Proceedings Paper:
Abdelzaher, Tarek F., Baruah, Sanjoy, Bate, Iain John orcid.org/0000-0003-2415-8219 et al. (3 more authors) (2023) Scheduling Classifiers for Real-Time Hazard Perception Considering Functional Uncertainty. In: RTNS '23: Proceedings of the 31st International Conference on Real-Time Networks and Systems. 31st International Conference on Real-Time Networks and Systems, 06-08 Jun 2023 ACM , pp. 143-154.

https://doi.org/10.1145/3575757.3593649

Scheduling Classifiers for Real-Time Hazard Perception Considering Functional Uncertainty

Tarek Abdelzaher
zaher@illinois.edu
University of Illinois
USA

Sanjoy Baruah
baruah@wustl.edu
Washington University in Saint Louis
USA

Iain Bate
iain.bate@york.ac.uk
University of York
UK

Alan Burns
alan.burns@york.ac.uk
University of York
UK

Robert I. Davis
rob.davis@york.ac.uk
University of York
UK

Yigong Hu
yigongh2@illinois.edu
University of Illinois
USA

ABSTRACT

This paper addresses the problem of real-time classification-based machine perception, exemplified by a mobile autonomous system that must continually check that a designated area ahead is free of hazards. Such hazards must be identified within a specified time. In practice, classifiers are imperfect; they exhibit functional uncertainty. In the majority of cases, a given classifier will correctly determine whether there is a *hazard* or the area ahead is *clear*. However, in other cases it may produce *false positives*, i.e. indicate hazard when the area is clear, or *false negatives*, i.e. indicate clear when there is in fact a hazard. The former are undesirable since they reduce quality of service, whereas the latter are a potential safety concern. A stringent constraint is therefore placed on the maximum permitted probability of false negatives. Since this requirement may not be achievable using a single classifier, one approach is to (logically) OR the outputs of multiple disparate classifiers together, setting the final output to hazard if any of the classifiers indicates hazard. This reduces the probability of false negatives; however, the trade-off is an inevitably increase in the probability of false positives and an increase in the overall execution time required.

In this paper, we provide optimal algorithms for the scheduling of classifiers that minimize the probability of false positives, while meeting both a latency constraint and a constraint on the maximum acceptable probability of false negatives. The classifiers may have arbitrary statistical dependences between their functional behaviors (probabilities of correct identification of hazards), as well as variability in their execution times, characterized by typical and worst-case values.

CCS CONCEPTS

• **Computer systems organization** → **Real-time systems**; **Real-time systems**; • **Software and its engineering** → **Real-time schedulability**; **Real-time schedulability**; *Software reliability*.

RTNS 2023, June 07–08, 2023, Dortmund, Germany

ACM ISBN 978-1-4503-9983-8/23/06.
https://doi.org/10.1145/3575757.3593649

KEYWORDS

Real-Time, Classifiers, Optimal Ordering, DNN, arbitrary dependences

ACM Reference Format:
Tarek Abdelzaher, Sanjoy Baruah, Iain Bate, Alan Burns, Robert I. Davis, and Yigong Hu. 2023. Scheduling Classifiers for Real-Time Hazard Perception Considering Functional Uncertainty. In *The 31st International Conference on Real-Time Networks and Systems (RTNS 2023), June 07–08, 2023, Dortmund, Germany*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3575757.3593649

1 INTRODUCTION

The importance of obtaining assurance for safety-critical systems that incorporate machine learning has been recognized in several large-scale initiatives including: the Assured Autonomy Program [24] of the United States Defense Advanced Research Projects Agency (DARPA); the Assuring Autonomy International Programme [25], funded by Lloyd's of London; and the Bounded Behavior Assurance initiative [20], led by Northrop Grumman Corporation.

The research reported in this paper is motivated by the problem of real-time classification-based machine perception in a mobile autonomous system. This system must continually check that a designated area ahead is free of hazards. Such hazards must be identified within a specified time. In practice, classifiers are imperfect; they exhibit functional uncertainty. In the majority of cases, a high performance classifier will correctly determine whether there is a *hazard* or the area is *clear*. However, in other cases it may produce *false positives*, i.e. indicate hazard when the area is clear, or *false negatives*, i.e. indicate clear when there is in fact a hazard. The former are undesirable since they reduce quality of service, whereas the latter are a potential safety concern. A stringent constraint is therefore placed on the maximum permitted probability of false negatives. Since this requirement may not be achievable using a single classifier, one approach is to (logically) OR the outputs of multiple disparate classifiers together, setting the final output to hazard if any of the classifiers indicates hazard. This reduces the probability of false negatives; however, the trade-off is an inevitable increase in the probability of false positives and an increase in the overall execution time required.

Previous research in this area assumes that collections of classifiers designed to solve the same machine perception problem are *independent*; however, this is rarely the case in practice. Classifiers

can exhibit related behaviors for a variety of reasons. Statistical dependences[1] may be induced by the environment (a hazard 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 to train multiple classifiers), and by common components and algorithms (the same Deep Neural Network approach may be applied in a subset of the classifiers). These dependences result in observable behavior that is to some degree correlated.

In this paper, we provide a methodology for characterizing the arbitrary statistical dependences between the functional behaviors of different classifiers. Building on this characterization, we present a typical-case optimal algorithm for the scheduling of classifiers that minimizes the probability of false positives, while meeting both a latency constraint and a constraint on the maximum permitted probability of false negatives. The algorithm provides the optimal solution, i.e. the minimum probability of false positives, compliant with the constraints, assuming that the classifiers execute for their typical-case execution times, but crucially are not guaranteed to do so. As is the case in practice, each classifier is assumed to have a variable but bounded execution time, characterized by worst-case and typical-case values. (Full details of the system model, terminology, and definitions used are given in Section 2).

To support arbitrary statistical dependences, it is *necessary* to capture information relating to each possible combination of n classifiers that could be run. Since there are 2^n such distinct combinations, the methodology presented in this paper has the *minimum* space complexity of $O(2^n)$ needed to support such arbitrary dependences. All further operations used in both the initial profiling phase, described in Section 3, and in the off-line computations required by the typical-case optimal algorithm, presented in Section 4, are at most quadratic in the number of combinations and hence $O(4^n)$ overall[2]. In practice, we do not expect applications to require more than approximately 12 distinct classifiers to solve the same hazard detection problem, while the methodology presented is viable for up to $n = 20$ classifiers, equating to around 20 minutes of processing time on a single core of an Intel i5-8265U 1.6 GHz laptop computer. In summary, off-line processing with exponential time complexity is necessary to support the arbitrary dependences between classifier behaviors that exist in real systems, but does not prevent the methodology presented from being effective in practice.

The main contribution of the paper is the derivation of a typical-case optimal algorithm for the scheduling of classifiers to solve the hazard detection problem. This algorithm determines a preferred sequence of classifiers to run, along with a corresponding series of *trigger times*, derived from the typical-case execution times of the classifiers and the latency constraint, and *escape sets*, i.e. the subsets of classifiers to run if the preferred classifiers do not complete by the trigger times. The trigger times and escape sets are determined such that the constraint on the probability of false negatives and the constraint on the overall latency are guaranteed to be met. In other words, if the preferred classifiers complete by the trigger times, then the preferred sequence executes, otherwise an escape set is employed; either way the constraint on the probability of false negatives and the latency constraint will be met. This algorithm requires substantial off-line computation as discussed above, however, it then permits minimally dynamic run time operation with $O(1)$ overheads at each scheduling point, corresponding to the completion of a classifier. The performance of the typical-case optimal algorithm is evaluated, in Section 5, by comparison to that of a *statically* optimal algorithm and a hypothetical *clairvoyant* optimal algorithm, on a real-world case study. The paper concludes, in Section 6, with a summary and directions for future work.

[1]Two events X and Y are said to be statistically dependent if the probability of X occurring given that Y has occurred, i.e. $p(X|Y)$, is different from the separate probability of X occurring, i.e. $p(X)$. Two events are statistically *independent* if $p(X|Y) = p(X)$.
[2]This is achieved by using a binary representation for sets of classifiers, which enables operations such as set intersection, set union, and set difference to be achieved in $O(1)$ time at least up to $n = 64$ on a 64-bit computer.

