Optimal Synthesis of Fault-Tolerant IDK Cascades for Real-Time Classification

Sanjoy Baruah* Iain Bate† Alan Burns† Robert I. Davis†
*Washington University in St Louis. Email: baruah@wustl.edu
†University of York. Email: {iain.bate, alan.burns, rob.davis}@york.ac.uk

***Abstract*—An *IDK classifier* is a computational element that classifies an input provided to it into one of a set of predefined categories provided that it can achieve the necessary confidence level to do so; otherwise, it outputs "I Don't Know" (IDK). The concept of *IDK classifier cascades* has emerged as a strategy for striking a balance between the requirements of rapid response and precise classification in machine perception. Effective algorithms for constructing IDK classifier cascades have recently been developed. Here we extend these prior approaches by incorporating fault-tolerance: enabling classification that is concurrently rapid and accurate even in the event of some of the IDK classifiers exhibiting faulty behavior.**

I. Introduction

A classifier is a software component that assigns each input it receives to one of a predefined set of classes. In the realm of autonomous mobile Cyber-Physical Systems (CPS), the process of perception increasingly relies on classifiers founded on Deep Learning and related AI technologies [8], [15], [6]. These classifiers are required to make accurate real-time predictions while working with limited computational resources. A significant proportion of current machine learning research tends to emphasize accuracy at the expense of real-time considerations. This has led to the development of classifiers with high accuracy that can be quite time-consuming when processing even simple inputs that should be straightforward to classify. For example, Wang et al. [17] demonstrated that for a substantial proportion of the ImageNet 2012 benchmark [14], a tenfold increase in classifier execution time resulted in only a marginal enhancement in prediction accuracy. Their recommendation was to strike a balance between accuracy and processing speed by employing advanced (and hence slower) classifiers exclusively for the more challenging cases, so that the overall time required for successful classification would be reduced without sacrificing accuracy. One approach aimed at achieving appropriate accuracy-latency trade-offs is the use of IDK classifiers [9], [16]. An IDK classifier can be derived from any pre-existing base classifier; if the base classifier fails to decide upon a classification with a confidence level surpassing a specified confidence threshold, it instead outputs a placeholder class, labeled as 'IDK,' signifying 'I Don't Know.'

For a given classification problem, it is possible to create multiple distinct IDK classifiers, each with differing execution times and probabilities of producing an actual class instead of IDK. Wang et al. [17] suggested the organization of these IDK classifiers into what they termed *IDK cascades*. These cascades consist of sequences of IDK classifiers that function as follows:

1) The initial classifier in the IDK cascade is the first to be invoked when classifying any input.
2) If this classifier outputs a real class as opposed to IDK, then the IDK cascade concludes, and the input is categorized under the identified class.
3) If, on the other hand, the classifier outputs IDK, then the next classifier in the IDK cascade is invoked, and the process continues from step 2.

In scenarios where it is imperative that all inputs be classified (i.e., an IDK response from the cascade is unacceptable), IDK classifiers need to work in conjunction with a more traditional *deterministic* classifier designed for the same classification task. By placing a deterministic classifier as the final component of an IDK cascade, it is ensured that the IDK cascade will always yield a real class, since if all of the IDK classifiers produce IDK for a particular input, then the deterministic classifier will step in to provide a definitive classification, or alternatively, instigate an appropriate degraded behavior.

This work. Recent research in the real-time systems community (e.g., [1], [2], [4], [5]) has studied IDK classifiers from the perspective of synthesizing IDK cascades that minimize the expected execution duration needed to obtain a real (i.e., non-IDK) classification, optionally within a specified latency constraint. In this paper we additionally allow for the possibility that individual IDK classifiers may occasionally exhibit *faulty behavior* on some inputs; in the sense that they may incorrectly classify the input as belonging to a class that does not match the ground truth. We seek to synthesize IDK cascades that are tolerant to such faults. Specifically, the **contributions** of this paper include the following:

1) In order to obtain an understanding of fault modeling for real-time classification problems, we ***conduct a systematic study*** that characterizes the sources of failures and the means of mitigation.
2) Based on this study, we ***propose a framework*** for studying fault-tolerance in IDK Cascades, by identifying a set of distinguishing characteristics of any classification problem that may be solved using IDK cascades, and seeking to understand how these characteristics dictate the specific modeling and mitigation strategy that needs to be used.

3) We ***instantiate this framework*** for a particular fault-tolerant classification problem, deriving an algorithm for synthesizing optimal fault-tolerant IDK cascades for this instantiation (i.e., this particular problem).
4) We ***evaluate*** our algorithm on real-world data in order to demonstrate its performance in practice.

Organization. The remainder of the paper is organized as follows. In Section II, we describe the system model (largely adapted from prior real-time systems research on IDK classifiers [1], [2], [4], [5]) that we use, and briefly state how we plan to extend it to incorporate considerations of fault tolerance. In Section III, we survey prior publications related to our work. In Section IV, we identify fault models for IDK classifiers, and in Section V, outline a framework for defining fault-tolerant IDK cascades. In Section VI, we explore a particular instantiation of the framework proposed in Sections IV and V. We specify a concrete fault-tolerance problem involving IDK classifiers that we aim to solve, provide a detailed illustrative example, and derive an optimal algorithm for solving the problem. We provide an experimental evaluation of this algorithm in Section VII, thereby demonstrating its applicability on a real case study. We conclude in Section VIII with some directions for future research.

II. Model

We first describe the formal model for IDK classifiers that has previously been considered in the real-time literature, before briefly discussing how we propose to extend the model in order to account for the possible occurrence of faults.

Consider a scenario in which we have n IDK classifiers denoted by $K_1, K_2, \ldots, K_n$, all for the same classification problem. As mentioned in Section I, we assume that there is a probability associated with each of these classifiers successfully classifying any given input. These classifiers may exhibit various mutual dependences between their behaviors; they are not obliged to be independent. In a conceptual framework, it proves useful to envision the probability space for these n IDK classifiers as a Venn Diagram divided into 2^n distinct regions. Each of these regions corresponds to one of the 2^n potential combinations of the n individual classifiers returning either a real class or IDK for an input, see Figure 1 for the case of $n = 3$ classifiers.

Abdelzaher et al. [1] provide a description of how profiling using representative input data can be employed to estimate the probability values linked to each of these 2^n regions. In essence, this methodology involves the following steps: (i) maintaining a counter, initially set to zero, corresponding to each of these 2^n regions; (ii) processing each input sample from the profiling data by having each of the n IDK classifiers process it and observing the outcome, whether it is IDK or not; (iii) incrementing the relevant counters based on these outcomes; and (iv) after evaluating all profiling input samples, dividing each counter value by the total number of samples, hence computing the estimated probability for the precise outcome associated with the respective region. For example, a region such as $(K_1, \overline{K}_2, \overline{K}_3)$, indicating that classifier K_1 reports a real class while classifiers K_2 and K_3 output IDK, is assigned an estimated probability of occurrence. Thus the methodology characterizes the IDK classifiers by:

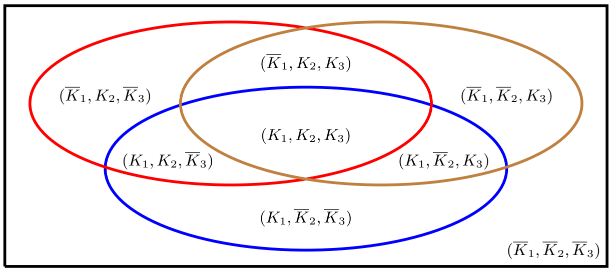

$(\overline{K}_1, K_2, K_3)$
$(\overline{K}_1, K_2, \overline{K}_3)$
$(\overline{K}_1, \overline{K}_2, K_3)$
(K_1, K_2, K_3)
$(K_1, K_2, \overline{K}_3)$
$(K_1, \overline{K}_2, K_3)$
$(K_1, \overline{K}_2, \overline{K}_3)$
$(\overline{K}_1, \overline{K}_2, \overline{K}_3)$

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

1) The expected (i.e., average) and the worst-case execution times of each of the n classifiers; and
2) The 2^n probability estimates, one corresponding to each of the 2^n possible combinations of the n individual classifiers returning either a real class or IDK.

