Timely Classification of Hierarchical Classes

Sanjoy Baruah Alan Burns Tarek Abdelzaher Yigong Hu

***Abstract*— An IDK classifier is a learning-enabled software component that attempts to categorize each input provided to it into one of a fixed set of base classes, returning IDK ("I Don't Know") if it is unable to do so to a required level of confidence. We consider the use of IDK classifiers in applications where it is natural to consider the base classes as comprising the leaves of a class hierarchy. Classification into higher levels of such a hierarchy may be easier than classification into base classes. Given a collection of different IDK classifiers that have been trained to classify at different levels of a class hierarchy, we derive algorithms for determining the order in which to use these classifiers so as to minimize the expected duration to successful classification (whilst guaranteeing to meet a hard deadline).**

***Keywords*— Classification; Deep Learning; Edge AI; Safe AI**

I. Introduction

This paper develops algorithms for real-time classification, subject to latency and confidence constraints, in the important case where the entities to be classified fall naturally into a predefined class hierarchy. The work is timely as modern intelligent Cyber-Physical Systems (CPS) and Internet of Things (IoT) applications are increasingly equipped with *perception modules* for understanding their environment, context, or user. Examples range from automatic target recognition [1], [2] to human activity classification [3]. A recent trend is to perform such processing *at the point of need*, meaning on an embedded platform in the field [4], where the sensor data originate, as opposed to a resource-rich server in the cloud. Such *in situ* sensor data processing improves local autonomy and reduces reliance on communication with a remote device, but introduces a need for resource economy to reduce latency.

To reduce the average classification latency (especially on low-end embedded platforms), it was suggested to replace large monolithic classifiers with *IDK classifier cascades*, inspired by caching systems. An *IDK classifier* [5], [6] is a well-calibrated classifier that returns a class if classification confidence meets or exceeds a predefined threshold, and returns an "IDK" (i.e., "I don't know") otherwise. An *IDK cascade* [5] is a sequential arrangement specifying the order in which IDK classifiers are applied to an input until a non-IDK classification is obtained.[1] In a typical IDK cascade, computationally efficient classifiers with a more limited accuracy or repertoire (optimized for the common case) are generally used first, resorting to more complex and resource-intensive classifiers only when upstream classification fails (i.e., returns an IDK). Besides reducing the mean latency to a successful classification (by answering quickly in the common case), IDK classifier cascades can also improve reliability and trustworthiness [7], [8] by deferring uncertain or ambiguous examples to a more reliable (albeit slower) fallback system such as a larger model, a human expert, or a downstream verifier. Note that, IDK classifier cascades can be regarded as a special case of the popular *mixture of experts* models [9], with the additional restriction that the experts are queried sequentially (e.g., due to platform resource constraints).

Techniques for empirically calibrating arbitrary classifiers to return confidence values that match their actual probability of correctness have been proposed in literature [10] and can be leveraged in this work. Note that, in the case of an IDK classifier, only the desired confidence threshold needs to be well-calibrated. Given labeled training data and any classifier that returns, for each class, c, an output O_c that monotonically increases with confidence in that class, calibration simply entails finding the output threshold, O_c^{thresh}, such that for instances where $O_c \geq O_c^{\text{thresh}}$, the empirical probability that the input is of class c (according to the labeled data) is equal to the desired confidence. At run-time, the classifier returns a non-IDK class, c, only when $O_c \geq O_c^{\text{thresh}}$. The calibration ensures that non-IDK answers satisfy the desired confidence threshold as long as run-time observations are drawn from the same distribution as the training data (i.e., in the absence of domain shift).

This paper extends prior work on optimizing the order of classifiers in IDK cascades [11]–[15] to the important case of *class hierarchies*. To appreciate the advantages of exploiting class hierarchies, consider a toy example application that requires us to classify automobiles, each of which is guaranteed to be of one of four different models. Two of the models, the Mazda MX 5 and the Ford MUSTANG, are COUPEs and the other two, the Mazda CX 30 and the Mercedes Benz GLE 350, are Sports Utility Vehicles (SUVs), as shown in Figure 1. Two observations are due:

- It is often easier to distinguish higher-level classes (e.g., an SUV from a COUPE) than it is to directly distinguish all lower-level classes (e.g., a MUSTANG, a MX 5, a CX 30 and a GLE 350). Indeed our experiments, described in Section V-A, bear this out. The intuition is that higher-level categories are more distinct from one another (and thus easier to distinguish) than lower-level ones.
- It is also generally the case (again, borne out by our experiments) that *given* an identified higher-level category (i.e., whether a vehicle is a COUPE or an SUV), one can use more accurate specialized classifiers for distinguishing between the different members of that one category (e.g., different COUPEs or different SUVs). Such specialized classifiers are trained exclusively on the specific

[1] In applications where every input must eventually receive a valid classification, a *deterministic* classifier is added as the final stage of the IDK cascade. If this deterministic classifier also fails to classify an input, the system registers a fault, potentially triggering recovery mechanisms.

high-level category. Intuitively, they are generally more accurate because the number of subclasses to distinguish among is reduced (e.g., only SUVs or only COUPEs, as opposed to both types), thus simplifying the classification problem.

