IDK Cascades for Time-Series Input Streams

Kunal Agrawal* Sanjoy Baruah* Alan Burns† Jinhao Zhao*
*Washington University in St. Louis, USA. Email: {kunal, baruah, jinhaoz}@wustl.edu
†The University of York, UK. Email: alan.burns@york.ac.uk

***Abstract*—An IDK classifier is a software component that attempts to categorize each input provided to it into one of a fixed set of classes, returning IDK ("I Don't Know") if it is unable to do so with the required level of confidence. Several different IDK classifiers may be available for the same classification problem, each offering a different trade-off between execution duration and the likelihood of successful classification. Algorithms have been obtained for determining the order in which such classifiers should be called such that the expected duration to successfully classify an input is minimized – such an ordering of classifiers is called an IDK cascade. Cascade-synthesis algorithms make the assumption that each input to be classified is drawn from the same underlying distribution. We derive runtime algorithms that seek to further reduce the expected response time of IDK cascades upon input sequences for which successive inputs are 'similar' in the following sense: if a particular classifier successfully classifies some input it is likely to also be able to classify the next input. We evaluate the effectiveness of our algorithms in the context of the *algorithms using predictions* framework by showing that it significantly reduces expected response time when the desired similarity between successive inputs exists, while suffering only a minor increase in expected response time in the absence of such similarity. We describe how our algorithm is able to learn during runtime whether similarities exist (and if so, to what degree) amongst its inputs.**

I. Introduction

A classifier is a software component that classifies its input as belonging to one of a predefined set of classes. Perception in autonomous mobile Cyber-Physical Systems (CPS) increasingly relies on classifiers founded on Deep Learning and related AI technologies [1]. Such perception requires classifiers to make accurate real-time predictions when implemented upon platforms with limited computational resources; however, most current machine learning research emphasizes accuracy over real-time considerations. We consequently have classifiers that are highly accurate, but quite time-consuming even upon simple inputs – e.g., it was shown [2] that a ten-fold increase in classifier execution time yields only a marginal improvement in prediction accuracy for much of the ImageNet 2012 benchmark [3].

IDK Cascades. IDK classifiers aim to strike a balance between accuracy and timeliness by using slower advanced classifiers only for more challenging cases. An IDK classifier can be derived from any pre-existing base classifier: if the base classifier fails to decide upon a classification with a confidence level surpassing a specified confidence threshold, it instead outputs a placeholder class labeled as IDK, signifying "I Don't Know." Multiple distinct IDK classifiers may be trained for a given classification problem, each with differing execution times and probabilities of producing an actual class instead of IDK. Wang et al. [2] proposed that such IDK classifiers be organized into an *IDK cascade*: a linear sequence of classifiers that is deployed as follows upon any input to be classified:

1) The first classifier in the sequence is invoked.
2) If it outputs a real class then the IDK cascade concludes and the input is categorized under the identified class.
3) If, however, it outputs IDK, then the next classifier in the IDK cascade is invoked, and the process repeats from step 2 above.

For use-cases where it is mandatory that each input must always be classified with a real class, a *deterministic* classifier, that returns a real class on all inputs, is placed as the final component of an IDK cascade. (The deterministic classifier being unable to classify an input would constitute a system fault and may trigger a recovery mechanism.)

Recent research in the real-time systems community (e.g., [4]–[7]) has studied the problem of synthesizing IDK cascades that minimize the expected duration needed to obtain a real (i.e., non-IDK) classification, optionally within a specified latency constraint. This prior work assumes a model for IDK classifiers (which we describe in greater detail in Section II) that characterizes each classifier with a probability of it successfully classifying any given input. This model does not require the different classifiers to be mutually independent with regards to the probability of successful classification; it does however, assume that for each individual classifier the probability of successful classification is the same for each input (i.e., that all inputs are drawn independently from the same underlying probability distribution).

This work. Most mobile perception pipelines require that sequences of readings [8] from sensors such as accelerometers, cameras, and microphones [9], each be classified. It is reasonable to hypothesize some dependence amongst successive inputs in such time-series readings from a single sensor source. Consider, for instance, a stream of frames captured by a camera in a moving car, with each identified Region of Interest (RoI) tracked individually across successive frames. It is plausible that if a RoI is successfully classified by a particular classifier, then that classifier can also classify the same RoI in the following several frames until perhaps eventually failing because the object has moved too far away and needs a more advanced classifier. Such potential dependencies are, to our knowledge, not currently exploited when using IDK classifier cascades, which always start out invoking the

2576-3172/24/$31.00 ©2024 IEEE
DOI 10.1109/RTSS62706.2024.00017

Original Size		New Size			
		32	64	128	256
256	Acc.	0.673	0.881	0.985	1.00
	Time	0.004	0.008	0.013	0.028
128	Acc.	0.667	0.854	1.00	
	Time	0.004	0.008	0.013	
64	Acc.	0.760	1.0		
	Time	0.004	0.008		
32	Acc.	1.0			
	Time	0.004			

TABLE I
From [10]: Accuracy (normalized) versus execution duration tradeoff when image resizing is used for visual machine perception. (Durations include the time required to downsize an image.)

first classifier in the cascade and moving on through the cascade until a non-IDK classification is returned. In this paper, *we propose run-time algorithms that exploit potential dependencies between successive inputs* of time-series data. We show that our proposed algorithms further reduce the expected duration to successful classification in the event that such dependencies exist, while simultaneously guaranteeing to not do too much worse than prior run-time algorithms (that always start out invoking the first classifier in the cascade) if the anticipated dependencies are absent. Specifically, we *characterize and formally define* (Definition 2) a particular kind of dependence that we hypothesize is widely present in time-series sensor input streams, and *derive algorithms* that are capable of exploiting such dependencies when present to speed up expected classification duration, whilst guaranteeing bounded degradation in the event the anticipated dependencies are not present. Additionally, we propose and evaluate *a schema for learning* the degree of dependence in an input stream, and for adapting to dynamic changes in this degree of dependence.

Organization. We start out in Section II with a discussion on IDK cascades, briefly describing formal models and surveying relevant prior work, and state the specific problem that is the subject of this paper. We design an algorithm for exploiting dependencies in Section III; in Section IV, we provide a thorough formal characterization of this algorithm's performance both when the expected dependence is present and when it is not. In Section V we propose a generalization of the algorithm of Section III that enables us to make certain kinds of QoS guarantees (that the original algorithm could not); more importantly, we will see that it possesses properties that enable us to use it to *learn* whether, and to what degree, dependencies exist in the input stream. We exploit this feature to develop a learning-based algorithm in Section VI. We conclude in Section VII by placing this work within the larger context of using learning-based components in safety-critical systems, and suggesting some directions for future research.

II. Background and Problem Statement

As mentioned in Section I, IDK cascades have recently been the subject of several studies in the real-time computing community (see, e.g., [4]–[7], [11] — this list is by no means exhaustive). Let us consider an illustrative example application in order to motivate IDK cascades from a practical perspective: the *image resizing* approach advocated by Hu et al. [10] for enhancing efficiency in visual machine perception. Hu et al. examined the tradeoff between execution duration and accuracy in image classification when the images may be downsized and classified using DNNs with smaller input sizes – please see [10] for details. Some of their findings are presented in Table I; from this table it can be seen, for instance, that downsizing an image with original size 256×256 to 128×128 reduced the execution duration by more than a factor of two (from 0.028 to 0.013) whilst causing only a small reduction in accuracy from 1.00 to 0.985. For purposes of illustration, let us suppose that the individual DNNs are retrained to output IDK (rather than an erroneous class) when their confidence in their classification is low, and that by so doing, we obtain four different IDK classifiers for inputs of size 256×256, 128×128, 64×64, and 32×32 respectively, that have execution durations of $0.028, 0.013, 0.008$, and 0.004 respectively and normalized probabilities of successful classification of 1.00, 0.985, 0.881, and 0.673 respectively. Given a stream of images of original size 256×256 to be classified, we could process each such image through a cascade of these four IDK classifiers (after appropriate downsizing as needed); by so doing, the expected execution duration of a successful classification is

$$\Big(0.004 + \big(0.008 \times (1 - 0.673)\big) + \big(0.013 \times (1 - 0.881)\big) + \big(0.028 \times (1 - 0.985)\big)\Big) = \mathbf{8.583 \times 10^{-3}} \quad (1)$$

which is smaller than the 0.028 (or 28×10^{-3}) execution duration of the classifier that uses the full-sized images.

