Davis, The University of York, UK; emails: {alan.burns, rob.daviss}@york.ac.uk.

1539-9087/2023/09-ART150 $15.00
https://doi.org/10.1145/3609129

used LECs do not provide performance guarantees of the form, and at the levels of assurance, required in the verification and validation processes used in safety-critical systems engineering.

A promising approach for incorporating low-assurance predictions of the kind that are typically provided by LECs into algorithms designed for use in safety-critical systems can be found in a recent line of work in the algorithms community: *algorithms using predictions*, see [11] for a review and survey. Algorithms using predictions are designed to make use of low-assurance predictions: they perform very well when the predictions are accurate, while simultaneously guaranteeing to provide an acceptable level of performance regardless of the quality of the predictions.

In this paper we examine the applicability of the algorithms using predictions framework to the analysis of real-time systems that use LECs. As an exemplar, we focus on a design problem concerning a form of LECs called IDK classifiers that have recently been the subject of several studies [1, 2, 4, 5] in the real-time systems community. IDK classifiers are computing components that categorize input samples from sensors into one of a set of a classes, for example, part of a camera image is classified as a 'pedestrian,' 'car,' 'stop sign,' etc., if they are able to do so with the required level of confidence, otherwise they return "I Don't Know" (IDK). Such LEC-based classifiers are generally not able to guarantee their classification decisions at a high enough level of assurance and so should be backed up by deterministic classifiers, i.e., non-learning-enabled classifiers, including perhaps human intervention. In this paper, we examine how, given predictions (i.e., estimates) of the probabilities that individual IDK classifiers will return "I Don't Know" and therefore need to fall back on a backup deterministic classifier, the framework of algorithms using predictions can provide improved performance in the form of lower expected execution durations when the predictions are accurate, while simultaneously guaranteeing to be robust (i.e., limiting expected execution durations) when predictions turn out to be inaccurate.

**Contributions.** In this work, we *explicitly* account for the fact that the probabilities describing the efficacy of the classifiers obtained via estimation (for example using the methodology detailed in [1]) may deviate significantly from the true values that accurately describe the efficacy of the classifiers in a given operational context.[1] We therefore revisit the problems, approaches, and algorithms of the prior real-time systems work on IDK classifiers [1, 2, 4, 5]. This prior work derives algorithms for finding the optimal IDK classifier cascades that minimize the expected execution duration required for successful classification, subject to a latency constraint. To date, the most advanced method [1] provides a solution that is applicable to classifiers that exhibit arbitrary dependences between their behaviors. We integrate the algorithms using predictions framework into this method to obtain an algorithm ALG that provides *consistency* in terms of close to optimal performance when the predictions (i.e., probabilistic characterizations) are accurate, while also providing *robustness* in that the departure of the algorithm's performance from optimal is bounded if the predictions turn out to be inaccurate. We also explore *the smoothness properties* of the algorithm: the degree of performance degradation as a function of the error between the true and the predicted probabilities, thereby providing a way to understand the impact of inaccurate predictions.

**Organization.** The remainder of the paper is organized as follows. Our work applies the algorithms using predictions framework to system design and analysis problems involving IDK classifiers. In Section 2, we provide the necessary background on both IDK classifiers (Section 2.1) and algorithms using predictions (Section 2.2), by briefly and non-exhaustively reviewing prior work on these topics. In Section 3, we derive an algorithm that ensures against severe degradation in performance in the event that the predicted probabilities turn out to be inaccurate (Section 3.1),

[1]Indeed, the underlying phenomena may not even be epistemic in nature and thus inherently cannot be accurately characterized using probability distributions.

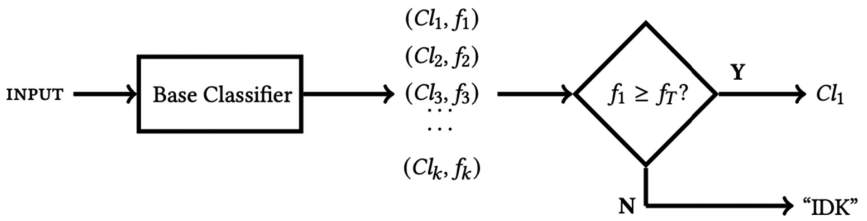



Fig. 1. How to obtain an IDK classifier with specified confidence threshold parameter $f_T$ from a given base classifier. Upon any input, the base classifier outputs up to $k$ ordered pairs $(Cl_i, f_i)$, indicating that it believes that the input belongs to the class $Cl_i$ with confidence score $f_i$. (It is assumed that $Cl_1, Cl_2, \ldots, Cl_k$ are the $k$ most likely classes in non-increasing order of confidence; i.e., $f_1 \geq f_2 \geq \cdots \geq f_k$.)

and use the predictions to achieve excellent performance when they are accurate, with graceful degradation as their accuracy falls off (Section 3.2). Section 4 provides details of the algorithm's implementation, and applies it to a proof-of-concept case study based on data from four existing classifiers that use the ResNet deep neural network architecture. We conclude in Section 5 with a discussion of some promising directions for possible further work.

## 2 BACKGROUND AND RELATED WORK

The research presented in this paper builds on two recent prior topics: (i) IDK classifiers, and (ii) algorithms using predictions.

### 2.1 IDK Classifiers

A classifier is a software component that categorizes each input provided to it into one of a fixed number of classes. Perception in autonomous mobile Cyber Physical Systems (CPS) is increasingly being performed using classifiers that are based on Deep Learning and related AI technologies. Classifiers that are used in this way must be able to make accurate predictions in real time using limited computational resources. However, much of the current work in mainstream machine learning research relegates timing issues to the background, and focuses primarily on improving the accuracy of classification. This focus on accuracy rather than timing has resulted in highly accurate classifiers that take substantial time to process even simple inputs that should be straightforward to classify. For instance, Wang et al. [15] showed that for a considerable fraction of the ImageNet 2012 benchmark [13], an order-of-magnitude increase in classifier execution time has yielded only a negligible improvement in the accuracy of predictions. They suggested a trade-off between accuracy and latency, based on the insight that if advanced but slower classifiers were only used in the more challenging cases, then the time taken to achieve successful classification could be reduced on average, without any reduction in accuracy. One approach aimed at achieving appropriate accuracy-latency trade-offs is the use of IDK classifiers [9]. An IDK classifier is obtained from an existing base classifier in the manner depicted in Figure 1. If the base classifier is unable to arrive at a classification decision with a level of confidence that exceeds a predefined confidence threshold, then a dummy class, IDK (meaning "I Don't Know") is output instead. Multiple different IDK classifiers, with different execution times and probabilities of success (i.e., returning a real class rather than IDK), may be devised for the same classification problem. Wang et al. [15] proposed arranging such IDK classifiers into *IDK cascades*, which are sequences of IDK classifiers designed to work as follows:

(1) The first classifier in the IDK cascade is invoked first, for any input that needs to be classified.
(2) If the classifier outputs a real class, rather than IDK, then the IDK cascade terminates and characterizes the input as being of the identified class.

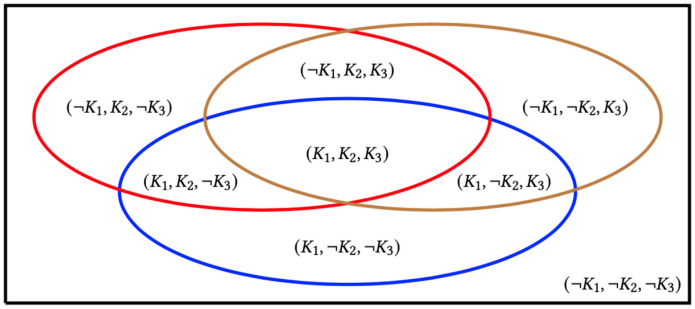



Fig. 2. The $2^n$ disjoint regions in the probability space for three IDK classifiers ($n = 3$) and one deterministic classifier. The blue, red, and brown ellipses respectively denote the parts of the space where the classifiers $K_1$, $K_2$, and $K_3$ are successful (i.e., do not output IDK). The enclosing rectangle can be thought of as denoting the region in which the deterministic classifier is successful (i.e., all inputs). Each of the $2^3 = 8$ disjoint regions into which the probability space is partitioned by the three ellipses is labeled with a 3-tuple, with $K_i$ ($\neg K_i$, respectively) denoting that the the IDK classifier $K_i$ returns a real class (resp. IDK) in this region.

(3) Otherwise (i.e., the classifier outputs IDK), the subsequent classifier in the IDK cascade is invoked and the process continues from step 2.

In application scenarios where classification of all inputs is required (i.e., IDK is not an acceptable response from the cascade), IDK classifiers must be used in conjunction with more traditional *deterministic* classifiers for the same classification problem. By having a deterministic classifier as the last classifier in an IDK cascade, then it is assured that the IDK cascade will always succeed, i.e., return a real class. If all of the IDK classifiers output IDK for some particular input, then the deterministic classifier will be called upon to provide an authoritative classification.

**Characterizing Classifiers.** As part of ongoing efforts to provide a scheduling-theoretic framework that enables the use of LECs in hard real-time safety-critical systems, the real-time scheduling theory community has since 2021 begun studying IDK classifiers [1, 2, 4, 5], with the goal of being able to synthesize IDK cascades that minimize the expected execution duration needed to obtain a successful classification, optionally within a specified latency constraint. Employing IDK cascades in this way provides an effective trade-off between the worst-case execution duration and the average-case (i.e., expected) execution duration required for successful classification, in practical systems [1]. One of the challenges is the possibility of correlated behaviors, i.e. arbitrary dependences, between different IDK classifiers; Abdelzaher et al. [1] observed that:

> Different IDK classifiers may exhibit related behaviors for a variety of reasons. Dependences may be induced by the environment (an object that is difficult for one classifier to identify may also be difficult for another classifier to identify), by the training process (the same data may be used in the training of all classifiers), and by common components and algorithms (the same Deep Neural Network approach may be applied in a subset of the classifiers).