These observations suggest that a hierarchical classification approach can break a complex global classification problem into a series of simpler ones. In the context of IDK cascades, it might therefore appear that one can use prior optimization results for IDK cascades [11] first to sequence high-level IDK classifiers (e.g., those that classify vehicle as being either SUVs or COUPEs), then (depending on the identified high-level class) sequence the corresponding specialized IDK classifiers, thereby solving the original global problem.

Although one can indeed deal with class hierarchies by solving a number of individual classification problems, one at the top level of the hierarchy and another for each intermediate class, it turns out that one can do better (in the sense of achieving smaller average duration to successful classification) by instead taking a holistic perspective: we show this in Section II via a series of simple examples that also serve to expose the reasons why this is the case. Intuitively, in practice, the used higher-level categories might not always be very well-separated (or there may be a lot of diversity within one of the subcategories), rendering the hierarchical decomposition less beneficial. The succession of intermediate and specialized classifiers might take more time than a larger global classifier. Hence a good cascade optimization algorithm should have the flexibility to use not only classifiers designed for individual levels of the class hierarchy, but also those that flatten it into a single global classification problem. Moreover, when classifiers at multiple levels of the hierarchy are used, the IDK cascade sequencing problems at the different levels might not be independent. The best order for classifiers considered in a higher-level IDK cascade (to tell the higher-level category) might correlate with the best order of IDK classifiers to consider at the next level (e.g., due to common influences such as external weather conditions). And while our examples that illustrate these cases in Section II are contrived, we have observed similar phenomena in several real-life classification use-cases (some of which are discussed in Section V). In short, the optimization of IDK cascades for solving classification problems involving hierarchical classes needs to be approached *holistically*, as it is not decomposable into sequences of classification problems at individual levels of the hierarchy. *However, prior work does not allow for holistic modeling or the exploitation of such naturally-occurring class hierarchies*. Our evaluation, based on empirical data obtained from two different application use-cases shows that non-trivial improvements in latency are attained when holistic IDK cascade optimization is used for hierarchical classes.

To summarize, in this work, we seek to better understand how to exploit class hierarchies, when they exist, in order to achieve faster classification. Our specific *contributions* are:

1) We *propose a formalization of the notion of a hierarchy* of classes in the context of classification.
2) We develop a generalization of current *formal models* for representing collections of IDK classifiers, to account for class hierarchies and to allow for the more accurate modeling of classifier properties upon such hierarchies.
3) We obtain *algorithms* for optimal (in the sense of minimum average duration, whilst subject to a hard deadline) classification of inputs into classes of such class hierarchies.
4) We provide an *experimental evaluation*, including upon real-world workloads, demonstrating the efficacy of our classification algorithms.

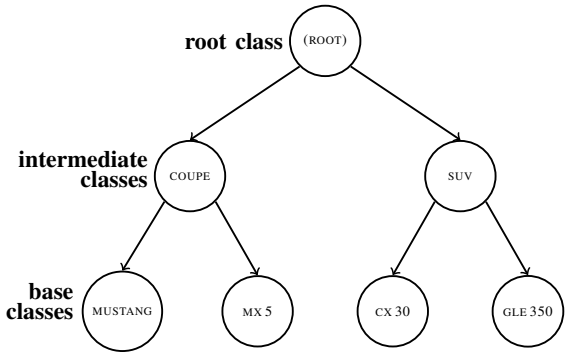


Fig. 1: An example class hierarchy

Organization. The remainder of this manuscript is organized in the following manner. In Section II we use a pair of examples to illustrate both opportunities that arise in classification due to the presence of the class hierarchy, and challenges to exploiting these opportunities. In Section III we formalize the notion of a class hierarchy, and propose a hierarchy-cognizant model for representing IDK classifiers. In Section IV, we present pre-processing and runtime algorithms for hierarchical classification using IDK classifiers. In Section V, we describe experiments evaluating our algorithms upon real-world case studies. We conclude in Section VI by summarizing our contributions and suggesting directions for followup research.

II. Some Illustrative Examples

We now step through a pair of simple examples on the automobile example class hierarchy of Figure 1 that highlight some of the challenges that arise in multilevel classification using IDK classifiers, beyond those that were identified (and dealt with) in prior work on classification using IDK classifiers.