Note that the *size of the model used to characterize the n IDK classifiers is* $\Theta(2^n)$. This exponential model size is a necessity when dealing with classifiers that exhibit arbitrary inter-dependences between their behaviors. To encompass these arbitrary dependences between all pairs of classifiers, it is vital to quantify the probability of each of the 2^n subsets of classifiers successfully classifying inputs, as is done in [1]. The number of IDK classifiers, denoted as n, available for solving a specific classification problem, is clearly application dependent. For example, in the extensive Multi-Modal case study detailed in [1], there are 9 classifiers. These classifiers are constructed from various neural-network architectures and employ multiple input modes (vision, acoustic, and seismic). As a general guideline, values of n exceeding approximately 10 to 12 are unlikely to be commonly encountered in practical applications. Therefore, the exponential model size, which is viable up to approximately $n = 20$, is not a concern in practice. We note that it is important to consider arbitrary dependences between classifiers in this way, since classifiers exhibit a range of dependences and degrees of correlation in their outputs. By contrast, classifier execution times are typically independent, as evidenced in [1].

Incorporating Faults. As stated in Section I, the main objective of this paper is to extend the IDK classifier model described above (and studied in previous work such as [1], [2], [4], [5]) to incorporate fault tolerance. Specifically, we say that a *fault* occurs whenever, on some input, an IDK classifier returns a real (i.e., non IDK) class that does not match the ground truth (i.e., the true class to which that input actually belongs). A *fault model* specifies what kinds of faults may occur, and how faults in the outputs of different IDK classifiers are correlated. This is discussed in detail in Section IV, where the notion of *exclusivity sets* is introduced to formalize the potential for common failures in different IDK classifiers.

III. Related Work

In their pursuit of establishing a scheduling-theoretic framework that facilitates the incorporation of AI-based algorithms into hard real-time safety-critical systems, the real-time scheduling theory community, starting from 2021, has directed its attention towards the investigation of IDK classifiers [1], [2], [4], [5]. Their objective is to develop the capability to construct IDK cascades that minimize the expected execution duration required to achieve successful classification, while optionally adhering to a specified latency constraint. The workload model employed here (elaborated in Section II above) is adopted from [1], which also provides a comprehensive overview of previous research concerning the synthesis of IDK cascades.

Simpler models compared to the one outlined in [1] have been defined, but these models rely on specific assumptions concerning the inter-dependences among the behaviors of different IDK classifiers. For example, an assumption is made in [5] that all classifiers exhibit pair-wise independence, whereas [4] allows for classifiers that are fully dependent and also considers a combination of classifiers with both independent and fully dependent behaviors.

A recognition that IDK classifiers do not perform perfectly in such an idealized manner is taken into account in [2]. This work focuses on binary classes where false-negative and false-positive behaviors are possible. IDK cascades are synthesised that minimize the probability of false positives while meeting both a latency constraint and a constraint on the maximum acceptable probability of false negatives. Our work also considers fault tolerance, but uses a more expressive fault model, as described in Section IV.

A different perspective on non-idealized behavior was recently examined in [3]. There, the probability values characterizing the behavior of the IDK classifiers are taken to represent *predictions* (i.e., estimates) of their true values, which are assumed to be unknown. The objective is then to synthesize IDK cascades that are robust to incorrect predictions. It would be interesting to additionally incorporate the possibility of faults into the model from [3]; however, doing so is outside of the scope of this paper, and hence left for future research.

IV. Fault Models for Real-Time Classifiers

This section provides an understanding of the categories of failures that affect IDK classifier behavior, and hence how to produce IDK cascades that mitigate the effects of the failures, so that the overall classification process is fault tolerant.

A. Understanding the Categories of Failures

The first step in understanding the categories of failure is to understand the faults that lead to them. A classical approach for understanding how failures can occur is a fault tree. Fault trees are widely used in the critical systems industry having first been developed as part of justifying that Inter-Continental Ballistic Missiles could not be launched inadvertently [18], [7]. A survey [13] of over 150 uses of the technique illustrates its widespread adoption. Figure 2 shows a simplified version of a fault tree for an IDK classifier. The top-level event starts with the hazard of concern, *Individual classifier provides plausible incorrect result*, which is then decomposed into failures that cause the higher-level failure. At the bottom level are either basic failure events, i.e., those that are not decomposed further, or a failure that could be extended elsewhere in the case of *Insufficient training in the current context*. The reason the fault tree is considered simplified is that in practice all of the bottom-level failures, e.g., *Sensor(s) provides a distorted representation of the object*, have not been decomposed further into what causes them.

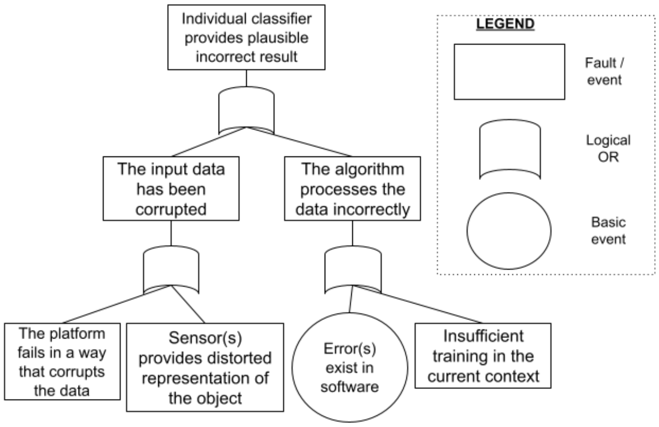

Individual classifier provides plausible incorrect result
The input data has been corrupted
The algorithm processes the data incorrectly
The platform fails in a way that corrupts the data
Sensor(s) provides distorted representation of the object
Error(s) exist in software
Insufficient training in the current context
LEGEND
Fault / event
Logical OR
Basic event

Fig. 2. Fault Tree for an IDK classifier

The key information that can be taken from this fault tree is that the left-hand side consists of *random* failures and the right-hand side consists of *systematic* failures. This distinction is important for a number of reasons. The main one being that random failures are unpredictable in terms of the inputs that they affect, hence these failures are more difficult to test for and to eliminate. Random failures common to more than one classifier can occur due to failure of, or interference with, shared hardware components. For example, a fault or dirt on an optical sensor distorts the image, wind over a microphone distorts the audio signal, local vibrations interfere with the readings from a seismometer. These failures have the potential to affect the output from every classifier that uses that type of input data. Systematic failures tend to be predictable in terms of manifesting repeatedly given the same or similar inputs. Systematic failures common to more than one classifier can occur due to errors in the same large data sets used to train them, inadequate training data for the specific operational context, and software bugs affecting a common neural network architecture.

B. Characterizing Mitigation Strategies

It is important to understand how failures may be tolerated. System functions typically fall into two categories: ones where *fail safe* is sufficient, i.e., in the event of a failure stopping executing the function is acceptable; or *fail operational* where even after a bounded number of failures the system should continue operating as expected. The difference can be explained using the classical fault-tolerance architectural patterns that are applied in this paper, see Figure 3. The pattern on the left-hand side is the fail-safe version where in the case of disagreement there is a deterministic classifier that can decide which class should be output or instigate a fail-safe behavior, e.g., an autonomous vehicle may need to park itself as soon as possible or the driver must take control. The pattern on the right-hand side is where in the case of a single failure, even with a disagreement, a real class can be output and the faulty classifier can be identified. We note that with more than three classifiers, the pattern on the right-hand side can be extended, so a real class and the faulty classifiers can be identified [10]. The fact that the faulty classifier can be identified means that the system can be fail operational, e.g., an autonomous vehicle can continue to operate as expected.

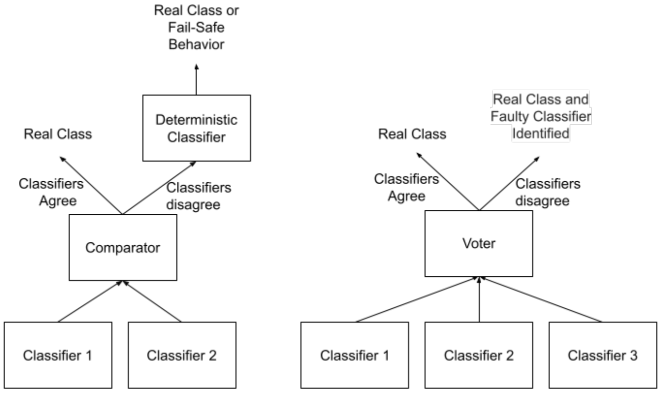


Fig. 3. Architectural Failure Patterns

Within the context of IDK classifiers, we know that they do not necessarily exhibit independent behavior. It is therefore not adequate to just run two arbitrary classifiers in order to identify a fault. Rather, we require that common failure of the classifiers is sufficiently unlikely for the application under consideration. To derive the required evidence, the two categories of failure, random and systematic, are dealt with separately:

1) Random – A fault tree is produced for each classifier. Each of the Basic Errors (events) is examined to identify which classifiers have common error events i.e., the same single error can cause two or more classifiers to produce the same incorrect results.
2) Systematic – The profiling data is used directly to compute the pair-wise correlation coefficients for each pair of classifiers failing on the same input data.