Suppose that we have $n$ IDK classifiers $K_1, K_2, \ldots, K_n$ for the same classification problem. (Note, these classifiers may exhibit arbitrary dependences between their behaviors, and so are not required to be independent). Conceptually, it is convenient to think of the probability space for these $n$ IDK classifiers as a Venn Diagram partitioned into $2^n$ distinct regions, with each region corresponding to one of the $2^n$ possible combinations of the $n$ individual classifiers successfully classifying an input or returning IDK. (See Figure 2 for the case of $n = 3$ classifiers.) Abdelzaher

et al. [1] describe how profiling using representative input data can be used to estimate the probability value that should be associated with each of these $2^n$ regions. In essence, this methodology: (i) maintains a counter, initialized to zero, corresponding to each of these $2^n$ regions; (ii) processes each input sample of the profiling data by having all $n$ IDK classifiers process it individually and observing whether each outcome is IDK or not; (iii) based on these outcomes increments the appropriate counter; and (iv) after all the profiling input samples have been looked at, each counter value is divided by the total number of samples to obtain the estimated probability that precisely the outcome associated with the region (e.g., $(K_1, \neg K_2, \neg K_3)$ indicating that classifier $K_1$ reports a real class, and classifiers $K_2$ and $K_3$ output IDK) will occur. Thus the methodology characterizes the IDK classifiers by:

(1) Their execution duration[2] parameters $C_1, C_2, \ldots, C_n$.
(2) The $2^n$ probability estimates, one corresponding to each of the $2^n$ possible combinations of the $n$ individual classifiers either successfully classifying an input or returning IDK.

We point out that the *size of the model for characterizing the n IDK classifiers is* $O(2^n)$. Such an exponential-sized model appears unavoidable, when considering classifiers that have arbitrary dependences between their behaviors. To account for arbitrary dependences between all pairs of classifiers, it is necessary to quantify the probability of each of the $2^n$ subsets of classifiers successfully classifying inputs, as is done in [1]. Simpler models are possible, but rely on assumptions regarding mutual dependences between the behaviors of the different IDK classifiers, which may not hold in practice. For example, it is assumed in [5] that all classifiers are pairwise independent, whereas [4] also permits classifiers that are fully dependent, as well as a mix of classifiers with independent and fully dependent relationships. The number of IDK classifiers, $n$, that are available for solving a specific classification problem is clearly application dependent; as a general rule values of $n$ that are much greater than about 10 to 12 are unlikely to be commonly encountered in practice, the exponential size of the model should therefore not be an issue in practice.[3] As an example, the large Multi-Modal case study described in [1] includes 9 classifiers, composed from multiple neural-network designs utilizing multiple input modes (vision, acoustic, and seismic).

As is evident on examining the methodology for estimating the probability values [1], the accuracy of these estimates is dependent on the quality and representativity of the profiling data used. In particular, how well does the profiling data represent the kinds of inputs experienced by the classifier during operation of the deployed system across different operational environments or *contexts*? Wang et al. [14] showed that the training of machine learning classifiers can lead to over-fitting, and hence performance that can be significantly degraded in some practical contexts. By comparison, simpler traditional classifiers are less likely to suffer from this problem [14]. Since it is difficult to make authoritative assertions that the profiling data is accurately representative of the underlying distribution from which inputs are drawn during operation, if indeed a single such distribution even exists, in this paper we view the $2^n$ probability estimates obtained as *predictions* of the true (unknown – perhaps unknowable) underlying probabilities, and apply the algorithms using predictions framework to deal with the possibility that these predictions are inaccurate.

[2]The methodology of [1] actually characterizes each classifier by both its worst-case and its measured average execution durations. However in this paper we make the simplifying assumption that each invocation of classifier $K_i$ actually takes a duration of exactly $C_i$ time units; i.e., these $C_i$'s represent the actual execution times of the classifiers. This simplifying assumption allows us to focus on LEC-specific aspects of prediction. The model is easily generalized to also characterize uncertainty in, and predictions of, the actual execution times of the classifiers; however the associated analysis becomes more complex.

[3]The computational limits of the techniques described in this paper on a laptop computer are approx. $n = 20$, since $2^{20}$ implies approximately $10^6$ nodes in the graphs that the algorithm employs.

Note, we do not assume that the classifiers are infallible. Successful designation of a class rather than returning IDK may in some cases result in incorrect classification. We assume that for each IDK classifier, the confidence threshold used to determine if IDK should be returned is set at an appropriately high level that the proportion of such misclassifications is within the design parameters of the system. In this paper, when we refer to successful classification, we mean returning a real class rather than IDK.

**Notation:** For any subset $S$ of the collection of IDK classifiers,

- let $\mathcal{P}[S]$ denote the *predicted* probability that at least one of the classifiers in $S$ returns a real class, rather than IDK.
- let $\mathcal{Q}[S]$ denote the *true* (but unknown) value of this same probability.

In terms of the Venn diagram representation of the probability space, shown in Figure 2, $\mathcal{P}[S]$ denotes the estimated probability measure inside the union of the circles corresponding to each of the classifiers in $S$. For example, in Figure 2, $\mathcal{P}[\{K_1, K_2\}]$ would include all the regions of the Venn diagram except the two regions that are labeled $(\neg K_1, \neg K_2, K_3)$ and $(\neg K_1, \neg K_2, \neg K_3)$. As was pointed out in [1], the $2^n$ $\mathcal{P}[S]$ values corresponding to each of the $2^n$ subsets of the set of $n$ IDK classifiers $K_1, K_2, \ldots, K_n$ are easily computed in $O(4^n)$ time from the $2^n$ predicted probability values for these classifiers that are determined as part of the profiling phase.

Prior work on IDK classifiers [1, 2, 4, 5] assumes that the predicted probability values are perfectly accurate, i.e. $\mathcal{P}[S] = \mathcal{Q}[S]$. This work relaxes that assumption, and assumes instead that the predicted probabilities may differ from the true values.

**Expected Execution Duration:** Consider an IDK cascade, with the classifiers in the following order, where $K_1$ to $K_n$ are IDK classifiers and $K_d$ is the deterministic classifier:

$$\left\langle K_1, K_2, \ldots, K_n, K_d \right\rangle \tag{1}$$

The first IDK classifier, $K_1$, will execute on each input; however, the second IDK classifier, $K_2$, will only execute in the event that the first classifier outputs IDK. Similarly, the third IDK classifier, $K_3$, will only execute in the event that the first two classifiers both output IDK, and so on. Since $C_i$ denotes the execution time of classifier $K_i$, the true expected execution duration $\mathbb{E}^T$ of the IDK cascade is equal to:

$$\begin{aligned}\mathbb{E}^T = \sum_{i=1}^{n} \left(C_i \times (1 - \mathcal{Q}[\{K_1, K_2, \ldots, K_{i-1}\}])\right) \\ + C_d \times (1 - \mathcal{Q}[\{K_1, K_2, \ldots, K_n\}])\end{aligned} \tag{2}$$

where $\mathcal{Q}[S]$ is the true probability that at least one of the classifiers in $S$ makes a real classification.

Note, we use $\mathbb{E}^T$ when referring to an expected execution duration computed using the true probabilities, $\mathcal{Q}[S]$, and $\mathbb{E}^P$ when referring to an expected execution duration computed using the predicted probabilities, $\mathcal{P}[S]$.

### 2.2 Algorithms Using Predictions

Safety-critical systems should have their correctness properties verified prior to deployment; such verification is currently typically done via some form of worst-case analysis. Worst-case analysis tends to lead to very conservative system designs that make inefficient use of computing resources almost all of the time. One approach to overcoming such conservatism is to go "Beyond Worst-Case Analysis" [12] by using *predictions* to guide an algorithm. Such predictions may be drawn from a variety of sources including machine learning or human intuition; they are often of uncertain provenance and so algorithms should not trust them entirely. Informally speaking, an algorithm

that uses predictions to make decisions should be designed in such a manner that it achieves the best of both worlds: providing improved performance when the prediction is accurate, without suffering too much of a performance degradation, in comparison with algorithms that are developed using traditional worst-case methods, when the prediction is inaccurate. The algorithmic framework of *algorithms using predictions* (see [11] for a comprehensive introduction and survey) offers a systematic approach to doing so. Algorithms designed according to this framework are characterized by three[4] properties (defined more formally later in this section):

(1) Consistency: When the predictions are accurate, the performance of the algorithm is excellent, often near-optimal.
(2) Robustness: When the predictions are inaccurate, the performance of the algorithm is not much worse than that of an algorithm that does not use predictions.
(3) Smoothness: The performance of the algorithm does not fall off drastically when the predictions have small errors: "the algorithm interpolates gracefully between the robust and consistent settings" [11].

In other words, the *consistency* of an algorithm that uses predictions characterizes its performance when the predictor is perfectly accurate, while *robustness* characterizes its performance guarantee regardless of the quality of the predictions.[5] *Smoothness* is related to the concept of sensitivity analysis as it pertains to safety-critical systems, and the general notion of stability: a minor change to the prediction should not cause a significant change in the algorithm's performance.

*Competitive analysis* is the standard approach to characterizing the performance of algorithms that are designed to operate in an online setting where the algorithm must make decisions without knowing all pertinent information. Recall that our goal is to develop an algorithm for synthesizing IDK cascades for which the expected execution duration is minimized. The competitive ratio of an algorithm that solves a minimization problem such as this is defined as follows.[6]

*Definition 1 (Competitive Ratio).* For any problem instance $X$, let $\texttt{OPT}(X)$ denote the cost achieved by an optimal clairvoyant algorithm, and let $\texttt{ALG}(X)$ denote the cost achieved by the online algorithm ALG, on this problem instance. The competitive ratio of ALG is the ratio

$$\max_{\text{all problem instances } X} \left\{ \frac{\texttt{ALG}(X)}{\texttt{OPT}(X)} \right\}$$

In the context of this paper:

- $X$ is a problem instance comprising the specification of a collection of several IDK classifiers and one deterministic classifier that all solve the same classification problem;
- OPT is clairvoyant in the sense that it knows the true probability values, $\boldsymbol{Q}[\cdot]$, associated with problem instance $X$, but it does not know what the behavior of each IDK classifier will be on any specific input.
- $\texttt{OPT}(X)$ is equal to the true expected execution duration, i.e., the cost, of the cascade that is constructed by OPT.
- $\texttt{ALG}(X)$ is equal to the true expected execution duration, i.e., the cost, of the cascade that is constructed by ALG.

[4] A fourth property, *learnability*, is sometimes also included in this list; we do not consider learnability in this paper.
[5] This property is commonly referred to as *resilience* [6] in the real-time and safety-critical systems literature; we call it 'robustness' in this paper, since that term is used in the algorithms using predictions literature.
[6] Competitive analysis for maximization objectives are defined similarly.

One can think of the algorithms using predictions framework as a generalization of competitive analysis. In this framework, the online algorithm gets a prediction, $P_X$, alongside the problem instance $X$. This prediction should enable the algorithm to make better decisions than it would be able to in the absence of this extra information. We want the algorithm to perform well even with inaccurate predictions; therefore, robustness is simply the competitive ratio of an algorithm which uses predictions:

*Definition 2 (Robustness).* Given problem instance $X$ and a prediction $P_X$ for $X$, let $\texttt{ALG}(X, P_X)$ denote the cost achieved by the online algorithm using this prediction, and $\texttt{OPT}(X)$ the cost achieved by the optimal clairvoyant algorithm. ALG's robustness on problem instance $X$ is defined as

$$\max_{\text{all predictions } P_X \text{ for } X} \left\{ \frac{\texttt{ALG}(X, P_X)}{\texttt{OPT}(X)} \right\}$$

In this and the following definition ALG is an algorithm using predictions; hence for the problem considered in this paper, $\texttt{ALG}(X, P_X)$ denotes the true expected execution duration of the cascade that is constructed by ALG when provided with the prediction $P_X$.