We emphasize that these examples have been explicitly constructed for the purposes of exposing unique aspects of the underlying classification problem – they are *not* intended to be realistic. Real-world case studies are discussed in Section V.

A. Example I: Inter-level Dependences

Let us suppose that we have a pair of IDK classifiers K_0 and K_1 that have been trained to distinguish COUPEs from SUVs. Suppose that K_0 and K_1 exhibit complementary behaviors upon inputs where the ground truth is some COUPE: K_0 tends to return IDK on all those COUPEs for which K_1 returns a base class, and vice versa – see the first two columns of Table I.

K_0	K_1	K_2	K_3
COUPE	IDK	(base class)	IDK
IDK	COUPE	IDK	(base class)

TABLE I: Example discussed in Section II-A. Classifiers K_0 and K_1 classify inputs as either COUPE or SUV; classifiers K_2 and K_3 are specialized to only classify COUPEs. This table shows the different combinations of outcomes that may occur for some application, for inputs for which the ground truth is some COUPE (i.e., MUSTANG or MX 5).

Suppose that we also have another pair K_2 and K_3 of IDK classifiers that have been trained only on COUPEs, and hence are suitable for use when the intermediate class has already been identified as being COUPE. It further so happens that K_2 tends to return some base subclass of COUPE (i.e., MUSTANG or MX 5) on those inputs for which K_0 returns COUPE and IDK on the other COUPE inputs; K_3 in contrast, returns MUSTANG or MX 5 on those inputs for which K_1 returns COUPE and IDK on the other COUPE inputs – see the remaining columns of Table I. This condition may naturally occur, for example, if classifiers use different modalities (e.g., images versus sound). It could be that, say, K_0 and K_2 use vision and thus both work better at daytime, whereas K_1 and K_3 use sound and thus both work better at night.

Assuming that K_0 and K_1 (K_2 and K_3, respectively) have similar execution durations, it is evident that

- if K_0 is the classifier that determines some input to be a COUPE, then K_2 should be used for determining the base class for that input; whereas
- if K_1 is the classifier that determines some input to be of a COUPE, then K_3 should instead be used for determining the base class for that input.

This example illustrates that *the transition from the higher-level classification problem* (COUPE versus SUV) *to the lower-level one* (MUSTANG versus MX 5) *is not history-free*. Optimality is consequently lost if the hierarchical classification problem is broken up into multiple non-hierarchical classification problems (first determining which intermediate class, and then one classification problem per intermediate class).

B. Example II: Additional "Global" Classifiers

Again with our automobile example class hierarchy of Figure 1, let us suppose that: (1) A classifier K_1 has been trained to distinguish between SUVs and COUPEs; (2) Another classifier, K_{COUPE}, has only been trained on COUPEs; (3) Analogously, classifier K_{SUV} has only been trained on SUVs; and (4) A deterministic classifier K_{det} has been trained on all four of the base classes and so classifies an input as belonging to one of the four base models. Suppose the execution durations of these classifiers are as follows:

Classifier	K_1	K_{COUPE}	K_{SUV}	K_{det}
WCET	10	50	20	100

Let us make the simplifying assumption that the classifiers K_1, K_{COUPE}, and K_{SUV} are all (almost) perfect at their respective classification tasks, in that they are extremely unlikely to return IDK on any input. Furthermore, suppose that in our intended application one is twice as likely to encounter an SUV as one is to encounter a COUPE; hence on an input that is drawn uniformly at random from the underlying probability distribution characterizing the intended application, K_1 returns COUPE with probability $(1/3)$ and SUV with probability $(2/3)$. It is evident that with these three classifiers, the optimal classification strategy[2] on any input is as shown in Figure 2: first use K_1 to classify the input as being either a COUPE or an SUV, and then use the appropriate classifier K_{COUPE} or K_{SUV} to obtain a final base classification at an average duration of

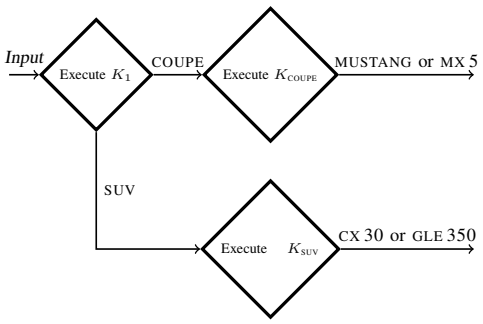


Fig. 2: A classification strategy for the example of Section II-B

$$\left(C_1 + \frac{1}{3}C_{\text{COUPE}} + \frac{2}{3}C_{\text{SUV}}\right) = \left(10 + \frac{50}{3} + \frac{2 \times 20}{3}\right) = \mathbf{40} \tag{1}$$