1.1 Related Work

System safety is concerned with the identification and subsequent mitigation of potentially hazardous events [21]. The key point is that a system is expected to be acceptably, but not necessary completely, safe. Any event that could lead to a hazard should be mitigated, and even if a hazardous event occurs then this should not necessarily mean that a serious accident will happen. In this context, two types of hazardous event are: (i) not detecting an object that is in, or on a trajectory to be in, a potentially dangerous place [11]; or (ii) erroneously detecting such an object that is not in fact there. In the Uber accident [11] the pedestrian was repeatedly mis-classified and was only correctly classified when it was too late. The second type of hazardous event could lead to unnecessary avoidance actions creating other hazardous events or reduced trust in the system. Classical examples of where the wrong actions were taken based on incorrect information and key information not being readily available, or routinely ignored due to its unreliability, are the Bhopal [12] and Kegworth accidents [28]. Any detection of a potentially hazardous state therefore needs to be both trustworthy and timely.

Machine perception has a fundamental role in intelligent real-time systems, such as in drones [17], autonomous cars [10, 26], and medical IoT systems [4, 15]. Perception in such systems is typically 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 include: (i) adaptive neural network approximations aimed at meeting latency constraints [9, 14, 19, 31], and (ii) adaptive model-switching systems that pick one of multiple neural network versions depending on the time available [16].

Another direction is the use of "I Don't Know" (IDK) classifiers [18, 27]. Like model-switching, IDK classifier cascades [30] 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, which will then prompt the system to choose another classifier, thereby executing a situation-dependent sequence similar to adaptive approximation approaches. Analytical results have previously been developed [5, 8] 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. Further, extensions have also been made considering arbitrary dependences between IDK classifiers [1], with a focus on minimizing the expected duration required for successful classification. The research presented in this paper employs a similar

methodology for characterizing arbitrary dependences, but solves a substantially different problem.

In 2020, Agrawal et al. [3] proposed a model whereby Learning Enabled Components take a fixed time to execute, and return a *value* that is not known prior to execution, but for which worst-case and typical-case bounds are known. The model assumes multi-stage computations where there is a choice of components for each stage. The algorithms presented seek to determine a schedule that minimizes the overall latency, while ensuring that the sum of the values produced by the components that are executed is no less than a specified target value. By summing the values output by the components, this work implicitly assumes that the behaviors of the components are *independent*. Building upon the earlier work of Agrawal et al. [3], in 2022 Baruah et al. [7] considered a model whereby components take a fixed time to execute and produce an output indicating hazard or clear along with an associated *confidence* or probability[3] that the output is correct. This probability is assumed to be no less than a worst-case value under all circumstances, and no less than a typical-case value in non-pathological circumstances. The algorithms presented in [7] seek to schedule the components in sequence such that a target confidence is reached, with the overall confidence that is achieved calculated as a multiplicative function of the probabilities involved. In other words, assuming that the behaviors of the components are *independent*. Both static and *semi-adaptive* algorithms are presented. With the former, the sequence of components is predetermined prior to run time, whereas with the latter the sequence is adapted as information becomes available when each component completes. A similar distinction between such strategies was previously made in the context of graph routing problems [2, 6].

The problem addressed and the solutions provided in this paper build upon the work of Baruah et al. [7]. However, a more practical approach is taken in this paper: (i) Components (classifiers) are assumed to take a variable amount of time to execute up to some worst-case bound, rather than a fixed time; (ii) classifiers are not trusted to output an accurate measure of their own confidence (uncertainty) for each problem instance, since such measures cannot in general be relied upon to give a correct indication in individual cases [13], rather a long run frequency interpretation is used to determine the probabilities of false negatives and false positives based on a representative data set; and (iii) the methodology presented supports arbitrary statistical dependences between the behaviors of different classifiers, rather than implicitly assuming independence.

2 SYSTEM MODEL

We consider a collection of n classifiers $K_1, K_2, \ldots, K_n$ that are all designed to solve the same hazard identification problem. When invoked on an input, classifier K_i returns either 1 indicating *hazard* or 0 indicating *clear*. Each classifier is assumed to take a variable but bounded time to execute. If a classifier returns 1 and the ground truth is 0, then that is referred to as a *false positive*, similarly if a classifier returns 0 and the ground truth is 1, then that is referred to as a *false negative*. When more than one classifier is employed on the same problem, then we assume that the outputs of the classifiers are (logically) OR-ed together. Hence if *any* of the classifiers indicates 1, then the overall output is 1 indicating hazard. Only if *all* employed classifiers indicate 0, is the overall output 0 indicating clear.

We use S to denote a set or subset of classifiers. Since there are n classifiers, there are 2^n such distinct subsets, including the empty set $\emptyset$. We use the following functions to describe the characteristics of each set or subset of classifiers. The probability of the classifiers in S returning a false negative is denoted by $\mathrm{FN}(S)$. Similarly, the probability of the classifiers in S returning a false positive is denoted by $\mathrm{FP}(S)$. In Section 3, we describe how these probabilities can be obtained. The fact that the outputs of the employed classifiers are OR-ed together means that if set V is a subset of S, i.e. $V \subseteq S$, then $\mathrm{FN}(V) \geq \mathrm{FN}(S)$ and $\mathrm{FP}(V) \leq \mathrm{FP}(S)$. In other words, adding classifiers cannot increase the probability of false negatives, but typically decreases it; whereas adding classifiers cannot decrease the probability of false positives, but typically increases it.

The sum of the worst-case execution times of the classifiers in S is denoted by $\mathrm{WCET}(S)$. Similarly, $\mathrm{TCET}(S)$ denotes the sum of their typical-case execution times, and $\mathrm{ACET}(S)$ denotes the sum of their *actual* execution times for a specific run-time instance of the problem.

An ordered *sequence*[4] of classifiers $\langle K'_1, K'_2, K'_3, \ldots, K'_k \rangle$ can be represented by a set of the incrementally increasingly larger sets of classifiers employed, for example $\{\emptyset, S_1, S_2, \ldots S_k\}$ where $S_1 = \{K'_1\}$, $S_2 = \{K'_1, K'_2\}$, $S_3 = \{K'_1, K'_2, K'_3\}$ and so on. At each stage in the sequence, the classifier that has been added can be recovered as $\{K'_j\} = S_j - S_{j-1}$. (While this method of representing sequences is somewhat cumbersome, it has significant advantages in describing and understanding the algorithms proposed in Section 4).

The problem that we aim to solve is defined as follows:

Definition 1. ***Hazard detection classifier sequencing problem:*** *Given a* latency constraint *L, specifying the maximum time available for classifier execution, and a* constraint on false negatives *H, specifying the maximum permitted probability of false negatives, select a sequence of classifiers to run that is compliant with the constraints and minimizes the probability of false positives.*

A subset S of classifiers complies with the constraint on false negatives if $FN(S) \leq H$, otherwise it does not. We use $ESCAP(S)$ to denote the set of classifiers, referred to as the *escape set*, that provides the shortest guaranteed time to meet the constraint on false negatives following the completion of the classifiers in S. If $FN(S) \leq H$, then $ESCAP(S) = \emptyset$, since S already meets the constraint on false negatives. Otherwise, $ESCAP(S)$ equates to the subset V with the smallest value of $WCET(V)$ such that $S \cap V = \emptyset$ and $FN(S \cup V) \leq H$. In other words, there are no classifiers in V that are also in S, and once the classifiers in both S and then V have executed then the constraint on false negatives can be guaranteed.

The optimal classifier sequencing problem admits different classes of solution, from static solutions that are computed off-line to fully-dynamic solutions that may select a different classifier to run each time a classifier finishes and its actual execution time for that instance becomes known. In this paper, we are mainly interested in

[3] Baruah et al. [7] use the term *uncertainty* to mean $1 - p$, where p is the estimated probability that the output is correct.

[4] Note, we use K'_1 to represent the 1st classifier in the sequence, K'_2 the 2nd classifier in the sequence, and so on. This is different from the classifier indexing, and so it may be the case that, for example, $K'_1 = K_3$, $K'_2 = K_1$ etc.

static and minimally dynamic algorithms that pre-compute a course of actions that can be followed at run-time with $O(1)$ overheads at each scheduling point, corresponding to the completion of a classifier. For comparison purposes, we also consider the performance of a hypothetical clairvoyant algorithm that knows, before they are run, what the actual execution times of the classifiers will be for each run time instance of the problem, but does not know what their outputs will be. Note, we do not pursue the characterization of fully-dynamic solutions further in this paper, since the run-time overheads are likely prohibitive.