Could one do better – reduce the expected execution duration even further by perhaps removing one or more of the classifiers from the cascade? For this particular example, it turns out that one cannot; however if the initial images to be classified are of size 128×128 (and so the second row of Table I is the relevant one) we see that the cascade of all three classifiers yields an expected execution duration equal to

$$\Big(0.004 + \big(0.008 \times (1 - 0.667)\big) + \big(0.013 \times (1 - 0.854)\big) = \mathbf{8.562 \times 10^{-3}}$$

whereas the cascade obtained by dropping the middle classifier (the one for images of size 64×64) has a smaller expected execution duration of

$$\Big(0.004 + \big(0.013 \times (1 - 0.667)\big)\Big) = \mathbf{8.329 \times 10^{-3}}$$

These examples serve to illustrate both that (i) IDK cascades are useful in the sense that they have the potential to significantly reduce expected execution duration; and (ii) it is

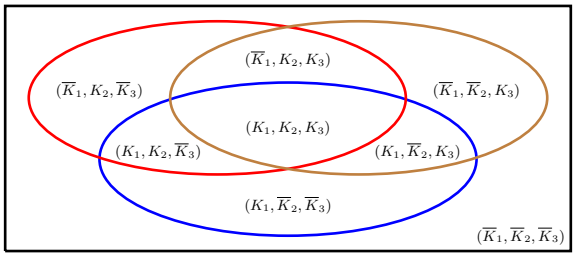

Fig. 1. (From [6].) The 2^n disjoint regions in the probability space for three IDK classifiers ($n = 3$) and one deterministic classifier. The blue, red, and brown ellipses respectively denote the regions of the probability space where the classifiers K_1, K_2, and K_3 are successful (i.e., do not output IDK). The enclosing rectangle denotes the region in which the deterministic classifier is successful (i.e., all inputs). Each of the $2^3 = \mathbf{8}$ disjoint regions into which the probability space is partitioned by the three ellipses is labeled with a 3-tuple, with K_i ($\overline{K}_i$, respectively) denoting that the the IDK classifier K_i returns a real class (resp. IDK) in this region.

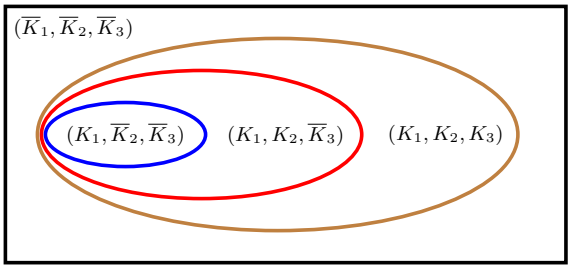

Fig. 2. Venn diagram for three contained IDK classifiers (plus one deterministic classifier). As in Figure 1, the blue, red, and brown ellipses respectively denote the regions of the probability space where the classifiers K_1, K_2, and K_3 are successful (i.e., do not output IDK).

not always obvious what the optimal cascade –the one with minimum expected duration– is. As mentioned earlier, considerable recent research in the real-time systems community has addressed the latter of these questions; we briefly discuss this prior research next, in Sections II-A and II-B.

A. A Model for IDK Classifiers

We now describe the formal model for IDK classifiers that has previously been considered in the real-time scheduling literature. Let us suppose that we have n IDK classifiers denoted $K_1, K_2, \ldots, K_n$, all for the same classification problem. We assume that there is a probability P_i of each classifier K_i successfully classifying any given input. The probabilistic behaviors of the different classifiers are not, in general, assumed to be independent – the classifiers may exhibit various mutual dependences between their behaviors. It is sometimes helpful to visualize the probability space for the n IDK classifiers as a Venn Diagram divided into 2^n distinct regions. Each of these regions corresponds to one of the 2^n potential combinations of the n individual classifiers returning either a real class or IDK for an input, see Figure 1 (taken from [6]) for the case of $n = 3$ classifiers. Abdelzaher et al. [4], [12] describe a measurement-based methodology for estimating the probability values associated with each of these regions, by conducting profiling experiments using representative training data. This methodology characterizes the collection of n IDK classifiers by 2^n probability values, one labeling each region of the Venn diagram, as well as the WCET values $C_1, C_2, \ldots, C_n, C_{n+1}$ with C_i denoting the worst-case execution duration of IDK classifier K_i, $1 \leq i \leq n$, and C_{n+1} denoting the worst-case execution duration of the deterministic classifier.

B. Synthesizing IDK Cascades

Algorithms have been derived for constructing IDK cascades that minimize the expected duration required to achieve successful classification, while optionally adhering to a specified latency constraint. To our knowledge, the most general algorithm for this purpose [4] has a worst-case running time of $\mathcal{O}(4^n)$ where n denotes the number of available IDK classifiers. More efficient algorithms have been obtained [5] for the special case where the different available IDK classifiers are a priori known to satisfy a particular property that we will refer to here as the *containment* property – this paper primarily restricts attention to this special case. (Containment was called 'full dependence' in [5]; however, this causes confusion with the notion of dependence amongst inputs.) The classifiers in a contained collection of classifiers can be strictly ordered from least to most powerful, such that any input successfully classified by a particular classifier in the collection is also successfully classified by all more powerful classifiers (see Figure 2 for a Venn-diagram representation, and to appreciate the rationale for the name "containment" for this property). For synthesizing optimal IDK cascades from a collection of contained classifiers, a polynomial-time algorithm with $\mathcal{O}(n^2)$ worst-case running time was derived in [5], as was a pseudopolynomial one with worst-case running time $\mathcal{O}(n^2 D)$ if a latency constraint D must also be satisfied.

A different aspect of IDK cascades was studied in [7]: what if the underlying probabilities (of successful classification by individual IDK classifiers) may have been incorrectly estimated? Assuming this is a possibility, the Algorithms Using Predictions framework [13]–[15] was applied to develop algorithms that perform well when these probability estimates are correct, while also being robust to incorrect estimates.

C. The Problem Considered

All the results discussed in Section II-B made the underlying assumption that there is no dependence between different inputs that are to be classified[1] (equivalently, that each is drawn independently from the same underlying probability

[1]Indeed, such non-use of potential dependencies is explicitly pointed out by Abdelzaher et al. [4]: "This paper considers the use of IDK cascades as a single-shot solution to the machine perception problem. Such solutions are also viable for systems where inputs are generated recurrently, i.e. periodically or sporadically, but no account is taken of the input data or results from previous time frames, i.e. *each machine perception or classification job is [assumed to be] effectively independent*."

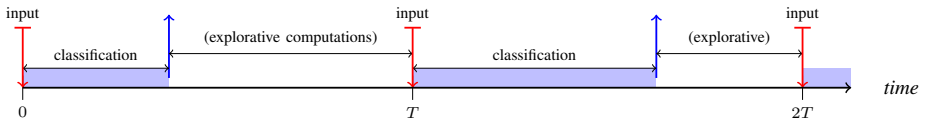

Fig. 3. Inputs to be classified arrive at time-instants $k \cdot T$ for all $k \in \mathbb{N}$. Upon successful classification, the determined class is immediately communicated to the next stage of the perception pipeline; the remainder of the duration until the arrival of the next input may be used to perform speculative exploratory computations aimed at speeding up classification of future inputs.

distribution). The main issue we investigate in this paper is this: *what if dependence between successive inputs is likely* (but not guaranteed to be present); and additionally, *the precise degree of such dependence is a priori unknown*? We focus only on the case of contained classifiers, in which each classifier in a cascade is successively more powerful than the prior ones.

As in prior work, we assume that there is a stream of inputs arriving (from, e.g., a sensor) at the classifier with successive inputs exactly T time units apart – see Figure 3. Each input must be classified before the next input arrives – we thus have a hard deadline of time-duration T for successful classification. Our *performance objective* is to reduce the expected duration to classification, subject to this hard deadline always being met.

We assume that we do not know, nor can precisely characterize, the possible dependence between successive inputs in the input stream.[2] Hence, in our approach, we assume that IDK cascades have already been pre-constructed exactly as discussed in Section II-B.[3] In particular, the *construction* of IDK cascades assumes that the inputs are independent. In this work, we demonstrate how one might adapt the use of a cascade during runtime in order to exploit the presence of dependencies to further optimize the expected runtime (still subject to the hard deadline).