A global classifier. Now, suppose we also train another classifier, K_0, as a "global" classifier —one that attempts to directly classify its input as belonging to one of the four base classes (or return IDK if it is unable to do so to the desired level of confidence). As we'd stated in Section I, it is more challenging to achieve accuracy in classifiers of this kind; let us therefore suppose that on representative data, K_0 returns a base class with probability just 0.5, returning IDK with the remaining probability (also 0.5). Let C_0 denote K_0's WCET. If we execute K_0 prior to executing the classifiers as depicted in Figure 2, K_0 would return a base class with probability 0.5 and so the classifiers as depicted in Figure 2 would only execute with probability 0.5. We'd previously determined (Expression (1) above) that the average duration to successful classification of the classifiers as depicted in Figure 2 is 40; we may conclude that consequently, the overall average duration to successful classification is $(C_0 + 0.5 \times 40)$, which, for values of $C_0 < 20$, is smaller than the average duration to successful classification when executing only the classifiers as depicted in Figure 2. This argues in favor of using K_0 prior to using the classifiers as depicted in Figure 2 if $C_0 < 20$.

But is this conclusion valid? Let us suppose that closer inspection of classifier K_0 reveals that it actually performs abysmally on COUPEs, return IDK with probability 1.0; in contrast, it is far better at classifying SUVs, for which task it returns IDK with probability only $(1/4)$. So if we execute K_0 prior to executing the classifiers as depicted in Figure 2,

[2]Indeed, since K_{COUPE} and K_{SUV} are only specialized to classify COUPEs and SUVs respectively, the only alternative strategy is to directly use the deterministic classifier K_{det} (at an expected execution duration of 100).

K_{COUPE} would be called with probability $\left(\frac{1}{3} \times 1\right)$ and K_{SUV} with probability $\left(\frac{2}{3} \times \frac{1}{4}\right)$ and hence the average duration would actually be

$$\begin{aligned} & C_0 + \frac{1}{2}C_1 + \left(\frac{1}{3} \times 1\right) C_{\text{COUPE}} + \left(\frac{2}{3} \times \frac{1}{4}\right) C_{\text{SUV}} \\ = & \left(C_0 + \frac{1}{2} \times 10 + \frac{1}{3} \times 50 + \frac{1}{6} \times 20\right) \\ = & \left(C_0 + 5 + \frac{50}{3} + \frac{10}{3}\right) = \boldsymbol{C_0 + 25} \end{aligned}$$

which is smaller than the average duration to successful classification when executing the classifiers as depicted in Figure 2 for $C_0 < 15$ (and not 20, as concluded above). In other words, executing classifier K_0 prior to the configuration of Figure 2 *increases* (rather than decreasing) average duration for values of C_0 satisfying $15 < C_0 < 20$.

This example illustrates that *while global classifiers* —those that do not exploit the class hierarchy but rather attempt to classify an input as belonging to one of the base classes— *may be useful despite their far more limited accuracy, their appropriate use requires careful analysis.*

III. A MODEL FOR IDK CLASSIFIERS

We now formalize the notion of class hierarchies (Sec. III-A) and propose extensions (Sec. III-B) to the formal model for representing collections of IDK classifiers that is commonly used in the real-time literature including [11]–[15], so as to enable the accurate modeling of additional aspects of the collections that were revealed, in the examples of Section II, to be salient in the context of hierarchical classification problems.

We start out with a brief description of the currently-used model. In the standard model for IDK classifiers [11]–[15], an instance comprises n distinct IDK classifiers denoted by $K_0, K_1, K_2, \ldots, K_{n-1}$, as well as a deterministic classifier (see footnote 1) K_{det}, all for the same classification problem. The probabilistic behaviors of the different classifiers are not assumed to be independent; rather, they are collectively specified in tabular form in a table with 2^n rows, with each row corresponding to one of the 2^n potential combinations of the n IDK classifiers returning either a real class or IDK for an input. While the exponential number of combinations may seem like a limitation, the total number of used classifiers in practice is small (not unlike the case with the number of levels used in a cache hierarchy). After all, the goal is to *save* resources. Table II depicts the table for the case $n = 3$. In this table, p_0 denotes the probability that on some input that is drawn randomly from the underlying distribution characterizing the application for which these classifiers have been trained, all three IDK classifiers K_0, K_1, and K_2 will return IDK (and hence only K_{det} is able to classify this input), while p_7 denotes the probability that all three classifiers would return a base class on such a randomly-drawn input. Similarly, p_5 denotes the probability that classifiers K_2 and K_0 would return a base class, but K_1 would return IDK, on some randomly-drawn input. Abdelzaher et al. [11] describe a measurement-based methodology for accurately estimating the p_i probability values associated with each row of the table, by conducting profiling experiments using representative training data. This methodology characterizes the instance with the 2^n probability values and $(n+1)$ WCET values $C_0, C_1, \ldots, C_{n-1}, C_{\text{det}}$, with C_i denoting the worst-case execution duration[3] of IDK classifier K_i, $0 \leq i < n$, and C_{det} denoting the worst-case execution duration of the deterministic classifier K_{det}.