Definition 2. ***Static optimality:*** *An algorithm is statically optimal if the set of classifiers Z that it selects off-line has the minimum probability of false positives $FP(Z)$ of any set of classifiers that comply with the constraint on false negatives, $FN(Z) \leq H$, and has a combined worst-case execution time that complies with the latency constraint, $WCET(Z) \leq L$. The classifiers in Z may run in any order.*

Definition 3. ***Clairvoyant optimality:*** *An algorithm is clairvoyant optimal if the set of classifiers Z that it selects for each run-time instance of the problem has the minimum probability of false positives $FP(Z)$ of any set of classifiers that comply with the constraint on false negatives, $FN(Z) \leq H$, and has a total actual execution time for that run-time instance that complies with the latency constraint, $ACET(Z) \leq L$. The classifiers in Z may run in any order.*

Definition 4. ***Typical-Case optimality:*** *An algorithm is typical-case optimal if the* sequence *of classifiers that it selects, represented by the set of the incrementally increasingly larger sets of classifiers employed $\{S_0 = \emptyset, S_1, S_2, \ldots S_k = Z\}$, has the minimum probability of false positives $FP(Z)$ of any set of classifiers that comply with the constraint on false negatives, $FN(Z) \leq H$, have a combined typical-case execution time that complies with the latency constraint, $TCET(Z) \leq L$, and each subset of classifiers $S_j = S_0$ to S_k has a valid* escape set, *$ESCAP(S_j)$, such that if the classifier in $S_j - S_{j-1}$ completes in more than its typical-case execution time, but within its worst-case execution time, then executing the classifiers in $ESCAP(S_j)$ is* sufficient *to guarantee that the constraints will still be met.*

Note that if there is some slack time available, for example because a previous classifier in the sequence executed in less than its typical-case execution time, then it may not be *necessary* to immediately switch to executing the classifiers in $ESCAP(S_j)$ when the classifier in $(S_j - S_{j-1})$ exceeds its typical-case execution time. The typical-case optimal algorithm presented in Section 4 takes advantage of any available slack time in this way.

3 PROFILING

In this section, we describe the profiling phase, which prepares a *profile table* that captures the probabilistic dependences between the behaviors of the n classifiers. Prior to this profiling phase, we assume that each of the classifiers has been trained and verified using representative input data. In many applications this data can be re-used directly in the profiling phase. Where new data is required, for example because the training and verification data is proprietary, then it must also be representative of the inputs expected during deployment.

During the profiling phase, all of the n classifiers are tested 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 assumed that each sample also provides information relating to the ground truth, i.e hazard or clear. There are no limitations 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, a single sample could include a high definition image, a 1 second acoustic sound clip recorded at 4KHz, and a 1 second seismic time-series measurement 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 image information, but differ in the resolution used and consequently in their execution time.

For each of the N input samples, we store the ground truth and the output of each of the n classifiers for further processing as described below. We also measure and store the execution time of each classifier on each input sample. We assume that the execution times of the different classifiers are *independent* of one another. This is typically the case because the neural networks used for such processing run on a dedicated GPU. Further, each neural network typically performs 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. We explore the correlations between the behaviors of different classifiers, and also the correlations between their execution times in Section 3.3. Note, in the case study used as a proof-of-concept in this paper, the classifiers are run on one core of a multi-core system, a Raspberry Pi 4.

Once we have considered all N inputs samples for all n classifiers, then we can determine how many times each of the 2^n binary patterns of possible classifier outputs occurs: (i) when the ground truth is 1, and (ii) when the ground truth is 0. From this information, we can then compute the probability of obtaining a false positive, and also the probability of obtaining a false negative, when employing each of the 2^n combinations of the n classifiers. Since the input data used in profiling is assumed to be representative of the input data when the system is operational, these probabilities are a correct reflection of the long run frequencies of occurrence of false positive and false negatives respectively for those sets of classifiers.

The method of computing the probabilities of false positives and false negatives is best illustrated via an example, as described below.

3.1 Profiling the Multi-Modal Case Study

The data used in this case study was collected previously [22] as part of a project that seeks to autonomously detect the presence of a potentially hostile enemy vehicle 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). The manner in which the input samples were collected is described by Liu et al. [22] 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."

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 generate an alert only when a specific type of target is present, while ignoring other passing traffic, hence the task is one of *binary* (i.e. combined) detection and classification.

There are seven classifiers of interest[5] in the case study:

- A: deepsense_both: Uses both seismic and acoustic data, and processes it using the DeepSense neural network architecture [32].
- B: deepsense_both_contras: Similar to A, but was trained using contrastive learning [23].
- C: deepsense_acoustic: Uses only acoustic data, and processes it using the DeepSense neural network architecture [32].
- D: deepsense_seismic: Similar to C, but uses only seismic data.
- E: cnn_both: Uses both seismic and acoustic data, and processes it using a standard convolutional neural network.
- F: cnn_acoustic: Uses only acoustic data, and processes it using a standard convolutional neural network.
- G: cnn_seismic: Similar to F, but uses only seismic data.

To illustrate the methodology and provide a sufficiently compact worked example, we first consider only the five classifiers A to E. The evaluation in Section 5 later includes all seven classifiers.

The output of classifiers A to E on 1800 randomly chosen input data samples[6] is summarized in Table 1. The first column shows a binary code identifying a subset of the classifiers, which is then enumerated in the second column. The fourth column, labeled $GT0$, indicates the number of times the binary code in the first column was obtained by running *all* of the five classifiers when the ground truth was 0, i.e. clear. Similarly, the third column, labeled $GT1$, indicates the number of times that the binary code in the first column was obtained by running *all* of the five classifiers when the ground truth was 1, i.e. hazard. For example, the first row in the table shows that there were 1107 input samples where none of the classifiers indicated hazard, when the ground truth was 0, i.e. clear. Similarly, there were 36 input samples where none of the classifiers indicated hazard, even though the ground truth was 1, i.e. hazard. Further, there was 1 input sample where *only* classifiers A and E (10001) indicated 1 and the ground truth was 0, and 3 input samples where *only* those classifiers indicated 1 and the ground truth was 1.

Once this summary of the classifier behavior on the input samples has been obtained, then the next step in profiling is to determine the probability of false positives and the probability of false negatives for each of the 2^n subsets of the n classifiers. (Since there are five classifiers in this initial example, there are 32 combinations of the classifiers and hence 32 rows in the profiling table). Recall that when multiple classifiers are employed, the overall output is obtained by (logically) OR-ing together the individual outputs. Hence any classifier indicating hazard results in an overall output of hazard, whereas all employed classifiers must indicate clear for the overall output to be clear.

Table 1: Profile table for the Multi-Modal case study

Binary	Classifiers S	$GT1$	$GT0$	$FP(S)$	$FN(S)$	$WCET(S)$
00000	∅	36	1107	0.0000	1.0000	0
00001	A	3	2	0.0075	0.1183	0.025121
00010	B	1	1	0.0042	0.1383	0.023854
00011	AB	0	0	0.0100	0.0933	0.048975
00100	C	3	18	0.0200	0.3600	0.017554
00101	AC	9	1	0.0250	0.0933	0.042675
00110	BC	2	0	0.0242	0.1067	0.041408
00111	ABC	5	0	0.0275	0.0850	0.066529
01000	D	9	36	0.0358	0.2083	0.01618
01001	AD	4	1	0.0417	0.0883	0.0413
01010	BD	1	1	0.0383	0.1067	0.040033
01011	ABD	22	1	0.0433	0.0750	0.065154
01100	CD	0	2	0.0542	0.0850	0.033734
01101	ACD	1	0	0.0575	0.0700	0.058854
01110	BCD	2	0	0.0567	0.0767	0.057587
01111	ABCD	13	0	0.0592	0.0667	0.082708
10000	E	4	22	0.0250	0.1850	0.0053
10001	AE	3	1	0.0292	0.0900	0.030421
10010	BE	1	1	0.0275	0.1083	0.029154
10011	ABE	3	1	0.0308	0.0800	0.054274
10100	CE	2	1	0.0425	0.1267	0.022854
10101	ACE	4	2	0.0458	0.0783	0.047975
10110	BCE	4	0	0.0450	0.0867	0.046708
10111	ABCE	45	0	0.0475	0.0750	0.071828
11000	DE	2	2	0.0592	0.0983	0.021479
11001	ADE	3	0	0.0617	0.0700	0.0466
11010	BDE	2	0	0.0600	0.0850	0.045333
11011	ABDE	122	0	0.0625	0.0650	0.070454
11100	CDE	0	0	0.0750	0.0667	0.039033
11101	ACDE	0	0	0.0767	0.0617	0.064154
11110	BCDE	2	0	0.0758	0.0650	0.062887
11111	ABCDE	292	0	0.0775	0.0600	0.088008
	Sum	600	1200			

To compute the probability of false positives for a given subset S of classifiers, we sum up the values in the $GT0$ column for all of the subsets in the table that have a *non-empty* intersection with S and then divide by the sum of all of the values in the $GT0$ column. This totals up the number of input samples where at least one of the classifiers in S indicates 1 when the ground truth is in fact 0 and divides it by the total number of input samples where the ground truth is 0. subsets of classifiers in the table. For example, consider $S = \{A, E\}$, the total number of input samples in the $GT0$ column where A or E appear in the set of classifiers is 35. As there are 1200 input samples with a ground truth of 0, it follows that $FP(\{A, E\}) = 35/1200 = 0.0292$.