Abstractly, our runtime strategy works as follows. As stated above, one input arrives each T time units. Hence at each time-instant $k \cdot T$ for all $k \in \mathbb{N}$, our run-time algorithm chooses which classifiers in the cascade to execute, and in what order, until a successful classification is obtained. When the successful classification occurs, our algorithm immediately reports the identified class; *the lapsed duration between the input's arrival instant and this classification instant constitutes the response time for this input* (the reduction of which is our performance objective). Once classification is complete, the remainder of the time-interval $[k \cdot T, (k+1) \cdot T)$ is used by the run-time algorithm for performing additional "exploratory" computations that are aimed at reducing the expected duration for classification of future inputs.

A design choice. Note that our run-time algorithm is designed to optimize for expected response-time rather than the overall computation time (i.e., for computing the classification and additionally doing the exploration) – this was a design choice on our part. By way of justification, we will see that the explorative computations essentially consist of executing some of the remaining classifiers upon the (already classified) input; hence the total execution duration of classification + exploration is guaranteed to not exceed the cumulative worst-case execution duration of the entire cascade (which, by design of the cascade, is guaranteed to not exceed T). The rationale behind our design choice is that since the computing capacity is anyway available, we may as well use it to possibly improve future response times. (In Section VII we briefly discuss alternative approaches in which there is also a cost associated with exploration and one must trade off the cost of current exploration versus future benefit.)

D. A Model for Dependence

Since our primary goal is to exploit dependences between successive inputs in an input stream, let us define a quantitative metric of such dependence. In order to precisely and clearly define this metric, we first define a notion of *equivalent inputs*.

Definition 1 (equivalent inputs). Consider a time series $\vec{x} \stackrel{\text{def}}{=} \langle x_1, x_2, x_3 \ldots \rangle$ of inputs drawn from a distribution $\mathcal{D}$ and to be classified by a cascade consisting of a set of classifiers $\mathcal{S}$. We say that two inputs in $\vec{x}$ are *equivalent* iff the same (sub)set of classifiers in $\mathcal{S}$ can successfully classify both inputs (or equivalently, the same (sub)set of classifiers return the class IDK for both inputs).

Intuitively, two inputs are equivalent if they belong to the same region in the Venn Diagram representations of the classifiers as in Figures 1 and 2.

We now define the dependence parameter which defines the probability that two successive inputs are equivalent.

Definition 2 (dependence parameter λ). Consider a time-series stream of inputs $\vec{x} \stackrel{\text{def}}{=} \langle x_1, x_2, x_3 \ldots \rangle$, an underlying probability distribution $\mathcal{D}$ and a cascade $\mathcal{K}$ constructed for $\mathcal{D}$. We say that $\vec{x}$ has dependence parameter λ iff it possesses the property that for each $t \geq 1$ the input x_{t+1} is

- equivalent to input x_t with probability λ, and
- with probability $(1-\lambda)$, drawn from the underlying probability distribution $\mathcal{D}$. □

Thus a small value of λ indicates little dependence between successive inputs ($\lambda = 0$ implies that each input is independently drawn from the underlying distribution $\mathcal{D}$) whereas a

[2] In Section VI we will explore avenues towards overcoming this assumption by *learning* the degree of dependence during runtime.

[3] Since we are considering contained classifiers and have a hard deadline, we use the pseudo-polynomial time algorithm from [5] to construct the cascade.

larger λ denotes greater dependence (if $\lambda = 1$ then all the inputs are equivalent).

We point out that the value of *the dependence parameter is not used* while constructing the cascade since we assume that the cascade is preconstructed and assumed to be optimal with respect to the expected execution time if the inputs are drawn independently from the underlying probability distribution $\mathcal{D}$. (Thus, λ is a very different kind of parameter from the probability estimates discussed in Section II-A, that are estimated using the technique of Abdelzaher et al. [4], [12] and subsequently explicitly used by the algorithms of [5] to synthesize the IDK cascade.) This parameter, however, is a characteristic property of the input stream which may be unknown to us. Our algorithms do not explicitly use this parameter and it is used only during the analysis of the algorithms. In later sections, we show how this parameter may be estimated during runtime and can be used to further optimize our runtime algorithms.

Implication of λ on cascades for contained classifiers. In this paper, we focus only on cascades constructed using a set of contained classifiers. In these cascades, each successive classifier is more powerful than the previous one — that is, it can classify everything that the prior classifier could classify. (For instance, if we consider the example in Figure 2, then the cascade would consist of K_1, K_2, K_3 in order or it might choose to skip one of these, but would never put them out of order.) Therefore, for these cascades, two equivalent inputs are always successfully classified by an identical suffix of the cascade.

III. An Algorithm for Exploiting Dependencies

Let us suppose we have a stream of inputs that need to be classified, and we have a collection of contained IDK classifiers (i.e., a collection of IDK classifiers that can be strictly ordered from least powerful to most powerful such that any input successfully classified by a classifier is also successfully classified by all more powerful classifiers — see Figure 2) all trained to solve this classification problem. As previously mentioned (Section II-B), an algorithm has been derived in [5] to construct an optimal IDK cascade from such a collection; we will use this prior algorithm to synthesize a cascade that is guaranteed, by the results in [5], to be optimal in the absence of dependencies between successive inputs. In the remainder of this section, let

$$\mathcal{K} \stackrel{\text{def}}{=} \langle K_1, K_2, \ldots, K_N \rangle \tag{2}$$

denote this optimal cascade, with K_N denoting the deterministic classifier. Let C_i denote K_i's WCET, and P_i the probability that K_i does not return IDK upon a randomly-drawn input. (Note that $C_{i-1} < C_i$ and $P_{i-1} < P_i$ for each $i, 1 < i \leq N$, and $P_N = 1$, in any optimal cascade.) For notational convenience, let *(i)* K_0 denote a hypothetical classifier that has WCET $C_0 = 0$ and $P_0 = 0$; and *(ii)* $P_i = 1$ for all $i > N$.

Algorithm 1: Dynamic ($\mathcal{K} = \langle K_1, K_2, \ldots, K_N \rangle$)

```
1 b = 1 // Start executing the cascade here
2 for each input do
3     Execute classifiers K_b, K_{b+1}, ... until K_i (i ≥ b)
        classifies the input with a non-IDK class
4     output this non-IDK class
5     if (i == b) then // Try and improve b
6         Execute classifiers K_{b-1}, K_{b-2}, ... until K_i
            classifies the input as IDK
7         b = (i + 1)// updating b
8     else
9         b = i
```

Exploiting dependencies. To understand the benefit of exploiting potential dependencies amongst successive inputs in the input stream, let us revisit the image resizing scenario IDK cascade considered in Section II (that was adapted from [10]). If an object being tracked in an image has not changed much in size between successive frames, then using a classifier trained for a particular size that successfully classifies a particular input image is likely to perform successful classifications upon multiple successive inputs before eventually returning IDK.[4] We had seen in Section II (Expression (1)) that the expected duration for the IDK cascade constructed from the first row of Table I to successfully classify an image is 8.583×10^{-3}. Let us assume that if a particular stage of the IDK cascade constructed from the first row of Table I is the first to successfully classify an image, then there is a 50% probability that it will also successfully classify the following image; in the presence of such dependencies, we will see in Example 1 that Algorithm 1, which we have designed to explicitly exploit dependencies (when present), further reduces the expected duration to successful classification to 7.896×10^{-3}.

Run-time algorithm. We now describe – see Algorithm 1 – how the cascade $\mathcal{K}$ of Expression (2) is used during run-time once it has been synthesized by the algorithm in [5], in order to exploit potential dependencies between successive inputs in the input stream. (Note that this algorithm will not explicitly use the dependence parameter λ.) For each input the first classifier to be called is the classifier of minimum WCET that was able to successfully classify the prior input – the index of this classifier is stored in the variable b (initialized to one; thus, classifier K_1 is the first to attempt to classify the first input). For each input, the time taken to execute lines 3–4 constitutes its response time, while the time taken to execute lines 5–7 represent exploration (during which progressively less powerful classifiers are executed until one that fails to classify the input is encountered; the index of the next-most-powerful classifier is then stored in variable b). (Observe that

[4]This is explained thus in [10, (p 171)]: "When the sampling rate of the sensors is relatively high, there will be a large amount of redundancy between frames. An object that appeared in the previous frame will likely also appear in the current frame."

exploration is not needed if $i \neq b$, i.e., K_b returned IDK, since in that case classifier K_i is the classifier of minimum WCET to successfully classify the input.)