Consistency is the competitive ratio of an algorithm when it is provided with accurate predictions:

*Definition 3 (Consistency).* Given problem instance $X$ and a prediction $P_X$ for $X$, let $\texttt{ALG}(X, P_X)$ denote the cost achieved by the online algorithm using this prediction, and $\texttt{OPT}(X)$ the cost achieved by the optimal clairvoyant algorithm. ALG's consistency on problem instance $X$ is defined as

$$\min_{\text{all predictions } P_X \text{ for } X} \left\{ \frac{\texttt{ALG}(X, P_X)}{\texttt{OPT}(X)} \right\}$$

In addition to consistency and robustness, we want an algorithm that can gracefully handle small errors in predictions, a property referred to as *smoothness*. Although the idea behind smoothness is intuitive, it is harder to define precisely and generally, and so we will not attempt to do so, settling instead for an informal discussion. Given problem instance $X$, let us suppose that the perfect prediction for $X$ —the one that minimizes $(\texttt{ALG}(X, P_X)/\texttt{OPT}(X))$— is $\mathcal{P}_X$. If ALG is given some actual prediction $P_X$ for problem instance $X$, the *prediction error* is defined to be some function of $\mathcal{P}_X$ and $P_X$. This function is problem and algorithm specific, but informally is some difference function between the two parameters which is 0 when $P_X = \mathcal{P}_X$ and increases as the difference between the two increases. The smoothness property of an algorithm reflects the desire that its competitive ratio relative to the optimal increases slowly as the prediction error increases.

**Application to Real-Time Systems.** The applicability of the Algorithms Using Predictions framework to the design of safety-critical real-time systems has recently received some attention. A paper at RTSS last year [17] presented an algorithm using predictions for soft real-time scheduling of independent jobs upon multiprocessors in order to minimize the average response time. Earlier this summer, an ECRTS paper [3] provided a tutorial introduction to the framework from the context of hard-real-time systems.

## 3 SYNTHESIZING IDK CLASSIFIER CASCADES

In this section we design an algorithm, ALG, for synthesizing IDK classifier cascades from a given collection of classifiers characterized as described in Section 2.1. Recall that such a characterization includes estimates (or predictions) of the true probabilities of certain underlying events. Given such a characterization of classifiers, a latency constraint or deadline $\mathcal{D}$, and a robustness constraint $\mathbf{\Gamma}$, the IDK classifier cascade $S$ that is synthesized by ALG has a worst-case execution duration $\leq \mathcal{D}$. In

the case that the predictions are accurate, then the cost i.e., the true expected execution duration of the IDK classifier cascade $S$ is close to that of the IDK classifier cascade that would be synthesized by an optimal algorithm that knows the true probability values. If however the predictions are inaccurate, then the true expected execution duration of the IDK classifier cascade $S$ is no greater than $\mathbf{\Gamma}$ times as large as that of the optimal IDK classifier cascade, regardless of the magnitude of the prediction error.

In Section 3.1 below, we first describe how ALG guarantees performance even when the predictions are inaccurate; later in Section 3.2, we extend the algorithm to make good use of accurate predictions. Finally, we characterize ALG's consistency and smoothness properties in Sections 3.3 and 3.4. But first, let us formally state the problem that we seek to solve.

**Problem Statement**. A *problem instance* comprises the specification of a collection of $n$ IDK classifiers $K_1, K_2, \ldots, K_n$ and one deterministic classifier $K_d$ that all solve the same classification problem[7]; latency constraint $\mathcal{D}$; and a robustness constraint $\mathbf{\Gamma}$. Given such an instance, our algorithm ALG constructs an IDK classifier cascade such that the sum of the $C_i$ parameters of all the classifiers used in the cascade is $\leq \mathcal{D}$, and it thus satisfies the *latency constraint*. A *robustness constraint* must also be satisfied, as follows. Let OPT denote a hypothetical optimal algorithm that knows the true probabilities, of which the $2^n$ probability values in our model are predictions (estimates), and constructs an IDK classifier cascade that minimizes the expected execution duration subject to the latency constraint. Let $\mathbb{E}^T_{\mathsf{OPT}}$ denote the true expected execution duration of the cascade that is constructed by the optimal algorithm OPT. Similarly, let $\mathbb{E}^T_{\mathsf{ALG}}$ denote the true expected execution duration of the cascade that is constructed by ALG. It is required that

$$\frac{\mathbb{E}^T_{\mathsf{ALG}}}{\mathbb{E}^T_{\mathsf{OPT}}} \leq \mathbf{\Gamma}$$

regardless of how (in)accurate the predicted probability values are. Subject to satisfying both the latency and robustness constraints, the *optimization objective* is to **minimize** $\mathbb{E}^P_{\mathsf{ALG}}$, where $\mathbb{E}^P$ is the expected execution duration when the predictions are accurate; and hence also minimize $\mathbb{E}^P_{\mathsf{ALG}}/\mathbb{E}^P_{\mathsf{OPT}}$, which equates to the *consistency* measure.

### 3.1 Robustness Considerations

Robustness demands that the expected execution duration $\mathbb{E}^T_{\mathsf{ALG}}$ of the IDK classifier cascade constructed by an algorithm ALG remains within a multiplicative factor $\mathbf{\Gamma}$ of the expected execution duration $\mathbb{E}^T_{\mathsf{OPT}}$ of the cascade constructed by the optimal algorithm OPT. This constraint rules out ALG using certain cascades regardless of the predicted probabilities, as shown in the following example.

*Example 1.* Throughout this section we will use as a running example a problem instance with two IDK classifiers $K_1$ and $K_2$, and a deterministic classifier $K_d$, with execution durations as follows:

| $K_i$ | $K_1$ | $K_2$ | $K_d$ |
|---|---|---|---|
| $C_i$ | 5 | 8 | 20 |

Suppose that a robustness constraint of 3 is specified, $\mathbf{\Gamma} = 3.0$. For this example problem instance, it is a potential violation of the robustness constraint to have the deterministic classifier $K_d$ as the first and hence only classifier in the cascade, since the true probability $Q[\{K_1\}]$ of the classifier $K_1$

[7]Recall from Section 2.1 that in this specification each classifier $K_i$ is characterized by an execution duration $C_i$. Additionally, the efficacy of the collection of $n$ IDK classifiers, and their inter-dependences, are characterized by $2^n$ probability values, each representing an estimate of the probability that one of the $2^n$ subsets of the $n$ IDK classifiers will output a class other than IDK on any given input.

successfully classifying the input (i.e., not returning IDK) may be arbitrarily close to 1.0 and OPT, knowing this, could place $K_1$ at the start of the cascade followed by $K_d$. Any algorithm ALG that chooses to construct a cascade with $K_d$ as the first classifier would then have a performance ratio of

$$\frac{\mathbb{E}^T_{\mathsf{ALG}}}{\mathbb{E}^T_{\mathsf{OPT}}} = \frac{C_d}{C_1 + (1 - Q[\{K_1\}]) \times C_d} \overset{\text{[Since } Q[\{K_1\}] \approx 1\text{]}}{\approx} \frac{20}{5} = 4 > \mathbf{\Gamma}$$

which exceeds the permitted robustness constraint $\mathbf{\Gamma}$. Thus, ALG simply cannot construct a cascade with $K_d$ first even if the predicted probabilities for $K_1$ and $K_2$ to be successful are arbitrarily close to zero, since these predicted probabilities may be inaccurate and the robustness constraint must protect against such inaccurate predictions.

We note that in the absence of any predictions as to the probability of success of the IDK classifiers, and with no consideration of robustness, system designers may simply assume the worst-case, i.e. a very low probability of success for the IDK classifiers, and thus choose to employ only the deterministic classifier. As shown above, this can result in a high (i.e., poor) competitive ratio.

Example 1 motivates the notion of *safe extensions* to a partially-constructed IDK classifier cascade.

*Definition 4 (Safe Extension).* Let $S$ denote the set of classifiers in a partially-constructed cascade that does not violate either the latency or the robustness constraint. (Initially, $S$ is the empty set.) Let $K_\ell$ denote some classifier not in $S$. We say that it is **safe** to extend the cascade with classifier $K_\ell$ if doing so does not result in violating either the latency or the robustness constraint.

Observe that the deterministic classifier $K_d$ is always the last classifier in a cascade: this follows from (i) the requirement that successful classification, defined as returning a real class rather than IDK, must be achieved for all inputs; and (ii) the observation that there is no benefit in adding another classifier to the cascade after $K_d$, since such a classifier will never execute.

Having already added the set $S$ of classifiers to a cascade (with $K_d \notin S$), how do we determine whether it is safe to extend the cascade with some classifier $K_\ell \notin S$? We must first ensure that *the latency constraint is not violated*; this can be achieved by checking two cases:
*Case 1: $K_\ell$ is an IDK classifier, $K_\ell \neq K_d$.* In this case, we must ensure that there is sufficient time within the latency constraint to include $K_\ell$, all of the classifiers in $S$, and subsequently the deterministic classifier $K_d$.

$$\left( C_\ell + \sum_{K_i \in S} C_i \right) + C_d \leq \mathcal{D}$$

*Case 2: $K_\ell$ is the deterministic classifier, $K_\ell = K_d$.* In this case, we need only ensure that there is sufficient time within the latency constraint to include $K_\ell$, which is the deterministic classifier, and all of the classifiers in $S$.

$$\left( C_\ell + \sum_{K_i \in S} C_i \right) \leq \mathcal{D}$$

Next, we must ensure that *the robustness constraint is not violated.* We once again separately consider the two cases considered above.

*Case 1: $K_\ell$ is an IDK classifier, $K_\ell \neq K_d$.* In this case, it can be shown that the worst case occurs when the true success probability of each IDK classifier in $(S \cup \{K_\ell\})$ is zero, and that of the classifier not in $(S \cup \{K_\ell\})$ with minimum execution duration is equal to one (and therefore the one selected

for execution by OPT). Even if ALG were to execute this classifier not in $(S \cup \{K_\ell\})$ with minimum execution duration next (i.e., immediately after executing $K_\ell$), it is necessary that

$$\frac{C_\ell + \sum_{K_i \in S} C_i + (\min_{K_i \notin S \cup \{K_\ell\}}\{C_i\})}{\min_{K_i \notin S \cup \{K_\ell\}}\{C_i\}} \leq \mathbf{\Gamma}$$

$$\Leftrightarrow \quad \left(\frac{C_\ell + \sum_{K_i \in S} C_i}{\min\limits_{K_i \notin S \cup \{K_\ell\}}\{C_i\}} + 1\right) \leq \mathbf{\Gamma} \tag{3}$$

for the robustness bound to not be violated.

