Noname manuscript No.
(will be inserted by the editor)

Scheduling IDK Classifiers with Arbitrary Dependences to Minimize the Expected Time to Successful Classification

Tarek Abdelzaher* · **Kunal Agrawal** · **Sanjoy Baruah** · **Alan Burns** · **Robert I. Davis** · **Zhishan Guo** · **Yigong Hu**

Received: November 2022 / Revised January 2023 / Accepted: date

* The author's names are listed in alphabetical order.

T. Abdelzaher
University of Illinois
Urbana Champaign,
USA
E-mail: zaher@cs.uiuc.edu

K. Agrawal
Washington University in Saint Louis
Saint Louis, MO,
USA
E-mail: kunal@wustl.edu

S. Baruah
Washington University in Saint Louis
Saint Louis, MO,
USA
E-mail: baruah@wustl.edu

A. Burns
University of York
York, UK
E-mail: alan.burns@york.ac.uk

R. I. Davis
University of York
York, UK
E-mail: rob.davis@york.ac.uk

Z. Guo
North Carolina State University
Raleigh,
USA
E-mail: zguo32@ncsu.edu

Y. Hu
University of Illinois
Urbana Champaign,
USA
E-mail: yigongh2@illinois.edu

Abstract This paper introduces and evaluates a general construct for trading off accuracy and overall execution duration in classification-based machine perception problems – namely, the *generalized IDK classifier cascade*. The aim is to select the optimal sequence of classifiers required to minimize the expected (i.e. average) execution duration needed to achieve successful classification, subject to a constraint on quality, and optionally a latency constraint on the worst-case execution duration. 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 "I Don't Know" (IDK) if it is unable to do so with the required level of confidence. An ensemble of several different IDK classifiers may be available for the same classification problem, offering different trade-offs between effectiveness (i.e. the probability of successful classification) and timeliness (i.e. execution duration). A model for representing such characteristics is defined, and a method is proposed for determining the values of the model parameters for a given ensemble of IDK classifiers. Optimal algorithms are developed for sequentially ordering IDK classifiers into an IDK cascade, such that the expected duration to successfully classify an input is minimized, optionally subject to a latency constraint on the worst-case overall execution duration of the IDK cascade. The entire methodology is applied to two real-world case studies. In contrast to prior work, the methodology developed in this paper caters for arbitrary dependences between the probabilities of successful classification for different IDK classifiers. Effective practical solutions are developed considering both single and multiple processors.

Keywords real-time; classifiers; optimal ordering; DNN; arbitrary dependences;

1 Introduction

This paper investigates a generalized approach for trading-off quality versus latency of classification in machine perception systems, called *generalized IDK classifier cascades*. The approach uses an ensemble of classifiers that can individually either return a class or say "I don't know" (IDK). The problem is to decide on the optimal order in which classifiers should be run in order to minimize the time to successful classification, optionally within a maximum latency constraint, given their different execution times, their individual success probabilities, and their dependences, i.e. the probability that one will succeed given that another has failed.

The broad challenge of investigating the trade-offs between latency and classification accuracy is motivated by the increasing role of machine perception in modern intelligent real-time applications, such as drones (Kangunde et al, 2021), autonomous cars (Shi et al, 2017; Bechtel et al, 2018), and medical IoT systems (Balaskas and Siozios, 2019; Hossain et al, 2022). Perception in such Cyber-Physical Systems is increasingly being performed using classifiers that are based on *Deep Learning*, thus generating interest in understanding and optimizing the trade-offs between the quality of

deep-learning-based perception and perception latency. Examples of such trade-off optimization approaches, proposed in recent literature, include: (i) adaptive neural network approximations aimed at meeting latency constraints (Bateni and Liu, 2018; Kim et al, 2020; Heo et al, 2020; Yao et al, 2020), and (ii) adaptive model-switching systems that pick one of multiple neural network versions depending on the time available (Hu et al, 2021a).

Another direction is to consider ensembles of IDK classifiers (Trappenberg and Back, 2000; Khani et al, 2016). When applied to neural networks, IDK classifiers build on the intuition, mentioned by Hu et al (2021a), that switching *entire* neural network models, using a hypothetical optimal model-switching algorithm, is in principle generally superior to adapting only *some* neural network parameters dynamically, as is done in adaptive approximation systems. This is because training an adaptive neural network must optimize any *non-adaptive parameters* for some *compromise* among all possible values of the adaptive parameters (Hu et al, 2021a). This compromise typically reduces run-time output quality for any specific instantiation of the adaptive parameters. One way to avoid this compromise is to train an entirely *separate neural network* for each different point in the quality/latency trade-off space. The problem with the latter approach is that optimal switching is impossible without a form of *clairvoyance*. This is because a decision on which neural-network version to execute must be made *ahead of time*, and not after some partial processing of the input has taken place. Since the optimal decision might depend on the level of difficulty of the input, having to decide before the input has been processed is a challenge.

IDK classifier cascades (Wang et al, 2018) address this challenge by taking a different design approach. Like model-switching, they use an ensemble of different classifiers; however, they assume that the chosen classifier, when not confident enough, can return an "I don't know" value, that will then prompt the system to choose another classifier, thereby executing a situation-dependent sequence similar to adaptive approximation approaches. Prior work by Baruah et al (2021, 2022) developed analytical results for the special cases of IDK classifiers where the probabilities of successful classification by the respective classifiers were either *independent*, or *fully dependent* of one another. This paper generalizes from those special cases to the case of a *general IDK classifier cascade*, meaning one with arbitrary dependences between the different classifiers. Further, prior work by Baruah et al (2021, 2022) also relies on the concept of a *deterministic classifier* that is guaranteed to always make a successful classification, albeit typically at the expense of a long execution time. This paper recognizes that such a construct may not always be possible in practice, and therefore also provides solutions based on a *classification threshold*. This classification threshold specifies the minimum probability, e.g. 0.925, such that in the long run any IDK cascade employed must be able to successfully classify at least 92.5% of its inputs. Finally, Baruah et al (2021, 2022) consider only single processor systems, whereas this paper also considers solutions for multiple processors. Here we use the term *processor* in the broad sense of an independent processing unit. Given that classifiers often make use of hardware

accelerators such as GPUs, such a *processor* may include both a CPU and a GPU.

Results from experimentation with real-life data sets, including vision as well as acoustic and seismic sensors, show that the use of different sensing modalities and neural network topologies often creates partial correlations between the behaviors of different classifiers. Exploiting those correlations, together with knowledge of relevant classifier execution times, one can arrive at an optimal order in which to execute the classifiers that appreciably improves the expected duration, i.e. the average time to successful classification.

The remainder of the paper is organized as follows. In Section 2 we review the essential background on IDK classifiers and elaborate the problem. In Section 3 we present our model for representing collections of IDK classifiers that may have arbitrary dependences between them. In Section 4 we explain how to assign values to the parameters in such a model, and illustrate the process on two real-world case studies. In Section 5 we present algorithms for synthesizing optimal IDK cascades on a single processor from individual IDK classifiers that are specified according to our model. In Section 6 we use these algorithms to synthesize optimal IDK cascades for the real-world case studies modeled in Section 4. Section 7 extends our approach to multiple processors, including synthesizing optimal IDK cascades for one of the real-world case studies on two, three, and four processors. Section 8 concludes the paper with a list of future research directions.

2 Background

Perception in autonomous mobile Cyber-Physical Systems is increasingly being performed using classifiers that are based on Deep Learning. Classifiers that are used in this way must be able to make accurate predictions in real time using limited computational resources. However, in mainstream machine learning research, much current work 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 very accurate classifiers that take substantial time to process even simple inputs that should be straightforward to classify. For example, Wang et al (2018) showed that for a considerable fraction of the ImageNet 2012 benchmark (Russakovsky et al, 2015a), 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.

2.1 IDK Classifier Cascades

One approach aimed at achieving appropriate accuracy-latency trade-offs is the use of *IDK classifiers* (Trappenberg and Back, 2000; Khani et al, 2016). An IDK classifier is obtained from an existing *base* classifier by attaching a computationally light-weight augmenting classifier that enables the IDK classifier to return an auxiliary "I Don't Know" (IDK) class depending on the degree of uncertainty in the predictions made by the base classifier[1]. In other words, an IDK classifier classifies an input as being in the IDK class if the base classifier is not able to predict some actual class for that input with a level of confidence that exceeds a predefined *confidence threshold*. (In Section 4 we describe in detail how to obtain an IDK classifier from a base classifier.) We define *success* for an IDK classifier as the act of outputting a non-IDK class. Note that, success only means that the true object class is recognized with a sufficiently high probability. It does not imply a complete absence of misclassifications with respect to the ground truth.

In principle, it is possible to generalize the notion of "success" of an IDK classifier to include meeting additional quality criteria. For example, we might refer to a neural network for target detection and classification as "successful" if it not only returned a high confidence in the target class but also in the object location. In machine learning, such a network (e.g. YOLO) is said to perform both *detection* and *classification*, however, for purposes of this paper, we use the term IDK classifier.

Multiple different IDK classifiers, with different execution times and probabilities of success (i.e. of not outputting IDK), may be devised for the same classification problem. Wang et al (2018) 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.
3. Otherwise (i.e. the classifier outputs IDK), the subsequent classifier in the IDK cascade is invoked and the process continues from step 2.

If it is a requirement that all inputs be successfully classified, in the sense of success defined above, then it must be the case that the last classifier in the cascade succeeds. We refer to a classifier that always succeeds as a *deterministic classifier*. There are various forms that the deterministic classifier may take. Wang et al (2018) proposed that all inputs that are not classifiable by the IDK classifiers be pushed up to a human expert, who thus takes on the role of the deterministic classifier. In some applications, a fully developed Deep Learning Neural Network may be sufficiently accurate to take on the role of deterministic classifier; however, its computational needs may be so large that it should only be executed when absolutely necessary, with other more

[1] This notion is similar to that of classifiers that *defer* (Madras et al, 2018).

efficient classifiers used if they can cater for typical inputs. Finally, to deal with applications that exhibit high levels of uncertainty, it may be necessary to introduce the class *unclassifiable* that the final arbiter in the cascade, the deterministic classifier, can output if a real class cannot be identified with the required level of confidence. Alternatively, a classification threshold may be specified requiring that any IDK cascade employed has an overall probability (long run frequency) of success that is no lower than the classification threshold. We return to the concept of a classification threshold in Section 6.4.

2.2 The Generalized IDK Classification Problem

Given a collection of several different IDK classifiers for a particular classification problem, this paper considers how they should be sequentially ordered for execution so as to minimize the expected duration to successfully classify an input, optionally subject to guaranteeing to meet a latency constraint on the worst-case execution duration of the IDK cascade.

Probabilistic characterization of classifiers. Observe that this problem, by seeking to minimize expected (i.e. *average*) duration, implicitly requires a probabilistic characterization of the likelihood of a classifier successfully classifying any given input, as opposed to outputting IDK. Obtaining such probabilistic characterizations that are accurate and useful is crucial to the successful solution of the problem. In addition to characterizing each classifier individually, the relationships between different classifiers must also be characterized. Two IDK classifiers may behave in a manner that is *independent* of one another. By independent, we mean that the probability that one classifier will output a real class is independent of whether it is run on all inputs or on only those inputs where the other classifier outputs IDK. For example, intuitively it is a reasonable hypothesis that an IDK classifier that processes camera images of a scene may perform very differently from an IDK classifier that processes radar signals of the same scene – these two classifiers may be expected to exhibit behavior that is mutually independent. Indeed, Madani et al (2012, 2013) provide ample and persuasive experimental evidence that very different sources of information such as text, audio, and video features obtained from the same scenario, and hence with the same ground truth, are effectively independent. At the other extreme, two image-based classifiers that use the same input image scaled to different resolutions (Hu et al, 2021b) may exhibit behavior that is *fully dependent*: the less powerful classifier is only able to successfully classify a strict subset of the inputs that the more powerful classifier can identify.

Prior research has proposed solutions to the problem of synthesizing IDK cascades out of collections of fully independent (Baruah et al, 2021) and fully dependent (Baruah et al, 2022) classifiers, as well as combinations of the two (Baruah et al, 2022). However, other forms of correlation and dependency between classifiers that are more complicated than independence or full

dependence can occur and are likely to be more common. For example, this is true for the two real-world case studies that we consider in this paper.

This work. This paper investigates the problem of optimally synthesizing IDK cascades from collections of classifiers that are correlated according to more general relationships than independence or full dependence. The main *contributions* are as follows:

- Extending the real-time model for IDK classifiers that was presented by Baruah et al (2021, 2022) for fully dependent and independent classifiers, by proposing a framework for specifying arbitrary forms of dependence between different classifiers.
- Extending current practice in the training and testing of classifiers: (i) to obtain IDK classifiers from base classifiers, and (ii) to obtain a probabilistic characterization of their expected run-time behavior, including their mutual dependences.
- Proposing algorithms for optimally synthesizing IDK cascades that are so characterized, thus extending the algorithmic framework that was initiated by Baruah et al (2021, 2022) for single processors, substantially enhancing its practical applicability.
- Proving key properties of optimal IDK cascades on multiple processors, and proposing algorithms for synthesizing them.
- Illustrating all of the contributions listed above —modeling; obtaining probabilistic characterizations; and synthesizing optimal IDK cascades— via two real-world case studies. The first case study is from the domain of image-recognition, which has been the focus of much research on classification using Deep Learning. The second case study comes from a Cyber-Physical Systems application that seeks to autonomously detect hostile presence in a battlefield environment, for use in future military systems.

In common with prior work (Baruah et al, 2021, 2022), 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 effectively independent.

3 System Model: Motivation and Definitions

We consider a collection of n IDK classifiers K_1, K_2, $\ldots, K_n$ that are all designed to solve a given classification problem. Prior work taking a real-time perspective on IDK cascades (Baruah et al, 2021, 2022) has characterized each classifier K_i by a pair of parameters (C_i, P_i), specifying its *execution time* C_i (assumed by Baruah et al (2021, 2022) to be constant) and its *success probability* P_i. These parameters denote that when invoked on an input, the classifier K_i takes C_i time units to complete execution and returns a real class, rather

than IDK, with probability P_i, where $0 < P_i \leq 1$. In this paper, we employ a more nuanced characterization of execution times, with $\bar{C}_i$ representing the average-case execution time and C_i representing the worst-case execution time of classifier K_i. We discuss how values for all of the parameters may be determined in Section 4.

Dependences amongst classifiers. While the parameters, $\bar{C}_i$, and P_i, characterize the expected behavior of each individual classifier, they do not address the relationship, i.e. the mutual dependences, between the behaviors of the different classifiers. Baruah et al (2021) assumed that all the classifiers are pairwise independent, while Baruah et al (2022) additionally considered the case where some classifiers are pairwise fully dependent. However, more complex relationships also arise in practice, with classifiers exhibiting 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). Dependences may also be experienced even among seemingly different modalities such as sound and vision. For example, a moving object that gets partially obscured by a barrier on an otherwise open plain may be harder to classify using both vision, due to occlusion, and acoustics, due to sound reflecting off the barrier thus reducing the signal volume). These dependences may give rise to observable behavior that is to some extent correlated.

Consider a simple example of two classifiers, K_1 and K_2. The probability of failure (i.e. an output of IDK) from K_1 is $(1-P_1)$; similarly for K_2 it is $(1-P_2)$. Let $P(F)$ denote the observed probability of K_1 and K_2 both outputting IDK. It is evident that $0 \leq P(F) \leq 1 - \max(P_1, P_2)$.

- If $P(F) = 1 - \max(P_1, P_2)$, then the classifiers are *fully dependent*.
- If $P(F) = (1 - P_1) \times (1 - P_2)$ then the classifiers are *independent*.
- If $(1 - P_1) \times (1 - P_2) < P(F) < 1 - \max(P_1, P_2)$ then there is partial dependency resulting from an *observed positive correlation*.
- If $0 \leq P(F) < (1-P_1) \times (1-P_2)$ then there is partial dependency resulting from an *observed negative correlation*.

Clearly it is best, if possible, for the classifiers developed or selected for use in an IDK cascade to behave with a negative correlation. If this is not possible then the closest to independence is desirable. The worst case, giving the highest probability of failure (i.e. returning IDK), is when the classifiers are fully dependent.

Conceptually, it is convenient to think of the probability space as a Venn Diagram partitioned into 2^n distinct regions, see Figure 1 for the case of $n = 3$ classifiers, with each region corresponding to one of the 2^n possible combinations of the n individual classifiers successfully classifying an input or returning IDK. In Section 4 we describe how to obtain the actual probabilities associated with each of these regions.

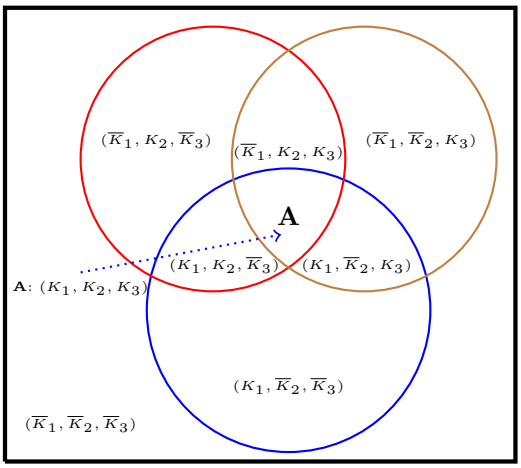

Fig. 1 The 2^n disjoint regions in the probability space for $n = 3$. The blue circle denotes the part of the space where K_1 is successful; the red circle, where K_2 is successful; and the brown circle, where K_3 is successful. These three circles partition the probability space into $2^3 = \mathbf{8}$ disjoint regions; each of these is labeled with a 3-tuple, with K_i ($\overline{K}_i$, respectively) denoting that the region corresponds to the IDK classifier K_i returning an actual class (resp. IDK). (Note that the entire space can be thought of as denoting the region in which the deterministic classifier is successful.)

General model. Summarizing the above, a collection of n IDK classifiers $K_1, K_2, \ldots, K_n$ for the same classification problem are characterized by their execution times (their $\bar{C}_i$ and C_i parameters) and the probability values associated with each of the 2^n disjoint regions in the Venn Diagram of the probability space. (Note, the parameter P_i in the model previously-proposed by Baruah et al (2021, 2022) is easily recovered from this model, by simply summing over the 2^{n-1} disjoint regions that lie within the circle of the Venn Diagram representing the IDK classifier K_i.)

The presence of a latency constraint. We also consider a variant of the problem in which the objective is to determine the optimal IDK cascade, that minimizes the expected duration for successful classification, subject to the worst-case execution duration not exceeding a specified latency constraint.

The presence of a classification threshold. We also consider a variant of the problem in which the overall probability of successful classification need not be guaranteed (i.e. 1), but rather must meet or exceed a specified classification threshold.

Reasonable values for n. The number of IDK classifiers, n, that could be used to form an IDK cascade to solve a specific classification problem is application dependent, and many different values may be found in the literature on classifier ensembles. As noted earlier, diversity comes from having classifiers with different types of input, different internal models and different training data. Even a single classifier, such as the image-based example described

previously, can have a number of different pixel resolutions defined and hence give rise to two, three or more distinct, though likely fully dependent, classifiers. Combining such ensembles of classifiers can lead to IDK cascades with eight or more components. In practice, however, we expect that values of n that are much greater than about 12 are unlikely to be commonly encountered in practice. This is important, since our model, comprising $O(2^n)$ parameters, is of a size exponential in n.

Table 1 provides a glossary of selected terminology along with brief informal definitions. These terms are discussed and defined in more detail in Sections 4 and 5.

Table 1 Glossary of selected terminology

Terminology	Informal definition
confidence	A base classifier's self-assessment of the probability that the class it has returned is correct.
confidence threshold	The minimum *confidence* required from a base classifier for the IDK classifier built upon it to return a real class rather than IDK.
classification threshold	The minimum *success probability* required of an IDK cascade,
latency constraint	The maximum permitted execution duration for an IDK cascade.
precision	The probability of correct classification (i.e. equating to the ground truth) when a real class is returned by an IDK classifier.
precision threshold	The minimum *precision* required of an IDK classifier.
successful classification	When an IDK classifier or IDK cascade returns a real class, rather than IDK.
success probability	The probability of *successful classification* for an IDK classifier or an IDK cascade.