Row#	K_2	K_1	K_0	Probability
0	0	0	0	p_0
1	0	0	1	p_1
2	0	1	0	p_2
3	0	1	1	p_3
4	1	0	0	p_4
5	1	0	1	p_5
6	1	1	0	p_6
7	1	1	1	p_7

TABLE II: Tabular representation of the 2^n disjoint regions in the probability space for three IDK classifiers ($n = 3$) and one deterministic classifier. A zero (one, respectively) in a particular column denotes that the classifier labeling that column returns IDK (a base class, respectively). p_0–p_7 are non-negative real numbers summing to 1.

Example 1. To illustrate the interpretation of these parameters, consider an instance with three IDK classifiers ($n = 3$) and one deterministic classifier – this is the instance of Table II. One possible cascade we could synthesize for this example instance is $\langle K_2, K_0, K_{\text{det}} \rangle$; for this cascade, we can compute its average duration to successful classification upon a randomly-drawn input by the following reasoning.

- K_2 will certainly execute.
- Rows 0–3 of Table II correspond to the outcome that K_2 returns IDK; hence there is a probability $(p_0 + p_1 + p_2 + p_3)$ that K_0 will need to execute.
- Of these four rows 0–3, Row 0 and Row 2 correspond to the outcome that K_0 returns IDK; hence there is a probability $(p_0 + p_2)$ that both K_2 and K_0 will return IDK and hence K_{det} will need to execute

Therefore the expected duration to successful classification of the cascade $\langle K_2, K_0, K_{\text{det}} \rangle$ upon a randomly-drawn input is equal to $C_2 + (p_0 + p_1 + p_2 + p_3) \times C_0 + (p_0 + p_2) \times C_{\text{det}}$.

Obtaining optimal cascades. Given the specifications of an instance as 2^n probability values and $(n+1)$ WCET values, Abdelzaher et al. [11] have derived an algorithm that synthesizes a cascade that is optimal in the sense that this cascade has the minimal average duration to successful classification (subject to guaranteeing to always meet a hard deadline if one is specified).

A. Class Hierarchies

The goal in classification is to categorize each input as belonging to one of a specified set of *base classes*. In this

[3]This is a simplifying assumption; all our results readily extend to the more general model [11, Sec. 3] that uses a pair of parameters, $\tilde{C}_i$ and C_i, with $\tilde{C}_i$ denoting the average execution duration and C_i the worst-case duration.

paper, we extend prior work by assuming that these base classes comprise a 2-level *class hierarchy* in the sense that they can be partitioned into a set of k *intermediate classes* denoted $I_0, I_1, \ldots, I_{k-1}$, such that all the base classes in each intermediate class possess a variety of common features. Figure 1 (already introduced in Section I) provides a visual representation of the class hierarchy for the example we have considered in Sections I and II — here the two intermediate classes (and hence, $k = 2$) are

$$\begin{aligned} I_0 &\stackrel{\text{def}}{=} \text{COUPE} = \{\text{MUSTANG}, \text{MX 5}\} \\ I_1 &\stackrel{\text{def}}{=} \text{SUV} \quad = \{\text{CX 30}, \text{GLE 350}\} \end{aligned}$$

It is often meaningful to define hierarchies that are deeper than two levels. For ease of presentation *we restrict our attention to two-level hierarchies* for now; extension to deeper hierarchies is discussed in Section IV-E.

B. Class Hierarchies: Modelling IDK Classificaton

We now extend the formal model of [11]–[15] to incorporate class hierarchies. Let k denote the number of intermediate classes in the sole intermediate level (i.e., intermdiate classes $I_0, I_1, \ldots, I_{k-1}$ partition the base classes). The collection of IDK classifiers is partitioned into $(k+2)$ distinct sets:

1) A set $\mathcal{K}_I$ of *intermediate classifiers*, each of which has been trained to return some intermediate class I_ℓ, $0 \leq \ell < k$, or IDK on any input;
2) A set $\mathcal{K}_\phi$ of *global classifiers*, each of which has been trained to return either a base class or IDK upon any input.
3) For each ℓ, $0 \leq \ell < k$, a set $\mathcal{K}_\ell$ of *specialized classifiers*, each of which has been trained to return either a base class that is $\in I_\ell$, or IDK, on any input that has a priori been determined to belong to the intermediate class I_ℓ (presumably by some classifier in $\mathcal{K}_I$).