A note on terminology: we refer to "normal" use of the cascade – begin at classifier K_1 on each input and continue executing classifiers in sequence until a non-IDK classification is obtained – as "static" to emphasize that the order of execution of the classifiers is pre-determined when the cascade is synthesized and is hence static at run-time, in contrast to the more dynamic execution order of the classifiers in the cascade in Algorithm 1. In the remainder of this manuscript we will use the term "*the static algorithm*" to refer to such static use of the cascade during runtime.

IV. Analysis of Algorithm 1

In this section we provide a multi-faceted characterization of the performance of Algorithm 1. The analysis is framed in terms of the dependence parameter λ — generally, the algorithm performs better for larger values of λ, as is to be expected. Lemma 1 below asserts that regardless of the value of the dependence parameter (i.e., irrespective of whether successive inputs have dependencies or not), each individual input can continue to be modeled as being drawn from the underlying distribution that was assumed by the algorithms in [5] whilst constructing the optimal cascade:

Lemma 1. Consider an input stream $\vec{x}$ characterized by dependence parameter λ drawn from some underlying probability distribution. Each individual input x_t in this input stream may be modeled as being drawn from the same underlying probability distribution.

Proof. This lemma is easily proved via induction. Observe first, for the *base case*, that the first input x_1 is drawn from the underlying probability distribution. Assume, for an *inductive hypothesis*, that input x_{t-1} is also drawn from the underlying probability distribution. For the *inductive step*, we have two possibilities for the input x_t:

1) with probability λ, x_t and x_{t-1} are equivalent. By the inductive hypothesis, x_{t-1} is drawn from the underlying probability distribution, and hence so is x_t.
2) with probability $(1-\lambda)$, x_t is drawn independently from the underlying probability distribution.

In either case, we conclude that x_t may also be modeled as being drawn from the underlying probability distribution. The lemma follows. □

We now introduce some notation —see Table II: for any random variable (RV) X, let $\mathbb{E}[X]$ denote its expectation (i.e., its average value). Let RV $C_{\text{static}}(\mathcal{K})$ represent the execution duration (on a single input) of the static algorithm upon the optimal cascade that was constructed using the algorithm of [5]. Since the static algorithm will execute the classifier K_i in the optimal cascade for some input only if all the preceding classifiers fail to successfully classify that input, we have

$C_{\text{static}}(\mathcal{K})$	RV denoting the execution duration of the cascade $\mathcal{K}$ upon a single input under 'normal' use (i.e., by "*the static algorithm*")
$C_{\text{dyn}}(\mathcal{K})$	RV denoting the execution duration of the cascade $\mathcal{K}$ under Algorithm 1 upon a single input
$C_{\text{dyn-k}}(\mathcal{K}, k)$	RV denoting the execution duration of the cascade $\mathcal{K}$ under Algorithm 2 upon a single input

TABLE II
Some Random Variables used in our analysis

$$\mathbb{E}[C_{\text{static}}(\mathcal{K})] = \sum_{i=1}^{N} (1 - P_{i-1})\, C_i \tag{3}$$

by summing over the expected execution durations of all the classifiers in the cascade.

Some additional notation (again see Table II): analogous to $C_{\text{static}}(\mathcal{K})$, let $C_{\text{dyn}}(\mathcal{K})$ be a random variable denoting the execution duration of Algorithm 1 upon a single input. The following theorem identifies the relationship of expected execution duration of Algorithm 1 upon each input in an input stream to the dependence parameter of the input stream.

Theorem 1. When classifying a stream of inputs with dependence parameter λ,

$$\mathbb{E}[C_{\text{dyn}}(\mathcal{K})] = \sum_{i=1}^{N} \Big[\Big(\big(P_i - P_{i-1}^2\big) - \lambda \big(P_{i-1} - P_{i-1}^2\big) \Big) \cdot C_i \Big] \tag{4}$$

Proof. Consider some arbitrary input x_t in the input stream $\vec{x}$ that is characterized by dependence parameter λ. Let K_b denote the classifier with minimum WCET to successfully classify the previous input, x_{t-1}. (Note that for each i, $1 \leq i \leq N$, the probability that classifier K_i is this classifier K_b is equal to $(P_i - P_{i-1})$ — this follows from Lemma 1, and the fact that the probability of a classifier K_i being the lowest-indexed one to classify a randomly-drawn input is equal to $P_i - P_{i-1}$.) On input x_t Algorithm 1 begins with classifier K_b. As previously observed (Definition 2), there are two possible cases:

Case 1. With probability λ, the dependence holds and K_b is successful (returns a real class, not IDK). (Note, the probability that K_b is the classifier with minimum WCET to have successfully classified the previous input is equal to $(P_b - P_{b-1})$.)

Case 2. With probability $(1-\lambda)$, there is no dependency between x_t and x_{t-1} and x_t is randomly drawn from the underlying distribution. Recall that Algorithm 1 executes classifiers $K_b, K_{b+1}, \ldots,$ until a successful classification is

obtained. Hence in this case, the probability that the classifier K_i must be executed in order to classify x_t is equal to

$$\begin{cases} 0, & \text{if } i < b \\ 1, & \text{if } i = b \\ (1 - P_{i-1}), & \text{if } i > b \end{cases}$$

which equals $\Pr\{b = i\} + (1 - P_{i-1}) \cdot \Pr\{b < i\}$, or

$$\begin{aligned} & \big(P_i - P_{i-1}\big) + \big((1 - P_{i-1}) \cdot P_{i-1}\big) \\ = \;& P_i - P_{i-1} + P_{i-1} - P_{i-1}^2 \\ = \;& P_i - P_{i-1}^2 \end{aligned}$$

Summing over the two cases, the probability that classifier K_i must be executed in order to classify the input x_t is equal to

$$\begin{aligned} & \lambda \cdot \overbrace{(P_i - P_{i-1})}^{\Pr\{b=i\}} + (1 - \lambda) \cdot (P_i - P_{i-1}^2) \\ = \;& \lambda P_i - \lambda P_{i-1} + P_i + -P_{i-1}^2 - \lambda P_i + \lambda P_{i-1}^2 \\ = \;& P_i - \lambda P_{i-1} - (1 - \lambda) P_{i-1}^2 \\ = \;& (P_i - P_{i-1}^2) - \lambda (P_{i-1} - P_{i-1}^2) \end{aligned}$$

and the theorem follows by summing, over all $i, 1 \leq i \leq N$, the product of classifier K_i's WCET with this probability of its being executed. □

A. AN ILLUSTRATIVE EXAMPLE

We can use the results obtained above to evaluate the runtime performance of Algorithm 1 upon any particular IDK cascade; we now illustrate the process of performing such evaluation. The analysis will be performed within the context of the *Algorithms Using Predictions* framework [13]–[15], by looking upon the claimed presence of dependencies in the input stream as a *prediction* that may or may not be correct. We begin with a very brief and informal primer on the Algorithms Using Predictions framework (please see [13] for additional details, or [16] for an introduction directed at a real-time computing audience). Algorithms are evaluated within this framework according to three characteristics[5]:

1) *Consistency*: When the prediction is accurate, the performance of the algorithm is better than one that does not use predictions. (Since our prediction is that the time-series input stream exhibits dependencies, the static algorithm, which ignores possible dependencies and always executes the classifiers in sequence beginning at K_1 until a non-IDK classification is returned, is the obvious candidate for the role of a good algorithm that does not use predictions.)
2) *Robustness*: When the prediction is 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 prediction has small errors: "the algorithm interpolates gracefully between the robust and consistent settings" [13].

[5]A fourth characteristic, *learnability*, is also considered – we defer discussion on this to Section VI.

For any particular IDK cascade $\mathcal{K}$, the P_i and C_i values are all fixed and therefore Expression (4), the expected classification duration $\mathbb{E}[C_{\text{dyn}}(\mathcal{K})]$ of Algorithm 1, can be written in the form $\big(A - B\,\lambda\big)$ for non-negative constants A and B. When $\mathbb{E}[C_{\text{dyn}}(\mathcal{K})]$ is plotted as a function of λ, this is a straight line with negative slope (see the solid blue line in Figure 4 for an example). We see from Expression (3) that $\mathbb{E}[C_{\text{static}}(\mathcal{K})]$, the expected classification duration of the static algorithm, is constant for a given cascade; hence, the ratio

$$\rho_{\text{dyn}}(\mathcal{K}, \lambda) \stackrel{\text{def}}{=} \frac{\mathbb{E}[C_{\text{dyn}}(\mathcal{K})]}{\mathbb{E}[C_{\text{static}}(\mathcal{K})]} \tag{5}$$

is also a straight line with negative slope. For any cascade $\mathcal{K}$ $\rho_{\text{dyn}}(\mathcal{K}, 1)$ can be looked upon as an indicator of the consistency, and $\rho_{\text{dyn}}(\mathcal{K}, 0)$ of the robustness, of Algorithm 1 when used to schedule that cascade at runtime, while its smoothness is indicated by the slope of the straight line. We illustrate on an example.