4 Populating the Model

Given n different base classifiers for the same classification problem, we now discuss our methodology for first converting each to an IDK classifier, and subsequently determining the model parameters that characterize the expected behavior of these IDK classifiers with regards to their execution times ($\bar{C}_i$ and C_i), their probabilities of success (P_i) and their mutual dependences. We refer to this step as the *profiling phase*.

Prior to this profiling phase each of the classifiers will have been trained and verified using representative input data. In many applications this data can then be used directly in the profiling phase, i.e. no further data is required. Where new data is required for profiling, for example because the training and verification data is proprietary, then it must of course also be representative of the inputs expected during deployment.

Gathering data. During the profiling phase, we test each of the n base classifiers on the same N input samples. Each input sample is a data structure that includes information collected from all sensing modalities used by the respective classifiers. It is expected that each sample comprises information relating to a valid ground-truth class, i.e. a class that the respective classifiers were trained to identify. There is no constraint on the format of the respective modalities, other than being consistent with the input format expected by the respective classifiers. For example, in a scenario involving vision, acoustic, and seismic sensing and classification (as in Section 4.2), a single sample could include an image of a target, a 1 second acoustic sound clip of the same target, recorded at 4KHz, and a 1 second seismic time-series measurement of the target, recorded at 100Hz. Further, the number of classifiers used may be different from the number of modalities present in the input sample. For example, a joint acoustic plus seismic classifier would make use of both the acoustic and the seismic information within the sample. By contrast, three different image classifiers could be used that all act on the same video information, but differ in their resolution and execution time.

For each of the N input samples, each classifier outputs an ordered pair $(\mathit{class}, \mathit{confidence})$, where *class* is the class that the classifier has determined most likely matches the input sample, and *confidence* is a real number in the range $[0, 1]$ that indicates how confident the classifier is that the input sample is indeed of the class returned. We store all N of these ordered pairs output by each of the n classifiers for further processing as discussed below. The ground truth, i.e. the actual class, for each of the N input samples is also known and stored.

During the profiling phase, we also measure and store the execution time of each classifier on each input sample. In this paper, we assume that the execution time of each classifier is independent of its actual input, but nevertheless is subject to variation due to other factors related to the hardware platform. This is typically the case because the neural networks used for such processing run on a dedicated GPU. Furthermore, the neural networks generally perform the same computations on each input, resulting in an execution time that depends primarily on the neural network architecture, input size, and GPU type, but not on the actual data values. In principle, some optimizations, such as the use of sparse matrix algebra to accelerate processing of sparse inputs, e.g. a row of all zeros, can result in shortened execution times for some inputs, but the common case of having a dense input has a consistent execution time. In the case studies considered in this paper, we use the average execution time

observed for each classifier K_i over the large number of input samples used in the profiling phase to determine its average execution time $\bar{C}_i$.

Obtaining IDK classifiers. An IDK classifier K_i is obtained from the i'th base classifier, $1 \leq i \leq n$, by defining a *confidence threshold* H_i, $0 \leq H_i \leq 1$, that is used in the following way. Suppose the classifier outputs the ordered pair (X, f) for some input sample, denoting that it believes that the input sample belongs to class X with a confidence equal to f. If $f \geq H_i$ then the IDK classifier K_i outputs the class X, whereas if $f < H_i$ then it outputs IDK.

It remains to explain how to derive a value for the confidence threshold H_i of each classifier K_i. We assume that the application requirements specify a minimum *precision threshold* for successful classification. The precision threshold is a lower bound on the fraction of a classifier's non-IDK classification decisions that must be correct. For example, a precision threshold of 0.95 indicates that at least 95% of the non-IDK classification decisions made by a classifier must be correct, i.e. in agreement with the ground truth. For each base classifier, we sort the N output pairs by their confidence values f, and then choose the lowest value of H_i such that the fraction of samples with $f \geq H_i$ where the classifier returns the correct class, as determined by the ground truth, is no smaller than the precision threshold.

Assigning values to the probability parameters. Once the confidence thresholds have been determined for each of the n IDK classifiers $K_1, K_2, \ldots, K_n$, then we are ready to determine the probabilities associated with each of the 2^n disjoint regions in the Venn diagram representation of the probability space, as illustrated by Figure 1. We do so by processing the data collected during the profiling phase as follows:

1. Suppose that on the j'th input sample ($1 \leq j \leq N$), classifier K_i outputs a real class, then it is deemed to have been successful on the j'th input sample. Otherwise, K_i outputs IDK and is deemed to have been unsuccessful.
2. Once we have considered all N inputs samples for all n IDK classifiers, then we can determine, for each input sample, which of the n classifiers were successful and which were not, thus associating each input sample with exactly one of the 2^n regions in the probability space. We can therefore determine, for each of the 2^n regions in the probability space, what fraction of the N inputs were successfully classified by *only* those classifiers associated with that region of the probability space.

4.1 ResNet Case Study as an Example

We now illustrate the methodology discussed above for obtaining IDK classifiers from base classifiers, and for determining the parameter values that characterize their dependences via probabilities, on a case study drawn from the domain of image classification. We examined five classifiers, which are all variants of the popular ResNet Deep Residual Network (He et al, 2015): ResNet-18,

ResNet-34, ResNet-50, ResNet-100, and ResNet-152. (The number x in ResNet-x denotes the number of layers in the network; larger values of x tend to yield more accurate classifiers that have greater execution times.) We tested all five classifiers on a representative data set of $50,000$ test images drawn from the validation set[2] of the ImageNet Large Scale Visual Recognition Challenge data set (Russakovsky et al, 2015b), and recorded the Top-1 correctness (i.e. the top class to which the classifier matches the input) and the associated confidence for each classifier on each of the test images.

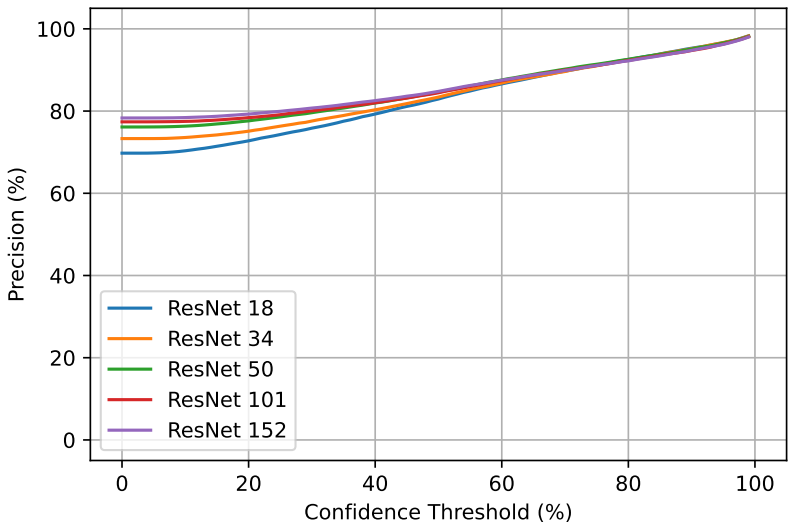

Fig. 2 Populating the model: Setting the confidence threshold.

Figure 2 shows the observed *precision*, i.e. the proportion of non-IDK outputs that are correct, as a function of the confidence threshold. We use the data summarized in this graph to set the confidence threshold, H_i, by assigning to it the minimum value that yields an observed precision that is no lower than the specified *precision threshold*. The confidence thresholds determine the probability that the IDK classifiers will output a real class rather than IDK. Figure 3 shows that as the selected confidence threshold becomes larger, so the IDK classifiers tend to output IDK more frequently. Thus we see that there is a clear relationship between precision and success probability for IDK classifiers: the greater the precision required, the more likely they are to output IDK rather than a real class.

[2]The validation and test data for ImageNet includes 200,000 photographs collected from Flickr and other search engines, hand labelled with the presence or absence of 1000 object categories. The validation set is a random subset consisting of 50,000 of these images with labels.

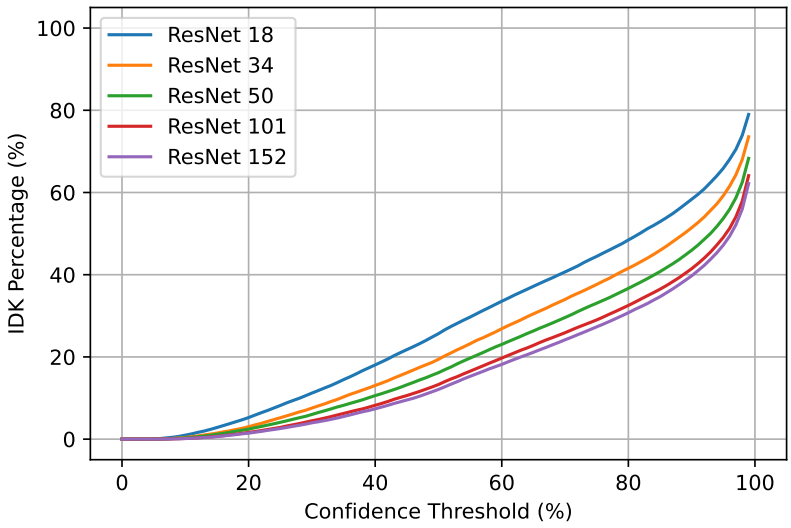

Fig. 3 Populating the model: Percentage of IDKs.

Having set the confidence thresholds, H_i, we can then compute the probabilities associated with each of the 2^n disjoint regions of the probability space. These probabilities are shown in Table 2; to keep the table and this example to a manageable size, we have omitted ResNet-101 from consideration and so are left with four classifiers, and hence $2^4 = 16$ regions of the probability space for which we computed the probabilities. Table 2 can be interpreted as follows: The first four columns correspond to the four classifiers ResNet-18, ResNet-34, ResNet-50, and ResNet152. In each of these columns a 0 indicates that the classifier returns IDK, whereas a 1 indicates that it returns a real class. Thus, each of the 16 rows of the table represents one of the 16 disjoint regions of the probability space. The column entitled "Count" denotes how many of the $N = 50,000$ input samples fall into each of the 16 regions; the entries in the column entitled "Prob-S" are obtained by dividing these counts by N. Prob-S therefore denotes the probability that *exactly* the specific pattern of IDK classifiers indicated by 1's will be able to classify an input, and those indicated by 0's will not and so will return IDK.

For subsequent use in more efficiently calculating the expected duration of an IDK cascade (see Section 6) we require a further set of probabilities given in the column entitled "Prob-A". Prob-A denotes the probability that *at least one* of IDK classifiers indicated by 1's will be able to classify an input, and is calculated from the Prob-S values. For example, in the fourth row of the table, the IDK classifiers denoted by C and D are indicated by a 1. The associated value of Prob-A is therefore the sum of all of the Prob-S values where there is a 1 in either column C or D. (Obtaining all of the Prob-A values takes a time

Table 2 ResNet example

ResNet						
-18	-34	-50	-152	Count	Prob-S	Prob-A
A	*B*	*C*	*D*			
0	0	0	0	15880	0.3176	0
0	0	0	1	3011	0.06022	0.5902
0	0	1	0	1423	0.02846	0.545
0	0	1	1	2465	0.0493	0.64564
0	1	0	0	914	0.01828	0.49216
0	1	0	1	960	0.0192	0.63488
0	1	1	0	545	0.0109	0.60016
0	1	1	1	3382	0.06764	0.66942
1	0	0	0	649	0.01298	0.4284
1	0	0	1	452	0.00904	0.62476
1	0	1	0	304	0.00608	0.5847
1	0	1	1	1208	0.02416	0.66412
1	1	0	0	275	0.0055	0.54442
1	1	0	1	609	0.01218	0.65394
1	1	1	0	500	0.01	0.62218
1	1	1	1	17423	0.34846	0.6824
Totals				50000	1.00	

Classifier	*A*	*B*	*C*	*D*	*E*
H_i	0.890	0.895	0.896	0.910	1
$\bar{C}_i$ (ms)	16.9	27.8	37	101.1	1000
C_i (ms)	22.6	37.5	49.5	125.1	1000

that is quadratic in the number of rows, i.e. in $O(2^{2n}) = O(4^n)$ time, given that there are 2^n rows in the table.)

Table 2 also reports the average-case and worst-case execution time parameters ($\bar{C}_i$ and C_i) of the classifiers on an NVIDIA Jetson TX2, considering the 50,000 runs, as well as the confidence thresholds, H_i, set assuming a required precision threshold of 0.95. Since focus of this paper is not on obtaining definitive worst-case execution times for classifiers, we use the 95-percentile execution time for each classifier as a proxy for its worst-case execution time C_i. (Note, this choice does not impact the methods subsequently presented; higher values for C_i could be used if a high reliability in meeting latency constraints were required). Table 2 also lists a hypothetical deterministic classifier E that always returns a real class, never IDK, and that has a significantly larger (arbitrarily assigned) execution time than any of the IDK classifiers. In Section 6, we use the information presented in Table 2 to synthesize optimal IDK cascades for this case study.

Figure 4 illustrates the execution time distributions for the ResNet classifiers, for the 50,000 input samples, normalized to the mean value for each classifier.

Observe that almost all of the execution times lie between 0.75 and 1.5 times the mean value.

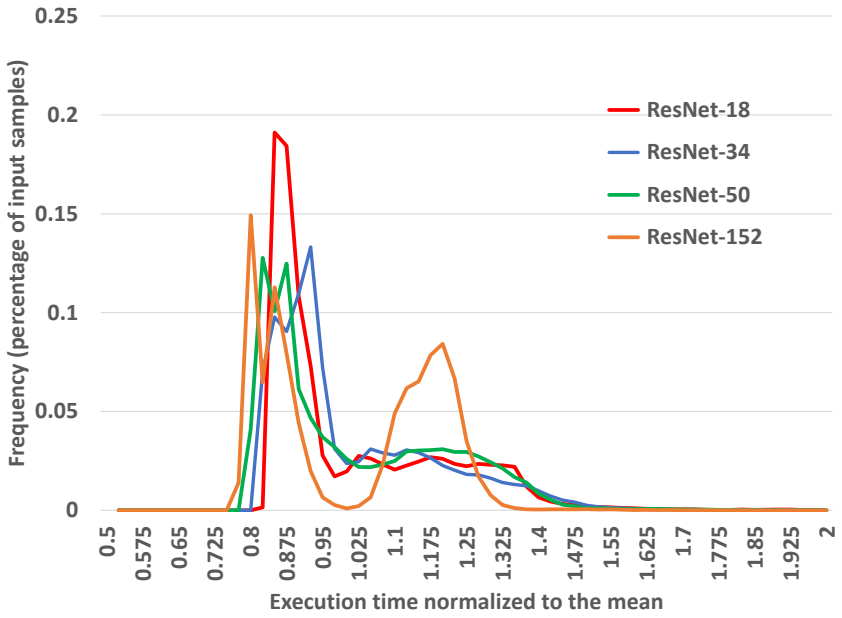

Fig. 4 Execution time distributions for the ResNet classifiers normalized to the mean.

4.2 Multi-Modal Case Study as an Example

All of the classifiers in the ResNet case study discussed in Section 4.1 operate on the same type of information: camera images. In this section, we explore another case study in which different classifiers use very different kinds of information. Here, it is a reasonable hypothesis (Madani et al, 2012, 2013) that some of the classifiers may behave independently. The data used in this case study was collected previously by Liu et al (2022) as part of a project that seeks to autonomously detect the presence of potentially hostile enemy vehicles in a battlefield environment. Three different kinds of sensors were deployed for this purpose: acoustic (a microphone array), seismic (a Raspberry Shake, comprising a Raspberry Pi plus a vertical-axis geophone), and vision (a camera). Each sensor is paired with its own neural network, which acts as the classifier. All three classifiers have adjustable parameters, in particular, each classifier can down-sample to reduce the resolution of the input, thereby trading off a fine granularity of information for faster processing times. The

manner in which the input samples were collected is described by Liu et al (2022) as follows:

"*We deployed our devices on the grounds of the DEVCOM Army Research Laboratory Robotics Research Collaboration Campus [...] and collected seismic and acoustic signals, while different ground vehicles were driven around the site. Data of three different targets: a Polaris all-terrain vehicle, a Chevrolet Silverado, and Warthog UGV were collected. Each target repeatedly passed by the sensors. The total length of the experiment was 115 minutes, spread roughly equally across the three targets. [...] A camera was employed to simultaneously record video of the target.*"

From the data provided by Liu et al (2022), we considered four classifiers as follows:

- A: deepsense_both_contras: This classifier used both seismic and acoustic sensor data as input, and processed the data using the DeepSense neural network architecture (Yao et al, 2017a), trained using contrastive learning (Liu et al, 2021).
- B: cnn_acoustic: This classifier used only acoustic data, and processed it using a standard convolutional neural network.
- C: deepsense_seismic: This classifier used only seismic data, and processed it using the DeepSense neural network architecture (Yao et al, 2017a).
- D: yolov5s-compressed: This classifier used only image data. It was derived from the YOLOv5 neural network (*small* version), after further image compression using the DeepIoT neural network architecture compression framework (Yao et al, 2017b).

We processed the profiling data (class, confidence) outputs for each of the four classifiers for each of the 1800 randomly chosen input samples[3], as described in Section 4.1. First, we assumed a required precision of 0.95 and used this value to compute the confidence threshold for each classifier[4]. Having computed the confidence thresholds, we used these values in the construction of the probabilities (Prob-S) associated with each of the 2^n disjoint regions of the probability space, and subsequently the probabilities (Prob-A) that each distinct subset of the IDK classifiers will be able to successfully classify an input. These probabilities are shown in Table 3. Table 3 also reports the average-case and worst-case execution time parameters ($\bar{C}_i$ and C_i) of the classifiers on a Raspberry Pi 4, considering the 1800 runs, as well as the (arbitrarily assigned) execution time of a hypothetical deterministic classifier E that always returns an actual class. In Section 6, we use the information presented in Table 3 to synthesize optimal IDK cascades for this case study.

Figure 5 illustrates the execution time distributions for the Multi-Modal classifiers, for the 1800 input samples, normalized to the mean value for each

[3]From each input sample, the different classifiers used as their input the different kinds of information that were obtained by the different sensors.

[4]The yolov5s-compressed classifier was accurate in all cases where it returned a non-zero confidence value, hence an arbitrarily small confidence threshold of 0.01 was set.

Table 3 MultiModal example

A: deepsense_both_contras; B: cnn_acoustic;
C: deepsense_seismic; D: yolov5s-compressed

A	B	C	D	Count	Prob-S	Prob-A
0	0	0	0	56	0.031111111	0
0	0	0	1	33	0.018333333	0.298333333
0	0	1	0	35	0.019444444	0.736111111
0	0	1	1	18	0.01	0.816666667
0	1	0	0	11	0.006111111	0.221666667
0	1	0	1	5	0.002777778	0.461111111
0	1	1	0	5	0.002777778	0.807777778
0	1	1	1	4	0.002222222	0.868333333
1	0	0	0	181	0.100555556	0.907222222
1	0	0	1	76	0.042222222	0.940555556
1	0	1	0	698	0.387777778	0.941666667
1	0	1	1	304	0.168888889	0.962777778
1	1	0	0	82	0.045555556	0.921111111
1	1	0	1	31	0.017222222	0.949444444
1	1	1	0	195	0.108333333	0.950555556
1	1	1	1	66	0.036666667	0.968888889
TOTALS				1800	1.00	

Classifier	A	B	C	D	E
H_i	0.705	0.543	0.86	0.01	1
$\bar{C}_i$ (ms)	17.0	3.9	11.4	1440.8	5000
C_i (ms)	19.6	5.3	13.7	1613.2	5000

classifier. Observe that almost all of the execution times lie between 0.75 and 1.5 times the mean value.

4.3 Characterizing Classifier Dependences

With the data that is available in the profiling phase it is possible to estimate the level of dependence, i.e. the degree of correlation, between the behavior of the different classifiers. This can be characterized in a number of different ways. First, Pearson's correlation coefficient[5] can be calculated for each pair of classifiers. This coefficient r is given by:

$$r = \frac{\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \overline{y})^2}}$$

where x_i and y_i are the paired results for the two classifiers on input sample $i = 1 \ldots N$, while $\overline{x}$ and $\overline{y}$ are the respective means of the N results.