To compute the probability of false negatives for a given subset S of classifiers, we sum up the values in the $GT1$ column for all of the subsets in the table that have an *empty* intersection with S and then divide by the sum of all the values in the $GT1$ column. This totals up the number of input samples where none of the classifiers in S indicate 1 when the ground truth is in fact 1 and divides it

[5]We did not consider the vision based YOLOv5 classifiers, since their execution times were orders of magnitude greater than those of the other classifiers.

[6]An input data sample comprises an image, a 1 second acoustic sound clip recorded at 4KHz, and a 1 second seismic time-series measurement recorded at 100Hz.

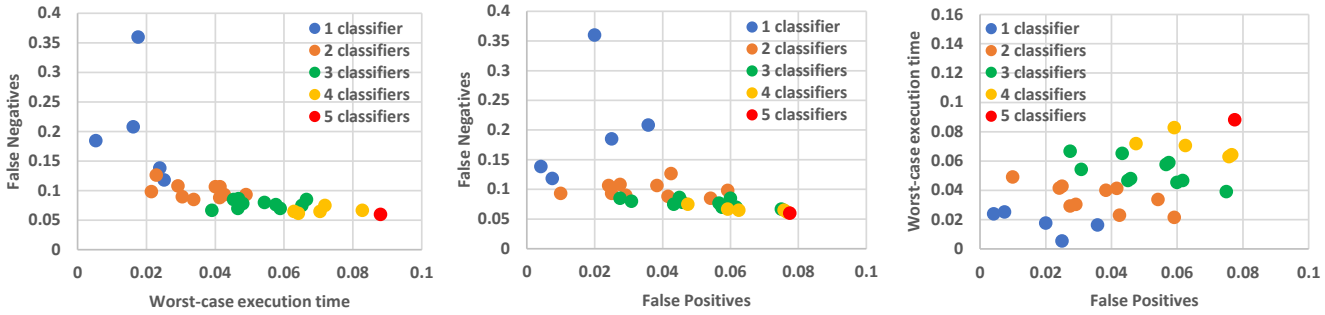

Figure 1: Multi-Modal case study: Relationships between $FN(S)$, $FP(S)$, and $WCET(S)$.

by the total number of input samples where the ground truth is 1. For example, consider $S = \{A, E\}$, the total number of inputs samples in the $GT1$ column where neither A nor E appear in the set of classifiers is 54. As there are 600 input samples with a ground truth of 1, it follows that $FN(\{A, E\}) = 54/600 = 0.09$.

Also shown in Table 1 is the total worst-case execution time, $WCET(S)$, measured in seconds for the classifiers in S. The worst-case execution times for each classifier was set to the 99-percentile of the values obtained via profiling on a Raspberry Pi 4[7].

The relationships between $FN(S)$, $FP(S)$, and $WCET(S)$ for the subsets S of classifiers listed in the profile table are illustrated in Figure 1. Observe that lowering the probability of false negatives typically requires employing more classifiers, which in turn increases both the worst-case execution time required and the probability of false positives. For example, employing just one classifier, the lowest probability of false negatives that can be achieved is 0.1183 with classifier A, with a worst-case execution time of 0.0251 and a probability of false positives of 0.0075. Using two classifiers, the probability of false negatives can be reduced to 0.085 with classifiers CD, with $WCET(\{C, D\}) = 0.0337$ and $FP(\{C, D\}) = 0.0542$. Further, with three classifiers the probability of false negatives can be reduced to 0.0667 with classifiers CDE, with $WCET(\{C, D, E\}) = 0.0390$ and $FP(\{C, D, E\}) = 0.0750$. With four classifiers the probability of false negatives can be reduced to 0.0617 with classifiers $ACDE$, with $WCET(\{A, C, D, E\}) = 0.0642$ and $FP(\{A, C, D, E\}) = 0.0767$. Finally, with five classifiers $ABCDE$, the minimum probability of false negatives of 0.06 is reached, with $WCET(\{A, B, C, D, E\}) = 0.088$ and $FP(\{A, B, C, D, E\}) = 0.0775$.

3.2 Static and Clairvoyant algorithms

Once the profiling table has been constructed, and populated with the probabilities of false positives and false negatives as described above, then it becomes trivially simple to formulate a statically optimal algorithm (see Definition 2).

Statically Optimal Algorithm: Select the set of classifiers S from the 2^n entries in the profiling table that has the minimum value of $FP(S)$ such that $FN(S) \leq H$ and $WCET(S) \leq L$. The classifiers in S may then be run in any order.

[7]If a higher reliability estimate was required, then such values could be obtained via static or measurement-based timing analysis. Potential overruns could also be dealt with via budget enforcement and an assumption that hazard is returned in the rare cases that a classifier exceeds its designated WCET.

Similarly, assuming that a clairvoyant algorithm also has access to a further column of information indicating the total actual execution time $ACET(S)$ for each subset of classifiers S the next time that they run, then a clairvoyant optimal algorithm (see Definition 3) is also trivially simple to formulate.

Clairvoyant Optimal Algorithm: Select the set of classifiers S from the 2^n entries in the profiling table that has the minimum value of $FP(S)$ such that $FN(S) \leq H$ and $ACET(S) \leq L$. The classifiers in S may then be run in any order.

In Section 4, we present a minimally dynamic algorithm that is typical-case optimal (see Definition 4). The statically optimal algorithm and the clairvoyant optimal algorithm are used as reference points for assessing the quality of the typical-case optimal algorithm in Section 5. These algorithms provide, respectively, bounds on the best possible solutions that can be achieved statically and dynamically. First, however, we discuss correlations between both the behaviors and the execution times of the classifiers.

3.3 Correlations

With the data that is available from profiling, it is possible to estimate the level of statistical dependence, i.e. the degree of correlation, between the behaviors of the different classifiers. This can be characterized by calculating Pearson's correlation coefficient[8] for each pair of classifiers. This coefficient r_{xy} is given by:

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

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.

Pearson's correlation coefficient r_{xy} can take values in the range $[-1, +1]$, with $r_{xy} = 0$ implying no correlation, and hence possibly independence.[9] The value $r_{xy} = +1$ implies identical behavior, and at the other extreme $r_{xy} = -1$ implies exactly opposite behavior.

Table 2 shows the coefficients computed for all seven classifiers in Multi-Modal case study, color-coded by the degree of correlation between the outputs of the distinct classifiers: red indicating a

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

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

Table 2: Behavior: Pearson Correlation Coefficients

	A	B	C	D	E	F	G
A	1	0.931	0.725	0.815	0.860	0.717	0.687
B	0.931	1	0.721	0.832	0.872	0.723	0.6944
C	0.725	0.721	1	0.570	0.687	0.819	0.440
D	0.815	0.832	0.570	1	0.747	0.579	0.699
E	0.860	0.872	0.687	0.747	1	0.711	0.717
F	0.717	0.723	0.819	0.579	0.711	1	0.433
G	0.687	0.694	0.440	0.699	0.717	0.433	1

strong degree of correlation ($abs(r_{xy}) > 0.5$), orange a moderate degree of correlation ($0.1 < abs(r_{xy}) \leq 0.5$), and green a weak degree of correlation ($abs(r_{xy}) \leq 0.1$). As expected, the outputs of the classifiers in the case study mostly show a strong positive correlation of between 0.433 and 0.909 for each pair, with 95% confidence intervals[10] for these correlation coefficients of $[0.39, 0.47]$ and $[0.90, 0.92]$ respectively.

We also examined the dependences 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. Table 3 shows these coefficients for all seven classifiers in the Multi-Modal case study.