For each classifier the above operations are used to identify those other classifiers that may exhibit dependent failures. Suppose, for example, that two classifiers both take input from the same sensor; a failure of that sensor would cause both classifiers to fail. We refer to such pairs of classifiers as *exclusivity pairs* to denote that they cannot be used to validate each other's classification decisions. We note that exclusivity pairs are not necessarily transitive: it is possible that classifiers K_i and K_j form one exclusivity pair and classifiers K_j and K_k another exclusivity pair, but classifiers K_i and K_k do not form an exclusivity pair. (This can easily happen in practice, if classifier K_j uses readings from two sensors, one of which is also read by K_i, and the other by K_k).

We refer to the set of all classifiers that cannot be used to validate the classification decisions of a classifier K_i as its **exclusivity set**; denoted by $\mathcal{E}(K_i)$. Hence, if classifier K_i outputs a real class, rather than IDK, then to identify a single fault will require some other classifier **not** contained in $\mathcal{E}(K_i)$ to also output a real class. If these classes are identical then there is no fault; if they differ then one classifier has failed; however, which one cannot be determined.

Since the exclusivity sets are based on exclusivity pairs, it follows that if $K_i \in \mathcal{E}(K_j)$, then $K_j \in \mathcal{E}(K_i)$. We also assume that $K_i \in \mathcal{E}(K_i)$; meaning that executing the same classifier twice does not enable a fault to be identified; though for some random failures this may not be the case.

V. A Framework for Defining Fault Tolerant IDK Cascades

In this section we introduce a framework that allows a wide range of classification problems to be specified. Each specific problem, that requires a cascade of IDK classifiers to be derived, is defined by the following characteristics:

- Output Class
- Timing Constraint
- Fault Model
- Data Model
- Fault Recovery Technique
- Optimization Criteria

These characteristics are described below.

Output Class. The output class can either be binary (e.g., Hazard or Clear) or multi-class (e.g., Pedestrian, Cyclist, Car, Truck). It can also include a class that indicates that the input is void (e.g. Cat, Dog, NONE). The output can be a single class, or an ordered list of the most likely classes.

Timing Constraint. The classification may be subject to a latency constraint, i.e., an output must be produced by a specified deadline. This deadline will usually apply to the execution of the complete IDK cascade including the deterministic classifier; however, in some problem instances the deterministic classifier may have a later deadline.

Fault Model. A fault is an output that is not IDK and not the ground truth. It could be caused by a random error (e.g., a sensor failure) or a systematic error (e.g., an issue with the training data). The fault model defines the number of faults to be tolerated (fail safe or fail operational). The faults may be considered independent or correlated.

We are concerned with systems in which one or more potential failures may be critical. For example, with a class that is either 'Hazard' or 'Clear', then outputting 'Clear' when there is a 'Hazard' is usually critical. Outputting 'Hazard' when the correct output is 'Clear' may in some problem instances also be critical, while in others it just represents degraded performance, to be minimized.

Data Model. During profiling it is necessary to gather data that will enable the optimal IDK cascade to be constructed. The profiling process may constrain what data can be collected, and this will restrict the choice of fault recovery techniques and optimization criteria. Typical data collected for each classifier during profiling includes: expected (i.e. average) execution time, worst-case execution time, probability of outputting IDK, and overall success rate. There may also be data collected related to the correlated behavior of the classifiers, for example enabling probability estimates to be determined corresponding to each possible combination of the n individual classifiers returning either a real class or IDK, as well as pair-wise correlation coefficients describing the failure behavior.

Fault Recovery Technique. To identify a single fault requires the execution of two classifiers whose failure behaviors are sufficiently independent. To identify F faults requires $F + 1$ such classifiers. To recover from a single fault it is usually enough to execute three sufficiently independent classifiers. For F faults, $2F + 1$ classifiers must be invoked. Within the context of IDK classifiers, the above numbers apply to classifiers outputting real classes, rather than IDK. Any IDK output will require more classifiers to be executed. At any stage during this process the deterministic classifier can be employed to complete classification. The deterministic classifier may use information from a further classifier, extrapolate from previous outputs, or invoke an application specific degraded level of service that remains safe; for example assuming that the class is 'Hazard' and responding accordingly.

Optimization Criteria. Given a set of classifiers, a fault model, a data model, and a fault recovery technique, an IDK cascade of classifiers must be synthesised that satisfies the requirements of the fault model and the timing constraints. As, in general, there is likely to be more than one solution, this synthesis can also optimize some other aspect of the problem (such as the likelihood of false positives/negatives or the minimization of fault free execution duration). The IDK cascade may be a linear sequence or more generally it may have branching capabilities, and thus be represented by a DAG.

In the remainder of this paper, we focus on a simple specific problem. The output class is binary, the fault model requires F faults to be identified, and the recovery technique is handled by the deterministic classifier. The optimization criterion is to minimize the expected execution duration of fault free behavior, i.e., the overall expected execution time of the IDK cascade when no fault occurs. The data model consists of classifier execution times (average case and worst case), and classification output in terms of success/failure/IDK over the profiling data. The required IDK cascade is a linear sequence, with no branching. Dynamic solutions, skipping classifiers or altering the sequence of classifiers based on the outputs of previous classifiers in the cascade, are left for future research.

VI. Specific Example and its Analysis

In this section we illustrate, on a small example, how to synthesize an optimal IDK cascade. As noted above, we consider a simple fault model, with F faults to be tolerated. Recovery is handled by the deterministic classifier, which by definition of the fault model is itself assumed to be fault-free. Initially we assume no latency constraints (i.e., hard deadlines), but later show how they can be incorporated.

A. The Problem Considered

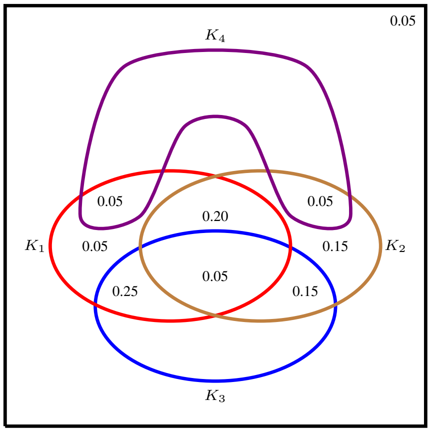
Assume we have a collection $K_1, K_2, \ldots, K_n$ of different IDK classifiers, as well as a deterministic classifier K_d, for the same classification problem. This collection of classifiers is characterized by:

1) The average-case (i.e., expected) execution time and the worst-case execution time for each of the classifiers, with C_i denoting the expected execution time of classifier K_i.
2) The 2^n probability estimates, one corresponding to each of the 2^n possible combinations of the n individual IDK classifiers either returning a real class or IDK.
3) The n *exclusivity sets* $\mathcal{E}(K_1), \mathcal{E}(K_2), \ldots, \mathcal{E}(K_n)$, with the exclusivity set $\mathcal{E}(K_i)$ denoting the set of classifiers that cannot be used to validate, for the purposes of fault tolerance, a classification made by K_i.

Given this collection of classifiers, our goal is to synthesize an IDK cascade, with the fault-free deterministic classifier K_d as the last classifier in the cascade, that is able to tolerate F **faults.** Since a faulty classification by F classifiers must be tolerated, then either (i) $F + 1$ different IDK classifiers that are not in each other's exclusivity sets must agree upon the real (i.e., non IDK) class to which any input belongs, or (ii) the deterministic classifier must make an authoritative classification. Our **performance objective** is to have this IDK cascade perform the classification with the minimum expected execution duration *across all fault-free behaviors*[1].

At run-time, classifiers are executed sequentially in the order in which they appear in the IDK cascade, until either: (i) $F+1$ classifiers that are not in each other's exclusivity sets have returned real (i.e., non-IDK) classes; or (ii) the deterministic classifier K_d is executed, i.e., we have reached the end of the IDK cascade. If K_d is indeed executed, then we return the class that it outputs. Otherwise, once $F + 1$ classifiers not

[1] Here, we are making the implicit assumption that faults are relatively rare, and hence optimizing for expected execution duration in the presence of faults is unlikely to be useful.