[5]See `https://en.wikipedia.org/wiki/Pearson_correlation_coefficient`

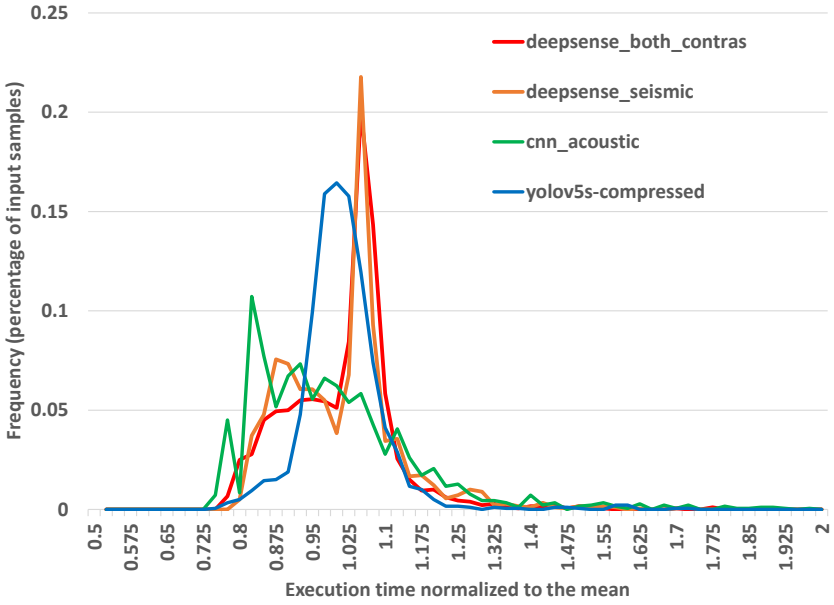

Fig. 5 Execution time distributions for the Multi-Modal classifiers normalized to the mean.

Pearson's correlation coefficient r can take values in the range $[-1, +1]$, with $r = 0$ implying no correlation, and hence possibly independence.[6] The value $r = +1$ implies identical behavior, and at the other extreme $r = -1$ implies exactly opposite behavior.

For the purposes of assessing correlations, the results for each classifier were converted into binary form, with 1 indicating a non-IDK output and 0 indicating IDK. Tables 4 and 5 show the coefficients computed for the ResNet and Multi-Modal case studies respectively, color-coded by the degree of correlation between distinct classifiers: red indicating a strong degree of correlation ($abs(r) > 0.5$), orange a moderate degree of correlation ($0.1 < abs(r) \leq 0.5$), yellow a weak degree of correlation ($0.05 < abs(r) \leq 0.1$), and green a very weak degree of correlation ($abs(r) \leq 0.05$).

As expected, the classifiers in the ResNet case study show a consistent strong positive correlation of between 0.579 and 0.686 between each pair. By comparison the classifiers in the Multi-Modal case study, with its different types of information (acoustic, seismic, and camera), demonstrate lower levels of correlation. Here. classifiers A and C (i.e. deepsense_both_contras and deepsense_seismic) show a moderate degree of positive correlation of 0.265, whereas the other pairs of classifiers have either weak degrees of correlation ($0.05 < abs(r) \leq 0.1$), or very weak degrees of correlation ($abs(r) \leq 0.05$).

[6] Although independence implies a correlation of zero, a correlation of zero does not necessarily imply independence.

Table 4 ResNet Classification: Pearson Correlation Coefficients

	A	B	C	D
A	1	0.668	0.630	0.579
B	0.668	1	0.678	0.639
C	0.630	0.678	1	0.686
D	0.579	0.639	0.686	1

Table 5 Multi-Modal Classification: Pearson Correlation Coefficients

	A	B	C	D
A	1	0.055	0.265	-0.042
B	0.055	1	-0.071	-0.037
C	0.265	-0.071	1	-0.009
D	-0.042	-0.037	-0.009	1

Interestingly, although close to 0 a number of the correlations are negative implying slightly better than independent performance.

Another method of assessing dependences is to compare the probability that all of the IDK classifiers return IDK obtained from the profiling data, with the probability computed assuming: (i) that all of the classifiers are independent, and (ii) that all of the classifiers are fully dependent. These values for the two case studies are shown in Table 6. With the ResNet case study the failure rate computed for arbitrary dependences is somewhat closer to fully dependent behavior than independent behavior. By contrast, with the Multi-Modal case study the failure rate computed for arbitrary dependences is at an intermediate level between the values expected if the IDK classifiers were independent or fully dependent.

Table 6 Probability of Failure

Case Study	Independent	Fully Dependent	Arbitrary
ResNet	0.054125569	0.4098	0.3176
Multi-Modal	0.013371	0.092778	0.031111

The fact that the classifiers comprising the case studies are neither independent nor fully dependent, means that prior approaches (Baruah et al, 2021, 2022) based on those assumptions cannot be used to synthesize optimal IDK cascades.

The above analysis examines dependences between the behavior of the classifiers in terms of successfully classifying an input or returning IDK. We also examined the dependencies between the execution times of the classifiers. For each of the N input samples, we recorded the execution time of each

classifier and categorized these execution times as either: 1 indicating above the median value or 0 indicating equal to or below the median value. We then computed Pearson's correlation coefficient for each pair of classifiers based on this binary data. Recall that the coefficients can range from -1 to $+1$, with a value of 0 implying no correlation. Tables 7 and 8 show these coefficients for the ResNet and Multi-Modal case studies respectively.

Table 7 ResNet Execution Times: Pearson Correlation Coefficients

	A	B	C	D
A	1	-0.003	-0.005	0.007
B	-0.003	1	0.040	0.039
C	-0.005	0.040	1	0.018
D	0.007	0.039	0.018	1

Table 8 MultiModal Execution Times: Pearson Correlation Coefficients

	A	B	C	D
A	1	-0.013	0.024	0.013
B	-0.013	1	0.062	-0.044
C	0.024	0.062	1	-0.013
D	0.013	-0.044	-0.013	1

Observe that for the ResNet classifiers, the correlation coefficients in Table 7 for all pairs of distinct classifiers indicate very weak correlation ($abs(r) \leq 0.05$). Similarly, for the Multi-Modal classifiers, the correlation coefficients in Table 8 indicate either weak correlation ($0.05 < abs(r) \leq 0.1$) or very weak correlation ($abs(r) \leq 0.05$).

Applying a Chi-squared test of independence on such a large sample size, e.g. either 50,000 or 1800 samples, means that the test will deem significant ($p < 0.05$) even small deviations from perfectly independent behavior. Despite this sensitivity, for some pairs of classifiers there was no evidence against a null hypothesis of independence, i.e. the observed variability had a probability $p > 0.05$ of occurring by chance. For other pairs of classifiers there was evidence against a hypothesis of independence. The weak degree of correlation observed in both case studies implies that the majority of the execution time of each classifier is effectively independent of the execution time of other classifiers, with a small effect size of less than 7% that is dependent. Hence regarding the execution time behavior of the classifiers as independent is a reasonable approximation. In the analysis derived in the following sections we assume that the execution times of the classifiers are independent.[7]

[7]This assumption is implicit in the way in which the average execution times are summed together weighted by probabilities.

5 Synthesizing Optimal IDK Cascades on a Single Processor

Once the mutual dependences among the classifiers have been characterized via the tables of probabilities as detailed in Section 4 and illustrated in Tables 2 and 3, we proceed to use this information to determine the optimal IDK cascade: the one with the minimum expected duration, optionally subject to a latency constraint that the worst-case execution duration of the IDK cascade is not permitted to exceed. Initially, we assume a single processor system. The synthesis of optimal IDK cascades for systems with multiple processors is addressed later in Section 7.

Recall that we we have a collection of n IDK classifiers $K_1, K_2, \ldots, K_n$ that solve the same classification problem. In addition, we may also have a single deterministic classifier denoted by K_d that also solves the problem, and which if employed will always be the last classifier in the IDK cascade to complete execution. Initially, we assume that such a deterministic classifier is both available and must be used, i.e. successful classification is a prerequisite. Subsequently, we relax this limitation and require instead that a specified classification threshold must be met. In other words, only IDK cascades with an overall probability of successful classification that meets or exceeds the classification threshold can provide feasible solutions. Before describing the solution, we first introduce some notation and preliminary computations.

Definition 1 *For any subset S of the collection of IDK classifiers, $\widehat{P}[S]$ denotes the probability that at least one of the classifiers in S is successful, i.e. does not return IDK.*

In terms of the Venn diagram representation of the probability space, $\widehat{P}[S]$ denotes the probability measure inside the union of the circles corresponding to each of the classifiers in S. Hence $\widehat{P}[\emptyset] = 0.0$, while for sets containing only a single classifier, K_i, we have $\widehat{P}[\{K_i\}] = P_i$, i.e. the probability that K_i returns a real classification, and not IDK. Note that the $\widehat{P}[S]$ values depend only on the members of the set S, and not on their order. For efficiency we pre-compute the values of $\widehat{P}[S]$ for each of the 2^n distinct subsets S of the IDK classifiers; these are the Prob-A values shown in Tables 2 and 3.

Now consider any given cascade of classifiers

$$\left\langle K'_1, K'_2, \ldots, K'_{n'}, K_d \right\rangle \tag{1}$$

for some $n' \leq n$. The first classifier in this cascade will execute on each input sample. However, the second classifier, K'_2, will only execute in the event that the first classifier outputs IDK, the third classifier K'_3 will only execute in the event that the first two classifiers both output IDK, and so on. Letting $\bar{C}'_i$ denote the average execution time of classifier K'_i, the expected duration of

the IDK cascade is therefore equal to:

$$\sum_{i=1}^{n'} \left(\bar{C}'_i \times \left(1 - \widehat{P}[\{K'_1, K'_2, \ldots, K'_{i-1}\}] \right) \right) + \bar{C}_d \times \left(1 - \widehat{P}[\{K'_1, K'_2, \ldots, K'_{n'}\}] \right) \quad (2)$$

Hence determining the expected duration of any given IDK cascade is a straightforward operation; simply apply (2), looking up the required probability values $\widehat{P}[S]$ in the table.

Unfortunately, the number of possible IDK cascades that need to be considered grows very rapidly with the number of available IDK classifiers k. We can obtain an upper bound of the number of IDK cascades as follows. Since the number of r-permutations of k distinct objects is $\frac{k!}{(k-r)!}$, it follows that for each $r, 0 \leq r \leq k$, there are $\frac{k!}{(k-r)!}$ possible distinct IDK cascades comprising r IDK classifiers followed by the deterministic classifier. Hence the total number of IDK cascades that we need to consider is given by

$$\sum_{r=0}^{k} \frac{k!}{(k-r)!} \quad (3)$$

5.1 DAG-based Representation and Algorithm

To avoid having to evaluate every permutation and thus incur complexity that is factorial in the number of classifiers, we employ a graph-based representation in the form of a Directed Acyclic Graph (DAG). Each vertex in the graph corresponds to a unique subset of the IDK classifiers. There are $2^n - 1$ such subsets of n IDK classifiers and hence $2^n - 1$ such vertices. In addition, there is start vertex, denoted by X, that represents the empty set of classifiers, and an exit vertex, denoted by E. The vertices are connected via directed edges. A directed edge connects each vertex representing a subset of IDK classifiers with each of the vertices 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 vertex AB to each of the vertices ABC and ABD. In addition to this there is a directed edge from all other vertices to the exit vertex E.

Figure 6 illustrates the DAG representation for the case of four IDK classifiers A, B, C, and D. When a deterministic classifier is considered, then it is represented by the exit vertex E, since all possible IDK cascades end with the deterministic classifier.

Observe that each unique permutation forming an IDK cascade corresponds to a unique path through the DAG, from start to exit. On a given path the corresponding IDK cascade can be recovered by collecting the classifiers that are added in moving from one vertex to the next. For example the path

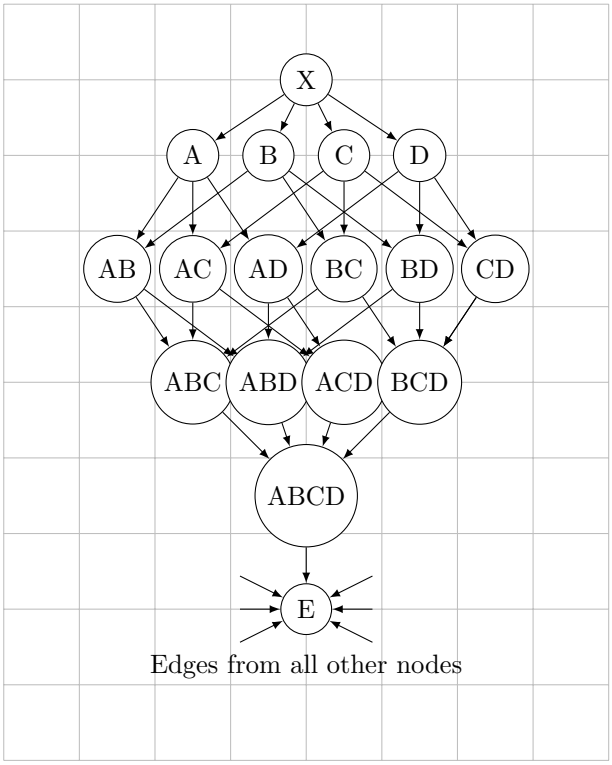

Fig. 6 DAG representation of the subsets of IDK classifiers (vertices), with arrows (directed edges) representing the addition of a further classifier and the associated increase in the expected execution duration.

$X \rightarrow A \rightarrow AC \rightarrow ACD \rightarrow ABCD \rightarrow E$ corresponds to the IDK cascade $\langle A, C, D, B, E \rangle$. (Note that when a deterministic classifier is not used, then the exit vertex and the graph remain the same, however E is not included in the corresponding IDK cascade).

Recall that the directed edge from a vertex corresponding to a subset S of IDK classifiers to another vertex corresponding to a subset S' represents the addition of a single classifier, $K'_i = S' - S$. Moreover, that edge represents the addition of classifier K'_i to any of the sub-paths (sub-sequences) of IDK classifiers that are permutations of S. Further, the increase in expected execution duration by adding classifier K'_i to any sub-sequence formed from all of the classifiers in S is given by $\bar{C}'_i \times \left(1 - \widehat{P}[S]\right)$. For example, the directed edge from ACD to $ABCD$ in Figure 6 represents the addition of IDK classifier B, and the increase in expected execution duration (cost) is given by $\bar{C}_B \times \left(1 - \widehat{P}[\{A, C, D\}]\right)$. This is the case irrespective of which one of the 6 possible paths is taken to reach vertex ACD from the start vertex X. The additional cost depends only on the set of classifiers, and not on their order.

This is the crucial point that facilitates constructing an algorithm with lower complexity. We therefore annotate the DAG with the cost associated with each edge. (Note, when a deterministic classifier is not used, then the cost associated with each of the incoming edges of the exit vertex is zero).

Now the problem of determining the IDK cascade with minimum expected execution duration is reduced to finding the shortest path from the start to the end of the DAG. 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[8], in time that is linear in the number of edges plus vertices. Since there are 2^n vertices, not counting the exit vertex, and each vertex can have at most n outgoing edges, the number of edges is upper bounded by $n2^n$. Hence, once the graph has been constructed, finding the optimal IDK cascade has $O(n2^n)$ complexity.

Considering the overall complexity starting from the results of profiling, deriving the Prob-A (i.e. $\widehat{P}[S]$) values can be achieved in $O(4^n)$ time[9], while construction of the DAG and computation of the optimal IDK cascade can then be done in $O(n2^n)$ time. Thus the overall complexity is $O(4^n)$. Recall that we do not expect practical applications to require more than approximately 12 distinct classifiers running in sequence to solve the same classification problem. By comparison, the DAG-based approach introduced in this section is viable for up to $n = 20$, requiring less than 20 minutes processing time on a single core of an Intel i5-8265U 1.6 GHz laptop computer, see Section 7.4 for further details of our proof-of-concept implementation.

The DAG-based representation can be adapted to cater for both a latency constraint on the worst-case execution duration, and a classification threshold denoting the minimum required overall probability of successful classification. This is achieved by first pruning away vertices and edges that are only present on infeasible paths corresponding to IDK cascades that do not meet those requirements, and then running the topological ordering algorithm to determine the optimal IDK cascade.

To cater for a latency constraint, for each vertex (except for the exit) we first sum up the worst-case execution duration for the corresponding set of classifiers plus the deterministic classifier if there is one. Any vertices with a worst-case execution duration that exceeds the latency constraint are then deleted from the graph along with their incoming and outgoing edges. The crucial point here is that any path which passes through such a deleted vertex cannot comply with the latency constraint, since the worst-case execution duration is a property of the set of classifiers, irrespective of their order.

To cater for a classification threshold L (without a deterministic classifier), for each vertex and corresponding subset of classifiers S we evaluate the total probability of successful classification given by $\widehat{P}[S]$. If this probability is less than the classification threshold, i.e. $\widehat{P}[S] < L$, then we delete the edge from

[8]See `https://en.wikipedia.org/wiki/Topological_sorting`

[9]Assuming that logical (bit-wise) AND operations can be performed on n-bit values in $O(1)$ time, which is certainly possible for $n \leq 64$.

that vertex to the exit vertex. Again, the crucial point is that any path that traverses such a deleted edge cannot comply with the threshold, since the total probability of successful classification depends on the set of classifiers in S irrespective of their order. Alternatively, to cater for a classification threshold with a deterministic classifier, we simply treat the deterministic classifier as if it were another IDK classifier, and do not represent it by the exit vertex. Edges are then deleted as described above, thus ensuring that any path that reaches the exit vertex complies with the classification threshold, irrespective of whether or not it includes the deterministic classifier.

We note that if following deletion of vertices and edges to ensure compliance with the latency constraint and classification threshold, no complete paths remain from the start to the exit vertex, then this means that no feasible solution to the problem exists.

6 Case Studies: Synthesizing Optimal IDK Cascades

In this section, we revisit the two case studies, previously examined in Sections 4.1 and 4.2, for which we obtained probabilistic models, summarized in Tables 2 and 3. For each case study, we use these probabilistic models to synthesize optimal IDK cascades with the minimum expected execution duration. First, in Sections 6.1 and 6.2, we consider the case where a deterministic classifier is available, and there is no latency constraint on the worst-case execution duration of the IDK cascades. Second, in Section 6.3, we introduce such a latency constraint, Finally, in Section 6.4, we relax the limitation of having a deterministic classifier and instead specify a classification threshold, i.e. a minimum required overall probability of successful classification.

6.1 The ResNet Case Study

In Section 4.1, we described how the profiling data was processed for several instantiations of the ResNet Deep Residual Network (He et al, 2015) to: (i) define IDK variants of the base classifiers; and (ii) obtain a probabilistic characterization of the effectiveness of these IDK classifiers, summarized for four IDK classifiers in Table 2. We now discuss how to use this probabilistic characterization to obtain, via the algorithm described in Section 5, an optimal IDK cascade with the minimum expected execution duration from these four IDK classifiers (labelled A, B, C, and D) and a deterministic classifier, E.

Since $k = 4$, by (3) there are 65 potential IDK cascades, each ending with the deterministic classifier. We computed the expected duration for each of these 65 IDK cascades using (2); these values are depicted in Figure 7. Note, the results are arranged left to right grouped and color-coded according to the number of classifiers in the IDK cascade, and in lexicographic order within each such group.

The expected durations ranged from a high of 1 second for the default IDK cascade $\langle E \rangle$ comprising only the deterministic classifier, to a low of 405.39ms, depicted in green and highlighted by a red arrow in Figure 7, for $\langle A, C, B, D, E \rangle$, which is the optimal IDK cascade, as determined by the DAG-based algorithm described in Section 5. That is, the expected execution duration is minimized when we call the ResNet-18, ResNet-50, ResNet-34, and ResNet-152 IDK classifiers followed by the deterministic classifier.