*Case 2: $K_\ell$ is the deterministic classifier, $K_\ell = K_d$.* When $K_\ell = K_d$, the cascade constructed by ALG is guaranteed to terminate after $K_d$ executes. In this case, the worst case occurs when the true probability of each IDK classifier in $S$ is equal to zero, and hence the expected duration for ALG equals $(C_\ell + \sum_{K_i \in S} C_i)$. And what about OPT's strategy? — this depends upon whether $S$ had included all the IDK classifiers or not.

- If not, then an argument essentially identical to the one above holds: in the worst case, the classifier not in $(S \cup \{K_\ell\})$ with minimum execution duration would have its true success probability equal to one and would therefore be the one that OPT executes.
- If $S$ includes all of the IDK classifiers, however, then in the worst case each IDK classifier's true success probability is equal to zero and OPT directly executes the deterministic classifier.

Putting both these possibilities together (and adopting the notational convention that the min of an empty set is $\infty$), we obtain the condition that

$$\frac{C_d + \sum_{K_i \in S} C_i}{\min\left(\min_{K_i \notin S}\{C_i\}, C_d\right)} \leq \mathbf{\Gamma} \tag{4}$$

for the robustness bound to not be violated.

*Example 2.* Consider again the instance of Example 1, comprising two IDK classifiers $K_1$ and $K_2$ with execution durations $C_1 = 5$ and $C_2 = 8$, and a deterministic classifier $K_d$ with execution duration $C_d = 20$. Initially the set $S$ of IDK classifiers in the cascade is equal to the empty set (i.e., $S = \{\}$), and we have a choice of all three classifiers to consider as the first classifier in the cascade:

- Suppose we choose the IDK classifier $K_1$, then the left hand side of (3) is greater than the left hand side of (4) and evaluates to:
$$\left(\frac{C_1}{C_2} + 1\right) = \left(\frac{5}{8} + 1\right) = \mathbf{1.625}$$
- Suppose we choose the IDK classifier $K_2$, then the left hand side of (3) is greater than the left hand side of (4) and evaluates to:
$$\left(\frac{C_2}{C_1} + 1\right) = \left(\frac{8}{5} + 1\right) = \mathbf{2.6}$$
- Suppose we chose the deterministic classifier $K_d$, then the left hand side of (4) evaluates to:
$$\left(\frac{C_d}{\min(C_1, C_2, C_d)}\right) = \left(\frac{20}{5}\right) = \mathbf{4}$$

Hence if the specified value of $\mathbf{\Gamma}$ is $\geq 4$, then any of the three classifiers may be at the start of the cascade without violating the robustness constraint. However if $2.6 \leq \mathbf{\Gamma} < 4$, then the deterministic classifier $K_d$ cannot be at the start of the cascade. Similarly if $1.625 \leq \mathbf{\Gamma} < 2.6$ then $K_1$ is the only

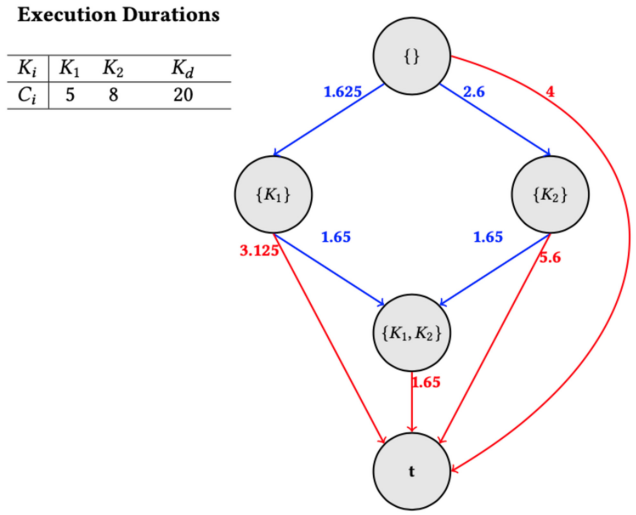



Fig. 3. Subset graph for the problem instance of Examples 1 and 2. Blue edges represent adding an IDK classifier, whereas red edges represent adding the deterministic classifier. The state labeled **t** represents all sets of classifiers that include the deterministic classifier $K_d$. Edge labels denote the robustness bound for that edge – the LHS's of (3) and (4) respectively for blue and red edges.

classifier that may be at the start of the cascade without violating the robustness constraint, while values of $\mathbf{\Gamma} < 1.625$ are *infeasible*, since no guarantee can be made that the robustness constraint will be met.

Figure 3 provides a graphical representation, which we refer to as the ***subset graph***, of the problem instance considered in Examples 1 and 2. (The execution durations of the classifiers –their $C_i$ parameters– are also listed in the figure for convenience.) In the subset graph, we represent each of the four subsets $\{\}$, $\{K_1\}$, $\{K_2\}$, and $\{K_1, K_2\}$ of the set of IDK classifiers as a separate node that is labeled with the set that it represents, while the node labeled **t** represents all sets of classifiers that include the deterministic classifier $K_d$. The edges in this figure correspond to extending a cascade:

- For any subset $S$ of the IDK classifiers and any IDK classifier $K_i \notin S$, the edge from the node labeled $S$ to the node labeled $(S \cup \{K_i\})$ denotes extending a cascade containing all the classifiers in $S$ (in any permutation) with the classifier $K_i \notin S$; such edges are depicted in blue.
- For any subset $S$ of the IDK classifiers, the edge from $S$ to **t** denotes extending a cascade containing all of the classifiers in $S$ (in any permutation) with the classifier $K_d$; such edges are depicted in red.

Each edge in the subset graph is labeled with a number that corresponds to the numerical value of the left hand side of the appropriate equation, i.e., (3) or (4). Observe that the labels on the outgoing edges from the node labeled "{}" are as computed in Example 2. The nodes and edges in the subset graph constitute a Directed Acyclic Graph (DAG). Each path in this DAG from the node labeled "{}" to the node labeled "**t**" corresponds to a complete IDK classifier cascade. Further, the robustness of any such complete IDK classifier cascade is given by the largest edge-label on the path.

### Identifying Feasible Cascades

Given an instance comprising $n$ IDK classifiers $K_1, K_2, \ldots, K_n$ and a single deterministic classifier $K_d$, ALG first constructs a subset graph that is the generalization of the one for $n = 2$ in Figure 3.

That is, this subset graph has a total of $2^n + 1$ nodes: one corresponding to (and labeled by) each of the $2^n$ subsets of the $n$ IDK classifiers and one, labeled as "**t**", to represent all the sets of classifiers that include the deterministic classifier $K_d$. *Feasible cascades* correspond to paths that traverse the subset graph from the node labeled "{}" to the node labeled "**t**", where: (i) each edge is labeled with a number $\leq \mathbf{\Gamma}$, and (ii) letting $S$ denote the second-last node on this path, i.e., the one immediately prior to the node labeled "**t**", the following holds:

$$\left(\sum_{K_i \in S} C_i\right) + C_d \leq \mathcal{D} \tag{5}$$

Note, multiple feasible cascades may exist for given values of $\mathbf{\Gamma}$ and $\mathcal{D}$.

*Example 3.* The table below lists the feasible cascades for the set of classifiers in Example 1 as a function of the robustness constraint $\mathbf{\Gamma}$ and latency constraint $\mathcal{D}$. Recall that Example 1 comprises two IDK classifiers $K_1$ and $K_2$ with execution durations $C_1 = 5$ and $C_2 = 8$, and a deterministic classifier $K_d$ with execution duration $C_d = 20$.

| | Deadline $\mathcal{D}$ | | | |
|---|---|---|---|---|
| $\mathbf{\Gamma}$ | $\geq 20$ | $\geq 25$ | $\geq 28$ | $\geq 33$ |
| $\geq 1.65$ | | | | $\mathcal{K}_{12d}$ |
| $\geq 2.6$ | | | | $\mathcal{K}_{21d}$ |
| $\geq 3.125$ | | $\mathcal{K}_{1d}$ | | |
| $\geq 4$ | $\mathcal{K}_d$ | | | |
| $\geq 5.6$ | | | $\mathcal{K}_{2d}$ | |

This table should be read in the following manner: for a given robustness constraint $\widehat{\mathbf{\Gamma}}$ and a given latency constraint $\widehat{\mathcal{D}}$,

(1) Identify the row closest to the bottom of the table for which $\widehat{\mathbf{\Gamma}}$ is at least as large as the value of $\mathbf{\Gamma}$ labeling the row.
(2) Identify the rightmost column for which $\widehat{\mathcal{D}}$ is at least as large as the value of $\mathcal{D}$ labeling the column.
(3) All cascades that are listed in the table at, above, and to the left of, the cell defined by the identified row and column are feasible cascades.

For instance, suppose that $\widehat{\mathbf{\Gamma}} = 6$ and $\widehat{\mathcal{D}} = 30$. The last row and the third column (the one labeled "$\geq 28$") are identified; hence the cascades $\mathcal{K}_d, \mathcal{K}_{1d}$, and $\mathcal{K}_{2d}$ are feasible cascades.

Since different permutations of the $n$ IDK classifiers result in distinct cascades, the total number of feasible cascades that can be synthesized may be as large as $\Theta(n!)$, which is $\Omega(n^n)$. Exhaustively considering each feasible cascade is computationally intractable even for relatively small $n$. (Recall that the number of classifiers used in any practical problem is likely to be limited to at most approximately $n = 12$, and while $2^{12} = 4096$, $12! \approx 480 \times 10^6$). In Section 3.2 below we describe how the predicted values of the probabilities characterizing the classifiers may be used to identify an appropriate IDK cascade in $O(n2^n)$ time.

### 3.2 Incorporating the Predictions

We saw above that certain paths from the node labeled "{}" to the node labeled "**t**" in the subset graph represent feasible IDK cascades that satisfy both the latency and the robustness constraints. We now describe how to compute the expected execution duration of any such feasible IDK cascade.

Observe that on any particular input, some prefix of the classifiers[8] in the cascade will execute for a duration equal to their $C_i$ parameters, while the remaining classifiers will not execute and so have an execution duration equal to zero. Let $S_o$ denote the classifiers comprising the cascade and $C$ the random variable that represents the execution duration of the cascade, we have

$$C = \sum_{K_i \in S_o} C_i \tag{6}$$

where the random variable $C_i$ denotes the execution duration of the classifier $K_i$ in the cascade, and hence takes a value of either zero or $C_i$ on any given execution of the cascade.