Row No.	Classifiers K_1	K_2	K_3	K_4	Prob-S (Probabilities)	Prob-E (Probabilities)
0	0	0	0	0	0.05	0
1	0	0	0	1	0	0
2	0	0	1	0	0	0
3	0	0	1	1	0	0
4	0	1	0	0	0.15	0
5	0	1	0	1	0.05	0.05
6	0	1	1	0	0.15	0.20
7	0	1	1	1	0	0.25
8	1	0	0	0	0.05	0
9	1	0	0	1	0.05	0.05
10	1	0	1	0	0.25	0
11	1	0	1	1	0	0
12	1	1	0	0	0.20	0.25
13	1	1	0	1	0	0.35
14	1	1	1	0	0.05	0.40
15	1	1	1	1	0	0.50

IDK Classifier K_i	Exclusivity set $\mathcal{E}(K_i)$
K_1	$\{K_1, K_3\}$
K_2	$\{K_2\}$
K_3	$\{K_1, K_3\}$
K_4	$\{K_4\}$

Exclusivity Sets

Fig. 4. The 4-classifier example, discussed in Section VI-B. **Left**: Venn-diagram representation of the probabilities of each classifier returning a real class. **Center**: The same information, in tabular form (Prob-S values), as well as computed intermediate probabilities (Prob-E). **Right**: The exclusivity sets.

in each other's exclusivity sets return real classes, we check whether or not these classifiers have returned the same class. If so, we return this class and are done, otherwise *a fault is detected* and we immediately call the deterministic classifier K_d and return the class that is output by K_d.

B. An Example

We now work through a simple example in order to illustrate the process of computing the expected execution duration, in fault-free operation, of a given IDK cascade.[2] To ease understanding, the worked example considers that a single fault must be tolerated, i.e., $F = 1$.

Suppose we have four IDK classifiers K_1, K_2, K_3, and K_4, and a deterministic classifier K_d. The probabilities associated with each of the $2^4 = 16$ possible combinations of the four IDK classifiers are provided in Figure 4, both as a Venn diagram and in tabular form. In the table, the column entitled *Prob-S* represents the probabilities associated with the different combinations of outcomes *when all of the classifiers are run*. This terminology is taken from [1]: "Prob-S [...] 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.". Their exclusivity sets are also specified in Figure 4 — these essentially state that K_1 and K_3 comprise an exclusivity pair, and that the remaining classifiers are independent from the perspective of susceptibility to faulty behavior.

We also introduce an intermediate probability *Prob-E* (shown in the table in Figure 4), with the following meaning. Prob-E is the probability that when *only* those classifiers indicated by 1's in the associated row of the table are executed, then at least $F+1$, i.e. two, of those classifiers that are not in each other's exclusivity sets will return a real class. In other words, Prob-E is the probability that under fault-free operation, no further classifiers will need to execute. We return to how the Prob-E values are computed later.

Let us compute the expected execution duration, assuming fault-free operation, of the IDK cascade

$$\langle K_1, K_2, K_3, K_4, K_d \rangle \tag{1}$$

Since we seek 1-fault tolerance, classifiers K_1 and K_2 will always execute. It is evident from Figure 4 that they will both succeed, i.e., return real classes, with probability $0.20 + 0.05 = 0.25$. (Note, this is the Prob-E value in row 12 of the table, corresponding to $\{K_1, K_2\}$). Hence, the probability that classifier K_3 will execute is equal to

$$1 - 0.25 = \mathbf{0.75}$$

Further, classifier K_4 will execute unless either or both of K_1 and K_3, as well as K_2, have returned real classes. From the Venn diagram in Figure 4, we see that the probability of this happening is equal to

$$1 - (0.20 + 0.15 + 0.05) = \mathbf{0.60}$$

Here, 0.20 is the probability that only K_1 and K_2 return real classes, 0.15 is the probability that only K_3 and K_2 return real classes, and 0.05 is the probability that all three of K_1, K_2 and K_3 return real classes. (Note, these three values sum to 0.4, which is the Prob-E value in row 14 of the table, corresponding to $\{K_1, K_2, K_3\}$).

Finally, the deterministic classifier K_d executes, in fault-free operation, unless at least two of

$$(K_1 \vee K_3), K_2, K_4 \tag{2}$$

return a real class. Let us refer to this happening as event $\mathcal{A}$.

We see from Figure 4 that only K_1 and K_2 succeed with probability 0.2, only K_1 and K_4 succeed with probability 0.05, only K_3 and K_4 succeed with probability 0, and only K_3 and K_2 succeed with probability 0.15. Further, only K_2 and K_4 succeed with probability 0.05, and only K_1, K_2 and K_3 succeed with probability 0.05. Finally, there is zero probability

[2] This example is contrived for illustrative purposes and is not intended to be realistic. The probability values have been chosen to highlight salient features of the problem and our proposed solutions, rather than to be faithful to reality. Evaluations on real-world case studies are detailed in Section VII.

that K_4 succeeds along with two or more other classifiers. Hence the probability of event $\mathcal{A}$ happening is 0.5. (This is the Prob-E value in row 15 of the table, corresponding to $\{K_1, K_2, K_3, K_4\}$). Hence the probability of K_d being executed is equal to

$$(1 - 0.5) = \mathbf{0.50}$$

Putting the above pieces together, we have the following expression for the expected execution duration of the IDK cascade $\langle K_1, K_2, K_3, K_4, K_d\rangle$

$$\overbrace{(C_1 + C_2)}^{K_1 \text{ and } K_2} + \overbrace{0.75 \times C_3}^{K_3} + \overbrace{0.60 \times C_4}^{K_4} + \overbrace{0.5 \times C_d}^{K_d} \tag{3}$$

Assuming that the expected execution times of the classifiers K_1, K_2, K_3, K_4, and K_d are 10, 12, 20, 280, and 500 respectively, then the expected execution duration of the IDK cascade is **455**.

Next, let us repeat the exercise on the IDK cascade

$$\langle K_1, K_2, K_4, K_3, K_d\rangle \tag{4}$$

and compute its expected execution duration, also assuming fault-free operation. As before, K_1 and K_2 both always execute. Next, K_4 will execute unless both K_1 and K_2 return a real class, this happens with probability

$$1 - (0.20 + 0.05) = \mathbf{0.75}$$

Further, classifier K_3 will execute unless two or more of K_1, K_2, and K_4 return real classes. From the Venn diagram in Figure 4, we see that the probability of this happening is equal to

$$1 - (0.25 + 0.05 + 0.05) = \mathbf{0.65}$$

Here, 0.25 is the probability that, of those three classifiers, only K_1 and K_2 return real classes, the first 0.05 is the probability that only K_1 and K_4 return real classes, and the second 0.05 is the probability that only K_2 and K_4 return real classes. Further, there is zero probability that all three classifiers will return real classes. (The sum of these values is 0.35. This is the Prob-E value in row 13 of the table, corresponding to $\{K_1, K_2, K_4\}$).

Finally, we observe that the deterministic classifier K_d executes unless at least two of

$$(K_1 \vee K_3), K_2, K_4$$

return a real class. Note that this is exactly the event $\mathcal{A}$ defined in (2) above. As we have already computed the probability of event $\mathcal{A}$ to be equal to 0.5; putting the above pieces together, we have the following expression for the expected execution duration of the IDK cascade $\langle K_1, K_2, K_4, K_3, K_d\rangle$

$$\overbrace{(C_1 + C_2)}^{K_1 \text{ and } K_2} + \overbrace{0.75 \times C_4}^{K_4} + \overbrace{0.65 \times C_3}^{K_3} + \overbrace{0.5 \times C_d}^{K_d} \tag{5}$$

Assuming the same execution times for the classifiers as before, this IDK cascade has an expected execution duration of **495** compared to 455. for the IDK cascade $\langle K_1, K_2, K_3, K_4, K_d\rangle$.

C. Computing the Prob-E Probabilities

In the above example, we showed how the expected execution duration of a given IDK cascade may be computed. As part of that example, we referred to how the Prob-E values may be used as intermediate values, simplifying and speeding up the calculation. We now describe how these values can be computed from the Prob-S values, obtained via profiling. In doing so, we cater for the general case of F faults, and thus require that $F + 1$ classifiers that are not in each other's exclusivity sets return real classes in order to meet the requirements for fault tolerance.

First, we provide a mathematical formulation of how the Prob-E values can computed for each of the 2^n subsets of classifiers S. We then describe how these values can be computed efficiently.