Table 3: Execution Times: Pearson Correlation Coefficients

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

Observe that for the execution times of the Multi-Modal classifiers, the correlation coefficients in Table 3 for all pairs of distinct classifiers indicate weak correlation ($abs(r_{xy}) \leq 0.1$). The weak degree of correlation 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 10% that is dependent. Hence regarding the execution time behavior of the classifiers as independent is a reasonable approximation. In Table 3, the correlation coefficients range from −0.058 to 0.076; the 95% confidence intervals[10] for these coefficients are $[-0.104, -0.0118]$ and $[0.030, 0.122]$ respectively. (Note, the widest confidence interval of $[-0.046, 0.046]$ occurs when the correlation coefficient is 0.0).

Figure 2 illustrates the frequency distributions of the execution times of the classifiers. Observe that classifiers A and B, which use the DeepSense neural network architecture and operate on both

[10]Computed using https://www.statskingdom.com/correlation-confidence-interval-calculator.html

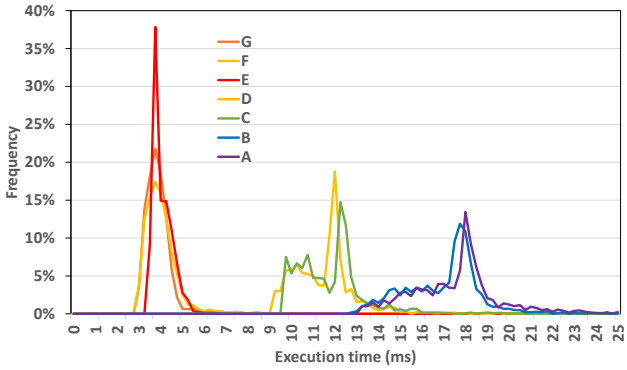

Figure 2: Frequency distribution of execution times.

seismic and acoustic data, have similar execution time distributions. This is also the case with classifiers C and D, which also use the DeepSense neural network architecture, but each operate on only a single form of input data. Finally, classifiers E, F, and G all use a standard convolutional neural network and have similar execution time distributions.

4 TYPICAL-CASE OPTIMAL ALGORITHM

In this section, we present a typical-case optimal algorithm (see Definition 4) for the hazard detection classifier sequencing problem. This algorithm has a substantial off-line component that then permits minimally dynamic run time operation with $O(1)$ overheads at each scheduling point. By construction, this typical-case optimal algorithm is guaranteed to find a feasible solution if and only if a static solution exists. We therefore assume that the typical-case optimal algorithm is only run on problems that admit a static solution.

The aim of the algorithm is to determine a *preferred sequence* of classifiers to run, along with a corresponding series of *trigger times* (derived from the typical-case execution times of the classifiers) and *escape sets*, i.e. subsets of classifiers to run if the preferred classifiers do not complete by the trigger times. The trigger times and escape sets are determined such that the constraint on false negatives and the constraint on overall latency are always guaranteed to be met. In other words, if the preferred classifiers complete by the trigger times, then the preferred sequence executes, otherwise the classifiers in an escape set are executed, either way the constraint on the probability of false negatives and the latency constraint will be met. The difference being that the preferred sequence will result in a lower probability of false positives.

We assume as input to the algorithm the *profile table* described in Section 3, populated with the $FP(S)$, $FN(S)$, and $WCET(S)$ values for each of the distinct subsets of classifiers.

As a prelude to the main off-line operation of the typical-case algorithm, values for the typical-case execution times $TCET(S)$ and the escape set $ESCAP(S)$ are added to the profile table for each subset of classifiers S. Table 4 extends Table 1, adding these values for the five classifiers A to E considered.

First, typical-case execution times are defined for each classifier. This is achieved by selecting an appropriate percentile from the

Table 4: Extended profile table for the Multi-Modal case study

Binary	Classifiers S	$FP(S)$	$FN(S)$	$WCET(S)$	$TCET(S)$	$ESCAP(S)$
00000	∅	0.0000	1.0000	0	0	CD
00001	A	0.0075	0.1183	0.025121	0.018166	DE
00010	B	0.0042	0.1383	0.023854	0.017788	DE
00011	AB	0.0100	0.0933	0.048975	0.035954	E
00100	C	0.0200	0.3600	0.017554	0.012263	D
00101	AC	0.0250	0.0933	0.042675	0.030429	E
00110	BC	0.0242	0.1067	0.041408	0.030051	D
00111	ABC	0.0275	0.0850	0.066529	0.048217	∅
01000	D	0.0358	0.2083	0.01618	0.011878	C
01001	AD	0.0417	0.0883	0.0413	0.030044	E
01010	BD	0.0383	0.1067	0.040033	0.029666	E
01011	ABD	0.0433	0.0750	0.065154	0.047832	∅
01100	CD	0.0542	0.0850	0.033734	0.024141	∅
01101	ACD	0.0575	0.0700	0.058854	0.042307	∅
01110	BCD	0.0567	0.0767	0.057587	0.041929	∅
01111	ABCD	0.0592	0.0667	0.082708	0.060095	∅
10000	E	0.0250	0.1850	0.0053	0.004112	CD
10001	AE	0.0292	0.0900	0.030421	0.022277	D
10010	BE	0.0275	0.1083	0.029154	0.0219	D
10011	ABE	0.0308	0.0800	0.054274	0.040066	∅
10100	CE	0.0425	0.1267	0.022854	0.016374	D
10101	ACE	0.0458	0.0783	0.047975	0.03454	∅
10110	BCE	0.0450	0.0867	0.046708	0.034162	D
10111	ABCE	0.0475	0.0750	0.071828	0.052328	∅
11000	DE	0.0592	0.0983	0.021479	0.01599	C
11001	ADE	0.0617	0.0700	0.0466	0.034156	∅
11010	BDE	0.0600	0.0850	0.045333	0.033778	∅
11011	ABDE	0.0625	0.0650	0.070454	0.051944	∅
11100	CDE	0.0750	0.0667	0.039033	0.028252	∅
11101	ACDE	0.0767	0.0617	0.064154	0.046418	∅
11110	BCDE	0.0758	0.0650	0.062887	0.046041	∅
11111	ABCDE	0.0775	0.0600	0.088008	0.064206	∅

execution time distribution obtained from running the classifiers on the N representative input samples during the profiling phase. Initially for the purposes of an illustrative worked example, we will assume that the 70-percentile value is used. We return to the selection of an appropriate percentile for the typical-case execution times in the evaluation in Section 5). The $TCET(S)$ values are then set to the sum of the typical-case execution times of the individual classifiers in S. This is a valid estimate of the typical-case execution times of each subset S, since the execution times of individual classifiers have a very weak degree of correlation, as shown in Section 3.3, and can therefore be modeled as independent. In any case correct operation of the algorithm does not rely on the precise values chosen for the typical-case execution times, since compliance with the constraints is guaranteed irrespective of the actual execution times realized at run time, provided that they do not exceed the worst-case execution times that were previously determined.

Second, the *escape set* $ESCAP(S)$ is computed for each subset of classifiers S. Recall that running the classifiers in the set $ESCAP(S)$ provides the shortest guaranteed time to meet the constraint on false negatives following the completion of the classifiers in S. $ESCAP(S)$ is determined by finding the subset V in the profile table that has the smallest value of $WCET(V)$ of those subsets where $S \cap V = \emptyset$ and $FN(S \cup V) \leq H$. (If $FN(S) \leq H$, then $ESCAP(S) = \emptyset$, since S already meets the constraint on false negatives). Since there are 2^n subsets S in the profiling table, computing the 2^n $ESCAP(S)$ values takes $O(4^n)$ time.

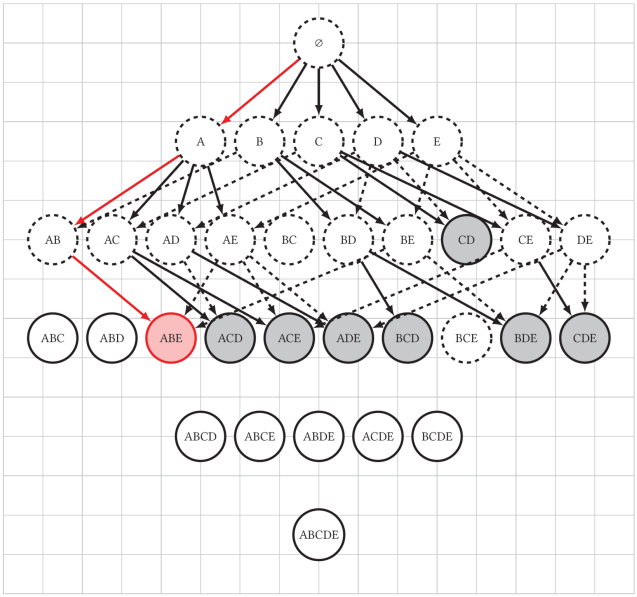

Figure 3: DAG representation.