As stated above, our goal is to find the feasible cascade with minimum expected execution duration. By taking expectations on both sides of (6) and appealing to the *linearity of expectation* property [7, p.1198], we find that $\mathbb{E}[C]$, the expected execution duration of the cascade, is equal to the sum of the expected execution durations of the individual classifiers comprising the cascade:

$$\mathbb{E}[C] = \mathbb{E}\left[\sum_{K_i \in S_o} C_i\right] = \sum_{K_i \in S_o} \mathbb{E}\left[C_i\right] \tag{7}$$

(Here we're using the linearity of expectation property in the second equality.)
The expected execution duration $\mathbb{E}[C_i]$ of any individual classifier $K_i$ in the cascade is easily obtained by reasoning as follows. Let $S \subseteq S_o$ denote the set of classifiers preceding $K_i$ in the cascade. Classifier $K_i$ executes for a duration $C_i$ if and only if none of the classifiers in $S$ are successful, i.e., returns a real class rather than IDK. Since the probability of this happening is, by definition, equal to $(1 - Q[S])$, the expected execution duration of classifier $K_i$ is given by:

$$\mathbb{E}[C_i] = C_i \times \Big(1 - Q[S]\Big) \tag{8}$$

Let us now return to the subset graph representation of a problem instance. For each set of IDK classifiers $S$ and each classifier $K_i \notin S$, we would like to assign the edge from the node labeled $S$ to the node labeled $(S \cup \{K_i\})$ an edge-cost equal to the expected execution duration computed according to (8). Once we do so, the feasible cascade of minimum expected execution duration is exactly the one corresponding to the feasible path from the node labeled "{}" to the node labeled "**t**", for which *the sum of the edge labels is minimized.*

Unfortunately, we do not know the $Q[S]$ values that are needed in order to compute the expected durations, and thus the edge-costs, according to (8), since these are the unknown true probabilities, of which the $\mathcal{P}[S]$ values, obtained as discussed in [1], are estimates. Our algorithm, ALG, therefore uses the $\mathcal{P}[S]$ values as predictions of the values of $Q[S]$ for this purpose, and computes the edge-costs in the subset graph as follows: the edge from the node labeled $S$ to the node labeled $(S \cup \{K_i\})$ is assigned an edge-cost equal to:

$$C_i \times \Big(1 - \mathcal{P}[S]\Big) \tag{9}$$

Once all edge-costs are assigned, ALG computes the shortest path from the node labeled "{}" to the node labeled "**t**", from all those that satisfy the latency constraint, via (5), and *using only edges labeled with robustness values* $\leq \mathbf{\Gamma}$. This is a standard problem in graph traversal. Since there is a single start vertex, the problem can be solved using a standard topological ordering algorithm[9] [7, Chapter 24.2], in time that is linear in the number of edges plus vertices, i.e., in $O(n2^n)$ time. Further details of the operation of the algorithm are given in Section 4.

[8] Such a prefix may include the entire cascade, as happens when the deterministic classifier at the end of the cascade is invoked.
[9] See https://en.wikipedia.org/wiki/Topological_sorting

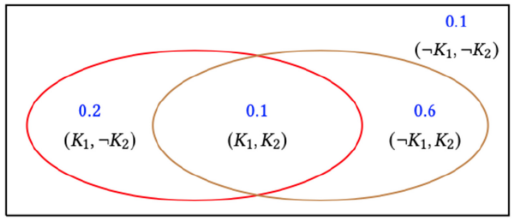



Fig. 4. Estimated probabilities for our running example.

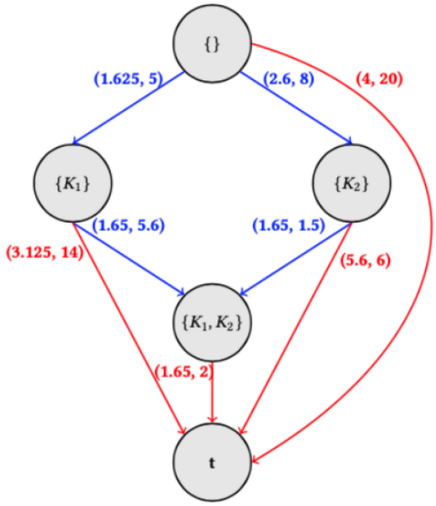



Fig. 5. Computing edge-costs for the problem instance of Examples 1 and 2. Each edge is labeled with the ordered pair (*robustness*, *edge-cost*), with the *robustness* values copied from Figure 3.

*Example 4.* We illustrate the process of computing edge-costs according to (9) for some of the edges in the subset graph, depicted in Figure 5, of our running problem instance when the probability estimates are as shown in the Venn diagram in Figure 4.

- Consider the edge leading from the vertex labeled "{}" to the one labeled "$\{K_2\}$". For this edge, $S \equiv \{\}$ and $C_i = C_2 = 8$; hence, the edge-cost is $8 \times (1 - 0) = \mathbf{8}$.
- Consider the edge leading from the vertex labeled "$\{K_2\}$" to the one labeled "$\{K_1, K_2\}$". For this edge, $S \equiv \{K_2\}$ and $C_i = C_1 = 5$; hence, the edge-cost is $5 \times (1 - 0.7) = 5 \times 0.3 = \mathbf{1.5}$.
- Consider the edge leading from the vertex labeled "$\{K_1, K_2\}$" to the one labeled "$t$". For this edge, $S \equiv \{K_1, K_2\}$ and $C_i = C_d = 20$; hence, the edge-cost is $20 \times (1 - 0.9) = 20 \times 0.1 = \mathbf{2}$.

We have also computed the edge-costs for all the other edges in the subset graph; these are shown in Figure 5.

### 3.3 Consistency

The *consistency* property of an algorithm using predictions refers to its performance when the predictions provided to it are accurate. An algorithm with good consistency performs well when given accurate predictions. While there is often a tension between consistency and robustness [11], it is typically fairly easy to directly trade one off against the other: "A practitioner who has high

confidence in the predictions may aim for high consistency and low robustness [...] a risk-averse decision maker may choose [to limit] the benefit of the predictions, but also the additional cost when they turn out to be incorrect" [11].

Unfortunately, for the problem considered in this paper, the trade-off between consistency and robustness is not so straightforward. To understand why, let us revisit the table in Example 3, listing feasible cascades as a function of the provided values of the robustness parameter $\mathbf{\Gamma}$ for our running example of two IDK classifiers and one deterministic classifier. Suppose that we have a latency constraint $\widehat{\mathcal{D}} = 35$, and hence all of the columns are in play. Additional cascades become feasible as the permitted robustness constraint $\widehat{\mathbf{\Gamma}}$ increases from $(1.65-\epsilon)$ to 1.65; from $(2.6-\epsilon)$ to 2.6; from $(3.125-\epsilon)$ to 3.125; from $(4-\epsilon)$ to 4; and from $(5.6-\epsilon)$ to 5.6, where $\epsilon$ is an arbitrarily small positive number. Hence, a slight change in robustness can yield an additional feasible cascade, which may have very different consistency properties — this may result in discontinuities in the graph plotting robustness versus consistency. To our knowledge, such discontinuities in the relationship between consistency and robustness are relatively rare in the prior literature on algorithms using predictions.

Once all the robustness constraints and the edge-costs in the subset graph have been computed (as illustrated in Figure 5 for our running example), ALG's consistency for a given value of the robustness constraint $\widehat{\mathbf{\Gamma}}$ can be determined as follows. (Recall that consistency is defined to be the ratio of $\mathbb{E}^P_{\mathsf{ALG}}$ to $\mathbb{E}^P_{\mathsf{OPT}}$ when the predictions are accurate).

- When the predictions are accurate, $\mathbb{E}^P_{\mathsf{ALG}}$ is exactly the sum of the edge-costs of the shortest path from the node labeled "{}" to the node labeled "**t**" that satisfies the latency constraint, (5), using only edges labeled with robustness values $\leq \widehat{\mathbf{\Gamma}}$.
- Since OPT does not need to worry about robustness constraints, $\mathbb{E}^P_{\mathsf{OPT}}$ is equal to the length of shortest path from the vertex labeled "{}" to the vertex labeled "**t**" that satisfies the latency constraint, (5), *regardless of the robustness* values on the edges.

We illustrate this consistency calculation in the example below.

*Example 5.* Consider our running example problem instance, with a latency constraint of $\widehat{\mathcal{D}} \geq 33$ and a robustness constraint of $\widehat{\mathbf{\Gamma}} \in [1.65, 2.6)$. It can be verified from Figure 5 that

- ALG constructs the cascade $(K_1, K_2, K_d)$, and hence $\mathbb{E}^P_{\mathsf{ALG}} = 5 + 5.6 + 2 = \mathbf{12.6}$; and
- OPT constructs the cascade $(K_2, K_1, K_d)$, and hence $\mathbb{E}^P_{\mathsf{OPT}} = 8 + 1.5 + 2 = \mathbf{11.5}$

ALG therefore has a consistency equal to

$$\frac{\mathbb{E}^P_{\mathsf{ALG}}}{\mathbb{E}^P_{\mathsf{OPT}}} = \frac{12.6}{11.5} \approx 1.1 \ .$$

in this case. If, however, the specified robustness constraint was $\widehat{\mathbf{\Gamma}} \geq 2.6$, then ALG can also construct the optimal cascade $(K_2, K_1, K_d)$, hence $\mathbb{E}^P_{\mathsf{ALG}} = \mathbb{E}^P_{\mathsf{OPT}}$ and ALG has a consistency of 1.0.

### 3.4 Smoothness

We next turn our attention to smoothness. As stated in Section 2.2, the smoothness property of an algorithm using predictions relates to its sensitivity to small errors in the predictions: a minor inaccuracy should not cause a large change to the algorithm's performance.[10]

[10] Although the *algorithms using predictions* framework generally requires performance to change linearly with prediction error, in this work we replace the need for linearity with a weaker continuity requirement.

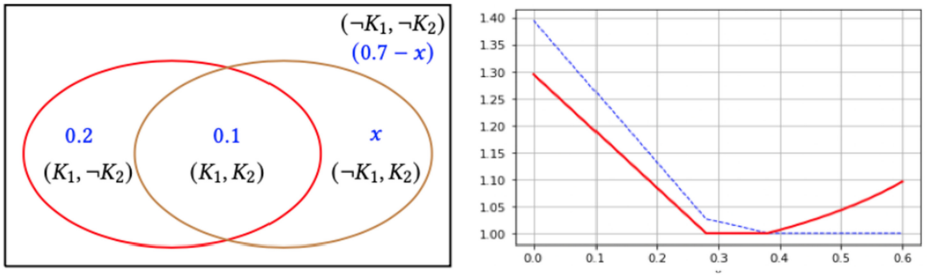



Fig. 6. Smoothness: plotting performance ratio as a function of prediction error, see Example 6.