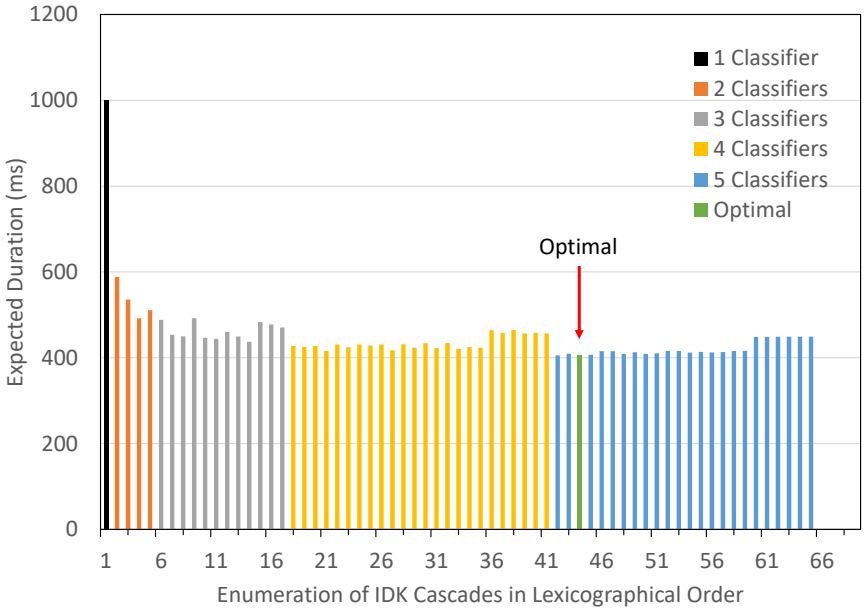

Fig. 7 ResNet: Expected duration for all 65 possible IDK Cascades.

Making an unfounded assumption that the IDK classifiers are independent and using the optimal algorithm defined by Baruah et al (2021, 2022) for such cases, would result in the selection of IDK cascade $\langle A, B, C, D, E \rangle$, which is not optimal in this case. Further, relying on the basic probability values, P_i, and computations assuming independence would underestimate the expected duration of that IDK cascade at 111ms rather than 405.44ms. Alternatively, making an unfounded assumption that the IDK classifiers are fully dependent and using the optimal algorithm defined by Baruah et al (2022) for such cases, would result in the selection of IDK cascade $\langle A, D, E \rangle$, which is also not optimal. Further, relying on the basic probability values, P_i, and computations assuming full dependence would overestimate the expected duration of that IDK cascade at 484.49ms rather than 449.93ms.

The optimal IDK cascade depends on the execution time, $\bar{C}_d$, of the deterministic classifier. Here, smaller values of $\bar{C}_d$ may reduce the need to execute the IDK classifiers, as illustrated in Table 9.

Table 9 ResNet: The optimal IDK cascade is different for deterministic classifiers with different execution times

$\bar{C}_d$	IDK Cascade	Expected Duration (ms)
2000	$\langle A, C, B, D, E\rangle$	722.99
1000	$\langle A, C, B, D, E\rangle$	405.39
500	$\langle A, C, B, E\rangle$	238.50
250	$\langle A, C, E\rangle$	141.87
100	$\langle A, E\rangle$	74.06

Note, an approach based on an assumption of independence would not select the optimal IDK cascade for any of the values of $\bar{C}_d$ in the table. Similarly, an approach based on an assumption of full dependence would not select the optimal IDK cascade for any of the values of $\bar{C}_d$ in the table, with the exception of $\bar{C}_d = 100$.

6.2 Multi-Modal Case Study

We repeated the process described in Section 6.1 on the Multi-Modal case study. Recall that this case study is notable in that the same input samples are classified by different classifiers that use information obtained from different sensors (acoustic, seismic, and camera), and hence the classifiers have substantially lower mutual dependence (see Section 4.3). The model that we constructed from the profiling data (as described in Section 4.2) is summarized in Table 3. Once again we have $k = 4$, and therefore a total of 65 possible IDK cascades. We again used (2) to compute the expected duration for each of these 65 IDK cascades, as depicted in Figure 8. Observe that in this case, there is a much larger variation in the expected duration of different IDK cascades, this is due to the larger differences in the execution times of the IDK classifiers used in this case study (see Table 3). The expected durations ranged from a high of 5000ms for the default IDK cascade $\langle E\rangle$, comprising only the deterministic classifier, to a low of 242.5ms for the IDK cascade $\langle C, B, A, D, E\rangle$, which is the optimal IDK cascade, as determined by the DAG-based algorithm described in Section 5.

Making an unfounded assumption that the IDK classifiers are independent and using the optimal algorithm defined by Baruah et al (2021, 2022) for such cases, would also result in the selection of IDK cascade $\langle C, B, A, D, E\rangle$ in this particular case. However, relying on the basic probability values, P_i, and computations assuming independence would underestimate the expected duration of that IDK cascade at 110.2ms rather than 242.5ms.

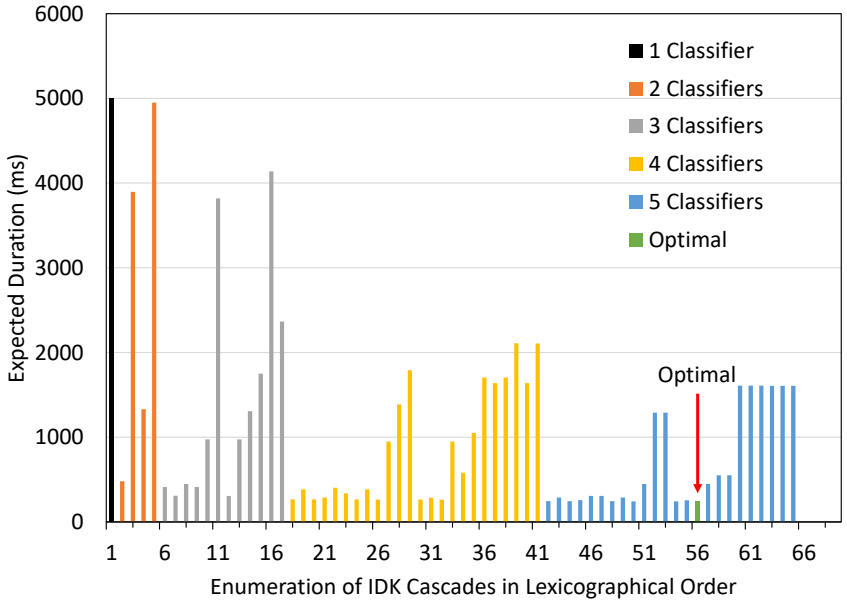

Fig. 8 Multi-modal: Expected duration for all 65 possible IDK Cascades.

As with the ResNet case study in Section 6.1, the structure of the optimal IDK cascade depends on the execution time, $\bar{C}_d$, of the deterministic classifier, as illustrated in Table 10.

Table 10 Multi-Modal:The optimal IDK cascade is different for deterministic classifiers with different execution times

$\bar{C}_d$	Cascade	Expected Duration (ms)
5000	$\langle C, B, A, D, E\rangle$	242.5
4000	$\langle C, B, A, D, E\rangle$	211.4
3000	$\langle C, B, A, E\rangle$	164.0
2000	$\langle C, B, A, E\rangle$	114.6

An approach based on an assumption of independence would not select the optimal IDK cascade for $\bar{C}_d = 4000$, but rather would select $\langle C, B, A, E\rangle$ instead. Similarly, an approach based on an assumption of full dependence would not select the optimal IDK cascade for any of the values of $\bar{C}_d$ in the table, but rather would select $\langle C, A, E\rangle$ instead.

As we have seen, approaches based on assumptions of independent or fully dependent behavior do not result in optimal or correct results for IDK classifiers with arbitrary dependences. In particular, relying on the basic probability values,

P_i, and computations assuming independence or full dependence can greatly underestimate or overestimate the expected duration of IDK cascades. In the remainder of this paper we therefore make no further comparisons with these techniques. They are not appropriate when the underpinning assumptions of independence or full dependence do not hold, and should not be used in those circumstances.

6.3 When a Latency Constraint Is Specified

As mentioned in Section 5, a specified latency constraint rules out from consideration those IDK cascades whose worst-case execution duration exceeds the constraint. Consider, for example, the ResNet case study from Section 6.1, with $\bar{C}_d = 1000$ms as specified in Table 2. In the absence of a latency constraint there were 65 feasible IDK cascades, with expected execution durations as depicted in Figure 7. When a latency constraint of 1100ms is specified, then the DAG representation described in Figure 6 in Section 5 is modified by deleting all of those vertices, and the edges connected to them, where the sum of the worst-case execution times of the corresponding IDK classifiers plus the deterministic classifier exceeds 1100ms. The result is that only vertices X, A, B, C, AB, BC, AC, and E remain and hence only 10 of the 65 possible IDK cascades remain feasible, i.e. are guaranteed to complete within the constraint. The IDK cascade $\langle A, C, B, D, E\rangle$, which was optimal in the absence of a latency constraint, is not one of them; instead, the IDK cascade $\langle B, C, E\rangle$, with an expected duration of 446.43ms becomes the optimal one. Similar observations can be made about the multi-modal case study: the added latency constraint may render some of the 65 IDK cascades, whose expected execution durations are shown in Figure 8, infeasible. How the expected duration of the optimal (feasible) IDK cascade varies with the latency constraint specified is represented graphically in Figure 9 for the ResNet case study, and Figure 10 for the Multi-Modal case study.

In these graphs of the Pareto Front, the x co-ordinate represents the specified latency constraint, while the y co-ordinate represents the expected duration of the optimal IDK cascade. In Figure 9, observe that there are steps in the graph after $x = 1100$ms. It is the presence of these steps that are responsible for the optimal IDK cascade for a latency constraint of 1100ms being different from that for no latency constraint or a larger latency constraint. Further, for a specified latency constraint of 1050ms, the optimal IDK cascade is $\langle C, E\rangle$, with an expected duration of 492ms. In fact, for the ResNet case study there are 11 Pareto optimal IDK cascades as detailed in Table 11. Similarly, for the Multi-Modal case study, there are 9 Pareto optimal IDK cascades as detailed in Table 12. Both Pareto graphs illustrate an interesting property of the proposed framework. The larger the latency that is permitted, the larger the worst-case duration can be, and hence the smaller the expected duration of the optimal IDK cascade, providing a clear trade-off between worst-case and average-case behavior.

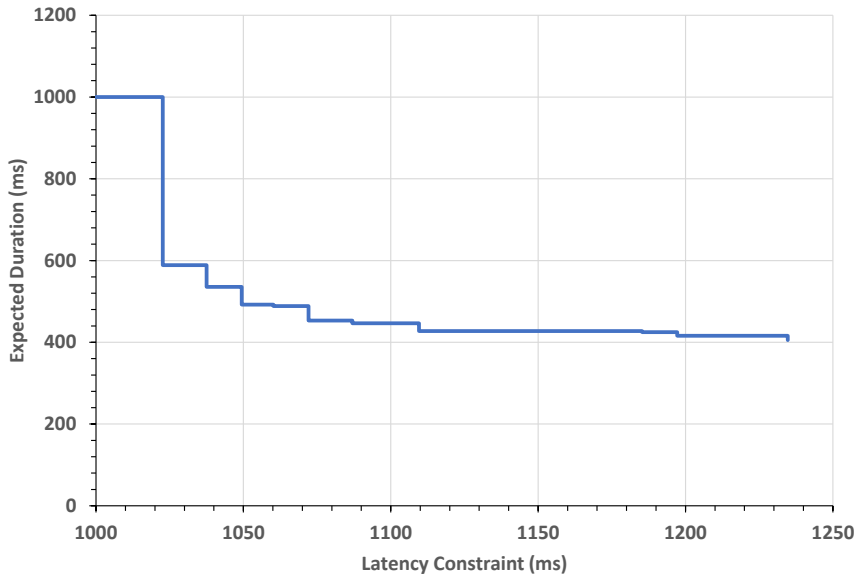

Fig. 9 ResNet: Pareto Front giving the expected duration in ms (y-axis) of the optimal IDK cascade for increasing latency constraints (x-axis).

Table 11 ResNet: Pareto Optimal IDK cascades

IDK Cascade	Worst-case (ms)	Expected Duration (ms)
$\langle E \rangle$	1000	1000
$\langle A, E \rangle$	1022.64	588.5
$\langle B, E \rangle$	1037.52	535.64
$\langle C, E \rangle$	1049.45	492
$\langle A, B, E \rangle$	1060.16	488.37
$\langle A, C, E \rangle$	1072.09	453.34
$\langle B, C, E \rangle$	1086.97	446.43
$\langle A, C, B, E \rangle$	1109.61	427.41
$\langle A, B, D, E \rangle$	1185.24	424.91
$\langle A, C, D, E \rangle$	1197.17	415.92
$\langle A, C, B, D, E \rangle$	1234.69	405.39

6.4 When a Classification Threshold is Specified

So far, we have assumed that in the event that all of the IDK classifiers in a cascade returned IDK, then a deterministic classifier would be called; however, it may not always be possible to have such an ultimate arbiter of all inputs. As an alternative, a *classification threshold* L, e.g. 0.925, can be specified, such that in the long run any IDK cascade employed must be able to successfully classify at least 92.5% of its inputs. The *classification threshold* thus acts as a

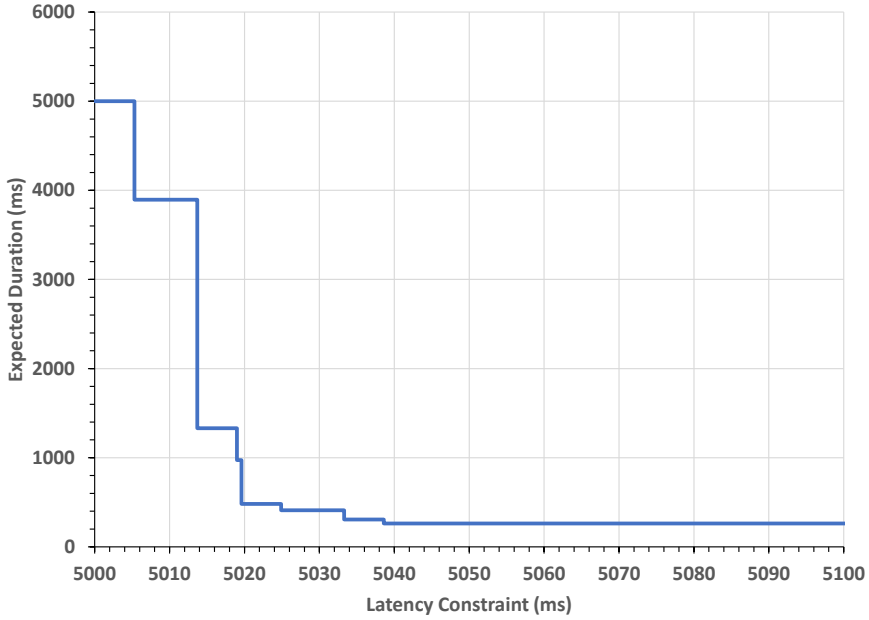

Fig. 10 Multi-modal: Pareto Front giving the expected duration in ms (y-axis) of the optimal IDK cascade for increasing latency constraint (x-axis).

Table 12 Multi-Modal: Pareto Optimal IDK cascades

IDK Cascade	Worst-case (ms)	Expected Duration (ms)
$\langle E \rangle$	5000	5000
$\langle B, E \rangle$	5005.3	3895.567
$\langle C, E \rangle$	5013.7	1330.844
$\langle C, B, E \rangle$	5019	973.540
$\langle A, E \rangle$	5019.6	480.889
$\langle B, A, E \rangle$	5024.9	411.576
$\langle C, A, E \rangle$	5033.3	307.553
$\langle C, B, A, E \rangle$	5038.6	262.919
$\langle C, B, A, D, E \rangle$	6651.8	242.492

constraint on the subsets of IDK classifiers that can form feasible IDK cascades, i.e. that meet all of the constraints.

Recall that when a classification threshold is used instead of a deterministic classifier, then the DAG representation described in Section 5 is modified as follows. Firstly, since there is no deterministic classifier, all incoming edges to the exit vertex are labelled with a cost of zero. Secondly, all edges to the exit vertex from vertices corresponding to some subset S of IDK classifiers where $\widehat{P}[S] < L$ are removed.

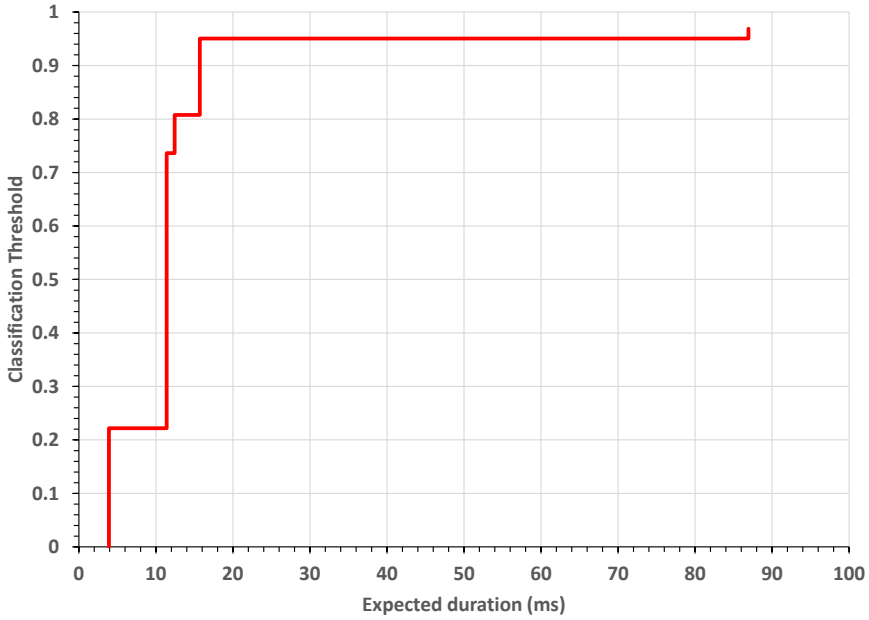

Fig. 11 Multi-modal: Pareto Front giving the expected duration (x-axis) of the optimal IDK cascade for increasing values of the classification threshold (y-axis).

With the ResNet case study, setting a classification threshold of 0.65 implies that the subset S of IDK classifiers used must be one of: $\{B, C, D\}$, $\{A, C, D\}$, $\{A, B, D\}$, or $\{A, B, C, D\}$, since these are the only ones where the $\widehat{P}[S]$ (i.e. Prob-A) values in Table 2 exceed the threshold. Hence the edges between all other vertices and the exit vertex are deleted. Subject to this threshold, the optimal IDK cascade is $\langle A, B, D\rangle$ with an expected duration of 788.5ms and a probability of successful classification of 0.65394.

With the Multi-Modal case study, setting a classification threshold of 0.925 implies that the subset S of IDK classifiers used must be one of: $\{A, D\}$, $\{A, C\}$, $\{A, C, D\}$, $\{A, B, D\}$, $\{A, B, C\}$, or $\{A, B, C, D\}$, since these are the only ones where the $\widehat{P}[S]$ (i.e. Prob-A) values in Table 3 exceed the threshold. Subject to this threshold, the optimal IDK cascade is $\langle C, B, A\rangle$ with an expected duration of just 15.697ms and a probability of successful classification of 0.951. Figure 11 shows the Pareto front illustrating how the expected duration of the optimal IDK cascade increases with an increasing classification threshold for the Multi-Modal case study, considering only IDK classifiers A, B, C, and D.

6.5 Validation

We validated the performance of the optimal IDK cascade $\langle A, B, D\rangle$ (i.e. ResNet-18, ResNet-32, and ResNet-152), along with three other plausible IDK cascades

$\{A, C, D\}$, $\{B, C, D\}$, and $\{A, B, C, D\}$ from the ResNet case study on $10,000$ images from the "TopImages" version of the ImageNetV2 data set. (Recall that $50,000$ images from the ImageNet Large Scale Visual Recognition Challenge data set (Russakovsky et al, 2015b) were used for profiling). The results are shown in Table 13. Observe that in each case, the actual average execution duration was between 2.24% and 2.82% lower that expected, and the actual frequency of successful classification was between 2.73% and 3.65% lower than the probability computed. This is a strong validation result given the disparate data sets used for profiling and validation. We were unable to validate the Multi-Modal case study on a separate data set, since no such additional compatible data set exists. The alternative of dividing up the limited number of samples available would potentially compromise the 2^n probability values computed from that data.