Recall that the probability space is divided into 2^n regions representing the probability, here referred to as Prob-S(T), of exactly the classifiers in the subset T returning a real classification, and those classifiers that are not in T returning IDK, when all of the classifiers are executed. Further, Prob-E(S) is the probability that when only those classifiers in the subset S are executed, then at least $F + 1$ of those classifiers that are not in each other's exclusivity sets will return a real class. Prob-E(S) can be defined as a summation over the regions of the probability space, represented by the 2^n unique subsets T, as follows:

$$\begin{aligned}&\text{Prob-E}(S) = \\ &\sum_{\forall T} \text{Prob-S}(T) : \exists Q : Q \subseteq (S \cap T) \wedge |Q| = F + 1 \\ &\wedge \nexists K_i, K_j \in Q, K_i \neq K_j, K_i \in \mathcal{E}(K_i)\end{aligned} \tag{6}$$

In the above formulation, each region T of the probability space contributes its probability, Prob-S(T), to Prob-E(S) if and only if there exists some subset Q of exactly $F + 1$ classifiers, K_i, K_j etc., that are not in each others exclusion sets, and the classifiers in Q are in S, i.e., they are all executed, and they are also in T, i.e., they all return a real class.

For each set of the 2^n sets of classifiers S (identified by 1's in the classifier columns in the table in Figure 4), we can efficiently compute the corresponding Prob-E(S) value as follows. First, we initialize Prob-E(S) to zero. Then, for each of the 2^n rows in the table representing a set of classifiers T, we determine the intersection $V = S \cap T$, which represents the set of classifiers in S that would return a real class in this case. To cater for F faults, we use a *validity table* of Boolean values, Valid(V), indicating whether all of the classifiers in V returning a real class would be sufficient to obtain at least $F + 1$ real classes from classifiers that are not in each other's exclusivity sets. The derivation of this lookup table is given below. If Valid(V) is `true`, then the Prob-S(T) value is added to Prob-E(S).

The validity table is computed from a further *sufficiency table* of 2^n Boolean values, Suffice(Q), which correspond to the 2^n distinct sets of classifiers Q. Suffice(Q) is defined to be `true` if and only if the set Q contains exactly $F+1$ classifiers and none of those classifiers are in each other's exclusivity sets. (This exclusivity requirement can be checked by determining if there exists a pair of classifiers $K_i, K_j \in Q$, $K_i \neq K_j$, such that $K_i \in \mathcal{E}(K_j)$).

The 2^n Boolean values, Valid(V), in the validity table are computed as follows. First, Valid(V) is initialized to `false`, then for each of the 2^n distinct sets Q, we determine if (i) Suffice(Q) is `true` and (ii) Q is a subset of V, i.e., $V \cap Q = Q$. If both of these conditions hold then Valid(V) is set to `true`. In other words, Valid(V) is `true` if and only if there exists some subset of V consisting of exactly $F+1$ classifiers that return real classes and are not in each others exclusivity sets.

Using a binary representation of the sets of classifiers, and assuming that set operations such as intersection, counting the number of members of a set, and table lookup based on set membership can be done in $O(1)$ time[3], then (i) the complexity of determining the 2^n Prob-E values from the Prob-S values and the validity table is $O(2^n \cdot 2^n) = O(4^n)$; (ii) the complexity of determining the 2^n values in the validity table from the 2^n values in the sufficiency table is $O(2^n \cdot 2^n) = O(4^n)$; and (iii) the complexity of determining the 2^n values in the sufficiency table is $O(n(n-1) \cdot 2^n)$, which is trivially upper bounded by $O(4^n)$, since $n(n-1) \leq 2^n$. It follows that the overall complexity involved in deriving the 2^n Prob-E values is $O(4^n)$. This is the same complexity as the case for zero faults derived in [1]. **Hence, this method caters for a requirement to tolerate F faults, with no increase in complexity over the no fault case**.

We now return to our worked example, assuming that one fault must be tolerated. Consider computing the Prob-E value for $S = \{K_1, K_2, K_3\}$, i.e., row 14 of the table in Figure 4. For each set T (row in the table) we determine the intersection set $V = S \cap T$, and whether there is a contribution from V to the Prob-E value for S.

- Rows 0-5, 8, and 9, the intersection set V contains zero or one classifiers, and so these rows trivially do not contribute.
- Rows 6 and 7, the intersection set V contains K_2 and K_3, since these classifiers are not in each other's exclusivity sets, these rows contribute their Prob-S values (0.15 and 0).
- Rows 10 and 11, the intersection set V contains only K_1 and K_3, since these classifiers *are* in each other's exclusivity sets, these rows *do not* contribute.
- Rows 12-15, the intersection set V contains K_1 and K_2, since these classifiers are not in each other's exclusivity sets, these rows contribute their Prob-S values (0.2, 0, 0.05, 0).

Summing the contributions, the Prob-E value for $S = \{K_1, K_2, K_3\}$ is 0.4 (see row 14 of the table in Figure 4).

D. A Naïve Algorithm

As shown above, it is straightforward to compute the expected execution duration, assuming fault-free operation, for any given IDK cascade. This observation immediately yields a naïve algorithm for determining the IDK cascade of minimum expected execution duration: simply examine all IDK cascades that can be constructed from the available classifiers, and choose the one with the smallest expected execution duration. Unfortunately, such an approach of exhaustive enumeration is computationally intractable even when the number of available distinct classifiers is relatively small, since it is straightforward to show that we can synthesize $\Theta(n!)$ distinct IDK cascades from a given collection of n classifiers.

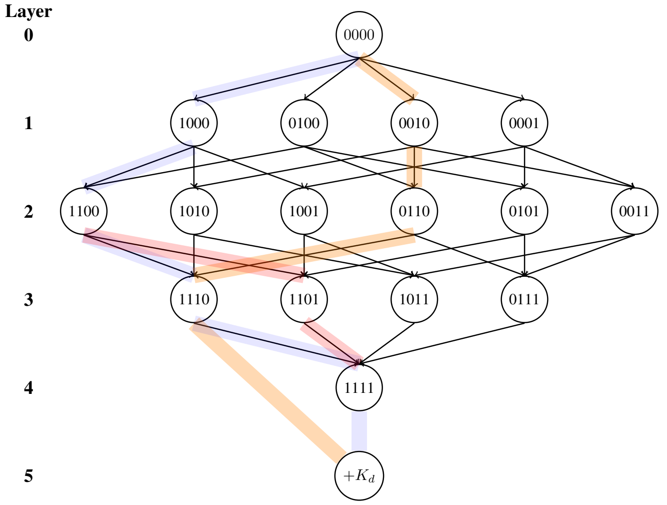


Fig. 5. DAG: The first vertex represents the empty set of IDK classifiers, and the last vertex represents the deterministic classifier K_d that terminates each IDK cascade.

However, our example computations above illustrate an important property: *the probability of a particular classifier in an IDK cascade executing or not is determined solely by the* *<u>set of classifiers</u>* *that preceded it in the IDK cascade, not the relative order in which they appear*. For example, observe that the terms corresponding to classifier K_d in both (3) and (5) are the same. This is because the set of classifiers preceding K_d in both the IDK cascades of (1) and (4) are the same ($\{K_1, K_2, K_3, K_4\}$), even though these classifiers occur in a different permutation in (1) and (4). A similar property had previously been observed [1] when fault tolerance was not considered. The fact that this property continues to hold even in the presence of potential faults enables us to exploit it. In Section VI-E, we derive an algorithm for identifying the IDK cascade of minimum expected execution duration in fault-free operation by implicitly examining only $O(2^n)$, rather than $\Theta(n!)$, distinct IDK cascades. As pointed out in [1], this improvement can be very significant; e.g., $2^{10} = 1024$ while $10! > 3.6$ million; 2^{20} is about one million whereas $20! \approx 2.4 \times 10^{18}$; etc.

E. Synthesizing Optimal Cascades

In the previous section, we observed the important property that the probability of a particular classifier in an IDK cascade executing or not depends only on the set of classifiers that preceded it in the IDK cascade, and not on their relative ordering. We now describe how this property allows us to

[3] This is easily achievable for $n \leq 64$ on a 64-bit computer architecture.

represent individual IDK cascades as *paths* through a Directed Acyclic Graph (DAG) in which the vertices represent sets of classifiers and the edges correspond to adding a classifier to the end of a partially-constructed IDK cascade.

Suppose we have n IDK classifiers $K_1, K_2, \ldots, K_n$, and a deterministic classifier K_d. We construct a DAG with (2^n+1) vertices, arranged in $(n+2)$ layers, as follows. (The DAG for $n = 4$ is depicted in Figure 5).