Example 1. Let us once again consider the IDK cascade constructed from the first row of Table I (the image resizing scenario from [10]), that was analyzed in Section II. Instantiating Expression (5) for this example, we'd already seen that the denominator (i.e., expected duration for the static algorithm) is 8.583×10^{-3}; it may be verified that the numerator evaluates to $(9.665 - 3.538\lambda) \times 10^{-3}$; hence for this example IDK cascade we have

$$\rho_{\text{dyn}}(\mathcal{K}, \lambda) = \frac{9.665}{8.583} - \frac{3.538}{8.583} \times \lambda \approx \big(\mathbf{1.126 - 0.4122\,\lambda}\big)$$

and hence its consistency (when $\lambda = 1$) is $\approx \mathbf{0.714}$ while its robustness (when $\lambda = 0$) is $\approx \mathbf{1.126}$, indicating that its expected classification duration may be as small as 71.4% that of the static algorithm when there is very high dependence in the input stream, at a cost of perhaps a 12.6% increase in expected classification duration when there is absolutely no dependence whatsoever. Its smoothness is depicted graphically in Figure 4 – it is visually evident from this figure that the transition between the extremes of consistency and robustness is indeed smooth. □

Since the plot of a cascade's expected classification duration as a function of λ provides a graphic illustration of smoothness, in the remainder of this manuscript we will refer to such plots as ***smoothness curves***.

B. WORST-CASE ANALYSIS

Example 1 illustrated how Algorithm 1, which has been designed to exploit dependencies, may suffer a performance degradation vis-à-vis the static algorithm in the absence of such dependencies; the upper bound on this performance degradation is given by its robustness $\rho(\mathcal{K}, 0)$. Theorem 2 below establishes an upper bound of 2 on the value of $\rho(\mathcal{K}, 0)$ for *any* cascade, even upon input streams that are carefully selected by a malevolent adversary to minimize the effectiveness of Algorithm 1.

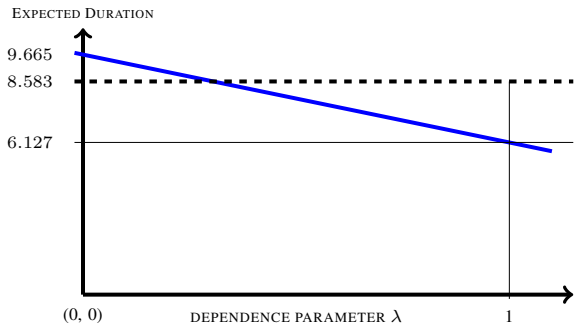

Fig. 4. Illustrating smoothness for Example 1: Expected execution duration versus λ. The solid line is for Algorithm 1; the dashed one, for the static algorithm.

Theorem 2. Consider any input stream $\vec{x} \stackrel{\text{def}}{=} x_1, x_2, \ldots, x_T$. For each $t, 1 \leq t \leq T$, let $C_{\text{dyn}}(\mathcal{K}, x_t)$ ($C_{\text{static}}(\mathcal{K}, x_t)$, respectively) denote the execution duration of Algorithm 1 (the static algorithm, resp.) in classifying the input x_t.

$$\left(\frac{\sum_{t=1}^{T} C_{\text{dyn}}(\mathcal{K}, x_t)}{\sum_{t=1}^{T} C_{\text{static}}(\mathcal{K}, x_t)} \right) \leq 2 \tag{6}$$

Proof. As in the proof of Theorem 1, let K_b denote the classifier with minimum WCET to successfully classify the previous input, x_{t-1}. Recall that Algorithm 1 begins with classifier K_b on input x_t, and continues executing the classifiers in sequence until some classifier returns a successful (non-IDK) classification. We consider two cases.

1) Input x_t is successfully classified by K_b. Since x_{t-1} is, by definition of "b", not successfully classified by classifier K_{b-1}, K_b's WCET term C_b must contribute to $C_{\text{static}}(\mathcal{K}, x_{t-1})$.
2) Input x_t is not successfully classified by K_b, and consequently classifiers $K_b, K_{b+1}, \ldots$ are all executed. Each such K_i's WCET term C_i must contribute to $C_{\text{static}}(\mathcal{K}, x_t)$.

We thus see that each WCET term contributing to $C_{\text{dyn}}(\mathcal{K}, x_t)$ must appear in at least one of $C_{\text{static}}(\mathcal{K}, x_t)$ or $C_{\text{static}}(\mathcal{K}, x_{t-1})$. We therefore conclude that

$$C_{\text{dyn}}(\mathcal{K}, x_t) \leq C_{\text{static}}(\mathcal{K}, x_t) + C_{\text{static}}(\mathcal{K}, x_{t-1}),$$

and the theorem follows. □

The upper bound of Theorem 2 assumes a malicious adversary that carefully selects the inputs in order to maximally degrade Algorithm 1's performance. What if the inputs are instead drawn from the underlying probability distributions that was assumed whilst constructing the optimal cascade (using the algorithm in [5])? The upper bound of Theorem 2 clearly continues to hold (since Algorithm 1's performance on such input cannot be poorer than on adversarial input). The following theorem shows that there are in fact IDK cascades for which we can come arbitrarily close to reaching this upper bound.

Algorithm 2: Dynamic-k ($\mathcal{K} = \langle K_1, K_2, \ldots, K_N \rangle, k$)

```
   // k is the skip-back factor
1  b = 1 for each input do
2      Execute the classifiers in K in order starting at
         K_max(b−k,1), until some K_i classifies the input
3      output this non-IDK class
4      if (i == b) then // Try and improve b
5          Execute classifiers K_{b−1}, K_{b−2}, ... until K_i
             classifies the input as IDK
6          b = (i + 1)
7      else
8          b = i
```

Theorem 3. There exist IDK cascades for which the ratio of Expression (6) is arbitrarily close to 2 upon an input stream drawn from the underlying distribution that was assumed whilst constructing the cascade.

Proof. Consider the following two-classifier cascade (recall that K_0 is a "dummy" classifier that was introduced for notational convenience):

	K_0	K_1	K_2
$\boldsymbol{P_i}$	0	$(1-\delta)$	1
$\boldsymbol{C_i}$	0	0	C_2

Let us suppose that this IDK cascade encounters an input stream exhibiting no dependencies ($\lambda = 0$). Since the expected classification duration by Algorithm 1 when the input stream exhibits no dependencies is seen to equal $\sum_{i=1}^{N} \left(\left(P_i - P_{i-1}^2 \right) \cdot C_i \right)$, (by setting $\lambda \leftarrow 0$ in Expression (4)), whilst the expected duration by the static algorithm is as given in Expression (3), their ratio is

$$\frac{\sum_{i=1}^{N} \left(\left(P_i - P_{i-1}^2 \right) \cdot C_i \right)}{\sum_{i=1}^{N} \left(\left(1 - P_{i-1} \right) C_i \right)} \tag{7}$$

For our IDK cascade, Expression (7) becomes

$$\left(\frac{((1-\delta) - 0^2) \cdot 0 + (1 - (1-\delta)^2) \cdot C_2}{(1-0) \cdot 0 + (1 - (1-\delta)) \cdot C_2} \right)$$
$$= \left(\frac{\delta(2-\delta)}{\delta} \right) = \quad \boldsymbol{(2-\delta)}$$

which approaches 2 as $\delta \to 0$. □

V. A Generalized Algorithm

We have seen (Theorems 2 and 3 above) that Algorithm 1 may have expected classification duration as large as twice that achieved by the static algorithm. This appears to be

a consequence of Algorithm 1 being quite aggressive in seeking to exploit dependencies; we now investigate a natural generalization of Algorithm 1, listed as Algorithm 2, that lets us tune down the degree of aggression. It does so by accepting an additional integer parameter k, $0 \leq k \leq N$ (here N is the number of classifiers in the cascade), with smaller values representing greater aggression. In fact, setting $k \leftarrow 0$ yields Algorithm 1 (and hence Algorithm 2 is a *generalization* of Algorithm 1), and it may be verified that setting $k \leftarrow N$ causes Algorithm 2 to behave as the static algorithm does.