For a given robustness constraint $\widehat{\Gamma}$ and latency constraint $\widehat{\mathcal{D}}$, ALG uses the predicted probability values to construct one particular cascade, whereas OPT may ignore $\widehat{\Gamma}$ and select a different cascade for different true probability values. ALG's performance ratio when the true probability values are exactly the same as the predicted ones (i.e., prediction error is equal to zero) is given by its consistency parameter. Let us now examine what happens to the performance ratio as the prediction error increases from zero: i.e., the true probability values differ from the predicted ones. When the prediction error is relatively small (i.e., the true probability values are not too far off from the predicted ones), OPT may continue with the same optimal cascade as it uses when prediction error is zero. However, as the error becomes larger and the true probability values drifts farther from the predicted values, OPT may choose to use a different cascade; when that happens, the rate of change in the expected execution duration of the cascade constructed by OPT may change at a different rate than with the cascade that was previously used by OPT. The impact of this on smoothness is best illustrated by considering an example.

*Example 6.* We revisit our running example, the predicted probabilities for which are depicted in Venn diagram form in Figure 4. For a specified latency constraint $\widehat{\mathcal{D}} \geq 33$ and a specified robustness constraint $\widehat{\Gamma} \in [1.65, 2.6)$, we saw in Example 5 that ALG synthesizes the cascade $(K_1, K_2, K_d)$ and OPT synthesizes the cascade $(K_2, K_1, K_d)$.

Now, suppose that the probability estimate for the region labeled "$(\neg K_1, K_2)$" in the Venn diagram in Figure 4 is in fact ***inaccurate***. Let us examine the impact of this inaccuracy as the true value of this probability, which we denote by $x$, takes values in the range $[0.0, 0.6]$, with the "slack", i.e., 0.6 minus this true probability, transferred to the region labeled "$(\neg K_1, \neg K_2)$", which consequently has a true probability of $(0.7 - x)$. These true probabilities are depicted in the Venn diagram to the left[11] in Figure 6. The red line in the graph to the right in Figure 6 plots the performance ratio (i.e., $\mathbb{E}^T_{\mathsf{ALG}}/\mathbb{E}^T_{\mathsf{OPT}}$) as a function of $x$. The dashed blue line in the same graph plots the performance ratio for the cascade $(K_2, K_1, K_d)$; recall that this cascade is infeasible from ALG's perspective since it violates the specified robustness constraint.
We highlight some observations from the plot:

- When the prediction error is zero (i.e., the true probability equals the predicted value: $x = 0.6$), the performance ratio is indeed 1.1 as we had determined in Example 5 when computing ALG's consistency.
- The performance ratio improves (i.e., decreases) as $x$ decreases from 0.6 to to 0.38. This decrease is because the expected execution duration of ALG's cascade, $(K_1, K_2, K_d)$, changes

[11] We note in passing that for $x = 0.0$ the IDK classifiers $K_1$, $K_2$ are *fully dependent*, whereas for $x = \left(\frac{7}{30}\right) = 0.2\bar{3}$, they are *independent*: since $\Pr[K_1] = 0.2 + 0.1 = \mathbf{0.3}$ and $\Pr[K_2] = 0.1 + \frac{7}{30} = \frac{1}{3}$, we have $\Pr[K_1 \wedge K_2] = 0.1 = \Pr[K_1] \times \Pr[K_2]$.

at a different rate with error than the rate at which the expected execution duration of OPT's cascade, $(K_2, K_1, K_d)$, changes.

- At $x = 0.38$, the performance ratio equals one. This is because it may be verified that for $x \in [0.28, 0.38]$, $(K_1, K_2, K_d)$ becomes OPT's favored cascade as well, hence changes in the value of $x$ impact both ALG and OPT equally.
- The ratio worsens, i.e., increases, as $x$ decreases below 0.28. It may be verified that $(K_1, K_d)$ is OPT's favored cascade for these values of $x$, and so the ratio is

$$\frac{C_1 + 0.7 \times C_2 + (0.7 - x) \times C_d}{C_1 + 0.7 \times C_d} = \frac{5 + 5.6 + 14 - 20x}{19} = \frac{24.6 - 20x}{19}$$

which increases as a straight line as $x$ decreases, from 1.0 to $\left(\frac{24.6}{19}\right) \approx 1.3$.

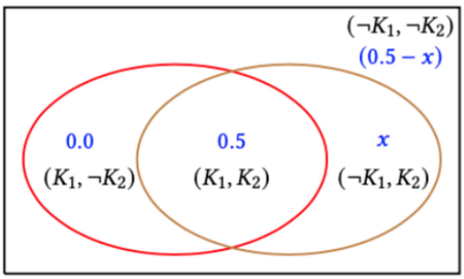
The plot also reveals that if the robustness constraint were large enough, i.e., $\widehat{\Gamma} \geq 2.6$, to allow ALG to construct the cascade $(K_2, K_1, K_d)$ then that cascade, whose performance ratio is depicted in blue, would only be superior to the cascade $(K_2, K_2, K_d)$ depicted in red for values of $x \in [0.38, 0.6]$; for $x < 0.38$, the latter cascade has a better, i.e., lower, performance ratio.

The apparent piece-wise linear nature of the lines in the graph to the right in Figure 6 is co-incidental; this is illustrated in the following example.

*Example 7.* Consider a problem instance with two IDK classifiers $K_1$ and $K_2$ and a deterministic classifier $K_d$, with execution durations

$$C_1 = (8 - \epsilon); C_2 = 8; \text{ and } C_d = 16$$

where $\epsilon \in \mathbb{R}_+$ is an arbitrarily small positive real number. *True* probabilities are as depicted in the Venn diagram below, with $x$ taking some specific value in the range $[0.0, 0.5)$:




For $\widehat{\mathcal{D}} \geq 32$ and $\widehat{\Gamma} = 2.0$, it can be verified that the only cascade satisfying the robustness constraint is $(K_1, K_2, K_d)$, and this is therefore the one constructed by ALG. Let us evaluate its smoothness, assuming that the predicted probability for the region $(\neg K_1, K_2)$ is arbitrarily close to 0.5. For values of $x$ close to 0.5, i.e., when the true probability for this region is also close to 0.5, it is evident that OPT would choose the cascade $(K_2, K_d)$, since $K_2$ would be have a true probability of successful classification close to 1. The performance ratio is therefore

$$\frac{\mathbb{E}^T_{\text{ALG}}}{\mathbb{E}^T_{\text{OPT}}} = \left(\frac{C_1 + 0.5 \times C_2 + (0.5 - x) \times C_d}{C_2 + (0.5 - x) \times C_d}\right) = \left(\frac{(8 - \epsilon) + 4 + (8 - 16x)}{8 + (8 - 16x)}\right) = \left(1 + \frac{4 - \epsilon}{16(1 - x)}\right)$$
$$\approx \left(1 + \frac{1}{4(1 - x)}\right) \tag{10}$$

Table 1. ResNet Case Study

| ResNet | | | | | |
|---|---|---|---|---|---|
| −18 | −34 | −50 | −152 | Count | $\mathcal{P}[S]$ |
| $A$ | $B$ | $C$ | $D$ | | |
| 0 | 0 | 0 | 0 | 15880 | 0 |
| 0 | 0 | 0 | 1 | 3011 | 0.5902 |
| 0 | 0 | 1 | 0 | 1423 | 0.545 |
| 0 | 0 | 1 | 1 | 2465 | 0.64564 |
| 0 | 1 | 0 | 0 | 914 | 0.49216 |
| 0 | 1 | 0 | 1 | 960 | 0.63488 |
| 0 | 1 | 1 | 0 | 545 | 0.60016 |
| 0 | 1 | 1 | 1 | 3382 | 0.66942 |
| 1 | 0 | 0 | 0 | 649 | 0.4284 |
| 1 | 0 | 0 | 1 | 452 | 0.62476 |
| 1 | 0 | 1 | 0 | 304 | 0.5847 |
| 1 | 0 | 1 | 1 | 1208 | 0.66412 |
| 1 | 1 | 0 | 0 | 275 | 0.54442 |
| 1 | 1 | 0 | 1 | 609 | 0.65394 |
| 1 | 1 | 1 | 0 | 500 | 0.62218 |
| 1 | 1 | 1 | 1 | 17423 | 0.6824 |
| Totals | | | | 50000 | 1.00 |

| Classifier | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| $C_i$ (ms) | 22.6 | 37.5 | 49.5 | 125.1 | 250 |

Since $x$ is in the denominator in the expression above, it is evident that the value of this ratio changes non-linearly with $x$. We may therefore conclude that the performance ratio as a function of error is not necessarily piece-wise linear.

## 4 CASE STUDY

In Section 3, we showed how the algorithms using predictions framework can be applied to the problem of choosing the optimum IDK classifier cascade, subject to a latency constraint and a constraint on robustness. Recall that the goal of our algorithm ALG is to determine the IDK classifier cascade that has the minimum expected duration in the case that the predictions are correct, among all IDK classifier cascades that meet the constraints.

In this section, we provide further details of the operation of ALG and apply it to a proof-of-concept case study based on real classifiers from the domain of image classification. This proof-of-concept is based on the ResNet case study[12] detailed in [1]. The case study comprises four classifiers, which are all variants of the ResNet Deep Residual Network [8]: ResNet-18, ResNet-34, ResNet-50, and ResNet-152. (The number $x$ in ResNet-$x$ denotes the number of layers of neurons in the network, with larger values typically yielding more accurate classifiers that have longer execution times.) In addition, we considered a hypothetical deterministic classifier for the same problem.

Abdelzaher et al. [1] used a validation set of 50,000 images with labels from the ImageNet Large Scale Visual Recognition Challenge data set [13] to populate the $\mathcal{P}[S]$ probability values for the $2^n$ subsets of the ResNet classifiers. Here, we reuse their data. The first four columns in Table 1 correspond to the four classifiers ResNet-18, ResNet-34, ResNet-50, and ResNet152. A zero in one of these columns indicates that the classifier returns IDK, whereas a 1 indicates that it returns a real class. Each of the rows thus represents one of the $2^4 = 16$ disjoint regions of the probability

[12]We would like to thank Abdelzaher et al. [1] for allowing us to use the data from this case study.