- For each $i \in \{0, 1, \ldots, n\}$, each vertex in layer i represents a different subset, of cardinality i, of the set of available IDK classifiers. The vertex in the lowest layer (layer $n+1$) has a different interpretation: it represents the addition of the deterministic classifier K_d to the end of an IDK cascade.
  In the DAG for $n = 4$ depicted in Figure 5, each vertex in layers 1-4 is labeled with a binary coding identifying the classifiers in the subset that the vertex represents. (For example, the vertex labeled "1011" represents the set of IDK classifiers $\{K_1, K_3, K_4\}$.)
- Let S denote the set of IDK classifiers represented by a particular vertex in layer i, $0 \le i < n$. We add an edge from this vertex to each vertex in layer $(i+1)$ that represents the union of S and one additional IDK classifier $\notin S$. The black arrows in Figure 5 represent these edges.
- We also add an edge from each vertex in the layers numbered $0, 1, \ldots, n$ to the sole vertex in the layer numbered $(n+1)$. (These edges are not explicitly depicted in Figure 5, in order to enhance clarity).
- Now every IDK cascade can be represented as a path in this DAG from the vertex in layer 0 to the vertex in layer $(n+1)$, with the vertex in this path that lies in the i^{th} level of the DAG representing the subset comprising the first i classifiers in the IDK cascade for each i, $1 \le i \le n$.
  Some such paths are highlighted in Figure 5. Those corresponding to the cascades (K_3, K_2, K_1, K_d) and $(K_1, K_2, K_3, K_4, K_d)$, are completely shown, while the path corresponding to the cascade $(K_1, K_2, K_4, K_3, K_d)$ overlaps with the path $(K_1, K_2, K_3, K_4, K_d)$ for the first two edges and the last edge, with the difference between the two paths, "1100"→"1101"→"1111", shown in red.

Next we label the edges of the DAG with the *edge-costs*, such that the expected execution duration of any IDK cascade is equal to the sum of the edge-costs on the corresponding path in the DAG. (The construction of the DAG, as described above, is essentially the same as described in [1]; however the computation of the edge-costs is different, as detailed below).

Determining the edge-costs. Let S denote any set of IDK classifiers, and consider the edge in the DAG from the vertex corresponding to the set S to the vertex corresponding to $\big(S \cup \{K_\ell\}\big)$, where $K_\ell \notin S$ is the classifier added along that edge. The edge-cost is equal to C_ℓ (the expected execution time of classifier K_ℓ) multiplied by the probability that classifier K_ℓ will be executed in any IDK cascade in which S is the set of classifiers that precede it. This probability is given by $(1 - P(S))$, where $P(S)$ is the Prob-E value associated with the set S in the probability table, hence the edge cost is given by $(1 - P(S))C_\ell$.

Determining the optimal IDK cascade. The problem of determining the IDK cascade with minimum expected execution duration over all fault-free executions is reduced to the problem of finding the shortest path from the sole start vertex in layer 0 to the exit vertex in layer $(n+1)$. This is a well-studied problem in graph theory, and algorithms are known for determining the shortest path in a DAG in time linear in the number of vertices plus edges. By using such an algorithm, we can determine the optimal IDK cascade, the one with the minimum expected execution duration over all fault-free executions, in $O(n \cdot 2^n)$ time[4], assuming $O(1)$ time for the look-up of each Prob-E value. Recall, from Section VI-C, that the pre-processing required to compute the 2^n Prob-E values from the 2^n Prob-S values in the probability table is $O(4^n)$. Thus the overall running time of the algorithm is dominated by the pre-processing stage, and is $O(4^n)$.

Incorporating hard deadlines. The algorithm may be generalized to additionally allow for the specification of a latency constraint, i.e., if it is required that classification must complete within a specified deadline under all circumstances. From the first $(n + 1)$ layers of the DAG, we simply remove all vertices for which the sum of the worst-case execution times of the set of classifiers represented by that vertex, plus the worst-case execution time of the deterministic classifier K_d, exceeds the specified deadline. Note, if this removes all of the vertices, then that implies that no feasible solution exists.

F. Challenges to Optimality and Fault-tolerance

Determining the optimal IDK cascade depends on obtaining profiling data and hence a table of probabilities that is representative of the behavior of the particular system of classifiers in its operational scenario. If the data is not representative, then the probability values may differ, and the expected execution time duration of the IDK cascade would not be precisely optimized; however, any deadlines would still be met, assuming that valid worst-case execution time values were used.

Obtaining the desired fault-tolerant behavior depends on the correctness of the exclusivity sets, which in turn depend on the analysis of systematic and random failures among the classifiers. It is questionable whether there would necessarily be sufficient evidence to identify the classifiers involved in systematic failures. The use of Fault Trees is intended to supplement this evidence by considering random failures. In both cases, 100% guarantees are not typically possible. Rather, the analysis aims to reduce the likelihood of using incorrect information As Low As Reasonably Practicable (ALARP).

VII. Evaluation: Multi-Modal Case Study

In this section we evaluate our approach by demonstrating its applicability to a real-world, multi-modal case study.

[4] There are 2^n vertices in the DAG, with at most n edges emanating from each vertex, and hence $O(n \cdot 2^n)$ edges in all.

The data used in this case study was collected previously [11] as part of a project that seeks to autonomously detect the presence of a potentially hostile enemy vehicle in a battlefield environment.[5] Three different kinds of sensors were deployed for this purpose: acoustic (a microphone array), seismic (a vertical-axis geophone), and vision (a camera). Based on this input data, the aim is for the classifiers to determine if a vehicle of the designated target type is present in the detection area. Such functionality is useful in "intelligent tripwire" scenarios, where the system must act only when a specific type of target is present, while ignoring other passing traffic, hence the output class is effectively binary. The application has safety implications, as false positives, i.e., incorrectly identifying a vehicle as hostile, could have consequences for the safety of friendly vehicles, as could false negatives, i.e., mis-classifying a hostile vehicle as friendly.

An IDK cascade is required that can deliver a strong constraint on the likelihood of a false-positive or false-negative output even when a single fault has occurred. If execution of the IDK cascade is fault free then its expected execution duration should be minimized. If a fault occurs, then the deterministic classifier must be called.

Data collection and pre-processing. The manner in which the input samples were collected is described in [11] 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.*"

Following the procedures detailed in Sections 4.1 and 4.2 of [1], we processed the raw data outputs (class and confidence) for each base classifier for each of 1800 randomly chosen input samples[6]. We assumed a required *precision* of 0.95 and used this value to compute a *confidence threshold* for each base classifier. An IDK classifier was then formed from each base classifier: for any given input sample, if the confidence level output by the base classifier met or exceeded the confidence threshold, then a real class was output, otherwise the output was IDK. The way in which the classification thresholds were chosen ensured that the long run probability of each IDK classifier outputting a real class that did not match the ground truth was no more than 1 minus the required precision, i.e. 0.05 in this case study.

Small Case Study. We first consider four IDK classifiers (out of nine in all labelled from A to I):

- B: deepsense_both: Based on the DeepSense neural network architecture [19], trained using contrastive learning [12], uses both seismic and acoustic data.
- D: deepsense_seismic: Based on the DeepSense neural network architecture [12], uses only seismic data.
- F: cnn_acoustic: Based on a standard convolution neural network, uses only acoustic data.
- I: yolov5s-compressed: Based on the YOLOv5 neural network (small version) with image compression using the DeepIoT neural network architecture compression framework [20], uses only image (video) data.

Following the procedures detailed in Sections 4.1 and 4.2 of [1], we used the profiling data for the 1800 input samples to construct, for each of the 16 possible outputs of the four IDK classifiers (1 = real class, 0 = IDK), the probability of occurrence, Prob-S. From those values, we computed the probability, Prob-E, that each distinct subset of classifiers would provide at least two real (i.e., not IDK) outputs from classifiers that are not in each other's exclusivity sets. This information is shown in Table I, along with the average-case and worst-case execution time[7] parameters 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 X that always returns a real class.

TABLE I
MULTIMODAL EXAMPLE

B	F	D	I	Count	Prob-S	Prob-E
0	0	0	0	56	0.0311	0
0	0	0	1	33	0.0183	0
0	0	1	0	35	0.0194	0
0	0	1	1	18	0.01	0.2178
0	1	0	0	11	0.0061	0
0	1	0	1	5	0.0028	0.0589
0	1	1	0	5	0.0028	0.15
0	1	1	1	4	0.0022	0.3489
1	0	0	0	181	0.1006	0
1	0	0	1	76	0.0422	0.265
1	0	1	0	698	0.3878	0
1	0	1	1	304	0.1689	0.2772
1	1	0	0	82	0.0456	0
1	1	0	1	31	0.0172	0.27
1	1	1	0	195	0.1083	0.15
1	1	1	1	66	0.0367	0.3911
TOTALS				1800	1.00	

Classifier	B	F	D	I	X
Average ET (ms)	17	3.9	11.4	1440.8	10000
WCET (ms)	19.6	5.3	13.7	1613.2	10000

Determining the exclusivity sets involves two distinct forms of analysis, one for random failures and one for systematic failures. First, fault trees for each classifier are constructed and the common mode failures are identified. Next the classifiers are profiled and Pearson's correlation coefficient used in a pair-wise analysis of failure independence. For this initial look at just four classifiers we derive the exclusivity sets by only considering the sensor failures. (In the larger case study that follows, we also consider the pair-wise correlations). We

[5] We thank the authors of [11] for providing us with the raw data for this case study.