The primary difference between Algorithm 2 and the prior Algorithm 1 is that whereas Algorithm 1 attempts to classify each input starting with the classifier of minimum WCET that was able to successfully classify the prior input, *the first classifier called by Algorithm 2 upon each input is k positions sooner in the cascade* (or the beginning of the cascade if that is fewer than k positions sooner). Intuitively, Algorithm 2 is being more conservative than Algorithm 1 regarding dependencies amongst successive inputs, in the sense that it starts out closer to the start of the IDK cascade on each input than Algorithm 2 does. We will now examine how this more conservative approach impacts the performance of Algorithm 2 as compared to that of Algorithm 1.

Some additional notation (also added to Table II): let $C_{\text{dyn-k}}(\mathcal{K}, k)$ be a random variable denoting the classifiction duration of Algorithm 2 upon a single input, when the skip-back factor is set equal to k. The following theorem is analogous to Theorem 1 in that it exposes the dependence of $\mathbb{E}[C_{\text{dyn-k}}(\mathcal{K}, k)]$ upon the dependence parameter of the input stream that it is tasked with classifying (its proof is similar to that of Theorem 1, and is provided in the appendix):

Theorem 4. When classifying a stream of inputs with dependence parameter λ,

$$\mathbb{E}[C_{\text{dyn-k}}(\kappa, k)] = \sum_{i=1}^{N} \Big(\big(P_{i+k} - P_{i+k-1}\, P_{i-1} - \lambda\, (P_{i-1} - P_{i+k-1}\, P_{i-1}) \big)\, C_i \Big) \tag{8}$$

(Recall the notational convention that $P_0 \stackrel{\text{def}}{=} 0$ and $P_i \stackrel{\text{def}}{=} 1$ for all $i > N$.) □

Worst-case analysis. Recall that Theorems 2 and 3 established an upper bound of two on the ratio of the execution durations of Algorithm 1 and the static algorithm upon an input stream, regardless of whether the input stream is generated randomly or by a malicious adversary. These bounds generalize in a straightforward manner to Algorithm 2: it is easily shown that for Algorithm 2 the RHS of Expression (6) becomes $\left(\frac{k+2}{k+1}\right)$, and the proof of Theorem 3 can be adapted to show that this upper bound also holds for some cascade upon randomly generated input streams. We omit detailed proofs.

k	$\mathbb{E}[C_{\text{dyn-k}}(\mathcal{K}, k)]$	Best ($\lambda = 1$)	Worst ($\lambda = 0$)
0	$9.4417 - 3.4567\,\lambda$	5.9850	9.4417
1	$8.7236 - 0.8716\,\lambda$	7.8520	8.7236
2	$8.5346 - 0.1266\,\lambda$	8.4080	8.5346
3	$8.5055 - 0.0215\,\lambda$	8.4840	8.5055
4	$8.5000 - 0.0000\,\lambda$	8.5000	8.5000

TABLE III
SMOOTHNESS CURVES, AND BEST-/ WORST-CASE BEHAVIORS, FOR THE CASCADE $\mathcal{K}$ OF EXAMPLE 1 ENHANCED WITH ADDITIONAL CLASSIFIER. (ALL NUMBERS ARE $\times 10^{-3}$.)

A. ANALYSIS OF ALGORITHM 2

We now provide a brief analysis of Algorithm 2 from the perspective of the Algorithms With Predictions framework [13]. Recall (Section IV-A) that in this framework an algorithm is characterized by its *consistency* in making good use of correct predictions, its *robustness* to incorrect predictions, and its *smoothness* in transitioning between the two extremes.

Let us start with smoothness. It is evident from Expression (8) that for any given IDK cascade $\mathcal{K}$ and fixed integer k, $\mathbb{E}[C_{\text{dyn-k}}(\kappa, k)]$ as a function of λ takes the form of a straight line with negative slope (i.e., $\Big(\mathbb{E}[C_{\text{dyn-k}}(\kappa, k)] = A - B\,\lambda\Big)$ for some non-negative constants A and B whose values depend on $\mathcal{K}$ and k). Since the line is straight for each value of k, we can conclude that *smoothness* holds for each value of k.

Turning next to robustness and consistency, observe that each i in the summation on the RHS of Expression (8) contributes an additive term $\Big(P_{i-1} \cdot (1 - P_{i+k-1}) \cdot C_i\Big)$ to the magnitude of the slope of this straight line. Note that as k increases, P_{i+k-1} becomes larger and hence $(1 - P_{i+k-1})$ becomes smaller; consequently, the value of B (the magnitude of the slope of the straight line) decreases. That leads to the conclusion that for larger values of k we will see smaller performance degradation when the predicted dependencies are missing (is *more robust* to mis-prediction), but conversely see smaller improvement when dependencies are present (displays *poorer consistency*).

We illustrate upon an enhancement of our running example – the IDK cascade of Example 1:

Example 2. In addition to all the IDK classifiers that were assumed in Example 1 to be available to us, suppose that we had another classifier K_i with $P_i = 0.996$ and $C_i = 15$. It turns out that the optimal IDK cascade generated by the cascade-synthesis algorithm of [5] includes all five classifiers[6] in the optimal cascade $\mathcal{K}$, and $\mathbb{E}[C_{\text{static}}(\mathcal{K})]$ falls to 8.500×10^{-3} (from 8.583×10^{-3} for the four-classifier cascade considered in Example 1).

We have instantiated Expression (8) for this cascade $\mathcal{K}$ for $k = 0, 1, 2, 3$, and 4 – see Table III. From the table it can be seen that when the predicted dependencies are strong ($\lambda \to 1$), smaller values of k exhibit smaller expected execution duration – this represents consistency. On the other hand when the pre-

[6] Indeed, the parameters of the added classifier were carefully selected in this contrived example in order to ensure that this is the case.

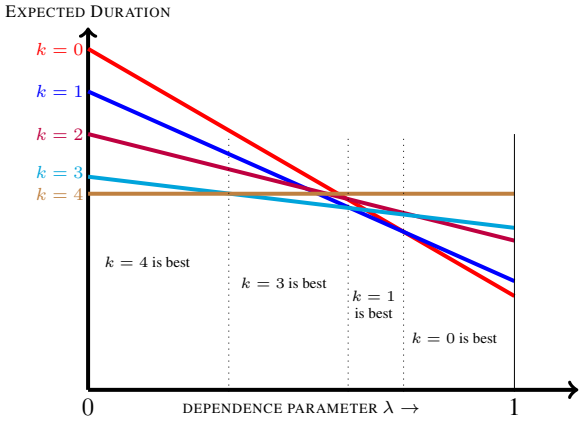

Fig. 5. Illustrating learning: smoothness curves for the same cascade, for different values of the skip-over factor k. Different smoothness curves are "best" – offer minimum expected classification duration – over different ranges of values of λ.

dicted dependencies are very weak or entirely absent ($\lambda \rightarrow 0$), larger values of k result in smaller expected execution duration (and are thus more robust to mis-prediction). □

Meeting robustness bounds. In addition to needing to meet the hard deadline of T time units – the inter-arrival duration between successive inputs – while processing each individual input, our cascade may additionally be subject to a *Quality of Service* (QoS) requirement: a need to guarantee that the expected duration to successful classification not exceed some specified bound.[7] By appropriate application of Theorem 4, Algorithm 2 is easily adapted to respect such bounds: we would only consider values of k for which Expression (8), when evaluated for $\lambda \leftarrow 0$, does not fall below the specified robustness bound. Continuing Example 2, let us suppose that we are subject to a SLA requiring that expected execution duration not exceed 9×10^{-3}. From Table III, we learn that $k = 0$ is unacceptable (since it would violate the SLA with an expected classification duration of 9.4417×10^{-3} if the anticipated dependencies are entirely absent: $\lambda = 0$); hence, we can only call Algorithm 2 with $k \geq 1$. If we anticipate significant dependencies, we would choose $k \leftarrow 1$, if we expect that the input stream will have little to no dependencies, we could choose $k \leftarrow 4$ and essentially default to the static algorithm.