A Directed Acyclic Graph (DAG) representation is used to derive a typical-case optimal solution. Figure 3 illustrates the DAG for the five classifiers A to E from the Multi-Modal case study, with a constraint on the probability of false negatives of $H = 0.085$, a latency constraint of $L = 0.05$ (i.e 50ms), and typical-case execution times assumed to be given by the 70-percentiles. The meaning of the dashed and solid lines and the color-coding used in the figure is explained below.

Each vertex in the DAG corresponds to a unique subset S of the classifiers, hence there are 2^n vertices, starting with the empty set ∅. The vertices are connected via directed edges. A directed edge connects each vertex representing a subset of classifiers with a vertex that represents the same subset extended via the addition of exactly one further classifier.

Any vertex corresponding to a subset S that complies with the constraint on false negatives, i.e. $FN(S) \leq H$, has $ESCAP(S) = \emptyset$. Such vertices cannot be improved upon by running further classifiers, since to do so cannot decrease either the overall execution time required or the probability of false positives. Such vertices therefore have no outgoing edges and are referred to as *exit* vertices. Exit vertices are indicated in Figure 3 via a solid (rather than dashed) boundary. While exit vertices represent potential solutions that meet the constraint on false negatives, not all such vertices are reachable due to the constraint on latency.

Along an edge, let P be the set of classifiers for the previous vertex and Q the set of classifiers for the subsequent vertex. Hence, an edge represents the addition of the single classifier in the set $Q - P$. An edge is only *valid*, i.e. can form part of a solution that is feasible when classifiers take their typical-case execution times, if $TCET(P) + WCET(Q - P) + WCET(ESCAP(Q)) \leq L$. In other words, the typical-case execution time for all completed classifiers plus the worst-case execution time of the classifier to run next

plus the worst-case execution time of the escape set after running that classifier, must be able to be completed within the latency constraint. All *invalid* edges are removed, since they cannot form part of a solution that is guaranteed not to fail. For example, on the left hand side of Figure 3 the edges between AB and ABC and between AB and ABD have been removed for this reason. As have the edges between BCE and $BCDE$ and between BCE and $ABCE$.

The *slack time* associated with an edge is given by $L-(TCET(P)+WCET(Q-P)+WCET(ESCAP(Q))$. When there are two or more valid incoming edges to a vertex Q, then this implies that there are multiple sub-sequences, i.e. permutations of the classifiers in the set Q, that could be utilized as part of a feasible solution that includes that vertex. We need only retain one such possibility, and therefore choose, without compromising solution optimality, to consider only the incoming edge with the maximum slack. Such edges are illustrated in Figure 3 by solid arrows, with other valid edges shown as dashed arrows.

The optimal solution is determined by choosing the vertex Z that has the minimum probability of false positives, $FP(Z)$, from all of the *exit vertices* Q corresponding to *feasible solutions*, i.e. that have $FN(Q) \leq H$ and are reachable via valid edges from the start vertex. For example, in Figure 3 the vertices corresponding to feasible solutions have a shaded background. They are CD, ABE, ACD, ACE, ADE, BCD, BDE, and CDE. Of these vertices, subset ABE, highlighted in red, has the smallest probability of false positives and so is the optimal subset Z.

The order in which the classifiers in Z should be run is recovered by tracing back the preferred incoming edges starting with vertex Z. Let $Z_0, Z_1, \ldots, Z_k$ be the k vertices (sub-sets) in sequence where Z_0 is the start vertex and $Z_k = Z$ is the optimal subset and an exit vertex. For example, in Figure 3, the typical-case optimal sequence of classifiers to run is ABE, as indicated by the red arrows.

To facilitate the best use of any available slack at run time, we set the trigger points to be as late as possible, while still ensuring that the escape sets can be completed in time if necessary. Hence $\forall i = 1 \ldots k\, TRIG(Z_i) = L-WCET(Z_i-Z_{i-1})-WCET(ESCAP(Z_i))$. Hence, the solution in the format for use in run time scheduling consists of $i = 1 \ldots k$ triplets of the form:

$(Z_i - Z_{i-1}, TRIG(Z_i), ESCAP(Z_{i-1}))$.

Each triplet indicates: (i) the preferred classifier $K'_i = Z_i - Z_{i-1}$ to execute next; (ii) the latest permitted start time $TRIG(Z_i)$ for that preferred classifier, sufficient to ensure that if it takes its worst-case execution time, then a feasible solution can still be guaranteed via the subsequent execution of $ESCAP(Z_i)$ in at most time $WCET(ESCAP((Z_i))$; and (iii) the escape set $ESCAP(Z_{i-1})$ to execute if it is too late to start the preferred classifier. By construction, provided that all classifiers comply with their worst-case execution times, then this scheduling point cannot be so late that escape set $ESCAP(Z_{i-1})$ is unable to complete within the latency constraint.

Considering the five classifiers A to E, the optimal solution is: ((A, 0.0034, DE), (B, 0.02085, DE), (E, 0.0447, E)). Meaning that if before any classifier runs, the current time t is no larger than 0.0034 then preferred classifier A should run. (Since $t = 0$ to begin with then this is always true). If classifier A completes before $t = 0.02085$, then preferred classifier B should run, otherwise employing escape set DE is guaranteed to meet the constraints, since $FN(ADE) = 0.07$ and $0.0034 + WCET(\{A\}) + WCET(\{D, E\}) = 0.05 = L$. If classifier B completes before $t = 0.0477$, then preferred classifier E should run, otherwise employing escape set E is guaranteed to meet the constraints, since $FN(ABE) = 0.08$ and $0.02085 + WCET(\{B\} + WCET(\{E\})) = 0.05 = L$. Note, in this example in the final stage, E is both the preferred classifier and also provides the escape set, since it is the only remaining classifier that can be guaranteed to meet the constraints in time. In general, however, this is not necessarily the case, a different escape set could be needed.

Observe that the typical-case optimal algorithm, by considering all possible paths through the DAG, covers all possible permutations of the classifiers. However, because the information about each vertex S (i.e. $FN(S)$, $FP(S)$, $WCET(S)$, $ESCAP(S)$, $TCET(S)$) only depends on the set of classifiers in S and not on the order in which they are run, then the complexity of the DAG-based approach is exponential in n, rather than factorial in n as would be the case if every permutation were actually considered separately.

The overall complexity of the off-line part of the typical-case optimal algorithm is $O(4^n)$, dominated by the construction of the extended profile table. Once that table has been populated, the DAG-based component of the algorithm has $O(n2^n)$ complexity, since there are 2^n vertices and at most n outgoing edges per vertex.

It is interesting to compare the statically optimal, clairvoyant optimal, and typical-case optimal solutions for the Multi-Modal case study, with the constraints set as described previously, and assuming in the clairvoyant case that $ACET(S) = TCET(S)$. The three solutions are as follows:

Static: ACE (any order):
$FP(S) = 0.0458$, $WCET(S) = 0.047975$, $TCET(S) = 0.03454$.
Typical: ABE (specific order):
$FP(S) = 0.0308$, $WCET(S) = 0.054274$, $TCET(S) = 0.040066$.
Clairvoyant: ABC (any order):
$FP(S) = 0.0275$, $WCET(S) = 0.066529$, $TCET(S) = 0.048217$.

Observe that the overall typical-case execution time of the clairvoyant solution fits within the latency constraint of 0.05; however, this does *not* mean that this solution is typical-case optimal, since it cannot be guaranteed not to fail. This can be seen by observing that if the first two classifiers run in their typical-case execution times, there is insufficient time left to guarantee that the final classifier can be completed within the latency constraint of 0.05 if it takes its worst-case execution time:

$TCET(\{A, B\}) + WCET(\{C\}) = 0.0535081$
$TCET(\{A, C\}) + WCET(\{B\}) = 0.0542823$
$TCET(\{B, C\}) + WCET(\{A\}) = 0.0551716$

By contrast, the typical-case optimal algorithm provides the best solution, i.e. with the lowest probability of false positives, when the classifiers take their typical-case execution times, *without permitting the possibility of failure if their execution times exceed those values.*

The hypothetical clairvoyant algorithm dominates the static and typical-case algorithms. However, there are scheduling anomalies that mean there is no clear dominance between the typical-case and static algorithms when actual execution times are considered. It is possible that for some actual-case execution times the solution chosen by the typical-case algorithm will result in a higher probability of false positives than the solution chosen by the static algorithm. This happens when the typical-case algorithm chooses a classifier to run first that is not present in the solution given by

the static algorithm. If this and all further classifiers require their worst-case execution times, then although a feasible solution is still guaranteed, it may result in a higher probability of false positives than running the set of classifiers chosen by the static algorithm.