Table 13 ResNet: Validation

IDK Cascade	$\langle A, B, D\rangle$	$\langle A, C, D\rangle$	$\langle B, C, D\rangle$	$\langle A, B, C, D\rangle$
Expected Duration (ms)	788.50	800.36	870.14	878.45
Average Duration (ms)	766.32	782.44	850.41	853.70
Percentage Difference	2.81%	2.24%	2.27%	2.82%
Probability of Classification	0.65394	0.66412	0.66942	0.6824
Frequency of Classification	0.6266	0.6322	0.6379	0.6459
Difference	2.73%	3.19%	3.15%	3.65%

We note that in practice it is necessary to have sufficient input samples to populate the probability table, i.e. Tables 2 and 3, created during the profiling phase. Since there are 2^n probabilities characterizing the model, ideally at least 100×2^n representative input samples would be used for profiling. With $n = 4$ IDK classifiers, there are 16 distinct regions in the probability space and hence 16 rows in the probability table. Since the ResNet and Multi-Modal case studies have $50,000$ and 1800 input samples respectively, they fulfill this criteria when four IDK classifiers are considered. For larger numbers of classifiers, as in the complete Multi-Modal case study discussed below, then having a relatively small number of input samples could impact the accuracy with which the dependences between the behaviors of different classifiers can be determined and represented.

6.6 The Complete Multi-Modal Case Study

In order to illustrate the proposed approach in full detail, so far we have limited the number of IDK classifiers in the case study examples to four. However, the Multi-Modal case study has nine different IDK classifiers that could be used. We recognize that considering all nine classifiers stretches the 1800 available input samples over $2^9 = 512$ regions of the probability space, which means there are fewer input samples used in construction of the probability table than

would ideally the case. Nevertheless, considering all nine classifiers provides a useful proof-of-concept in applying the method to larger numbers of classifiers. We note that the number of input samples is sufficient to assess the correlations between the behaviors and execution times of the nine classifiers, see below.

The initial characterization of the nine Multi-Modal IDK classifiers is given in Table 14.

Table 14 Multi-Modal: Characterization of all nine IDK classifiers.

Name	Index	Execution time (ms)	Confidence threshold	Success Probability
deepsense_both	A	17.5	0.66	0.899444
deepsense_both_contras	B	17.0	0.705	0.907222
deepsense_acoustic	C	11.7	0.715	0.213333
deepsense_seismic	D	11.4	0.86	0.736111
cnn_both	E	4.0	0.649	0.595556
cnn_acoustic	F	3.9	0.5433	0.220556
cnn_seismic	G	3.7	0.752	0.327222
yolov5s	H	3145.9	0.1	0.298889
yolov5s-compressed	I	1440.8	0.1	0.298333

The confidence thresholds for each classifier were set so as to meet a required *precision threshold* of 0.95. (Recall that the precision threshold is a lower bound on the fraction of a classifier's non-IDK classification decisions that must be correct, i.e. match the ground truth). Such a precision threshold of 0.95, combined with a classification threshold of 0.95, ensures an overall *accuracy* for feasible IDK cascades of at least $0.95 \times 0.95 \approx 0.9$ or 90%. Meaning that a minimum of 90% of all input samples will be correctly classified as defined by the ground truth, with approximately 5% of those remaining unclassified (i.e. IDK returned by the final classifier in the IDK cascade) and approximately 5% incorrectly classified.

With the classifications thresholds set as shown in Table 17, we examined the correlation between the categorized behaviors (1 = non-IDK, 0 = IDK) of each distinct pair of the nine classifiers using Pearson's correlation coefficient, r. Recall that this coefficient r takes values in the range $[-1, +1]$, with $r = 0$ implying no correlation, and $r = +1$ implying perfect positive correlation. The results are shown in Table 15, color-coded by the degree of correlation between distinct classifiers: red indicating a strong degree of correlation ($abs(r) > 0.5$), orange a moderate degree of correlation ($0.1 < abs(r) \leq 0.5$), yellow a weak degree of correlation ($0.05 < abs(r) \leq 0.1$), and green a very weak degree of correlation ($abs(r) \leq 0.05$). Observe that there is a wide range of different degrees of correlation between the behaviors of the different pairs of classifiers. For example, classifiers H (yolov5s) and I (yolov5s-compressed) are very strongly correlated ($r = 0.998$), since they apply essentially the same classifier on uncompressed and compressed video data. At the other

extreme, classifiers D (deepsense_seismic) and I (yolov5s-compressed) are almost completely uncorrelated ($r = 0.008$), since they use different classifiers on different types of data. Between these extremes, there are many pairs of classifiers with weak degrees of correlation ($0.05 < abs(r) \leq 0.1$), as well as pairs of classifiers such as A paired with either B, C, D, E, or G that have moderate degrees of correlation ($0.1 < abs(r) \leq 0.5$), while classifiers C (deepsense_acoustic) and F (cnn_acoustic) have a strong degree of correlation ($r = 0.701$), since both operate on the same type of data.

Table 15 Multi-Modal Classification: Pearson Correlation Coefficient

	A	B	C	D	E	F	G	H	I
A	1	0.377	0.111	0.282	0.177	0.080	0.143	0.052	0.048
B	0.377	1	0.059	0.265	0.209	0.055	0.121	-0.043	-0.043
C	0.111	0.059	1	-0.067	0.219	0.701	-0.103	-0.028	-0.030
D	0.282	0.265	-0.067	1	0.127	-0.071	0.219	-0.005	-0.008
E	0.177	0.209	0.219	0.127	1	0.231	0.326	0.035	0.037
F	0.080	0.055	0.701	-0.071	0.231	1	-0.066	-0.036	-0.035
G	0.143	0.121	-0.103	0.219	0.326	-0.066	1	0.155	0.151
H	0.052	-0.043	-0.028	-0.005	0.035	-0.036	0.155	1	0.988
I	0.048	-0.043	-0.030	-0.008	0.037	-0.035	0.151	0.988	1

We also examined the correlation between the categorized execution times (1 = above the median, 0 = equal to or below the median) of all distinct pairs of the nine classifiers. The results are shown in Table 16. In stark contrast to the classification behavior, the classifier execution times exhibit either very weak ($abs(r) < 0.05$) or weak ($0.05 < abs(r) \leq 0.1$) degrees of correlation. This indicates that the assumption that execution times are independent continues to be a reasonable approximation when all nine classifiers are considered.

Table 16 Multi-Modal Execution Times: Pearson Correlation Coefficient

	A	B	C	D	E	F	G	H	I
A	1	0.031	0.036	-0.011	-0.027	-0.009	0.022	0.007	0.004
B	0.031	1	0.009	0.024	-0.040	-0.013	-0.024	-0.011	0.013
C	0.036	0.009	1	0.000	0.029	-0.004	0.031	-0.002	0.049
D	-0.011	0.024	0.000	1	-0.020	0.062	-0.058	0.020	-0.013
E	-0.027	-0.040	0.029	-0.020	1	-0.007	0.076	-0.018	0.007
F	-0.009	-0.013	-0.004	0.062	-0.007	1	0.024	-0.009	-0.044
G	0.022	-0.024	0.031	-0.058	0.076	0.024	1	-0.011	0.024
H	0.007	-0.011	-0.002	0.020	-0.018	-0.009	-0.011	1	-0.013
I	0.004	0.013	0.049	-0.013	0.007	-0.044	0.024	-0.013	1

Applying the DAG-based algorithm described in Section 5, considering all nine classifiers and assuming a required *classification threshold* of 0.95, yields

$\langle E, D, B\rangle$ as the optimal IDK cascade, with an expected duration of 10.8962ms, a worst-case duration of 38.1ms (on a Raspberry Pi 4), and an overall success probability of 0.956667. Computing this optimal IDK cascade from the initial profiling data took less than 10ms on a single core of an Intel i5-8265U 1.6 GHz laptop computer.

Table 17 illustrates how the optimal IDK cascade changes as the classification threshold increases. Observe that while increasing the required classification threshold increases the minimum expected duration, this does not necessarily imply that the worst-case duration of the optimal IDK cascade also increases. For example, $\langle E, D, F, G, C\rangle$, which is the optimal IDK cascade for a classification threshold of 0.9, has a larger worst-case duration than $\langle E, D, B\rangle$, which is the optimal IDK cascade for a classification threshold of 0.95.

Table 17 Multi-Modal: Optimal IDK cascades considering all 9 IDK classifiers.

Classification Threshold	IDK cascade	Expected Duration (ms)	Worst-Case Duration (ms)	Probability of Classification
0.85	$\langle E, D\rangle$	8.61067	18.5	0.865556
0.9	$\langle E, D, F, G, C\rangle$	10.8212	42.8	0.9
0.925	$\langle E, B\rangle$	10.8756	24.4	0.932778
0.95	$\langle E, D, B\rangle$	10.8962	38.1	0.956667
0.975	$\langle E, D, A, B\rangle$	11.6546	59.5	0.981111
1	$\langle E, D, B, A, G, F, C, X\rangle$	89.7576	5083.8	1

We note that the optimal IDK cascades shown in Table 17 are non-trivial to find, in particular $\langle E, D, A, B\rangle$ matches none of the plausible heuristics such as ordering IDK classifiers by their average execution time $\bar{C}_i$, their probability of success P_i, or by the ratio of these two values, $\bar{C}_i/P_i$.

7 Multiprocessor IDK Cascades

In this section, we extend the model and synthesis of optimal IDK cascades to multiple processors.

As noted by Liu (1969) real-time scheduling on multiple processors is intrinsically a much more difficult problem than single processor scheduling:

"*Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors.*"

Even with non-preemptive execution, the scheduling problem on multiple processors is significantly more complex than on a single processor. The reason

for this is that both the allocation of classifiers to processors, and the scheduling of classifiers on each processor needs to be determined. Further, with multiple processors, the order in which classifiers finish executing does not necessarily match the order in which they start executing. This impacts allocation and scheduling, while also requiring a revised analysis.

In this first investigation of optimal IDK cascades on multiple processors, we reduce the difficulty of the problem by making the following simplifying assumption. We assume that the execution time of each classifier can be represented by a single value C_i. In other words that each classifier takes a near constant time to execute, or at least holds the processor for a near constant time. In this section, we therefore assume that $C_i = \bar{C}_i$, and leave removing this assumption to future work. Further, we assume non-preemptive and therefore partitioned scheduling on a system with multiple processors that are fully isolated from one another, in other words where there is no inter-processor interference caused by shared hardware resources.

In Section 7.1, we discuss the properties of optimal IDK cascades on multiple processors, and provide some theorems and proofs concerning their key characteristics. This analysis informs an exhaustive solution that involves considering every possible sequential ordering (i.e. permutation) of the n classifiers, converted into an allocation and schedule on multiple processors via a *list scheduler*. Second, in Section 7.2, to avoid having to evaluate every permutation and thus incur complexity that is factorial in n, we again employ a graph-based representation in the form of a Directed Acyclic Graph (DAG). that can be used to determine the optimal IDK cascade for multiple processors via standard topological ordering algorithms for graph traversal. The DAG-based approach has complexity that is exponential in n, which is to be expected given that the problem is NP-complete, as shown in the Appendix. Finally, Section 7.3 extends the Multi-Modal case study to multiple processors, while Section 7.4 outlines our implementation of the DAG-based algorithm and explores its scalability.

7.1 Multiprocessor Model and Analysis

Instead of a single processor running the IDK classifiers in sequence, we assume that there are m processors that can run up to m IDK classifiers in parallel[10].

Each IDK classifier K_i is assumed to take the same execution time $\bar{C}_i$, and to have the same probability P_i of successful classification, irrespective of which processor it executes on.

In the context of multiple processors, an IDK cascade is conceptually defined by a static allocation of classifiers to processors and an order in which each processor should run the classifiers allocated to it. Classifiers are assumed

[10]Recall that we use the term *processor* with the broad meaning of an independent processing unit. Given that classifiers often make use of hardware accelerators such as GPUs, such a processing unit may include both a CPU and a GPU. In this section, we assume that m such independent processing units can run up to m classifiers in parallel.

to execute non-preemptively on their allocated processor. As we will see, a more compact definition is also possible based on the concept of *list scheduling*. With list scheduling, whenever a processor becomes available, it simply runs the next classifier in the list. Since each classifier K_i is assumed to occupy a processor for a fixed duration[11] equal to its execution time $\bar{C}_i$, in the case of list scheduling a single global list suffices to define both the allocation of classifiers to processors and the running order of classifiers on each processor.

Similar to the single processor case, the problem is to determine the optimal IDK cascade, that is the allocation and running order of classifiers that minimizes the expected duration, *meaning the elapsed time to successful classification*, optionally subject to a maximum permitted latency constraint.

For a given IDK cascade, let f'_1 denote the finish time of the classifier that finishes first, f'_i denote the finish time of the i-th classifier to finish, and f'_n denote the finish time of the n-th and last classifier to finish, assuming that all n classifiers are executed. Thus $(f'_1, f'_2, \ldots f_i, \ldots f'_{n-1}, f'_n)$ is an ordered list of classifier finish times, with $(K'_1, K'_2, \ldots K_i, \ldots K'_{n-1}, K'_n)$ denoting the corresponding classifiers that finish at those times. (Note, for completeness, if two classifiers K_j and K_k, with $j < k$, finish at the same time, then they are ordered according to their indices, i.e. $K'_i = K_j$ and $K'_{i+1} = K_k$ and $f'_i = f'_{i+1}$).

The expected duration of an IDK cascade depends only on the classifiers and their finish times, irrespective of how many processors are used and whether or not the schedules on each processor are work conserving or not. Once classifiers in the set $\{K'_1, K'_2, \ldots K'_i\}$ have finished executing at time f'_i, then the probability of successful classification is given by $\widehat{P}[\{K'_1, \ldots, K'_i\}]$ (see Definition 1), where $\widehat{P}[S]$ denotes the probability that at least one of the classifiers in the set S is successful, i.e. does not return IDK. The expected duration is therefore given by:

$$f'_1 + \sum_{i=1}^{n-1} (f'_{i+1} - f'_i)(1 - \widehat{P}[\{K'_1, \ldots, K'_i\}]) \tag{4}$$

If a deterministic classifier is employed, then as soon as it finishes and is therefore included in the set S of classifiers that have completed, then $\widehat{P}[S] = 1$ and so no further terms contribute to the expected duration. (Note that such a deterministic classifier may or may not be the last classifier to finish).

Also, if two classifiers K'_{i+1} and K'_i finish at the same time $f'_{i+1} = f'_i$ then there is no change to the expected duration when considering the second of those two classifiers K'_{i+1}, since $f'_{i+1} - f'_i = 0$; however, the probability $\widehat{P}[S]$ considered for subsequent classifiers that finish later than f'_{i+1} accounts for the fact that both K'_{i+1} and K'_i have finished.

[11] This is a necessary assumption for list scheduling to produce a consistent schedule on multiple processors. Permitting a classifier to release a processor early could otherwise lead to a different schedule, resulting in timing anomalies where early completion of one classifier results in a longer overall expected duration.

The expected duration of an IDK cascade Q can also be expressed as follows:

$$\sum_{t=1}^{F(Q)} (1 - \widehat{P}[S(t,Q)]) \tag{5}$$

where t is measured in integer time units (e.g. clock cycles), $F(Q)$ is the last finish time of any classifier, $S(t,Q)$ is the set of classifiers that finish strictly before time t, and $\widehat{P}[S(t,Q)]$ denotes the probability that at least one of the classifiers in the set $S(t,Q)$ is successful. This formulation is not intended for use in computing the expected duration, rather it is helpful in reasoning about optimal IDK cascades.

Lemma 1 *An IDK cascade, meaning an allocation of classifiers to processors and a schedule of classifiers on each processor, exists that is optimal and is locally work conserving, i.e. no processor becomes idle until all classifiers allocated to it have finished.*

Proof We assume for contradiction that there is no such optimal IDK cascade, and instead there is an optimal IDK cascade Q such that some processor or processors have a schedule that is not work-conserving, i.e. the schedule contains inserted idle time between the execution of classifiers, or at the start. We modify the schedule for each such processor so that it is work-conserving by removing all of the inserted idle time, while retaining the order in which the classifiers on each processor execute. We refer to the transformed IDK cascade as V. Since the start and finish times of every classifier in IDK cascade V are no later than in IDK cascade Q, it follows that $\forall t\ S(t,Q) \subseteq S(t,V)$ and hence $\forall t\ \widehat{P}[S(t,Q)] \leq \widehat{P}[S(t,V)]$, and further that IDK cascade V finishes no later than IDK cascade Q, i.e. $F(Q) \geq F(V)$. It follows from (5) that the expected duration of IDK cascade V is no greater than that of IDK cascade Q, hence V must also be an optimal IDK cascade

Lemma 2 *An optimal IDK cascade exists that leaves no processor idle when there is a classifier to run, i.e. the global schedule is work-conserving.*

Proof We assume for contradiction that there is no such optimal IDK cascade, and instead there is an optimal IDK cascade Q that results in at least one processor being idle when there is at least one as yet un-started classifier allocated to some other processor. We first denote IDK cascade Q by V^1 and then iteratively transform IDK cascade V^i into V^{i+1} for $i = 1 \ldots z$ until IDK cascade V^z has a global schedule that is work-conserving. On each iteration, we show that the transformation is such that the new IDK cascade V^{i+1} must also be optimal, given that V^i is optimal.

Base step: $V^1 = Q$. By definition of Q, V^1 is an optimal IDK cascade for which there exists some processor x and some time t at which processor x is idle from time t to time $t+1$ and there is at least one classifier allocated to some other processor that does not start until time $t+1$ or later.

Iterative step: From IDK cascade V^i we select the processor x which becomes idle at the earliest time t such that at time $t+1$ there is un-started classifier allocated to some other processor. From Lemma 1, the local schedule for processor x must necessarily be work-conserving and hence processor x has no more classifiers to execute in IDK cascade V^i after time t. We make a new IDK cascade V^{i+1} by copying IDK cascade V^i. We then remove classifier K_j that has the latest start time of any classifier from its currently allocated processor, which cannot be x, and append it to the schedule for processor x, so that K_j starts at time t. Comparing IDK cascades V^{i+1} and V^i, all classifiers except K_j have unchanged start and finish times; however classifier K_j starts and finishes earlier in V^{i+1} than in V^i. It follows that $\forall t\ S(t, V^i) \subseteq S(t, V^{i+1})$ and hence $\forall t\ \widehat{P}[S(t, V^i)] \leq \widehat{P}[S(t, V^{i+1})]$, and further that $F(V^i) \geq F(V^{i+1})$. Hence from (5), the expected duration of IDK cascade V^{i+1} is no greater than that of V^i, and so V^{i+1} must also be an optimal IDK cascade, given that V^i is optimal. If V^{i+1} has a globally work-conserving schedule then iteration terminates, otherwise it continues.

Termination: Iteration must terminate within a finite number of steps z since on each iteration the start time of one classifier (K_j) is reduced by at least 1 time unit, which cannot continue to happen indefinitely without the overall schedule becoming globally work-conserving

Theorem 1 *List scheduling of an appropriate ordered list of classifiers suffices to provide an optimal IDK cascade.*

Proof Lemmas 1 and 2 show that the optimal IDK cascade implies a globally work-conserving schedule. Since the duration for which each classifier K_i occupies a processor is fixed at $\bar{C}_i$, and all classifiers are non-preemptable, it follows that list scheduling applied to all distinct ordered lists of the n classifiers generates all possible distinct globally work-conserving schedules, at least one of which must therefore be optimal

Theorem 1 suggests an exhaustive approach to determining an optimal IDK cascade for a system with m processors as follows:

- Create a list corresponding to each of the $n!$ permutations of the n classifiers. These $n!$ lists represent all possible IDK cascades.
- For each list (permutation) construct the schedules for all m processors, and hence determine the ordered list of classifier finish times $(f'_1, f'_2, \ldots f'_i, \ldots f'_{n-1}, f'_n)$ and the corresponding ordered list of classifiers $(K'_1, K'_2, \ldots K'_i, \ldots K'_{n-1}, K'_n)$. From these two lists compute the expected duration of the IDK cascade. Optionally, in the case of a maximum permitted latency constraint, then the feasibility of the corresponding IDK cascade is determined by comparing the finish time of the deterministic classifier with the latency constraint.
- Record the feasible IDK cascade with the minimum expected duration. This is an optimal IDK cascade.