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

[7] The worst-case execution times are 95-percentile estimates, as also used in [1]. They are close to the average-case execution times due to the fact that the path taken in neural network classifier code is typically not data dependent.

assume that the sensors have no built-in fault tolerance, and hence the allowed single failure, as sanctioned by the fault model, can cause either the seismic data, the acoustic data or the video data to be corrupted. Therefore, the exclusivity sets are: $\mathcal{E}(B) = \{B, D, F\}$, $\mathcal{E}(D) = \{B, D\}$, $\mathcal{E}(F) = \{B, F\}$, and $\mathcal{E}(I) = \{I\}$.

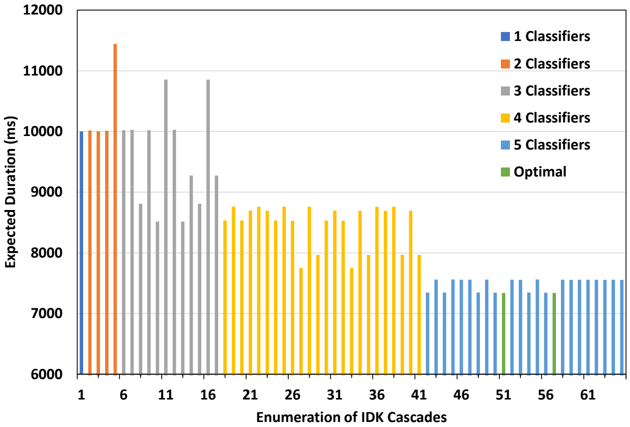


Fig. 6. Expected duration for all 65 possible IDK Cascades.

The DAG algorithm, described in Section VI-E, was used to synthesize the optimal IDK cascade. The expected execution duration of each of the 65 possible IDK cascades is depicted in Figure 6. The minimum expected execution duration is 7339.9ms and this is delivered by the pair of IDK cascades $\langle D, F, I, B, X\rangle$ and $\langle F, D, I, B, X\rangle$ that utilize all 5 classifiers, including the deterministic one. (Recall that the order of the first two classifiers has no effect on the expected execution duration). The best IDK cascade with one or two classifiers is simply $\langle X\rangle$, since no two IDK classifiers suffice on their own, and using one IDK classifier along with the deterministic classifier only adds to the expected duration. The best IDK cascades with three classifiers are $\langle D, F, X\rangle$ and $\langle F, D, X\rangle$ with an expected execution duration of 8515.3ms. Finally, the best IDK cascades with four classifiers are $\langle D, F, I, X\rangle$ and $\langle F, D, I, X\rangle$ with an expected execution duration of 7751.1ms.

Table II shows how the optimal IDK cascades and their expected execution durations vary for a range of different execution times C_X for the deterministic classifier. As the execution time of the deterministic classifier is reduced, so the optimal IDK cascade is composed of fewer classifiers.

TABLE II
SMALL CASE STUDY: THE OPTIMAL IDK CASCADE FOR DIFFERENT EXECUTION TIMES FOR THE DETERMINISTIC CLASSIFIER

C_X	Cascade	Expected Duration (ms)
10000	$\langle D, F, I, B, X\rangle$	7339.9
6000	$\langle D, F, I, B, X\rangle$	4904.4
5000	$\langle D, F, X\rangle$	4265.3
4000	$\langle D, F, X\rangle$	3415.3
1000	$\langle D, F, X\rangle$	865.3
250	$\langle D, F, X\rangle$	227.9

Table III illustrates how the optimal IDK cascade changes with a varying latency constraint or deadline. The Pareto optimal IDK cascades are given along with their expected execution durations and worst-case execution times. The latter corresponds to the value of a latency constraint below which the optimal IDK cascade changes. This table illustrates the trade-off between a reduction in expected execution duration and an increase in overall worst-case execution time.

TABLE III
SMALL CASE STUDY: PARETO OPTIMAL IDK CASCADES

IDK Cascade	Worst-case (ms)	Expected Duration (ms)
$\langle X\rangle$	10000	10000
$\langle D, F, X\rangle$	10019	8515.3
$\langle D, F, I, X\rangle$	11632.2	7751.1
$\langle D, F, I, B, X\rangle$	11651.8	7339.9

Large Case Study. We now turn to the complete case study with nine IDK classifiers and a deterministic classifier. The parameters of the IDK classifiers are given in Table IV. The final column shows the probability that each classifier will return a real class, as opposed to IDK. Note, these probabilities are *not* independent.

TABLE IV
CHARACTERIZATION OF ALL NINE IDK CLASSIFIERS.

Name	Classifier	Execution time (ms)	WCET (ms)	Probability of real class
deepsense_both	A	17.5	21.4	89.9%
deepsense_both_contras	B	17.0	19.6	90.7%
deepsense_acoustic	C	11.7	14.4	21.3%
deepsense_seismic	D	11.4	13.7	73.6%
cnn_both	E	4.0	4.8	59.6%
cnn_acoustic	F	3.9	5.3	22.1%
cnn_seismic	G	3.7	4.6	32.7%
yolov5s	H	3145.9	3475.9	29.9%
yolov5s-compressed	I	1440.8	1613.2	29.8%

The exclusivity sets were computed as follows. First, we considered random failures. Fault trees for each classifier were constructed and the common mode failures identified. These common mode failures stem from shared sensors: audio for classifiers A, B, C, E, F, seismic for classifiers A, B, D, E, G, and video for classifiers H, I. Considering systematic failures, the Pearson correlation coefficients between all pairs of classifiers were computed, see Table V.

TABLE V
LARGE CASE STUDY: PEARSON CORRELATION COEFFICIENTS

	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

Observe that for many classifiers this measure is indicative of pair-wise independent behavior, with absolute correlation values ≤ 0.1 (green cells). Other pairs of classifiers have correlated behavior and so cannot be trusted to verify that each other's outputs are correct. These pairs of classifiers are thus placed in each other's exclusivity sets.

Combining the information about random and systematic failures, the exclusivity sets are as follows:

$$\begin{array}{ll}
\mathcal{E}(A) = \{A, B, C, D, E, F, G\} & \mathcal{E}(F) = \{A, B, C, E, F\} \\
\mathcal{E}(B) = \{A, B, C, D, E, F, G\} & \mathcal{E}(G) = \{A, B, C, D, E, G\} \\
\mathcal{E}(C) = \{A, B, C, E, F, G\} & \mathcal{E}(H) = \{H, I\} \\
\mathcal{E}(D) = \{A, B, D, E, G\} & \mathcal{E}(I) = \{H, I\} \\
\mathcal{E}(E) = \{A, B, C, D, E, F, G\} &
\end{array}$$

Here, almost all of the exclusions due to correlated systematic failures are covered by the exclusions due to random (hardware) failures. The only additional ones are between classifiers C and G, where the correlation (-0.103) is only marginally outside of the bound used. This is unsurprising, since shared input types (audio, seismic, video) can also be a primary cause of correlated systematic failures.

TABLE VI
LARGE CASE STUDY: THE OPTIMAL IDK CASCADE FOR DIFFERENT EXECUTION TIMES FOR THE DETERMINISTIC CLASSIFIER

C_X	Cascade	Expected Duration (ms)
10000	$\langle G, F, D, C, I, E, A, B, X \rangle$	6883.5
6000	$\langle G, F, D, C, I, E, A, B, X \rangle$	4616.9
5000	$\langle G, F, D, C, I, E, A, B, X \rangle$	4050.2
4000	$\langle G, F, D, C, X \rangle$	3266.0
1000	$\langle G, F, D, C, X \rangle$	837.6
250	$\langle F, D, X \rangle$	227.9

Table VI shows how the optimal IDK cascades and their expected execution durations vary for a range of different execution times, C_X, for the deterministic classifier. As the execution time of the deterministic classifier is reduced, so the optimal IDK cascade is composed of fewer classifiers. Intuitively, the DAG algorithm is making a trade-off between fast classification (classifiers F and G), higher probability of returning real classes, but longer execution time (classifier D), and also importantly choosing to run classifiers that are not in each other's exclusivity sets, e.g., focusing on pairs such as (G, F), (F, D), and (D, C). Thus many of the IDK cascades in Table VI begin with $\langle G, F, D, C \rangle$. For longer durations of the deterministic classifier, it becomes more important to reduce the probability that it will run, and so the expensive and less effective video based classifier I is selected, followed by a series of classifiers, E, B, and A that have exclusivity sets that do not include I. Comparing Table VI with Table II for the small case study, it is evident that having additional classifiers to choose from, and the potential for longer IDK cascades, leads to improved solutions, in terms of a reduction in the expected execution duration.