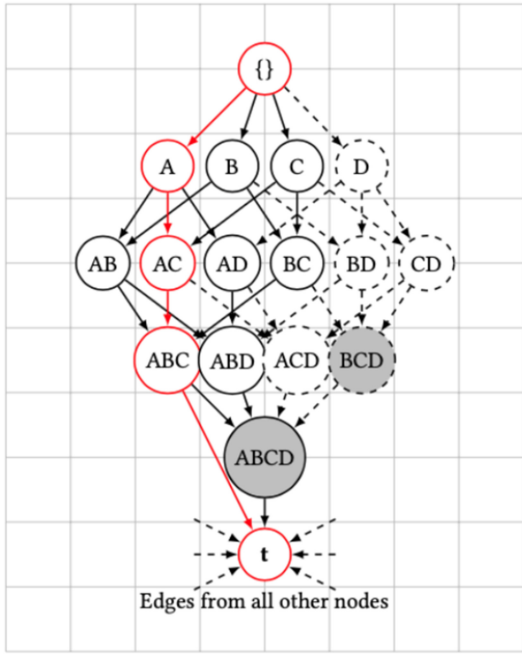



Fig. 7. Directed Acyclic Graph (DAG) representation of the subsets of IDK classifiers (nodes), with arrows (edges) representing the addition of a further classifier.

space. The column labeled "Count" indicates how many of the 50,000 input samples used in profiling resulted in that pattern of classifier outputs. Finally, the column labeled "$\mathcal{P}[S]$" contains the computed probability (i.e., prediction) that *at least one* of the subset of classifiers indicated by 1's in the first four columns will successfully classify any given input. Table 1 also details the execution times $C_i$ of the classifiers on an NVIDIA Jetson TX2, using the 95-percentile value as a proxy for the worst-case. Also listed is a hypothetical deterministic classifier $E$, with an arbitrarily assigned execution time of 250ms; longer than that of any of the IDK classifiers.

Figure 7 illustrates the complete subset graph for the ResNet case study, with the four IDK classifiers $A$, $B$, $C$, and $D$, and the deterministic classifier $E$. The subset graph is a Directed Acyclic Graph (DAG) where each node corresponds to a unique subset of the IDK classifiers. In addition, there is start node, denoted by {}, that represents the empty set of classifiers, and an exit node, denoted by **t**, that represents all the subsets that contain the deterministic classifier. The nodes are connected via directed edges. An edge connects each node representing a subset of IDK classifiers with each of the nodes that represents the same subset extended via the addition of exactly one further classifier. For example, with four IDK classifiers $A$, $B$, $C$, and $D$, there is a directed edge from the node $AB$ to each of the nodes $ABC$ and $ABD$. There is also a directed edge from each of the subset nodes to the exit node. We explain the meaning of the colors and dashed lines in Figure 7 later, in the context of a concrete problem instance.

**Algorithm Description.** We now give a detailed description of how algorithm ALG constructs the subset graph, computes the information associated with the nodes and edges, eliminates options that do not meet the constraints, and finally computes the optimal IDK classifier cascade that minimizes the expected execution duration based on the predictions (probabilities), subject to compliance with the latency and robustness constraints.

(1) The subset graph is constructed in topological order, starting from the node in layer zero representing the empty set. The nodes in subsequent layers are then added in order, starting with nodes in the first layer, representing subsets of cardinality 1, followed by nodes in the second layer, representing subsets of cardinality 2, and so on, until finally the exit node representing all subsets that include the deterministic classifier is added.
(2) On the addition of a node, all of the incoming edges to that node are also added. (Recall that an edge links a node associated with a subset $S$ with a node associated with that subset extended via the addition of exactly one further classifier). The worst-case execution duration of the new node is then calculated, along with the robustness and cost of its incoming edges.
(3) The worst-case execution duration of a node is calculated as the sum of the execution times of the IDK classifiers in the subset associated with the node, plus the execution time of the deterministic classifier, since that is always included in the cascade.
(4) The robustness value for an edge is calculated via (3) for edges that are incoming to a normal node, and via (4) for edges that are incoming to the exit node. The cost of an edge is calculated as $C_i \times (1 - \mathcal{P}[S])$ where $C_i$ is the execution time of the classifier added in going from the predecessor node to the successor node indicated by the edge, and $S$ is the subset of classifiers associated with the predecessor node.
(5) Once these parameters have been computed, then they are compared to the robustness and latency constraints. Any edges that do not meet the robustness constraint are removed, along with the new node if it is orphaned by the removal of those edges, i.e., if it no longer has any incoming edges. Further, the new node is removed if its worst-case execution duration exceeds the latency constraint. (Note, edges are not created to subsequent layers from nodes that have been removed).
(6) Once the above constraint-based pruning has taken place, if the new node has not been removed, then the minimum expected execution duration associated with it is computed. This is done by computing the minimum over the node's incoming edges of the cost of an incoming edge plus the minimum expected execution duration associated with the predecessor node that the edge originates from. A back pointer is used to record which predecessor node led to the minimum expected execution duration of the new node.
(7) Once the exit node is reached and processed as described above, then the optimal IDK classifier cascade compliant with the constraints can be recovered by tracing the back pointers back up the graph until the start node is reached. The expected execution duration of the optimal IDK classifier cascade is the value recorded in the exit node. Finally, if there is no path back to the start node, i.e. if the exit node has been removed, then this means that no feasible solution exists given the constraints.

We now return to the ResNet case study. For reasons of clarity, rather than annotate the subset graph in Figure 7 with the robustness values, these values are listed in Table 2. Note that due to the form of the formula for robustness, i.e., (3), the robustness values associated with incoming edges that represent the addition of a single IDK classifier and end at the same node have the same value. These values are given in the second column of Table 2. For example, the incoming edges from nodes *AB*, *AC*, and *BC* to node *ABC* all have robustness values of 2.31. The robustness values associated with edges that end at **t** typically take different values, as computed according to (4). These values are given in the third column of Table 2. For example, the edge from node *ABC* to **t** has a robustness value of 4.31, whereas the edge from node *ABCD* to **t** has a robustness value of 3.88. Table 2 also indicates, in the fourth column, the worst-case execution duration of the subset of classifiers listed in the first column plus the deterministic classifier, i.e., the worst-case execution duration of a complete IDK classifier cascade.

Table 2. Robustness Values for the ResNet Case Study

| Subset (Node) | Robustness of edges | | Worst-case duration (ms) (Node plus **t**) |
|---|---|---|---|
| | Incoming to node | Outgoing to **t** | |
| A | 1.60 | 7.27 | 272.64 |
| B | 2.66 | 12.70 | 287.52 |
| C | 3.18 | 13.23 | 299.45 |
| D | 6.52 | 16.57 | 375.08 |
| AB | 2.22 | 6.27 | 310.16 |
| AC | 2.92 | 8.58 | 322.09 |
| AD | 4.94 | 10.60 | 397.72 |
| BC | 4.84 | 14.88 | 336.97 |
| BD | 8.18 | 18.22 | 412.60 |
| CD | 8.71 | 18.75 | 424.53 |
| ABC | 2.31 | 4.31 | 359.61 |
| ABD | 4.75 | 8.80 | 435.24 |
| ACD | 6.26 | 11.92 | 447.17 |
| BCD | 10.37 | 20.41 | 462.05 |
| ABCD | 2.88 | 3.88 | 484.69 |

We now consider a concrete example based on the above ResNet case study data. We assume that there is a specified latency constraint, $\mathcal{D} = 450$ms, and a specified robustness constraint, $\Gamma = 5$. From the information in the fourth column of Table 2, we see that the latency constraint precludes the use of IDK classifier cascades composed from the subsets *BCD* and *ABCD*. In Figure 7, the nodes representing these subsets are shaded in gray indicting that they do not meet the latency constraint. From the information in the second and third columns of Table 2, we see that the robustness constraint implies that nodes *A*, *B*, and *C* in the subset graph can be reached from the start node; nodes *AB*, *AC*, *AD*, and *BC* can be reached from nodes *A*, *B*, and *C*; nodes *ABC* and *ABD* can be reached from nodes *AB*, *AC*, *BC*, and *AB*, *AD* respectively; and lastly node *ABCD* can be reached from nodes *ABC* and *ABD*. Further, the exit node **t** can only be reached from nodes *ABC* and *ABCD*. All other edges in the graph do not meet the robustness constraint and are shown In Figure 7 as dashed lines. Nodes that are orphaned by the removal of edges that do not meet the robustness constraint are shown with a dashed border. Together, the latency and robustness constraints imply that the only viable subset of classifiers that can form a feasible IDK classifier cascade is *ABCE*. There are however a number of different orders in which the classifiers in this subset could be run. These different orderings result in different expected execution durations.

It is not necessary to examine all possible orderings (permutations) of the classifiers in order to determine the optimal IDK classifier cascade. Rather, steps (6) and (7) of algorithm ALG enable the path through the subset graph that has the lowest expected execution duration, compliant with the constraints, to be obtained in $O(n2^n)$ time, since there are at most $n$ outgoing edges to each of $2^n$ nodes. The algorithm therefore operates in exponential time, rather than the factorial time required to fully examine all possible paths, i.e., permutations. This reduction in complexity is possible due to the fact that the cost of an edge represents the increase in the expected execution duration that occurs on adding an extra classifier $K_i$ to the subset of classifiers associated with the predecessor node that the edge originates from. Importantly this is the case irrespective of the path taken to reach the predecessor node, i.e., the actual sequence in which the classifiers associated with the predecessor node are run. For example, the expected execution duration of classifier *E* preceded by the classifiers in the subset *ABC* is 94.46ms, irrespective of which one of the six possible orders

Table 3. Selected Expected Execution Durations for the ResNet Case Study

| Edge | Cost of edge (ms) | Node | Min cost to node (ms) | From |
|---|---|---|---|---|
| $\{\} \rightarrow A$ | 16.90 | $A$ | 16.90 | $\{\} \rightarrow$ |
| $\{\} \rightarrow B$ | 27.80 | $B$ | 27.80 | $\{\} \rightarrow$ |
| $\{\} \rightarrow C$ | 37.00 | $C$ | 37.00 | $\{\} \rightarrow$ |
| $A \rightarrow AB$ | 15.89 | $AB$ | 32.79 | $A \rightarrow$ |
| $A \rightarrow AC$ | 21.15 | $AC$ | 38.05 | $A \rightarrow$ |
| $B \rightarrow AB$ | 8.58 | $BC$ | 46.59 | $B \rightarrow$ |
| $B \rightarrow BC$ | 18.79 | $ABC$ | 49.59 | $AC \rightarrow$ |
| $C \rightarrow AC$ | 7.69 | $\mathbf{t}$ | 144.05 | $ABC \rightarrow$ |
| $C \rightarrow BC$ | 12.65 | | | |
| $AB \rightarrow ABC$ | 16.86 | | | |
| $AC \rightarrow ABC$ | 11.55 | | | |
| $BC \rightarrow ABC$ | 6.76 | | | |
| $ABC \rightarrow \mathbf{t}$ | 94.46 | | | |

in which the classifiers in subset *ABC* may have been executed. This holds because the probability that all three of those classifiers will return IDK is the same irrespective of their running order.