5 EVALUATION

In this section, we evaluate the performance of the typical-case optimal algorithm compared to the hypothetical clairvoyant algorithm and also the static algorithm. The running example in the previous section considered specific values for the constraint on the probability of false negatives and the latency constraint, here we provide a systematic evaluation that examines the performance of the three algorithms for a range of different values of these constraints.

The evaluation is based on the Multi-Modal case study introduced in Section 3, considering classifiers A to G. This necessitates a profile table with 128 rows, covering all 2^7 distinct combinations of these seven classifiers. Due to the space that it would require, we do not reproduce this table here. The basic characteristics of the two additional classifiers F and G are as follows: $FP(\{F\}) = 0.004167$, $FN(\{F\}) = 0.42$, $WCET(\{F\}) = 0.00777$ and $FP(\{G\}) = 0.0675$, $FN(\{G\}) = 0.3167$, $WCET(\{G\}) = 0.0058$.

The experiment involved 1000 runs. In each run:

- The latency constraint was randomly selected in the range $[0.03333, 0.06667]$, i.e. 33 to 67ms, which is typically achievable using three classifiers.
- The constraint on the maximum probability of false negatives was randomly selected in the range $[0.06667, 0.08333]$, which is again typically achievable using three classifiers[11].
- Only pairs of constraints on latency and the probability of false negatives that admitted a static solution were used, otherwise further random constraints were generated.
- The set of actual execution times for the classifiers was selected at random from the sets of execution time values obtained for the 1800 input samples used during profiling.

On each run, we computed the solution produced by the static and the clairvoyant algorithms, and also that produced by the typical-case optimal algorithm assuming typical-case execution times equating to the 25−, 50−, and 75−percentiles. For the typical-case solutions, we simulated the schedule of classifiers obtained when the classifiers took their assigned actual-case execution times.

Figure 4 shows the Cumulative Distribution Function of the probability of false positives, computed for the 1000 runs, using the set of classifiers that were selected in each case. The smallest probability of false positives achieved was 0.0308333 for *ABE* and the largest was 0.104167 for *DEFG*. Figure 4 shows that the typical-case optimal algorithm results in performance that is on average significantly better than that of the statically optimal algorithm, roughly halving the gap to the hypothetical clairvoyant algorithm.

Out of the 1000 runs, the static algorithm outperformed the typical-case algorithm on 33, 41, and 94 occasions, i.e. just 3.3%, 4.1%, and 9.4% of the time, when the latter algorithm used the 75−, 50−, and 25−percentiles respectively for the typical-case execution times. The average probability of false positives across the experiment was

[11] No single classifier or pair of classifiers can achieve a value for $FN(S)$ in this range. The best that can be achieved using two classifiers is $FN(S) = 0.085$, using three classifiers, $FN(S) = 0.06667$, and using four classifiers, $FN(S) = 0.06$.

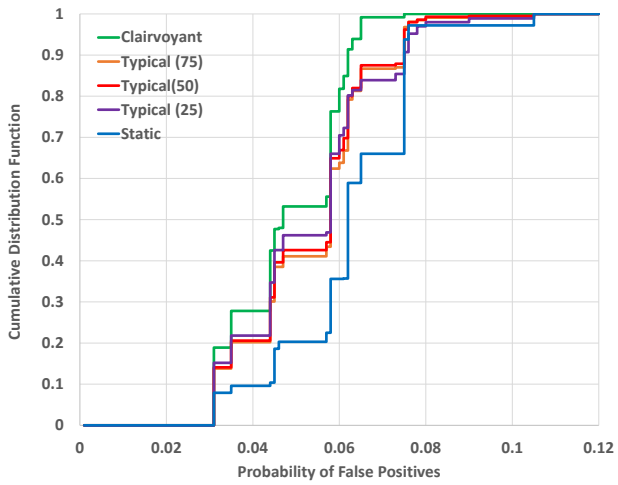

Figure 4: CDF: Multi-Modal case study, 7 classifiers.

0.062 for the static algorithm, 0.048 for the clairvoyant algorithm, and 0.052 for all three variants of the typical-case optimal algorithm. Again, illustrating the gains that can be made by employing a minimally dynamic solution rather than a static one.

The performance of the typical-case optimal algorithm was not especially sensitive to the values chosen for the typical-case execution times between the upper and lower quartiles. Choosing different percentiles (25%, 50%, or 75%) for the typical-case execution times resulted in small differences in performance for different constraint settings, resulting in the lines crossing in Figure 4. Further optimization in specific cases could be achieved by exploring a limited number of different percentiles for the typical-case execution times, and making a pragmatic choice of the one that provides the best overall performance in that case.

It is interesting to note that if all seven classifiers were employed, then the probability of false positives would be $144/1200 = 0.12$ and the probability of false negatives $32/600 = 0.053333$. If instead of (logically) OR-ing together the classifier outputs, we required a minimum of two classifiers to agree on a hazard designation, then in this case the probability of false negatives would increase to $49/600 = 0.081667$, while the probability of false positives would decrease to $41/1200 = 0.034167$. However, this configuration is outperformed by requiring only one classifier to indicate hazard and using only classifiers *ABE*, for a probability of false negatives of 0.08 and a probability of false positives of 0.0308. Nevertheless, considering how the outputs of different classifiers should be combined to balance requirements on the probabilities of both false negatives and false positives is an interesting avenue for future work.

Relaxing the latency constraint, i.e. $L = \infty$, the clairvoyant optimal, typical-case optimal, and statically optimal algorithms all choose the same solution for any given constraint on the probability of false negatives. Figure 5 illustrates the Pareto front characterizing this trade-off between the solution with the minimum probability of false positives and the constraint on the probability of false negatives, for the Multi-Modal case study with 7 classifiers. Observe that,

as expected, the tighter the constraint, the larger the probability of false positives.

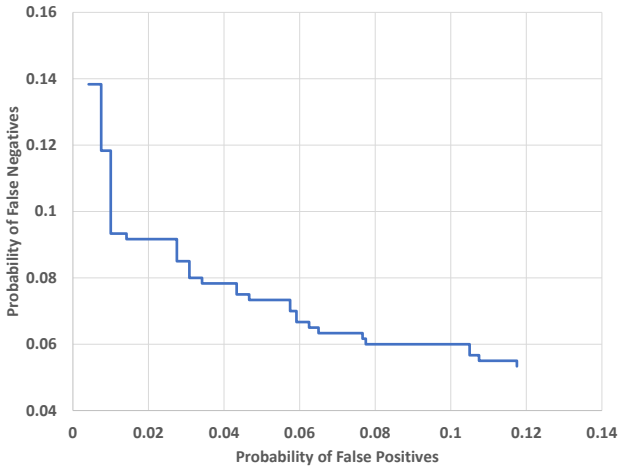

Figure 5: Pareto Front: Multi-Modal case study, 7 classifiers.

By contrast, relaxing the constraint on the probability of false negatives, i.e. $H = 0.999$, results in very few different solutions when the latency constraint is varied. In the Multi-Modal case study with 7 classifiers, the optimal static solution for latency constraints in the range $[0.0053, 0.00777]$ is to run classifier E, with $FP(\{E\}) = 0.025$. For all larger latency constraints, the optimal solution is to run classifier F, with $FP(\{F\}) = 0.004167$. The reason there are so few different solutions is that the relaxed constraint on the probability of false negatives can be met by using a single classifier, while adding further classifiers only increases the probability of false positives. Hence, finding the optimal solution with a relaxed constraint on the probability of false negatives effectively reduces to choosing the *single* classifier that minimizes the probability of false positives while also meeting the latency constraint.

6 CONCLUSIONS

The research described in this paper addressed a problem of real-time classification-based machine perception, specifically the hazard detection classifier sequencing problem (see Definition 1). The main contribution was in the derivation of optimal algorithms for the scheduling of classifiers that minimize the probability of false positives, while meeting both a latency constraint and a constraint on the maximum acceptable probability of false negatives (i.e. hazards not detected). The classifiers may have arbitrary statistical dependences between their functional behaviors (i.e. probabilities of correct detection of hazards), as well as variability in their execution times. The solutions proposed were both applicable to real-world scenarios and practical, with $O(1)$ run-time overheads. The effectiveness of the approach was illustrated via a case-study based on real Deep Learning classifiers operating on data from multiple sensors. The evaluation showed that the minimally dynamic, typical-case optimal algorithm provides a significant improvement over the best possible static solutions, approximately halving the performance gap to a hypothetical clairvoyant algorithm.