To cater for a *classification threshold* L (see Section 6.4) that negates the need for a deterministic classifier, the above algorithm is modified as follows:

The summation over values of i in the formula for the expected duration (4) terminates when $\widehat{P}[\{K'_1, \ldots, K'_{i+1}\}] \geq L$. In other words, the IDK cascade terminates once it achieves a success probability that meets the classification threshold L. Note, this happens after classifier K'_{i+1} completes a time f_{i+1}. Further, in the case of a maximum permitted latency constraint, the feasibility of the corresponding IDK cascade is determined by comparing the finish time f'_{i+1} of classifier K'_{i+1} with the latency constraint.

With the exhaustive approach described above, $n!$ lists (IDK cascades) are considered. Further, the calculation required to determine the schedule on the m processors and hence the total expected duration for each IDK cascade takes $O(nm)$ time. Hence, once the table of 2^n $\widehat{P}[S]$ probability values has been computed during the profiling stage in $O(4^n)$ time, then finding the optimal IDK cascade has $O(n!nm)$ complexity.

7.2 DAG-based Algorithm for Multiple Processors

To improve upon the exhaustive approach, which has factorial complexity, we developed a graph-based representation in the form of a DAG that can be used to determine the optimal IDK cascade for multiple processors via standard topological ordering algorithms for graph traversal. We first discuss the fundamental difference between classifier schedules on single and multiple processors that necessitates a more nuanced representation in the latter case. We then describe the representation used for vertices and edges along with how the DAG is constructed, how the costs associated with each edge are calculated, and finally how the DAG may be used to determine the optimal IDK cascade.

Throughout, we make use of a running example to aid understanding. This example considers two processors and five IDK classifiers A, B, C, D, and E, with execution times of 40, 60, 50, 20, and 15 respectively. Figure 12 illustrates the global work-conserving schedule for four different IDK cascades: $\langle A, B, C, D, E\rangle$, $\langle A, D, B, C, E\rangle$, $\langle A, D, E, B, C\rangle$, and $\langle A, E, D, B, C, \rangle$ on the two processors. In each schedule, the finish times of the first to fifth classifier to complete are indicated by f'_1 to f'_5.

In order to compute the expected execution duration of an IDK cascade, we need to consider the finishing time of each classifier. In the single processor case, the classifiers run sequentially and so the order in which they are specified in the IDK cascade determines not only the order in which they start, but also the order in which they finish. This means that the cost calculations can proceed directly as each classifier in the IDK cascade is considered in sequence. By contrast, in the multiple processor case, the classifiers can run in parallel, and so the order in which they are specified in the IDK cascade determines only the order in which they start; *the order in which they finish may be different.* This means that the cost calculation cannot proceed directly as each classifier in the IDK cascade is considered in sequence. Rather, it can only proceed as far as the current minimum *makespan* of the m processors, where the makespan of a processor is the total execution duration of the classifiers allocated to it

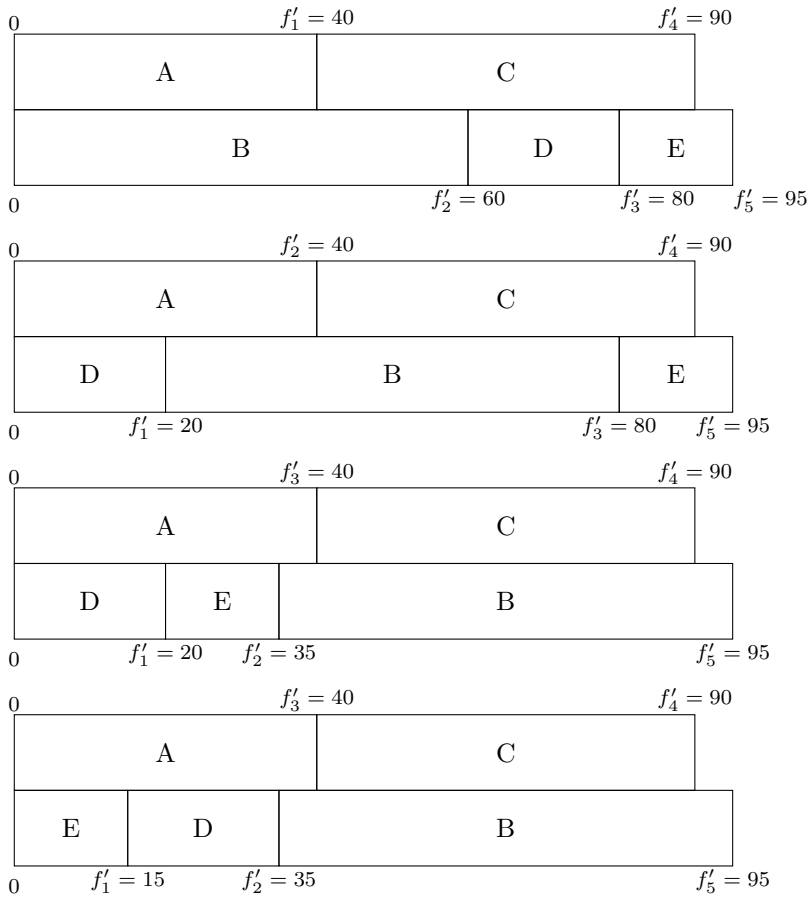

Fig. 12 Processor allocation and schedule for the following four IDK cascades: $\langle A, B, C, D, E\rangle$, $\langle A, D, B, C, E\rangle$, $\langle A, D, E, B, C\rangle$, and $\langle A, E, D, B, C, \rangle$ on two processors. In each schedule, the finish times of the first to fifth classifier to complete are indicated by $f'_1 \cdots f'_5$.

so far. Since list scheduling is employed, when x classifiers are considered on m processors, the minimum makespan corresponds to the finish time of the $(x - (m - 1))$-th classifier to finish, or zero when $x < m$. The reason for this is that there can be at most $m - 1$ unfinished, i.e. running, classifiers when a new one is added.

As an example, the first schedule at the top of Figure 12 is for the IDK cascade $\langle A, B, C, D, E\rangle$ running on two processors. Notice that although the classifiers start in the order: A, B, C, D, E, they finish in a different order: A, B, D, C, E. Once classifiers A, B, and C have been added, then the costs can only be calculated up to the minimum makespan considering those three classifiers. This minimum makespan equates to the finish time f'_2 of the second

classifier to complete, i.e. B. It is not until the fifth classifier, E, is added that we can calculate the costs up to the finish time f'_4 of classifier C, which was the third classifier added, but the fourth to finish.

Constructing the DAG: In the case of multiple processors, in order to perform the necessary cost calculations, we need to distinguish between the set of classifiers that have finished executing on a given processor, referred to as its *completed set*, and the last classifier that was added to that processor, referred to as its *running set*. Each vertex in the DAG therefore corresponds to $2m$ sets of classifiers, comprising one completed set and one running set for each of the m processors. Each of the n classifiers may appear in at most one of these $2m$ sets. Further, at most one classifier may appear in each of the m running sets, and each running set may contain at most one classifier.

Each vertex records:

- The contents of each of the m completed sets.
- The contents of each of the m running sets.
- A finishing time f'_i that equates to the minimum makespan of the m processors taking into account the classifiers in the completed sets and the running sets.
- The set S containing *all* classifiers that are in the completed sets, and the single classifier in the running set of the processor selected as having the minimum makespan.[12]
- The overall success probability $\widehat{P}[S]$. The $\widehat{P}[S]$ values are found via table lookup from the table of values determined during the profiling phase, see Section 4; these are the Prob-A values illustrated in Tables 2 and 3.

Directed edges join two vertices. We refer to the vertex where the edge is outgoing as the *previous vertex*, and the vertex where the edge is incoming as the *next vertex*. The DAG is constructed beginning with a single *start vertex* that has empty completed sets and empty running sets. Construction proceeds recursively, adding only those edges that are permitted by the rules set out below and the vertices that they lead to, or by linking to vertices that already exist.

With list scheduling, the first m classifiers in an IDK cascade are allocated to different processors, hence the DAG is bootstrapped by adding a first layer of vertices that account for *all* combinations[13] of m classifiers chosen from the n classifiers that are initially unused. Each vertex in the first layer therefore has one classifier in each of its m running sets, and is linked to by an incoming edge from the start vertex.

After the first layer of vertices have been constructed in the bootstrapping phase, the only *normal edges* that are permitted are those that add an as yet unused classifier to the processor that currently has the minimum makespan,

[12] When two processors have the same makespan, then the processor with the lowest index value is selected.

[13] It is sufficient to cover all combinations, rather than all permutations, since that avoids duplication where two vertices can be made equivalent by switching the processor numbering.

with ties broken in favor of the processor with the lowest index.[14] Adding solely to the processor with the minimum makespan ensures that only those allocations that result in a globally work-conserving schedule can be generated, and that *all* such distinct schedules can be generated from the DAG.[15] Stated otherwise, along a normal edge joining a previous vertex v to a next vertex w, an unused classifier is added to the selected processor with the minimum makespan. The next vertex w thus equates to the previous vertex v with the classifier in the running set of the selected processor first moved into the corresponding completed set, and then the new classifier added to the running set.

The number of outgoing normal edges from a vertex equates to the number of unused classifiers, i.e. the number of classifiers that are not in the completed or running sets of that vertex. Hence, vertices that contain all n classifiers have no outgoing normal edges. Rather, they may have outgoing *special edges* that represent the transfer of a classifier from the running set of a processor into its corresponding completed set.

Normal edges are used to handle cost calculations up to and including the $(n-(m-1))$-th classifier to finish, while special edges are used to handle the remaining cost calculations for the final $m-1$ classifiers to finish. We return to special edges after illustrating the basic construction of the DAG via the running example.

Recall that the running example assumes two processors and five IDK classifiers A, B, C, D, and E, with execution times of 40, 60, 50, 20, and 15 respectively. As a compact notation, we use $\emptyset$ to mean the empty set, and append the subset contents together separated by a colon and a vertical line. Thus AB:D|C:E indicates that classifiers A and B are in the completed set of processor 1 and classifier D is in its running set, while classifier C is in the completed set of processor 2, with classifier E in its running set. The start vertex is indicated by $\emptyset$:$\emptyset$|$\emptyset$:$\emptyset$, with no classifiers in any of its completed or running sets.

Figure 13 illustrates part of the DAG representation for this example. Note, the graph is incomplete and shows only those vertices and edges that are referred to in the text. Each edge is labelled with an identifier to aid discussion. A path from the start vertex to an exit vertex of the DAG corresponds to an IDK cascade, with the order in which the classifiers appear in the IDK cascade recoverable via the edges and vertices visited. The path on the left hand side of the DAG, via edges 1a, 2a, 3a, 4a, and 5a, represents the IDK cascade $\langle A, B, C, D, E\rangle$; the path to the left of centre, via edges 1b, 2b, 3b, 4b, and 5a, represents the IDK cascade $\langle A, D, B, C, E\rangle$; the path to the right of centre, via edges 1b, 2c, 3c, 4c, and 5b, represents the IDK cascade $\langle A, D, E, B, C\rangle$; and finally the path on the right hand side, via edges 1c, 2d, 3d, 4c, and 5b,

[14] This consistent tie-breaking avoids some of the duplication inherent in cases where two vertices can be made equivalent by switching the processor numbering.

[15] By distinct schedules, we mean schedules that cannot be made equivalent by switching the processor numbering.

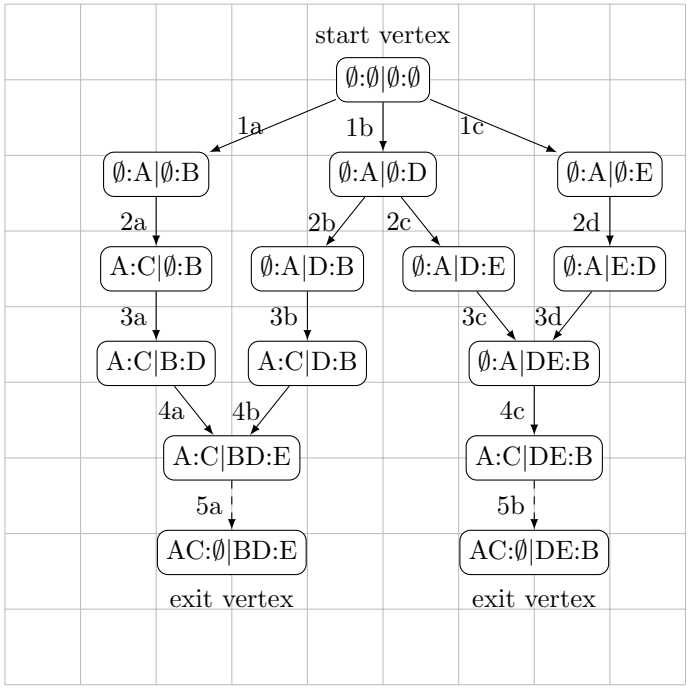

Fig. 13 DAG representation of the $2m$ subsets of IDK classifiers (vertices), with arrows (directed edges) representing the cost (contribution to the expected execution duration). Special edges are shown as dashed lines, and link to special vertices. Note, this DAG is incomplete and shows only the vertices and edges described in the text.

represents the IDK cascade $\langle A, E, D, B, C, \rangle$. The schedules for these four IDK cascades are illustrated in Figure 12.

Initially, the first layer of vertices are created containing all combinations of $m = 2$ classifiers chosen from the $n = 5$ that are available. Note, only three of these vertices are shown in Figure 13. Following the path on the left hand side of the graph, edge 1a, from the start vertex, adds classifiers A and B to the running sets. Recall that after the first layer, it is only permitted to add a classifier to the processor with the minimum makespan, hence at edge 2a, classifier C is added to processor 1, since at the previous vertex the two processors have makespans of 40 and 60 respectively. The converse is true for edge 2b, with the two processors having makespans of 40 and 20, and so in that case classifier B is added to processor 2. Observe that edges 4a and 4b have the same next vertex, even though the order in which classifiers B and D are added on the two paths that join at that vertex are different. Edge 5a is a special edge as the previous vertex already includes all n classifiers. Edge 5a links to the only special vertex on this path, which is also an exit vertex. It moves classifier C from the running set to the completed set of processor 1, since that processor has the minimum makespan of the processors that have classifiers in their running sets.

We now return to the construction of *special edges* and *special vertices*. Recall that special edges are used to handle the cost calculations for the final $m-1$ classifiers to finish. Special edges are therefore added to each vertex that includes all n classifiers and has more than one running set with a classifier in it. Along a special edge joining a previous vertex v to a next vertex w, of the processors with running sets that contain classifiers at vertex v, the one with the minimum makespan is selected, and the classifier in the running set of that processor is transferred to the corresponding completed set. Stated otherwise, the next vertex w equates to the previous vertex v with the classifier in the running set of the selected processor moved into the corresponding completed set, and that running set thus empty. It follows that traversing a chain of special edges moves $m-1$ classifiers, one by one, from the running sets into the corresponding completed sets, finally reaching an *exit vertex* that includes all n classifiers and has only one running set with a classifier in it. Exit vertices have no outgoing edges.

A vertex is referred to as *special* if its incoming edge(s) are special. Each special vertex records:

- The contents of each of the m completed sets.
- The contents of each of the m running sets.
- A finishing time f'_i that equates to the minimum makespan of the processors *with a classifier in their running set*, taking into account the classifiers in both the completed sets and the running sets of those processors.
- The set S containing *all* classifiers that are in the completed sets, and the single classifier in the running set of the processor selected as having the minimum makespan of those processors with a classifier in their running set.
- The overall success probability $\widehat{P}[S]$.

Cost calculations on edges: Each edge represents an increase in the expected execution duration (cost) for all paths (IDK cascades) that include it. Edges represent the cost increase as the time considered moves on from the minimum makespan computed at the previous vertex, equating to the finish time f'_i of some classifier or $f'_0 = 0$ in the case of the start vertex, to the minimum makespan computed at the next vertex, equating to the finish time f'_{i+1} of the next classifier to complete.

The cost of an edge is given by:

$$(f'_{i+1} - f'_i) \times \left(1 - \widehat{P}[S]\right) \tag{6}$$

where $\widehat{P}[S]$ is the overall success probability recorded for the previous vertex, with S defined as the set of all classifiers that are in any of the completed sets and the classifier in the running set of the single processor selected as having the minimum makespan at that vertex. Due to the way in which the graph is constructed by only ever adding classifiers to the processor with the minimum makespan, it follows that all of the classifiers in S are guaranteed to be finished by f'_i.

Returning to the example in Figure 13, traversing edge 1a, the minimum makespan increases from 0 to 40 (i.e. $f_1' = 40$) and the set S at the previous (i.e. start) vertex contains no classifiers, and so the cost of this edge is 40. Traversing edge 2a, the minimum makespan increases from 40 to 60 (i.e. $f_2' = 60$) and classifier A is added to the completed set for processor 1. The set S at the previous vertex contains only classifier A, and hence the cost of this edge is given by $20 \times \left(1 - \widehat{P}[\{A\}]\right)$. Traversing edge 3a, the minimum makespan increases from 60 to 80 (i.e. $f_3' = 80$) and classifier B is added to the completed set for processor 2. The set S at the previous vertex now contains classifiers A and B, hence the cost of this edge is given by $20 \times \left(1 - \widehat{P}[\{A, B\}]\right)$. Further, traversing edge 4a, the minimum makespan increases from 80 to 90 (i.e. $f_4' = 90$) and classifier D is added to the completed set for processor 2. The set S at the previous vertex now contains classifiers A, B, and D, hence the cost of this edge is given by $10 \times \left(1 - \widehat{P}[\{A, B, D\}]\right)$.

Observe that on any path reaching the layer of vertices that have all n classifiers allocated (i.e. the next vertices to edges 4a, 4b, and 4c), costs have been calculated as far as the minimum makespan of the m processors when all n classifiers are considered. However, this means that there are still some additional costs to be incurred as the classifiers in the running sets of each of the remaining $m - 1$ processors, that were not yet selected as having the minimum makespan, finish. This is taken care of via the cost calculation for the special edges that link to the special vertices.

Continuing with the example, traversing special edge 5a, processor 1 is selected as having the minimum makespan of 90, compared to 95 for processor 2, and hence for the next vertex, classifier C is removed from the running set of processor 1 and placed in the corresponding completed set. Since the next vertex of edge 5a is special, the minimum makespan increases from 90 to 95 (i.e. $f_5' = 95$). (Recall that for special vertices, the minimum makespan excludes those processors with empty running sets). The set S at the previous vertex now contains classifiers A, B, C, and D, hence the cost of this edge is given by $5 \times \left(1 - \widehat{P}[\{A, B, C, D\}]\right)$. Finally, the exit vertex, in the final layer, records the overall success probability $\widehat{P}[S]$ where S now contains all of the classifiers.

Note that since this example is for two processors, the first special vertices on any path are also exit vertices. With more processors additional special edges and special vertices would move further classifiers into the completed sets, and hence complete the calculations.

Observe that on the path on the left hand side of the graph in Figure 13, even though classifier C is added to the corresponding IDK cascade at edge 2a, we cannot complete all of the cost calculations up to the finish time of classifier C until later in the graph at edge 4a. This is a consequence of the fact that the start and finish times of classifiers are interleaved due to parallel execution, and is why we need to separately keep track of the classifiers in the completed sets and the running sets.

Finally, the two paths on the right hand side of the graph in Figure 13 and the corresponding two schedules at the bottom of Figure 12 illustrate how the DAG-based representation gains over examining all possible permutations. When two paths join because the sets of completed and running classifiers have become the same even though the classifiers did not execute in the same order, for example via edges 3c and 3d, then the subsequent evolution of these paths and the calculations required are identical and do not need to be duplicated.

Note that different to the DAG-based algorithm presented for a single processor in Section 5.1, in the case of multiple processors we make no distinction regarding the deterministic classifier if any. It is treated exactly the same as any other classifier.

Finding the optimal IDK cascade: The problem of finding the optimal IDK cascade is characterized by a latency constraint, which is assumed to be infinite if there is no such constraint, and a classification threshold, which is assumed to be 1 if successful classification has to be guaranteed. For there to be any feasible solutions, then the overall success probability considering the complete set of n classifiers must not be less than the classification threshold. Assuming that a feasible solution exits, then an optimal IDK cascade can be found as follows.