For the example discussed in Section II-B (and, as stated above, letting $I_0 \stackrel{\text{def}}{=} \text{COUPE}$ and $I_1 \stackrel{\text{def}}{=} \text{SUV}$), we would have

$$\mathcal{K}_I = \{K_1\}, \mathcal{K}_\phi = \{K_0\}, \mathcal{K}_0 = \{K_{\text{COUPE}}\}, \text{ and } \mathcal{K}_1 = \{K_{\text{SUV}}\}$$

In addition, there is a deterministic classifier K_{det} that never returns IDK – it classifies each input to some base class. Each classifier K_i is characterized by a WCET C_i. As in prior work [11], the probabilistic behaviors of the classifiers are specified in tabular form; we discuss the details below.

Initial table. This is a table with $\left(|\mathcal{K}_\phi| + |\mathcal{K}_I|\right)$ columns and $\left(2^{|\mathcal{K}_\phi|} \times (k+1)^{|\mathcal{K}_I|}\right)$ rows specifying the probability space for when one begins classifying an input and before the intermediate class to which it belongs becomes known. One column corresponds to each global classifier and each intermediate classifier. Each global classifier may return some base class or IDK, while each intermediate classifier may return one of the k intermediate classes or IDK (for a total of $(k+1)$ possibilities); hence the number of rows in the table is as stated above.

Specialized tables. For each ℓ, $0 \leq \ell < k$, there is a separate specialized table with $\left(|\mathcal{K}_I| + |\mathcal{K}_\phi| + |\mathcal{K}_\ell|\right)$ columns and $\left(2^{(|\mathcal{K}_\phi| + |\mathcal{K}_\ell|)} \times (k+1)^{|\mathcal{K}_I|}\right)$ rows for specifying the probability space when involving the intermediate class I_ℓ. Each column corresponds to one of the global classifiers, one of the intermediate classifiers, or one of the specialized classifiers in $\mathcal{K}_\ell$. (Recall, from the illustrative example of Section II-A, that the optimal choice of classifiers once the intermediate class is known may depend upon the outcomes of classifiers that were executed before the intermediate class became known – hence information about the classifiers in $\mathcal{K}_\phi$ and $\mathcal{K}_I$ needs to be maintained in this probability table.) The intermediate classifier either returns IDK or one of the k intermediate classes; the base and the specialized classifiers may return some base class or IDK, resulting in the number of rows in the table stated above.

Example 2. Table III lists the probability tables for our example of Section II-B. For this example $|\mathcal{K}_\phi| = 1$, $|\mathcal{K}_0| = |\mathcal{K}_1| = 1$, and $|\mathcal{K}_I| = 1$, and so the first table has 2 columns and $\left(2^1 \times (2+1)^1\right) = (2 \times 3) = \mathbf{6}$ rows, while each of the two specialized tables has 3 columns and $2^2 \times 4^1 = \mathbf{16}$ rows.

To see that these probability tables do indeed represent the instance described in Section II-B, observe that

- K_1 is assumed to never return IDK; hence, Rows 1–2 of the leftmost table have zero probability.
- SUVs are twice as likely as COUPEs, and hence in the leftmost table Rows 2–3 sum to 1/3, and Rows 4–5 to 2/3.
- K_0 is bad at COUPEs: Row 3 of the first table has probability zero; Row 9 of the second has probability 1.0
- K_0 returns IDK with probability $(1/4)$ on SUVs: Row 10 of the third table has probability 1/4 (and Row 14 as the remaining probability of $3/4$)

Generating these tables. These probability tables are obtained using a measurement-based methodology that is a generalization of the one developed by Abdelzaher et al. [11]. We record the classification decisions made by each of the IDK classifiers upon a large number of representative inputs.[4] We associate a count, instantiated to zero, with each row of the tables, and for each input increment the appropriate counts. Since each specialized classifier $K_i \in \mathcal{K}_\ell$ is only trained to classify inputs in I_ℓ, we derive its confidence threshold using only such inputs. However, when generating the probability tables, we include all inputs and record the true outputs, including those outside I_ℓ, to accurately capture the classifier's behavior in deployment. Once all the inputs have been processed in this manner, the probability value associated with each row of the probability table is set equal to its count divided by the total number of relevant inputs. (Section V-A details the application of this method upon one of our real-world use cases.)

A minor generalization. The model above is easily generalized to use-cases where some intermediate classifiers may additionally return some base classes. An example use-case of

[4]Each of the classifiers will have undergone training and validation upon representative inputs prior to this profiling phase; in many cases, this training data can be reused here, thereby eliminating the need for additional data. If new data is necessary it must also be representative of the expected inputs during deployment.