6.1 Directions for future work

Deriving an optimal fully dynamic algorithm and assessing its complexity is one possible avenue for future work. While such an algorithm is likely to have prohibitively high run-time overheads, it would provide a more precise reference for the performance of the typical-case optimal algorithm presented in this paper, and any heuristic algorithms that may be derived in future.

The work in this paper assumes that the performance of the classifiers, as characterized by the profile table, is consistent across different operational environments or *contexts*. Research in late 2022 [29] showed that this is not always the case, and that the training of machine learning classifiers can lead to over-fitting, and hence performance that can be significantly degraded in some practical contexts. By comparison, simpler traditional classifiers are less likely to suffer from this problem [29]. An interesting avenue for future research relates to catering for different operational contexts with associated classifier characterizations (i.e. multiple profiles for each classifier) within the same system. This could potentially be achieved by using the methodology presented in this paper to determine solutions appropriate to different operational modes, corresponding to the different contexts or environments.

ACKNOWLEDGMENTS

This research was funded in part by Innovate UK HICLASS project (113213), and the US National Science Foundation (Grants CPS-1932530, CNS-2141256, and CNS-2229290). EPSRC Research Data Management: No new primary data was created during this study.

REFERENCES

[1] Tarek Abdelzaher, Kunal Agrawal, Sanjoy Baruah, Alan Burns, Robert I. Davis, Zhishan Guo, and Yigong Hu. 2023. Scheduling IDK Classifiers with Arbitrary Dependences to Minimize the Expected Time to Successful Classification. *Real-Time Systems (to appear)* (2023). https://www-users.york.ac.uk/~rd17/papers/IDKarbitrary.pdf

[2] Kunal Agrawal and Sanjoy K. Baruah. 2019. Adaptive Real-Time Routing in Polynomial Time. In *IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*. IEEE, 287–298. https://doi.org/10.1109/RTSS46320.2019.00034

[3] Kunal Agrawal, Sanjoy K. Baruah, and Alan Burns. 2020. The Safe and Effective Use of Learning-Enabled Components in Safety-Critical Systems. In *32nd Euromicro Conference on Real-Time Systems, ECRTS 2020, July 7-10, 2020, Virtual Conference (LIPIcs, Vol. 165)*, Marcus Völp (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 7:1–7:20. https://doi.org/10.4230/LIPIcs.ECRTS.2020.7

[4] Konstantinos Balaskas and Kostas Siozios. 2019. ECG analysis and heartbeat classification based on shallow neural networks. In *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 1–4.

[5] Sanjoy Baruah, Alan Burns, Robert I. Davis, and Yue Wu. 2022. Optimally Ordering IDK Classifiers Subject to Deadlines. *Real-Time Systems* (2022). https://doi.org/10.1007/s11241-022-09383-w

[6] Sanjoy K. Baruah. 2018. Rapid Routing with Guaranteed Delay Bounds. In *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*. IEEE Computer Society, 13–22. https://doi.org/10.1109/RTSS.2018.00012

[7] Sanjoy K. Baruah, Alan Burns, and David Griffin. 2022. Functional Uncertainty in Real-Time Safety-Critical Systems. In *RTNS 2022: The 30th International Conference on Real-Time Networks and Systems, Paris, France, June 7 - 8, 2022*, Yasmina Abdeddaïm, Liliana Cucu-Grosjean, Geoffrey Nelissen, and Laurent Pautet (Eds.). ACM, 1–11. https://doi.org/10.1145/3534879.3534884

[8] Sanjoy K. Baruah, Alan Burns, and Yue Wu. 2021. Optimal Synthesis of IDK-Cascades. In *RTNS'2021: 29th International Conference on Real-Time Networks and Systems, Nantes, France, April 7-9, 2021*, Audrey Queudet, Iain Bate, and Giuseppe Lipari (Eds.). ACM, 184–191. https://doi.org/10.1145/3453417.3453425

[9] Soroush Bateni and Cong Liu. 2018. Apnet: Approximation-aware real-time neural network. In *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 67–79.

[10] Michael G Bechtel, Elise McEllhiney, Minje Kim, and Heechul Yun. 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, 11–21.

[11] National Transportation Safety Board. 2019. Collision Between Vehicle Controlled by Developmental Automated Driving System and Pedestrian, Tempe, Arizona, March 18, 2018. https://www.ntsb.gov/investigations/accidentreports/reports/har1903.pdf

[12] Edward Broughton. 2005. The Bhopal disaster and its aftermath: A review. *Environmental health : a global access science source* 4 (02 2005), 6. https://doi.org/10.1186/1476-069X-4-6

[13] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 1321–1330. http://proceedings.mlr.press/v70/guo17a.html

[14] Seonyeong Heo, Sungjun Cho, Youngsok Kim, and Hanjun Kim. 2020. Real-time object detection system with multi-path neural networks. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 174–187.

[15] Md Sanzid Bin Hossain, Joseph Dranetz, Hwan Choi, and Zhishan Guo. 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 (2022), 3906–3917. https://doi.org/10.1109/JBHI.2022.3165383

[16] Yigong Hu, Shengzhong Liu, Tarek Abdelzaher, Maggie Wigness, and Philip David. 2021. 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, 169–178.

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

[18] Fereshte Khani, Martin C. Rinard, and Percy Liang. 2016. Unanimous Prediction for 100% Precision with Application to Learning Semantic Mappings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics. https://doi.org/10.18653/v1/p16-1090

[19] Jung-Eun Kim, Richard Bradford, and Zhong Shao. 2020. Anytimenet: Controlling time-quality tradeoffs in deep neural network architectures. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 945–950.

[20] J. Lee, A. Prajogi, E. Rafalovsky, and P. Sarathy. 2016. Assuring Behavior of Autonomous UxV Systems. In *Safe and Secure Systems and Software Symposium (S5)*. The Air Force Research Laboratory (AFRL).

[21] Nancy G. Leveson. 1986. Software Safety: Why, What, and How. *ACM Comput. Surv.* 18, 2 (jun 1986), 125–163. https://doi.org/10.1145/7474.7528

[22] Dongxin Liu, Tarek 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. 2022. IoBT-OS: Optimizing the Sensing-to-Decision Loop for the Internet of Battlefield Things. In *2022 International Conference on Computer Communications and Networks (ICCCN)*. 1–10. https://doi.org/10.1109/ICCCN54977.2022.9868920

[23] Dongxin Liu, Tianshi Wang, Shengzhong Liu, Ruijie Wang, Shuochao Yao, and Tarek Abdelzaher. 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, 1–10.

[24] Dr. Sandeep Neema. Accessed: 2019-03-07. Assurance for Autonomous Systems is Hard. https://www.darpa.mil/attachments/AssuredAutonomyProposersDay_ProgramBrief.pdf

[25] University of York. Accessed: 2022-12-20. Assuring autonomy international programme. https://www.york.ac.uk/assuring-autonomy/

[26] Weijing Shi, Mohamed Baker Alawieh, Xin Li, and Huafeng Yu. 2017. Algorithm and hardware implementation for visual perception system in autonomous vehicle: A survey. *Integration* 59 (2017), 148–156.

[27] Thomas P. Trappenberg and Andrew D. Back. 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, 296–301. https://doi.org/10.1109/IJCNN.2000.859412

[28] E. Trimble. 1989. Report No: 4/1990. Report on the accident to Boeing 737-400, G-OBME, near Kegworth, Leicestershire on 8 January 1989. (1989).

[29] Tianshi Wang, Denizhan Kara, Jinyang Li, Shengzhong Liu, Tarek Abdelzaher, and Brian Jalaian. 2022. The Methodological Pitfall of Dataset-Driven Research on Deep Learning: An IoT Example. In *Military Communications Conference, MILCOMM 2022, Sydney, USA, 28 November - 2 December 2022*. https://edas.info/web/milcom2022/program.html#S1569610867

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

[31] Shuochao Yao, Yifan Hao, Yiran Zhao, Huajie Shao, Dongxin Liu, Shengzhong Liu, Tianshi Wang, Jinyang Li, and Tarek Abdelzaher. 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, 1–10.

[32] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. 2017. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th international conference on world wide web*. 351–360.