VI. Incorporating Learning

In Section V-A we saw that for any cascade $\mathcal{K}$, calling Algorithm 2 with different values of k yields different smoothness curves. In Figure 5 we have plotted the different smoothness curves for a hypothetical cascade $\mathcal{K}$, for $k = 0$ (the red line – highest intercept on the y axis) through $k = 4$ (the horizontal line: the larger the value of k, the lower its y-intercept).

As stated when defining it (Definition 2), the concept of the dependence parameter λ was introduced for analysis purposes only: it plays no role in either the cascade-synthesis algorithms in [5] or our run-time Algorithms 1 and 2. But the observation, evident in Figure 5, that different curves may offer the minimum expected classification duration for different ranges of values of λ, suggests there it may be worth attempting to learn λ's value at run-time: if this value can be discovered, then Algorithm 2 could be called with the corresponding value of k that minimizes the expected classification duration.

We can learn λ's value at run-time for a given cascade $\mathcal{K}$ in the following manner. Prior to run-time, we first pre-compute the "$(A - B\lambda)$" representations of the smoothness curves for each value of k, thereby obtaining the (mathematical equivalent of) Figure 5 for $\mathcal{K}$, and then choose an initial value for k as previously discussed in Section V-A (to maximally exploit dependencies subject to QoS robustness bounds). During runtime we observe the algorithm's behavior as discussed below over a series of inputs – we will refer to the interval (equivalently, the number of inputs in the interval) over which such observation is done as an *epoch*, and use these observations to obtain an estimate of λ's true current value. If our smoothness curves reveal that a different choice of k would yield a smaller expected execution duration for this value of λ, then we change k accordingly, and a new epoch commences.

Estimating λ's true value. There are several ways, of varying degrees of statistical sophistication and reliability, for estimating λ's true current value during an epoch. We could simply measure the average observed classification duration over the epoch and then determine, from the smoothness curve for the current value of k, the value of λ for which the expected classification duration equals this average observed duration.[8] For greater statistical accuracy, we can obtain a *maximum likelihood estimate*[9] of λ's true value directly, rather than via the expected classification duration. We could, for instance, count the following over the epoch for each i, $1 \leq i \leq N$:

1) A_i: the number of inputs x_t for which the classifier K_i is the minimum-WCET classifier to successfully classify

[7] Such statistical QoS requirements are frequently found in *Service Level Agreements* (SLAs) between service providers and clients; although they may not be safety-critical, failure to meet a SLA may have adverse consequences in the form of penalties and reputation loss.

[8] Note that the slope of a smoothness curve determines its sensitivity to detecting changes in the true value of λ from the observed average classification duration (at one extreme, the smoothness curve for the static algorithm – in Figure 5, the brown "$k = 4$" line – is completely horizontal and hence the observed average classification duration yields no information about the true underlying value of λ). The epoch size should reflect this: while relatively small epochs may be adequate for small values of k, the epoch size needs to be larger when k is larger and slopes, correspondingly smaller. (If the slope is very small, one should occasionally "explore" with a smaller k (larger slope) — the largest possible that satisfies robustness constraints — for a small duration in order to check whether the current estimate of λ needs updating.)

[9] The maximum likelihood estimate (MLE) of a parameter is the parameter value whose probability of generating the observations is the greatest – see, e.g., [17] for a tutorial introduction. Since discussion of fairly standard statistical techniques is not the purpose of this paper we only give a very high-level description of the MLE approach to estimating λ's value here.

$$\textbf{Dependencies:}\quad \Pr\{K_i \text{ executes}\} = \begin{cases} \Pr\{b \in [1,\ldots,k+1]\} = P_{k+1}, & i = 1 \\ \Pr\{b \in [i,i+1,\ldots,i+k]\} = P_{i+k} - P_{i-1}, & 2 \le i \le N-k \\ \Pr\{b \in [i,i+1,\ldots,N]\} = 1 - P_{i-1}, & i > N-k \end{cases}$$

$$\textbf{No dependencies:}\ \Pr\{K_i \text{ executes}\} = \begin{cases} \Pr\{b \in [1,\ldots,k+1]\} = P_{k+1}, & i = 1 \\ \overbrace{P_{i+k} - P_{i+k-1}}^{\Pr\{b=i+k\}} + \overbrace{P_{i+k-1}}^{\Pr\{b<i+k\}} \times \overbrace{(1-P_{i-1})}^{\Pr\{K_{i-1} \text{ fails}\}}, & 2 \le i \le N-k \\ \Pr\{K_{i-1} \text{ fails (since } b < i)\} = 1 - P_{i-1} & i > N-k \end{cases}$$

Fig. 6. Probability that a given classifier K_i executes in the presence/ absence of dependencies (see proof of Theorem 4)

both x_t and x_{t-1}.

2) B_i: the number of inputs x_t for which the classifier K_i is the minimum-WCET classifier to successfully classify x_t, but the minimum-WCET classifier to have successfully classified x_{t-1} is not K_i.

Having obtained these A_i, B_i values, it is straightforward to apply standard MLE techniques to estimate λ by constructing the following log likelihood function for λ:

$$L(\vec{x};\lambda) = \prod_{t=1}^{\#\vec{x}-1} \Pr[x_{t+1}|x_t]$$

Define $Q_i = \big(\lambda + (1-\lambda)\cdot(P_i - P_{i-1})\big)$, we also know that

$$\Pr[x_{t+1}|x_t] = \begin{cases} Q_{x_t} & , x_{t+1} = x_t \\ (1-Q_{x_t}) \cdot \frac{P_{x_{t+1}} - P_{x_{t+1}-1}}{1-(P_{x_t} - P_{x_t-1})} & , x_{t+1} \neq x_t \end{cases}$$

Therefore, we can express $L(\vec{x};\lambda)$ in terms of A_i's and B_i's and λ. We then obtain the lambda value that maximizes the log likelihood function, i.e., best explains the observed dependencies, by setting $\frac{\partial}{\partial\lambda}L(\lambda)$ equal to zero and solving for λ. The associated confidence interval is obtained by determining the Fisher information [18] of these observations, and an epoch terminates (and the value of λ gets updated) when the measured confidence exceeds a pre-specified threshold (e.g., 95%); a new epoch commences.

VII. Context and Future Research Directions

The real-time computing community has established a rich body of results on the real-time properties of IDK cascades. This prior work has all assumed that each input to be classified is drawn from the same underlying distribution. However, an important use-case for IDK cascades is perception, and many mobile perception pipelines require that sequences of readings obtained by some sensor each be classified. It is reasonable to hypothesize some dependencies between successive inputs in such time-series readings from a single sensor source. Hence in this paper we have extended prior work to allow for the exploitation of dependencies, if present, in the stream of inputs that an IDK cascade is tasked with classifying.

Since IDK classifiers are inherently learning-based, it is reasonable to apply recently-developed techniques from the algorithmic AI community to their analysis. A recent paper [7] had applied the Algorithms Using Predictions framework to the analysis of IDK cascades; we have also done this, but in a very different direction than in [7]: whereas [7] looked on the probability parameters as predictions, here we instead consider the presence and degree of dependence amongst successive inputs as a prediction.

We have also explored an aspect of the Algorithms Using Predictions framework that was not addressed in [7] – *learning*. In this we drew inspiration from another widely-studied topic in algorithmic AI: reinforcement learning, whereby we occasionally perform additional exploratory computations in the interest of perhaps improving future performance.[10]

Future work. There are multiple directions in which this work could be carried further forward; here we list a couple that we find particularly interesting:

1) It is not yet clear to us how to best extend the methods and results of this paper to cascades built using classifiers that do not possess the containment property (that the individual classifiers can be strictly ordered from least powerful to most powerful such that any input successfully classified by a classifier is also successfully classified by all more powerful classifiers). Indeed coming up with mathematically rigorous reasonable definitions of dependence parameters, that generalize Definition 2 to cascades of such classifiers, seems quite challenging and may require the use of more deep-learning detailed approaches (such as "feature vectors") to define the degree of similarity between inputs.
2) We are assuming hard-real-time environments in which run-time algorithms reserve adequate computation to be able to successfully classify each input even in the worst case. In the event of early successful classification, the entire remaining reserved computation is turned over for exploration for potentially improving future performance. It would be interesting to study algorithms for exploration that do not get "free" use of all the unused capacity, by coming up with cost measures for exploratory computation; perhaps the technique of *Explorable Uncertainty*, that has recently attracted attention in the algorithmic AI commu-

[10]We point out that we are not the first to have noticed the relationship between reinforcement learning and IDK cascades – Srikishan et al. [19] have independently also observed this.

nity (see, e.g., [20]), may help formalize the tradeoff of the cost of current exploration versus potential future benefits.