Row#	K_1	K_0	Prob
0	0	0	
1	0	1	
2	1	0	1/3
3	1	1	
4	2	0	1/6
5	2	1	1/2

Row#	K_{COUPE}	K_1	K_0	Prob
0	0	0	0	
1	0	1	0	
2	0	2	0	
3	0	3	0	
4	0	0	1	
5	0	1	1	
6	0	2	1	
7	0	3	1	
8	1	0	0	
9	1	1	0	1.0
10	1	2	0	
11	1	3	0	
12	1	0	1	
13	1	1	1	
14	1	2	1	
15	1	3	1	

Row#	K_{SUV}	K_1	K_0	Prob
0	0	0	0	
1	0	1	0	
2	0	2	0	
3	0	3	0	
4	0	0	1	
5	0	1	1	
6	0	2	1	
7	0	3	1	
8	1	0	0	
9	1	1	0	
10	1	2	0	1/4
11	1	3	0	
12	1	0	1	
13	1	1	1	
14	1	2	1	3/4
15	1	3	1	

TABLE III: Probability tables for the example of Section II-B. (Only non-zero probabilities shown – all other entries in the "Prob." columns are zeros.) A "**0**" in K_1's column denotes that it returns IDK, a "**1**" that it returns COUPE, and a "**2**" that it returns SUV. For any other classifier, a "**1**" in its column denotes that it returns the intermediate class labeling the column, a "**0**" denotes that it returns IDK.

this kind is discussed in Section V-A, where some intermediate classifiers return either COUPE or SUV, denoting a vehicle of the corresponding subclass, or "BACKGROUND", denoting the absence of a vehicle. In this case, "BACKGROUND" can be thought of as a base class: a "BACKGROUND" object needs no further classification.

IV. ALGORITHMS

In this section, we will develop a recursive formulation to the problem of determining an optimal strategy for choosing the order in which to execute IDK classifiers so as to minimize the average duration to successful classification whilst guaranteeing to always complete classification within a specified hard deadline $\mathcal{D}$. (For ease of presentation we first focus on minimizing the average classification duration; in Section IV-C we explain how to incorporate consideration of the deadline $\mathcal{D}$.) Algorithm 1 applies Dynamic Programming to speed up the recursion by avoiding unnecessary re-computations and thereby obtain a reasonably efficient implementation.

A. An Overview

We start out with a high-level overview of our approach. We assume a 2-level class hierarchy (as in Figure 1) with k intermediate classes $I_0, I_1, \ldots, I_{k-1}$. As described in Section III-B, there are $(k+2)$ disjoint sets of IDK classifiers: the global classifiers $\mathcal{K}_\phi$, the intermediate classifiers $\mathcal{K}_I$, and the specialized classifiers $\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_{k-1}$. In addition, there is a single deterministic classifier K_{det}. Each classifier is characterized by its worst-case execution time (WCET), with classifier K_i's WCET denoted C_i. The probabilistic behaviors of the IDK classifiers are characterized by $(k+1)$ probability tables: an *initial table*, and one *specialized table* for each of the k intermediate classes — see Section III-B.

Preprocessing. Prior to classification time we will construct, by a pre-processing algorithm that implements recursive search via dynamic programming

- An *initial cascade* that is an ordered list comprising a subset of $(\mathcal{K}_\phi \cup \mathcal{K}_I)$, terminated by K_{det}; and
- For each intermediate classifier K_i in this initial cascade and each intermediate class I_ℓ, a *specialized_cascade*(K_i, I_ℓ) that is an ordered list comprising a subset of $(\mathcal{K}_\phi \cup \mathcal{K}_\ell)$ and terminated by K_{det}.

Section IV-B explains how these cascades are synthesized.

Classification. Upon construction, these cascades are stored and used for classification during runtime in the following manner. Given an input that is to be classified,

- We start out attempting to classify this input via the classifiers in the initial cascade (in the order in which they appear in the cascade), until a non-IDK classification is obtained.
- If this non-IDK classification is to a base class, then we return this base class and are done. Else, some intermediate classifier K_i in the initial cascade must have returned some intermediate class I_ℓ.
- Henceforth we attempt to classify the input by using the classifiers in the specialized_cascade(K_i, I_ℓ) in order, until a non-IDK classification is obtained. This non-IDK classification is guaranteed to be some base class – this is returned, and we are done.

In Example 3 below, we discuss the example instances of Section II in terms of initial and specialized cascades.