Firstly, vertices and the edges to and from them are omitted from the DAG if their recorded finish time exceeds the latency constraint. Since the finish time of successor vertices is monotonically non-decreasing on any path through the DAG, this is done as the graph is constructed, i.e. all successor vertices of a vertex that breaks a latency constraint must also break that constraint and are therefore not required. Further, the remaining vertices are marked as *qualifying* if their recorded overall success probability meets or exceeds the classification threshold. The problem of determining the optimal IDK cascade, subject to a latency constraint and classification threshold, then amounts to finding the minimum cost path through the DAG from the single start vertex to any one of the qualifying vertices. This is a well-known problem in graph traversal. Since there is a single start vertex, the problem can be solved using a standard topological ordering algorithm, in time that is linear in the number of vertices plus edges. The complexity of the DAG-based algorithm hence depends on the number of vertices and edges and the operations involved in constructing them. The complexity of the problem and the DAG-based algorithm are discussed further in the Appendix. We note that if after catering for the latency constraint and classification threshold, no qualifying vertices remain, then this means that there is no feasible solution to the problem.

7.3 Complete Multi-Modal Case Study on Multiple Processors

In order to illustrate the proposed approach for multiple processors, we consider the complete Multi-Modal case study with all nine IDK classifiers as characterized in Table 14. For comparison purposes, Table 18 first sets out the equivalent results for the case of a single processor.

Table 18 Multi-Modal: Optimal IDK cascades for a single processor considering all 9 IDK classifiers and a deterministic classifier.

Classification Threshold	IDK cascade	Expected Duration (ms)	Worst-Case Duration (ms)	Probability of Classification
0.85	$\langle E, D\rangle$	8.61067	15.4	0.865556
0.9	$\langle E, D, F, G, C\rangle$	10.8212	34.7	0.9
0.925	$\langle E, B\rangle$	10.8756	21	0.932778
0.95	$\langle E, D, B\rangle$	10.8962	32.4	0.956667
0.975	$\langle E, D, B, A\rangle$	11.6546	49.9	0.981111
1	$\langle E, D, B, A, G, F, C, X\rangle$	89.7576	5069.2	1

Assuming a required *classification threshold* of 0.95, and considering all nine classifiers A to I, yields $\langle E, D, F, G, B\rangle$ as the optimal IDK cascade for a two processor system, with an expected duration of 8.16333ms, a worst-case duration of 21ms, and an overall success probability of 0.963889. Table 19 illustrates how, for a two processor system, the optimal IDK cascade changes as the classification threshold increases. The final row in the table sets the classification threshold to 1.0 and also includes a hypothetical deterministic classifier X with an assigned execution time of 5000ms. In this case the optimal IDK cascade is $\langle E, G, D, F, B, A, C, I, H, X\rangle$ with classifiers $\langle E, G, B, X\rangle$ running on the first processor and classifiers $\langle D, F, A, C, I, H\rangle$ running on the second processor. The expected duration is 70.4877ms and the worst-case duration is 5024.7ms. It is interesting to note that the previous rows in the table remain unaltered by the addition of such a deterministic classifier, since it is more efficient to rely upon a selection of IDK classifiers to reach the required classification thresholds, rather than use the deterministic classifier.

Tables 20 and 21 for three and four processors respectively, similarly illustrate how the optimal IDK cascade changes as the classification threshold increases. Tables 19, 20, and 21 for two, three, and four processors respectively, are directly comparable to Table 18 for a single processor.[16]

Assuming a classification threshold of 0.95, then by using two processors the expected duration is reduced to 74.9% of that required using a single processor (8.16333ms vs. 10.8962ms) and the worst-case duration is reduced to 64.8% (21ms vs. 32.4ms). Using three processors the expected duration is reduced to 67.3% of that required using a single processor (7.33195ms vs. 10.8962ms) and the worst-case duration is reduced to 63.9% (20.7ms vs. 32.4ms). Finally, Using four processors the expected duration is reduced to 63.5% of that required using a single processor (6.92311ms vs. 10.8962ms) and the worst-case duration is reduced to 52.5% (17ms vs. 32.4ms).

The above results and comparisons assume that the execution times for the classifiers are unchanged when the classifiers are run in parallel on multiple processors rather than serially on a single processor, i.e. assuming no interference

[16]Note that in computing Table 18, we made the same simplifying assumption about classifier execution times that was used in the analysis of multiple processors. For that reason, the worst-case durations differ from those given in Table 17 in Section 6.6.

Table 19 Multi-Modal: Optimal IDK cascades for a dual processor considering all 9 IDK classifiers and a deterministic classifier.

Classification Threshold	IDK Cascade	Processor 1	Processor 2	Expected Duration (ms)	Worst-Case Duration (ms)	Probability of Classification
0.85	$\langle E, G, D\rangle$	$\langle E, G\rangle$	$\langle D\rangle$	6.77911	11.4	0.876667
0.9	$\langle E, G, D, F, C\rangle$	$\langle E, G, C\rangle$	$\langle D, F\rangle$	7.69972	19.4	0.9
0.925	$\langle E, D, F, G, B\rangle$	$\langle E, B\rangle$	$\langle D, F, G\rangle$	8.16333	21	0.963889
0.95	$\langle E, D, F, G, B\rangle$	$\langle E, B\rangle$	$\langle D, F, G\rangle$	8.16333	21	0.963889
0.975	$\langle E, G, D, F, B, A\rangle$	$\langle E, G, B\rangle$	$\langle D, F, A\rangle$	8.5605	32.8	0.983889
1	$\langle E, G, D, F, B, A, C, I, H, X\rangle$	$\langle E, G, B, X\rangle$	$\langle D, F, A, C, I, H\rangle$	70.4877	5024.7	1

Table 20 Multi-Modal: Optimal IDK cascades for a tri processor considering all 9 IDK classifiers and a deterministic classifier.

Classification Threshold	IDK Cascade	Processor 1	Processor 2	Processor 3	Expected Duration (ms)	Worst-Case Duration (ms)	Probability of Classification
0.85	$\langle G, E, F, D\rangle$	$\langle G, F\rangle$	$\langle E\rangle$	$\langle D\rangle$	6.33206	11.4	0.892778
0.9	$\langle G, E, F, D, C\rangle$	$\langle G, C\rangle$	$\langle E, F\rangle$	$\langle D\rangle$	6.77161	15.4	0.9
0.925	$\langle G, E, F, D, C, B\rangle$	$\langle G, B\rangle$	$\langle E, F, C\rangle$	$\langle D\rangle$	7.33195	20.7	0.965556
0.95	$\langle G, E, F, D, C, B\rangle$	$\langle G, B\rangle$	$\langle E, F, C\rangle$	$\langle D\rangle$	7.33195	20.7	0.965556
0.975	$\langle G, E, F, D, B, C, A\rangle$	$\langle G, B\rangle$	$\langle E, F, A\rangle$	$\langle D, C\rangle$	7.50578	25.4	0.984444
1	$\langle G, E, F, D, B, A, C, I, H, X\rangle$	$\langle G, B, I\rangle$	$\langle E, F, A, C, H\rangle$	$\langle D, X\rangle$	69.2984	5011.4	1

Table 21 Multi-Modal: Optimal IDK cascades for a quad processor considering all 9 IDK classifiers and a deterministic classifier.

Classification Threshold	IDK Cascade	Proc. 1	Proc. 2	Proc. 3	Proc. 4	Expected Duration (ms)	Worst-Case Duration (ms)	Probability of Classification
0.85	$\langle G, F, E, D\rangle$	$\langle G\rangle$	$\langle F\rangle$	$\langle E\rangle$	$\langle D\rangle$	6.18794	11.4	0.892778
0.9	$\langle G, E, F, D, C\rangle$	$\langle G, F\rangle$	$\langle E\rangle$	$\langle D\rangle$	$\langle C\rangle$	6.36422	11.7	0.9
0.925	$\langle G, E, F, D, C, B\rangle$	$\langle G, F\rangle$	$\langle E, C\rangle$	$\langle D\rangle$	$\langle B\rangle$	6.92311	17	0.965556
0.95	$\langle G, E, F, D, C, B\rangle$	$\langle G, F\rangle$	$\langle E, C\rangle$	$\langle D\rangle$	$\langle B\rangle$	6.92311	17	0.965556
0.975	$\langle G, E, F, D, B, C, A\rangle$	$\langle G, F, C\rangle$	$\langle E, A\rangle$	$\langle D\rangle$	$\langle B\rangle$	7.09133	21.5	0.984444
1	$\langle G, E, F, D, B, A, C, I, H, X\rangle$	$\langle G, F, X\rangle$	$\langle E, A, H\rangle$	$\langle D, I\rangle$	$\langle B, C\rangle$	68.8584	5007.6	1

effects between the processors. A consideration of any such effects is beyond the scope of this paper.

7.4 Proof of Concept Implementation

We implemented the DAG-based algorithm, described in Section 7.2, in C++. The implementation built upon the algorithmic description in the following ways:

- Data structures were used to represent the DAG and each vertex.
- The DAG data structure recorded the input parameters, i.e. the number of processors, the number of classifiers and their execution times, and provided access to the pre-computed table of $\widehat{P}[S]$ probability values. Further, it provided access to the layers of vertices as they were created, and also recorded, as construction progressed, the lowest cost qualifying vertex that complied with the latency constraint and the classification threshold.
- The vertex data structure contained all of the information detailed in the algorithmic description, as well as fields to record the cost for the vertex (i.e. the total cost up to and including the vertex along the lowest cost path to it), and a pointer to the vertex in the previous layer on that lowest cost path.
- The cost for each vertex was computed on-the-fly as the DAG was constructed, i.e. as vertices were added, layer by layer. This had the advantage that no lasting representation of edges was required. Instead, each vertex required only a single pointer back to the vertex in the previous layer that was on the lowest cost path to it. Once the DAG was complete, this enabled the optimal IDK cascade to be recovered from the path back to the start vertex from the minimum cost vertex that complied with the latency constraint and the classification threshold.
- A large hash table was used to eliminate *equivalent* vertices that could otherwise occur in each layer. A 32-bit CRC was obtained from a binary representation of the completed sets and the running sets of each vertex.[17] The bottom 26 bits of the CRC was then used as a hash key into a hash table of size 2^{26}. On removing an equivalent vertex, the remaining vertex was given the minimum cost of the pair and its pointer back to the vertex in the previous layer was updated to correspond to the lowest cost path.
- Effective pruning of vertices was achieved by avoiding construction of unnecessary vertices in the first place. Any vertex exceeding the latency constraint was marked as a stopping point and was not extended to further vertices in the subsequent layer. Similarly, qualifying vertices that already complied with the latency constraint and the classification threshold were not extended, since they represent better solutions than any of their successors.

[17] Recall that vertices are effectively equivalent if their running sets and completed sets are the same.

A simplified algorithm was also implemented for the single processor case, based on the DAG-based algorithm described in Section 5.1. In this case the hash key used was simply the bottom 20-bits of the binary representation of the allocated classifiers, with a hash table of size 2^{20}.

Our DAG-based implementations were run on one core of a mid-range laptop PC (a Lenovo ThinkPad with an Intel Core i5-8265U CPU clocked at 1.60 GHz to 1.80 GHz, with 16 GBytes of RAM, running Microsoft Windows 10). A separate run was made to create each row in Tables 18, 19, 20, and 21. The longest run-time in each case was for the final row in the table, with a classification threshold of 1.0. The run-times measured using the C++ `clock()` function were approximately: 7ms, 521ms, 568ms, and 555ms, for one, two, three, and four processors respectively. (Note, that in each case the run-time was dominated by the initialization of the hash table for each layer of vertices). Similarly, the numbers of non-equivalent vertices created (not counting the start vertex) were: 1023, 83870, 221874, and 197975 respectively. The above run-times cover only the code used to determine the optimal IDK cascade using the DAG-based algorithms. The additional pre-processing needed to set up the table of $2^{10} = 1024$ $\widehat{P}[S]$ values took less than 1ms, and was the same in each case.

The Multi-modal case study with 9 IDK classifiers and a deterministic classifier presents only a limited computational challenge. To investigate the scalability of our DAG-based implementations, we devised a test based on n classifiers that all had the same execution duration, with disjoint probabilities of success each equating to $1/n$. Further, no latency constraint was specified. We selected these settings since a set of classifiers that all have the same execution duration maximizes the number of different work-conserving schedules, and hence the number of non-equivalent vertices created. Further, the disjoint probabilities of success meant that all n classifiers were required to meet a classification threshold of 1.0, so all vertices, with the exception of the exit vertices in the final layer, needed to be extended to vertices in the next layer.

Figure 14 shows the number of non-equivalent vertices that were created by our DAG-based implementations in solving the optimal IDK cascade problem for $m = 1$ to 6 processors and $n = 6$ to 20 classifiers. Where the lines stop before 20 classifiers, this indicates that more than 24 GBytes of memory would be required to store the vertices required for the next point. For $m = 1$, the simpler algorithm was employed and the number of vertices (not counting the start vertex) is given by $2^n - 1$. For $m = 2$ to 6, the more complex algorithm was employed. Observe that in this case the number of vertices grows faster (i.e. the lines have a steeper slope) for larger numbers of processors, but those lines start at a lower value. For example, with $m = 6$ processors and $n = 6$ classifiers the problem is trivial, there is only one vertex in the first layer and one special vertex in each of the five subsequent layers for 6 vertices in all, whereas with $m = 2$ processors and $n = 6$ classifiers there are far more distinct possibilities, leading to 699 vertices in all, not counting the start vertex. As the number of classifiers increases to 12 or more, so the number of distinct possibilities on

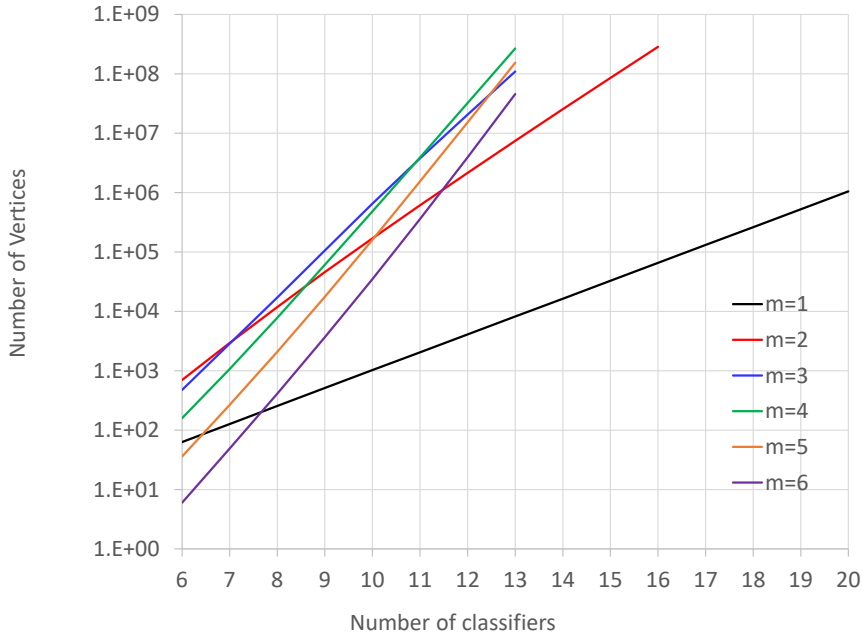

Fig. 14 Number of non-duplicated vertices created for m processors and n classifiers

$m = 6$ processors becomes greater than that on $m = 2$ processors and so the lines cross.

Figure 15 illustrates the corresponding run-times of our DAG-based implementations in solving the optimal IDK cascade problem for $m = 1$ to 6 processors and $n = 6$ to 20 classifiers, when run on one core of a laptop PC. For $n = 6$ to 10 classifiers, the run-times were dominated by the time taken to initialize the hash table for each layer of vertices. Recall that for the single processor case a much smaller hash table is used and this accounts for most of the difference between the single and multiple processor run-times for smaller values of n. Observe also that in each case, the run-time for 7 classifiers is lower than that for 6 classifiers; this is an artefact of cache warm-up. For more than $n = 10$ classifiers, the run-times reflect the number of non-equivalent vertices, shown in Figure 14. Also shown in Figure 15 is the $O(4^n)$ run-time required to set up the table of 2^n $\widehat{P}[S]$ i.e. Prob-A values from the table of 2^n Prob-S values. The run-time of this preliminary processing is independent of the number of processors. It is the dominant factor in the overall run-time when considering problems on a single processor ($m = 1$), but effectively negligible when considering problems on two or more processors ($m > 1$).

Within the limits of approximately 1200 seconds (20 minutes) run-time and 24 GBytes of (potentially paged) memory usage, the maximum number of classifiers that could be catered for by our implementation of the DAG-based algorithm for multiple processors was 16 classifiers for $m = 2$ and 13 classifiers

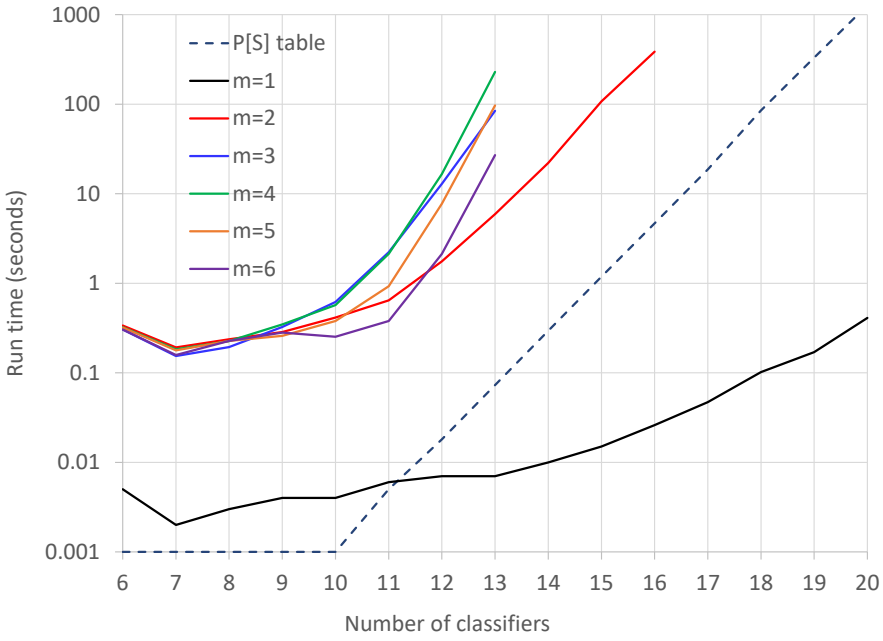

Fig. 15 Run-time of the DAG-based algorithm for m processors and n classifiers

for $m > 2$. Assuming a single processor, the maximum number of classifiers that could be catered for was at least 20, with the construction of the table of $\widehat{P}[S]$ probability values dominating the overall run-time in that case.

8 Conclusions and Future Research

The increasing use of machine perception in many forms of Cyber-Physical Systems (CPS) is leading to the application of a wide range of Deep Learning components whose role is to classify input data and thereby ensure the safe and effective behavior of the system. To achieve the levels of fidelity and reliability required, it is necessary to employ a collection of diverse classifiers. One method of managing this collection is to convert each classifier into an IDK classifier and to organize their execution into an IDK cascade that can perform the necessary classification.

Previous work showed how such IDK cascades can be analyzed and optimized, but made the unrealistic assumption that the behaviors of the classifiers, in terms of their probabilities of successful classification, are either completely independent or fully dependent. In this paper we removed this assumption and showed how representative profiling data can be used to characterize the level of mutual dependence exhibited by the classifiers, via a probabilistic representation that caters for arbitrary dependences. This

probabilistic representation was then used in the synthesis of optimal IDK cascades that have the minimum expected duration, with or without a latency constraint on the overall worst-case execution duration.

Previous work also relied on the concept of a deterministic classifier that is guaranteed to always make a successful classification. In this paper, we recognized that such a construct may not always be viable in practice, and therefore also provided solutions based on a classification threshold, equating to the minimum overall probability (long run frequency) of successful classification that is deemed acceptable.

Further, we developed solutions for both single processors and multiple processors that use DAG-based representations and topological ordering algorithms. The effectiveness of our proposed solution was demonstrated via two real-world case studies, using a variety of classifiers with inputs including image, seismic, and acoustic data.

Finally, our analysis of the behaviors of the classifiers from these case studies indicated a whole range of strong, moderate, and weak correlations between different pairs of classifiers. Thus demonstrating that assumptions of independence or full dependence do not in general hold. Rather the approach taken in this paper, catering for arbitrary dependences, is necessary in order to solve the optimal IDK cascade problem in practice.