3) As mentioned above, prior work [7] has considered the probability characterization of IDK classifier behavior as a prediction, while we have accepted that these are ground truth and instead consider the presence and degree of dependence amongst successive inputs as a prediction. In future, it would be interesting to incorporate both forms of predictions into a single holistic analysis.

Acknowledgements. This research was funded in part by the US National Science Foundation (Grant Nos CNS-2141256 and CPS-2229290), and Innovate UK SCHEME project (10065634). EPSRC Research Data Management: No new primary data was created during this study.

APPENDIX

We now provide a brief proof of the claim made in Theorem 4 that the expected classification duration $\mathbb{E}[C_{\text{dyn-k}}(\kappa, k)]$ of Algorithm 2 equals

$$\sum_{i=1}^{N}\Big(\big(P_{i+k} - P_{i+k-1}\,P_{i-1} - \lambda\,(P_{i-1} - P_{i+k-1}\,P_{i-1})\big)\,C_i\Big)$$

As in the proof of Theorem 1, we will express $\mathbb{E}[C_{\text{dyn-k}}]$ as a sum of the contributions of each individual classifier K_i to the expected classification duration, which is simply the product of its WCET C_i and the probability that it executes. Now, the total probability that any classifier executes equals

$$\begin{aligned}&\lambda \times \Pr\{\textit{it executes when dependencies happen}\}\\ &+(1-\lambda) \times \Pr\{\textit{it executes when dependencies do not happen}\}\end{aligned}$$

We separately compute the probability that each classifier executes in each of these two cases in Figure 6; the reasoning behind these computations is essentially identical to the reasoning used in the proof of Theorem 1.

It remains to compute, for each classifier, the total probability that it executes. For $i = 1$ and $i > N-k$, these are easily seen to equal P_{k+1} and $(1-P_{i-1})$ respectively. For $2 \leq i \leq N-k$, we have

$$\begin{aligned}&\Pr\{K_i \text{ executes}\}\\ &\quad= \lambda(P_{i+k} - P_{i+1})\\ &\qquad+(1-\lambda)\big(P_{i+k} - P_{i+k-1} + P_{i+k-1}(1-P_{i-1})\big)\\ &\quad= \lambda(P_{i+k} - P_{i+1}) + (1-\lambda)\big(P_{i+k} - P_{i+k-1}\,P_{i-1})\big)\\ &\quad= \big(P_{i+k} - P_{i+k-1}\,P_{i-1}\big) - \lambda\big(P_{i-1} - P_{i+k-1}\,P_{i-1}\big) \quad (9)\end{aligned}$$

Recalling that we have adopted the notational conventions $P_0 \stackrel{\text{def}}{=} 0$ and $P_i \stackrel{\text{def}}{=} 1$ for $i > N$, it may be verified that instantiating Expression (9) with $i \leftarrow 1$ yields P_{k+1} and instantiating it with any $i > N-k$ yields $(1-P_{i-1})$. Theorem 4 follows.

REFERENCES

[1] L.-H. Wen and K.-H. Jo, "Deep learning-based perception systems for autonomous driving: A comprehensive survey," *Neurocomputing*, vol. 489, pp. 255–270, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222003113

[2] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, and J. Gonzalez, "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*, A. Globerson and R. Silva, Eds. AUAI Press, 2018, pp. 580–590. [Online]. Available: http://auai.org/uai2018/proceedings/papers/212.pdf

[3] 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, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[4] T. Abdelzaher, K. Agrawal, S. Baruah, A. Burns, R. I. Davis, Z. Guo, and Y. Hu, "Scheduling IDK classifiers with arbitrary dependences to minimize the expected time to successful classification," *Real-Time Systems*, vol. 59, no. 3, pp. 348–407, Sep. 2023.

[5] S. Baruah, A. Burns, R. Davis, and Y. Wu, "Optimally ordering IDK classifiers subject to deadlines," *Real Time Syst.*, vol. 59, no. 1, pp. 1–34, 2023. [Online]. Available: https://doi.org/10.1007/s11241-022-09383-w

[6] S. Baruah, I. Bate, A. Burns, and R. Davis, "Optimal synthesis of fault-tolerant IDK cascades for real-time classification," in *Proceedings of the 30th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2024)*. IEEE, 2024.

[7] S. Baruah, A. Burns, and R. I. Davis, "Optimal synthesis of robust IDK classifier cascades," in *2023 International Conference on Embedded Software, EMSOFT 2023, Hamburg, Germany, September 2023*, C. Pagetti and A. Biondi, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3609129

[8] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A unified deep learning framework for time-series mobile sensing data processing," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017, p. 351–360. [Online]. Available: https://doi.org/10.1145/3038912.3052577

[9] D. Liu, T. F. Abdelzaher, T. Wang, Y. Hu, J. Li, S. Liu, M. Caesar, D. Kalasapura, J. Bhattacharyya, N. Srour, J. Kim, G. Wang, G. Kimberly, and S. Yao, "IoBT-OS: Optimizing the sensing-to-decision loop for the Internet of Battlefield Things," in *31st International Conference on Computer Communications and Networks, ICCCN 2022, Honolulu, HI, USA, July 25-28, 2022*. IEEE, 2022, pp. 1–10. [Online]. Available: https://doi.org/10.1109/ICCCN54977.2022.9868920

[10] Y. Hu, S. Liu, T. Abdelzaher, M. Wigness, and P. David, "On exploring image resizing for optimizing criticality-based machine perception," in *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021, pp. 169–178.

[11] S. Baruah, A. Burns, and Y. Wu, "Optimal synthesis of IDK-cascades," in *RTNS'2021: 29th International Conference on Real-Time Networks and Systems, Nantes, France, April 7-9, 2021*, A. Queudet, I. Bate, and G. Lipari, Eds. ACM, 2021, pp. 184–191. [Online]. Available: https://doi.org/10.1145/3453417.3453425

[12] T. F. Abdelzaher, S. K. Baruah, I. Bate, A. Burns, R. I. Davis, and Y. Hu, "Scheduling classifiers for real-time hazard perception considering functional uncertainty," in *Proceedings of the 31st International Conference on Real-Time Networks and Systems, RTNS 2023, Dortmund, Germany, June 7-8, 2023*. ACM, 2023, pp. 143–154. [Online]. Available: https://doi.org/10.1145/3575757.3593649

[13] M. Mitzenmacher and S. Vassilvitskii, "Algorithms with predictions," in *Beyond the Worst-Case Analysis of Algorithms*, T. Roughgarden, Ed. Cambridge University Press, 2021, pp. 646–662.

[14] ——, "Algorithms with predictions," *Commun. ACM*, vol. 65, no. 7, pp. 33–35, jun 2022. [Online]. Available: https://doi.org/10.1145/3528087

[15] A. Lindermayr and N. Megow, "Open source project: ALGORITHMS WITH PREDICTIONS," https://algorithms-with-predictions.github.io, accessed: 2023-08-02.

[16] K. Agrawal, S. Baruah, M. A. Bender, and A. Marchetti-Spaccamela, "The Safe and Effective Use of Low-Assurance Predictions in Safety-Critical Systems," in *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, ser. Leibniz International Proceedings in Informatics

(LIPIcs), A. V. Papadopoulos, Ed., vol. 262. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, pp. 3:1–3:19. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2023/18032

[17] I. J. Myung, "Tutorial on maximum likelihood estimation," *Journal of Mathematical Psychology*, vol. 47, no. 1, pp. 90–100, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022249602000287

[18] A. Ly, M. Marsman, J. Verhagen, R. Grasman, and E.-J. Wagenmakers, "A tutorial on fisher information," 2017.

[19] B. Srikishan, A. Tabassum, S. Allu, R. Kannan, and N. Muralidhar, "Reinforcement learning as a parsimonious alternative to prediction cascades: A case study on image segmentation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 13, pp. 15 066–15 074, Mar. 2024. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/29428

[20] C. Dürr, T. Erlebach, N. Megow, and J. Meissner, "Scheduling with Explorable Uncertainty," in *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 94. Cambridge, United States: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Jan. 2018, pp. 30:1–30:14. [Online]. Available: https://hal.science/hal-02074087