The selection of classifiers available in the large case study is effective in tolerating $F = 1$ faults. Considering $F = 2$ faults, then classifiers A, B, and E are effectively rendered useless by the fact that their outputs can only be verified by classifiers H and I, which are also in each other's exclusivity sets. For $F = 2$ faults, and an execution time of $C_X = 10000$ms for the deterministic classifier, the optimal IDK cascade is simply to run the deterministic classifier. However, with $C_X = 40000$ms, the optimal IDK cascade becomes $\langle G, F, I, D, C, X \rangle$ with an expected execution duration of 39426.4ms, compared to $\langle G, F, D, C, I, E, A, B, X \rangle$ with an expected execution duration of 23883.5 ms for $F = 1$ fault, and $\langle E, D, B, A, G, F, C, I, X \rangle$ with an expected execution duration of 478.8ms for no fault tolerance.

Finally, we note that the pre-processing needed to obtain the Prob-E values and the execution of the DAG algorithm for the large case study takes less than 4 milliseconds on a basic laptop PC (C++ implementation, debug version), illustrating the efficiency of the method for a practical problem of typical size.

VIII. CONCLUSIONS

Future Cyber-Physical Systems seem destined to incorporate a wide range of learning enabled components. Many of these components will involve various forms of classification. IDK classifiers are an effective way of addressing the real-time requirements of such systems. A cascade of IDK classifiers can be tailored to meet timing constraints, safety constraints and performance targets. In this paper we have shown how to construct IDK cascades that are tolerant of failures, such failures resulting from random faults (such as sensor errors) and systematic faults (such as those that can occur when training data does not match the operational environment).

The approach developed in this paper enables the expected execution time duration of the IDK cascade to be minimized in the non-fault case, which is important for meeting deadlines, minimizing system workload, and reducing energy consumption, while also ensuring that faults can be tolerated. The approach has been designed so that it can be generalized in a number of ways: (i) the fault model can involve fail-operational as well as fail-safe behavior; (ii) the criteria to minimize can involve more than just fault-free behavior, for example, a cascade could be designed to deliver fail-safe behavior if there are three concurrent faults, fail-operational behavior if there are one or two faults, and to minimize expected execution duration for fault-free or single-fault behavior; and (iii) the run-time IDK cascade can become more adaptive, for example, represented by a DAG rather than a sequence, this would allow different choices to be made depending on the actual run-time behavior of each individual classifier.

ACKNOWLEDGEMENTS

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

References

[1] Tarek Abdelzaher, Kunal Agrawal, Sanjoy Baruah, Alan Burns, Robert I. Davis, Zhishan Guo, and Yigong Hu. Scheduling idk classifiers with arbitrary dependences to minimize the expected time to successful classification. *Real-Time Systems*, March 2023. doi:10.1007/s11241-023-09395-0.

[2] Tarek F. Abdelzaher, Sanjoy K. Baruah, Iain Bate, Alan Burns, Robert Ian Davis, and Yigong Hu. Scheduling classifiers for real-time hazard perception considering functional uncertainty. In *Proceedings of the 31st International Conference on Real-Time Networks and Systems, RTNS 2023, Dortmund, Germany, June 7-8, 2023*, pages 143–154. ACM, 2023. doi:10.1145/3575757.3593649.

[3] Sanjoy Baruah, Alan Burns, and Robert I. Davis. Optimal synthesis of robust IDK classifier cascades. In Claire Pagetti and Alessandro Biondi, editors, *2023 International Conference on Embedded Software, EMSOFT 2023, Hamburg, Germany, September 2023*. ACM, 2023. doi:10.1145/3609129.

[4] Sanjoy K. Baruah, Alan Burns, Robert I. Davis, and Yue Wu. Optimally ordering IDK classifiers subject to deadlines. *Real Time Syst.*, 59(1):1–34, 2023. URL: https://doi.org/10.1007/s11241-022-09383-w, doi:10.1007/S11241-022-09383-W.

[5] Sanjoy K. Baruah, Alan Burns, and Yue Wu. Optimal synthesis of idk-cascades. In Audrey Queudet, Iain Bate, and Giuseppe Lipari, editors, *RTNS'2021: 29th International Conference on Real-Time Networks and Systems, Nantes, France, April 7-9, 2021*, pages 184–191. ACM, 2021. doi:10.1145/3453417.3453425.

[6] Michael Garrett Bechtel, Elise McEllhiney, Minje Kim, and Heechul Yun. Deeppicar: A low-cost deep neural network-based autonomous car. In *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018, Hakodate, Japan, August 28-31, 2018*, pages 11–21. IEEE Computer Society, 2018. doi:10.1109/RTCSA.2018.00011.

[7] H. Gilmore, R. Woodcock, and R. Jewell. Minuteman mark IIA reentry vehicle system safety engineering plan. Technical report, AVCO Corporation, 1964.

[8] Vemema Kangunde, Rodrigo S Jamisola, and Emmanuel K Theophilus. A review on drones controlled in real-time. *International journal of dynamics and control*, 9(4):1832–1846, 2021.

[9] Fereshte Khani, Martin C. Rinard, and Percy Liang. 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, 2016. URL: https://doi.org/10.18653/v1/p16-1090, doi:10.18653/V1/P16-1090.

[10] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

[11] Dongxin Liu, Tarek F. Abdelzaher, Tianshi Wang, Yigong Hu, Jinyang Li, Shengzhong Liu, Matthew Caesar, Deepti Kalasapura, Joydeep Bhattacharyya, Nassy Srour, Jae Kim, Guijun Wang, Greg Kimberly, and Shouchao Yao. Iobt-os: Optimizing the sensing-to-decision loop for the internet of battlefield things. In *31st International Conference on Computer Communications and Networks, ICCCN 2022, Honolulu, HI, USA, July 25-28, 2022*, pages 1–10. IEEE, 2022. doi:10.1109/ICCCN54977.2022.9868920.

[12] Dongxin Liu, Tarek F. Abdelzaher, Tianshi Wang, Yigong Hu, Jinyang Li, Shengzhong Liu, Matthew Caesar, Deepti Kalasapura, Joydeep Bhattacharyya, Nassy Srour, Jae Kim, Guijun Wang, Greg Kimberly, and Shouchao Yao. Iobt-os: Optimizing the sensing-to-decision loop for the internet of battlefield things. In *31st International Conference on Computer Communications and Networks, ICCCN 2022, Honolulu, HI, USA, July 25-28, 2022*, pages 1–10. IEEE, 2022. doi:10.1109/ICCCN54977.2022.9868920.

[13] Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Comput. Sci. Rev.*, 15:29–62, 2015. URL: https://doi.org/10.1016/j.cosrev.2015.03.001, doi:10.1016/J.COSREV.2015.03.001.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015. URL: https://doi.org/10.1007/s11263-015-0816-y, doi:10.1007/S11263-015-0816-Y.

[15] Weijing Shi, Mohamed Baker Alawieh, Xin Li, and Huafeng Yu. Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey. *Integr.*, 59:148–156, 2017. URL: https://doi.org/10.1016/j.vlsi.2017.07.007, doi:10.1016/J.VLSI.2017.07.007.

[16] Thomas P. Trappenberg and Andrew D. Back. 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*, pages 296–301. IEEE Computer Society, 2000. doi:10.1109/IJCNN.2000.859412.

[17] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E. Gonzalez. IDK cascades: Fast deep learning by learning not to overthink. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 580–590. AUAI Press, 2018. URL: http://auai.org/uai2018/proceedings/papers/212.pdf.

[18] Hugh A. Watson. Launch control safety study; vol. i and ii, 1962.

[19] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek F. Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 351–360. ACM, 2017. doi:10.1145/3038912.3052577.

[20] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek F. Abdelzaher. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In M. Rasit Eskicioglu, editor, *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, SenSys 2017, Delft, Netherlands, November 06-08, 2017*, pages 4:1–4:14. ACM, 2017. doi:10.1145/3131672.3131675.