There are a number of interesting directions for future work in this area:

1. Removing the simplifying assumption that each classifier holds a processor for a constant time, used in the analysis of optimal IDK cascades for the multiple processor case.
2. Considering the impact of the environment on the probability of success of each classifier, and deriving optimal IDK cascades that are sensitive to the current mode of the environment (e.g. daylight or darkness).
3. Allowing the permitted latency on the overall execution duration to be set dynamically, and hence facilitate switching between a collection of statically "optimal" IDK cascades at run-time.
4. Allowing the actual execution time of each IDK classifier to influence subsequent (dynamic) choices of which IDK classifier to run next, when there is a latency constraint.
5. Allowing the confidence threshold to have a mixed-criticality perspective (i.e. lower-criticality requirements having a lower threshold than higher-criticality ones).
6. Verifying that representative input data sets, used to provide the profiling data, properly capture the arbitrary dependences between the classifiers.
7. Considering the optimal order of execution of classifiers when some subsets of classifiers may be executed in parallel on the same GPU.
8. Considering the scheduling of classifiers when the input data is recurrent, i.e. forming a time-series, and the classifiers are executed periodically. In this case classification performance may be improved via knowledge of the input samples and confidence in the identification of the same object in prior time frames.

Appendix

In this appendix we discuss the complexity of the DAG-based algorithm to the optimal IDK cascade problem on multiple processors.

Complexity of the algorithm

In the DAG-based algorithm, described in Section 7.2, considering the m completed sets of a vertex, each classifier has $m+1$ possible states, it is either in none of the m completed sets or it is in exactly one of them. Hence an upper bound on the number of different variations for the m completed sets is $(m+1)^n$. Considering the m running sets of a vertex, each running set has $n+1$ possible states, it is either empty or contains exactly one of the n classifiers. Hence an upper bound on the number of different variations for the m running sets is $(n+1)^m$. It follows that the total number of different variations encompassing both completed sets and running sets is $(m+1)^n(n+1)^m$, which provides a simple upper bound on the number of vertices in the DAG.

The total number of edges per vertex is bounded by n as follows. Vertices in the first layer have $n-m$ outgoing edges, one for each unused classifier, as well as one incoming edge from the start vertex. Subsequent layers of normal vertices have at most $n-(m-1)-x$ outgoing edges, where x is the number of the layer from 2 to $n-m$. Finally, vertices with a full allocation of all n classifiers have at most one outgoing special edge, while exit vertices have no outgoing edges. The total number of edges is therefore upper bounded by $n(m+1)^n(n+1)^m$.

The amount of computation required for each vertex is $O(m)$. This comprises determining the minimum makespan of the m processors, and looking up the total success probability for the classifiers in the set S.[18] The amount of computation required for each edge is $O(1)$ based on the information available at the previous and next vertex.

It follows that, once the table of 2^n $\widehat{P}[S]$ probability values has been computed during the profiling stage in $O(4^n)$ time, then finding the optimal IDK cascade by constructing the DAG and then applying a standard topological ordering algorithm has at most $O((n+m)(m+1)^n(n+1)^m)$ complexity.

This exponential upper bound is far from tight, since some variations included in the vertex count are not permitted or cannot be generated as part of the DAG construction. These include variations where the same classifier appears in two running sets which is not permitted, and allocations that do not correspond to global work-conserving schedules and so cannot be generated, for example where one are more processors are unused. The bound does however suffice to show that for a fixed number of processors m, the complexity of

[18] Set membership can be encoded as a bit map, with set union operations taking linear time at least up to $n = 64$. From a bit-map representation of S, lookup of the corresponding $\widehat{P}[S]$ probability value also takes linear time.

finding the optimal IDK cascade is bounded by an exponential in n rather than a factorial in n.

A more accurate complexity bound can be obtained by explicitly counting the maximum number of vertices and edges in each layer. The number of vertices can be counted as follows:

- There is one start vertex.
- There are $n-(m-1)$ layers of normal vertices. Let i, from $i=1$ to $i=n-(m-1)$, be the layer index for these vertices. Vertices in these layers have a classifier in each of the m running sets and $i-1$ classifiers in the completed sets. Hence there are $\binom{n}{m} = n!/(n-m)!m!$ ways of choosing the running sets, $\binom{n-m}{i-1} = (n-m)!/((n-(m-1)-i)!(i-1)!)$ ways of choosing the $i-1$ classifiers that are in any of the completed sets, and $m^{(i-1)}$ ways of assigning those $i-1$ classifiers to the m completed sets. Hence the maximum number of normal vertices in layer i is given by: $m^{(i-1)}n!/((n-(m-1)-i)!(i-1)!)$.
- There are $m-1$ layers of special vertices. Each special vertex is linked to by a single special edge from a single vertex in the layer above. The maximum number of special vertices in any layer is therefore given by the number of normal vertices in layer $i=n-(m-1)$, which equates to $m^{(n-m)}n!/(n-m)!$. (Note the final layer of special vertices are exit vertices).

Further, the number of edges can be counted as follows:

- The start vertex has $\binom{n}{m} = n!/(n-m)!m!$ outgoing edges.
- Each normal vertex in layer i, from $i=1$ to $i=n-(m-1)$, has $n-(m-1)-i$ outgoing normal edges. (Note, each vertex in the last of these layers has no outgoing normal edges, but rather has a single outgoing special edge that is counted below).
- Each special vertex in layer i, from $i=n-(m-1)+1$ to $i=n$, has one incoming special edge.

The complexity measure is then given by the total number of edges plus m times the total number of vertices, since $O(m)$ operations are required at each vertex to determine the minimum makespan. The count obtained using the explicit method set out above is, however, an overestimate as it includes assignments of classifiers to the completed sets that do not represent globally work-conserving schedules.

Figure 16 shows the complexity measure for the optimal IDK cascade problem as given by the factorial, exponential, and counting bounds for $m = 2, 3, 4$ processors and $n = 3 \ldots 24$ classifiers. Also shown is the exponential bound, $O(n2^n)$, for the single processor case, $m=1$.

For multiple processors, finding an optimal solution has exponential complexity, since the problem of deciding if there is any IDK cascade that meets both a maximum latency constraint and a classification threshold is NP-complete, as shown below.

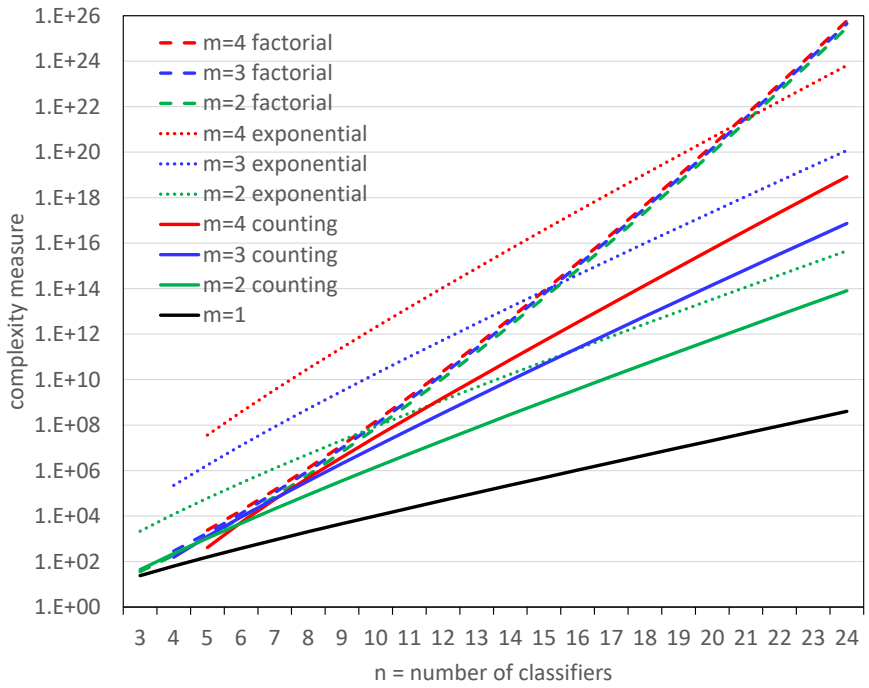

Fig. 16 Complexity measure for m processors and n classifiers

An NP-complete problem

We now show that the IDK cascade decision problem for $m > 1$ processors, with a fixed value of m, is NP-complete via reduction (Karp, 1972) from the BIN PACKING decision problem, which is known to be NP-complete.

The IDK cascade decision problem involves determining if there is an IDK cascade comprising up to n specified classifiers that when run on m processors meets a required classification threshold on the overall probability of successful classification and a maximum latency constraint on the overall elapsed time. Note the IDK decision problem can be solved by any algorithm that determines the optimal IDK cascade.

The BIN PACKING decision problem is characterized as follows: There is a finite set I of items with sizes $g_i \in \mathbb{Z}^+$, an integer bin capacity B, and an integer $K > 1$. The question is, is there a partition of I into disjoint sets $I_1, \ldots, I_K$ such that the sum of the sizes of the items in each I_j is B or less?

Theorem 2 *Given a set of n classifiers and $m > 1$ processors, determining if an IDK cascade exists that complies with a maximum latency constraint D and a classification threshold L is NP-complete.*

Proof First, we note that the feasibility of a solution to the IDK cascade decision problem for multiple processors may be trivially checked by a deterministic algorithm in polynomial time by constructing the processor allocation and schedule from the IDK cascade, determining the finish time of each classifier, and computing the overall success probability from the table of pre-computed $\widehat{P}[S]$ values. Hence deciding if there is a feasible IDK cascade that complies with a maximum latency constraint D and a classification threshold L is therefore in the NP complexity class.

Given an instance of the BIN PACKING problem, we construct an instance of the IDK cascade decision problem for multiple processors as follows. The number of classifiers n equates to the number of elements in the set I, with the execution time of each classifier given by the size of each item in the set, i.e. $\bar{C}_i = g_i$. The maximum latency constraint equates to the size of the bins, $D = B$, and the number of processors equates to the number of partitions (bins), $m = K$. Further, the probability of success for each of the n classifiers is *disjoint* and set to L/n. Thus the success probability for any subset of q classifiers is qL/n, and so execution of all n classifiers is necessary to meet the required classification threshold L.

Now assume that we have a black box that can solve the IDK cascade decision problem for multiple processors. Via the above construction, we may use this black box to solve the BIN PACKING decision problem. Correctness of this approach needs to be shown for both if and only if cases.

If case: For an instance of the BIN PACKING decision problem for which the answer is yes, there exists an IDK cascade that provides an allocation and schedule that executes all n classifiers on m processors within the latency constraint D. The black box, which can solve all IDK cascade decision problems for multiple processors, therefore gives the answer yes.

Only if case: If the black box returns yes, then there exists an IDK cascade that provides an allocation and schedule that executes all n classifiers on m processors within the latency constraint D. This implies that there is an equivalent partition of I into K subsets such that the sum of the sizes of the items in each of the disjoint sets $I_1, \ldots, I_K$ does not exceed B.

We have shown that our algorithm solves the BIN PACKING problem using the black box for the IDK cascade decision problem. Since the construction takes polynomial time, and we have shown that the IDK cascade decision problem is in the NP complexity class, we conclude that the IDK cascade decision problem for multiple processors is NP-complete.

Acknowledgments

This research was funded in part by Innovate UK HICLASS project (113213), and the US National Science Foundation (Grants CPS-1932530, CCF-2028481, and CNS-2141256). EPSRC Research Data Management: No new primary data was created during this study.

References

Balaskas K, Siozios K (2019) Ecg analysis and heartbeat classification based on shallow neural networks. In: 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST), IEEE, pp 1–4

Baruah S, Burns A, Davis RI, Wu Y (2022) Optimally ordering IDK classifiers subject to deadlines. Real-Time Systems DOI 10.1007/s11241-022-09383-w

Baruah SK, Burns A, Wu Y (2021) Optimal synthesis of idk-cascades. In: Queudet A, Bate I, Lipari G (eds) RTNS'2021: 29th International Conference on Real-Time Networks and Systems, Nantes, France, April 7-9, 2021, ACM, pp 184–191, DOI 10.1145/3453417.3453425, URL https://doi.org/10.1145/3453417.3453425

Bateni S, Liu C (2018) Apnet: Approximation-aware real-time neural network. In: 2018 IEEE Real-Time Systems Symposium (RTSS), IEEE, pp 67–79

Bechtel MG, McEllhiney E, Kim M, Yun H (2018) Deeppicar: A low-cost deep neural network-based autonomous car. In: 2018 IEEE 24th international conference on embedded and real-time computing systems and applications (RTCSA), IEEE, pp 11–21

He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition. CoRR abs/1512.03385, URL http://arxiv.org/abs/1512.03385, 1512.03385

Heo S, Cho S, Kim Y, Kim H (2020) Real-time object detection system with multi-path neural networks. In: 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE, pp 174–187

Hossain MSB, Dranetz J, Choi H, Guo Z (2022) Deepbbwae-net: A cnn-rnn based deep superlearner for estimating lower extremity sagittal plane joint kinematics using shoe-mounted imu sensors in daily living. IEEE Journal of Biomedical and Health Informatics 26(8):3906–3917, DOI 10.1109/JBHI.2022.3165383

Hu Y, Liu S, Abdelzaher T, Wigness M, David P (2021a) 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), IEEE, pp 169–178

Hu Y, Liu S, Abdelzaher TF, Wigness M, David P (2021b) On exploring image resizing for optimizing criticality-based machine perception. In: 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2021, Houston, TX, USA, August 18-20, 2021, IEEE, pp 169–178, DOI 10.1109/RTCSA52859.2021.00027, URL https://doi.org/10.1109/RTCSA52859.2021.00027

Kangunde V, Jamisola RS, Theophilus EK (2021) A review on drones controlled in real-time. International journal of dynamics and control 9(4):1832–1846

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA, Plenum Press, New York, The IBM Research Symposia Series, pp 85–

103, DOI 10.1007/978-1-4684-2001-2_9, URL https://doi.org/10.1007/978-1-4684-2001-2_9

Khani F, Rinard MC, Liang P (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, DOI 10.18653/v1/p16-1090, URL https://doi.org/10.18653/v1/p16-1090

Kim JE, Bradford R, Shao Z (2020) Anytimenet: Controlling time-quality tradeoffs in deep neural network architectures. In: 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, pp 945–950

Liu CL (1969) Scheduling algorithms for multiprocessors in a hard real-time environment. JPL Space Programs Summary 37–60:28–31

Liu D, Wang T, Liu S, Wang R, Yao S, Abdelzaher T (2021) Contrastive self-supervised representation learning for sensing signals from the time-frequency perspective. In: 2021 International Conference on Computer Communications and Networks (ICCCN), IEEE, pp 1–10

Liu D, Abdelzaher T, Wang T, Hu Y, Li J, Liu S, Caesar M, Kalasapura D, Bhattacharyya J, Srour N, Wigness M, Kim J, Wang G, Kimberly G, Yao S (2022) IoBT-OS: Optimizing the sensing-to-decision loop for the internet of battlefield things. In: Proceedings of the 31st International Conference on Computer Communications and Networks, ICCCN 2022, July 25-28, 2022, IEEE

Madani O, Georg M, Ross DA (2012) On using nearly-independent feature families for high precision and confidence. In: Hoi SCH, Buntine WL (eds) Proceedings of the 4th Asian Conference on Machine Learning, ACML 2012, Singapore, Singapore, November 4-6, 2012, JMLR.org, JMLR Proceedings, vol 25, pp 269–284, URL http://proceedings.mlr.press/v25/madani12.html

Madani O, Georg M, Ross DA (2013) On using nearly-independent feature families for high precision and confidence. Mach Learn 92(2-3):457–477, DOI 10.1007/s10994-013-5377-0, URL https://doi.org/10.1007/s10994-013-5377-0

Madras D, Pitassi T, Zemel RS (2018) Predict responsibly: Improving fairness and accuracy by learning to defer. In: Bengio S, Wallach HM, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R (eds) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pp 6150–6160, URL https://proceedings.neurips.cc/paper/2018/hash/09d37c08f7b129e96277388757530c72-Abstract.html

Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein MS, Berg AC, Fei-Fei L (2015a) Imagenet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252, DOI 10.1007/s11263-015-0816-y, URL https://doi.org/10.1007/s11263-015-0816-y

Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein MS, Berg AC, Fei-Fei L (2015b) Imagenet large scale

visual recognition challenge. Int J Comput Vis 115(3):211–252, DOI 10.1007/s11263-015-0816-y, URL https://doi.org/10.1007/s11263-015-0816-y

Shi W, Alawieh MB, Li X, Yu H (2017) Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey. Integration 59:148–156

Trappenberg TP, Back AD (2000) A classification scheme for applications with ambiguous data. In: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN 2000, Neural Computing: New Challenges and Perspectives for the New Millennium, Como, Italy, July 24-27, 2000, Volume 6, IEEE Computer Society, pp 296–301, DOI 10.1109/IJCNN.2000.859412, URL https://doi.org/10.1109/IJCNN.2000.859412

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

Yao S, Hu S, Zhao Y, Zhang A, Abdelzaher T (2017a) Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In: Proceedings of the 26th international conference on world wide web, pp 351–360

Yao S, Zhao Y, Zhang A, Su L, Abdelzaher T (2017b) Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In: Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, pp 1–14

Yao S, Hao Y, Zhao Y, Shao H, Liu D, Liu S, Wang T, Li J, Abdelzaher T (2020) Scheduling real-time deep learning services as imprecise computations. In: 2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), IEEE, pp 1–10

Author Biographies

Tarek Abdelzaher (Ph.D., UMich, 1999) is a Sohaib and Sara Abbasi Professor of CS and Willett Faculty Scholar (UIUC), with over 300 refereed publications in Real-time Computing, Distributed Systems, Sensor Networks, and IoT. He was Editor-in-Chief of J. Real-Time Systems for 20 years, an AE of IEEE TMC, IEEE TPDS, ACM ToSN, ACM TIoT, and ACM ToIT, among others, and chair of multiple top conferences in his field. Abdelzaher received the IEEE Outstanding Technical Achievement and Leadership Award in Real-time Systems (2012), a Xerox Research Award (2011), and several best paper awards. He is a fellow of IEEE and ACM.

Kunal Agrawal is a Professor at the Washington University in St. Louis, Prior to 2009, she worked with Professor Charles Leiserson in the Massachusetts Institute of Technology Supercomputing Technologies Group. The goal of her 2012 National Science Foundation CAREER Award, "Provably Good Concurrency Platforms for Streaming Applications," is to design platforms that will allow programmers to easily write correct and efficient high-throughput parallel programs.

Sanjoy Baruah is the Hugo F. & Ina Champ Urbauer Professor of Computer Science and Engineering at Washington University in Saint Louis. His research interests and activities are in real-time and safety-critical system design, scheduling theory, and resource allocation and sharing in distributed computing environments.

Alan Burns holds a Personal Chair in the Department of Computer Science at the University of York in the UK. He is a Fellow of the Royal Academy of Engineering and a Fellow of the IEEE. He has chaired the IEEE Technical Committee on Real-Time Systems and received their "Outstanding Technical Achievement and Leadership Award" in 2006. His research interests include real-time system scheduling, mixed-criticality systems and programming languages.

Robert I. Davis is a Reader in the Real-Time Systems Research Group at the University of York, UK. Robert received his PhD in Computer Science from the University of York in 1995. Since then he has founded three start-up companies, all of which have succeeded in transferring real-time systems research into commercial products. Robert's research interests include real-time scheduling and schedulability analyses for single-core, multi-core, and networked systems.

Zhishan Guo is an Associate Professor in the Department of Computer Science at NC State University. Prior to 2022, he was an Associate Professor in the Department of Electrical and Computer Engineering at University of Central Florida, where he directed the Real-Time Intelligent Systems Lab. Zhishan's research interests are in real-time scheduling theory, machine learning theory, and their applications to Cyber-Physical Systems.

Yigong Hu received the BS degree from Shanghai Jiao Tong University and the MS degree from Columbia University, in 2018 and 2020, respectively. He is currently working toward a PhD degree in computer science at the University of Illinois at Urbana-Champaign (UIUC). His research interests include intelligent Real-Time Systems, Internet of Things (IoT), and Cyber-Physical Systems (CPS).