Table 3 illustrates the costs associated with selected edges and nodes from the subset graph for the ResNet case study. The values in the second column are the costs corresponding to the increase in the expected execution time due to adding the classifier corresponding to traversing the edge specified in the first column. The values in the fourth column represent the minimum value of the expected execution time on completing execution of the subset of classifiers specified in the third column, compliant with the robustness constraints and latency constraints. The fifth column indicates the previous node and edge that led to that minimum cost value. Tracing back from node $\mathbf{t}$, representing all subsets that include the deterministic classifier $E$, recovers the sequence $A \rightarrow AC \rightarrow ABC \rightarrow \mathbf{t}$, and hence the optimal IDK classifier cascade compliant with the constraints, which is *ACBE* run in that order.

Given a robustness constraint, $\mathbf{\Gamma} = 5$, and a latency constraint, $\mathcal{D} = 450$ms, the IDK classifier cascade chosen by the algorithm ALG is *ACBE*, which has a robustness of 4.31, an expected execution duration of $\mathbb{E}^P_{\mathsf{ALG}} = 144.05$ms, and a worst-case execution duration of 359.61ms. Ignoring the constraints, algorithm OPT selects *ACE* as the optimal IDK classifier cascade, which has an expected execution duration of $\mathbb{E}^P_{\mathsf{OPT}} = 141.87$ms, and a worst-case execution duration of 322.09ms. ALG therefore has a consistency of:

$$\frac{\mathbb{E}^P_{\mathsf{ALG}}}{\mathbb{E}^P_{\mathsf{OPT}}} = \frac{144.05}{141.87} \approx 1.015 \ .$$

If the robustness constraint were relaxed to $\mathbf{\Gamma} \geq 8.58$, then ALG would also select *ACE* as the optimal IDK classifier cascade.

This proof-of-concept case study based on data obtained for four real classifiers using the ResNet deep neural network architecture illustrates both the detailed behavior of the algorithm ALG and the efficacy of the approach, integrating the algorithms with predictions framework with the methodology introduced in [1]. Thus enabling optimal IDK classifier cascades to be obtained that comply with both latency and robustness constraints. The case study makes apparent the trade offs in complying with a robustness constraint. Choosing the best IDK classifier cascade without considering robustness could potentially result in performance that was more than $\mathbf{\Gamma} = 8$ times worse than optimal if the predictions were incorrect. However, complying with a maximum

*robustness* constraint of $\Gamma = 5$ only reduces performance in the case where the predictions are correct by less than 2%, as indicated by the *consistency* measure.

## 5 CONCLUSIONS

The verification of safety-critical systems normally takes the form of worst-case analysis. Such analysis is safe, but is often very conservative and does not utilize lower-assurance knowledge (e.g., about the typical run-time behavior of the system). In this work we adapt the algorithms using predictions framework to obtain a benefit when a system behaves according to predictions, while bounding the performance loss that can occur if the predictions are inaccurate or even erroneous.

The application we have chosen to illustrate this approach is the optimization of the choice and ordering of the execution of a collection of IDK classifiers, by synthesizing a cascade out of some of them. An IDK classifier either outputs a real class or defers by outputting "I Don't Know" (IDK). An output of IDK means that a further classifier must be executed, whereas an output that is a real class terminates the sequence. An optimal ordering is one that minimizes the expected execution duration required for successful classification. IDK classifiers are commonly based to Deep Learning and related AI techniques. The extensive training data used in the development of these components allows their probabilities of success to be measured via profiling [1]. These measured values can then be used as predictors of the likely success that will be experienced by the IDK classifiers at run-time (with success defined as a non-IDK output).

The algorithm developed in this paper to order IDK classifiers in a cascade exhibits three key properties in terms of its performance, i.e., the expected execution duration of the cascade.

> CONSISTENCY: when the predicted probabilities are accurate, then the algorithm performs close to the optimal method.
> SMOOTHNESS: when the predictions have a small error then the algorithm performs relatively close to the optimal method.
> ROBUSTNESS: when the predictions have a large error, or are arbitrarily poor, then the algorithm's performance is bounded to be no more than $\Gamma$ times as large as the optimal method. The design parameter $\Gamma$ is termed the robustness constraint.

We believe the Algorithms using Predictions framework has potential to impact the system design and analysis process for safety-critical Cyber-Physical Systems (CPS's). Moving forward we plan to explore additional examples within the domain of safety-critical real-time CPS's in which uncertainty leads to elements within the design that cannot be known with complete confidence, but which can nevertheless be estimated and these estimates used to produce predictions of likely run-time characteristics. The predictions as well as worst-case parameters together with a specified robustness constraint can then be used to develop algorithms that perform much more effectively than ones that only take account of the worst-case parameter values.

There are multiple sources of uncertainty in complex safety-critical real-time CPS's in addition to those (of the kind considered in this manuscript) that are introduced by the use of ML-based components in safety-critical systems. For each such source, the availability of good predictions could potentially be used to improve performance. One particularly important source of uncertainty in such systems is the execution duration– the *worst-case execution time* or WCET [16] – of pieces of code: the WCET estimates that are used for verification and certification of safety-critical systems are notoriously conservative over-approximations, and it therefore behoves us to examine whether predictions of less conservative (and more realistic) execution times can be meaningfully exploited within the algorithms using predictions framework to obtain more resource-efficient implementations of safety-critical systems.

Another source of uncertainty in event-triggered systems where triggering events may occur repeatedly is the pattern of occurrence of triggering events; in many conservative worst-case models of real-time workload, this is characterized by having a *period* parameter [10] designate the minimum duration that must elapse between successive triggerings. Worst-case scheduling techniques must allocate resources assuming that successive triggerings will occur at the maximum rate (i.e., separated by exactly the period parameter); if predictions are available of a larger inter-triggering duration, the algorithms using predictions framework could perhaps be used to design algorithms that make efficient use of computational resources when such predictions are accurate, without facing catastrophic consequences if they are not.

Finally, we note that the work in this paper focused on the sequential execution of IDK classifiers cascades on a single compute resource. The reason for this is that each IDK classifier is typically implemented as a neural network that runs on a dedicated GPU. As such there is parallelism inherent in the execution of each classifier, with the classifiers running in sequence if IDK is returned. In future we also intend to explore the potential for extending the algorithms with predictions framework to the problem of running classifiers in parallel on multiple compute resources [1].

## REFERENCES


[1] T. Abdelzaher, K. Agrawal, S. Baruah, A. Burns, R. Davis, Z. Guo, and Y. Hu. 2023. Scheduling IDK classifiers with arbitrary dependences to minimize the expected time to successful classification. *Real-Time Systems* (13 Mar 2023). https://doi.org/10.1007/s11241-023-09395-0

[2] T. Abdelzaher, S. Baruah, I. Bate, A. Burns, R. Davis, and Y. Hu. 2023. Scheduling classifiers for real-time hazard perception considering functional uncertainty. In *Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS'23)*. Association for Computing Machinery, New York, NY, USA.

[3] Kunal Agrawal, Sanjoy Baruah, Michael Bender, and Alberto Marchetti-Spaccamela. 2023. The safe and effective use of low-assurance predictions in safety-critical systems. In *35th Euromicro Conference on Real-Time Systems, ECRTS 2023, July 12–14, 2021, Vienna, Austria (Leibniz International Proceedings in Informatics (LIPIcs))*, Alessandro V. Papadopoulos. (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 3:1–3:19. https://doi.org/10.4230/LIPIcs.ECRTS.2023.3

[4] S. Baruah, A. Burns, R. Davis, and Y. Wu. 2023. Optimally ordering IDK classifiers subject to deadlines. *Real Time Syst.* 59, 1 (2023), 1–34. https://doi.org/10.1007/s11241-022-09383-w

[5] S. Baruah, A. Burns, and Y. Wu. 2021. Optimal synthesis of IDK-cascades. In *RTNS'2021: 29th International Conference on Real-Time Networks and Systems, Nantes, France, April 7–9, 2021*, Audrey Queudet, Iain Bate, and Giuseppe Lipari (Eds.). ACM, 184–191. https://doi.org/10.1145/3453417.3453425

[6] A. Burns, R. Davis, S. K. Baruah, and I. Bate. 2018. Robust mixed-criticality systems. *IEEE Trans. Comput.* 67, 10 (2018), 1478–1491.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms*. (third ed.). MIT Press.

[8] K. He, X. Zhang, S. Ren, and J. Sun. 2015. Deep residual learning for image recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[9] F. Khani, M. Rinard, and P. Liang. 2016. Unanimous prediction for 100% precision with application to learning semantic mappings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7–12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics. https://doi.org/10.18653/v1/p16-1090

[10] C. Liu and J. Layland. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM* 20, 1 (1973), 46–61.

[11] M. Mitzenmacher and S. Vassilvitskii. 2021. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*, T. Roughgarden (Ed.). Cambridge University Press, 646–662. https://doi.org/10.1017/9781108637435.037

[12] T. Roughgarden (Ed.). 2020. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press. https://doi.org/10.1017/9781108637435

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei. 2015. ImageNet large scale visual recognition challenge. *Int. J. Comput. Vis.* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[14] T. Wang, D. Kara, J. Li, S. Liu, T. Abdelzaher, and B. Jalaian. 2022. The methodological pitfall of dataset-driven research on deep learning: An IoT example. In *Military Communications Conference, MILCOMM 2022, Sydney, USA, 28 November - 2 December 2022*. https://edas.info/web/milcom2022/program.html#S1569610867

[15] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, and J. Gonzalez. 2018. IDK cascades: Fast deep learning by learning not to overthink. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6–10, 2018*, Amir Globerson and Ricardo Silva (Eds.). AUAI Press, 580–590. http://auai.org/uai2018/proceedings/papers/212.pdf

[16] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. 2008. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.* 7, 3 (2008), 36:1–36:53. https://doi.org/10.1145/1347375.1347389

[17] Tianming Zhao, Wei Li, and Albert Y. Zomaya. 2022. Real-time scheduling with predictions. In *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5–8, 2022*. IEEE, 331–343. https://doi.org/10.1109/RTSS55097.2022.00036

Received 23 March 2023; revised 2 June 2023; accepted 13 July 2023