Example 3. Recall that the instance of Section II-A assumes that intermediate classifiers K_0 and K_1 both have about the same execution duration, while classifiers K_2 and K_3, both specialized for COUPEs, offer complementary coverage (see Table I); hence the initial cascade may be $\langle K_0, K_1, K_{\text{det}} \rangle$, and

- specialized_cascade(K_0, COUPE) = $\langle K_2, K_{\text{det}} \rangle$
- specialized_cascade(K_1, COUPE) = $\langle K_3, K_{\text{det}} \rangle$

For the instance of Section II-B, recall that we use the global classifier K_0 if its WCET $C_0 < 15$, and then the classification strategy of Figure 2. This translates to an initial cascade of $\langle K_0, K_1, K_{\text{det}} \rangle$, and

- specialized_cascade(K_1, COUPE) = $\langle K_{\text{COUPE}}, K_{\text{det}} \rangle$
- specialized_cascade(K_1, SUV) = $\langle K_{\text{SUV}}, K_{\text{det}} \rangle$

whereas if $C_0 \geq 15$, the initial cascade does not include K_0 (it's simply $\langle K_1, K_{\text{det}} \rangle$) while the specialized cascades remain the same.

B. The Cascade-synthesis Algorithms

We now describe our dynamic-programming algorithms for constructing the initial cascade and the specialized cascades. We focus here primarily on explaining the intuition behind our algorithms; the algorithms themselves are provided in pseudo-code form as Algorithms 1 and 2 (and as Python code in the separately-uploaded supplementary material).

When starting out classifying some input we will not generally know beforehand the intermediate class to which it belongs, and so may only attempt to classify it by using classifiers in $(\mathcal{K}_I \cup \mathcal{K}_\phi \cup \{K_{\rm det}\})$. Hence, these are the classifiers considered in synthesizing the initial cascade.

Let S denote the classifiers in the prefix of the initial cascade that we have constructed so far (initially, $S = \emptyset$). Suppose they all returned IDK on some input. Below we will define a function EXPAND(S) (in Algorithm 1) for determining the classifier $K_h \in ((\mathcal{K}_I \cup \mathcal{K}_\phi) \setminus S)$ to execute next, so as to minimize the average remaining duration to successful classification; a call to EXPAND(S) will additionally return this minimum average remaining duration to successful classification.

If this classifier K_h returns IDK, then we add K_h to S and repeat the call to EXPAND(S). If K_h determines that the input belongs to some base class, then we are done. Else, it must be the case that K_h is an intermediate classifier (i.e., $K_h \in \mathcal{K}_I$), and it has declared that the input belongs to some intermediate class I_ℓ. Below we will define another function, EXPAND$'(S, I_\ell, T, K_h)$ (Algorithm 1, line 21), for determining the classifier to execute in this case in order to minimize the average remaining duration to successful classification, where

- S has the same interpretation as above: it is the set of classifiers, $S \subset (\mathcal{K}_I \cup \mathcal{K}_\phi)$, that we have used thus far but they have all returned IDK;
- K_h is the intermediate classifier that has returned I_ℓ on the input; and
- T is the set of specialized classifiers for intermediate class I_ℓ (i.e., $T \subset \mathcal{K}_\ell$) that we have used in an attempt to classify the input (after K_h had identified it as belong to class I_ℓ), but they have all returned IDK (initially, $T = \emptyset$).

As was the case with EXPAND(S), a call to EXPAND$'(S, I_\ell, T, K_h)$ will additionally return the minimum average remaining duration to successful classification.

The function EXPAND(S)**.** Recall that S denotes the set of classifiers, $S \subset (\mathcal{K}_I \cup \mathcal{K}_\phi)$, that we have used in an attempt to classify the input but they have all returned IDK.

The first if-condition of EXPAND(S) (Line 2) checks whether we have already considered this case; if so, we would have stored the minimum average remaining time to successful classification in the variable S.cost. (Thus, this if-condition is essentially implementing a top-down – *memoized* – dynamic program.)

The next if-condition (Line 4) checks whether we have exhausted the available supply of global and intermediate classifiers; if so, we must use the deterministic classifier $K_{\rm det}$.

The remainder of the pseudo-code considers the remaining intermediate (the for-loop at line 8) and global (the for-loop at line 13) classifiers one at a time, seeking to identify the one to execute next in order to minimize the remaining average duration to successful classification.

If we execute classifier $K_j \in ((K_I \cup K_\phi) \setminus S)$, the probability of K_j returning a particular output (IDK or some intermediate class for $K_j \in \mathcal{K}_I$; IDK or some base class for $K_j \in \mathcal{K}_\phi$) can be computed as the ratio (*Numerator*/*Denominator*), where

Algorithm 1: EXPAND( ) and EXPAND$'$( )

```
1  EXPAND(S)
2  if S.cost has already been computed then
3      return S.cost
4  if (S == (K_φ ∪ K_I)) then
5      S.cost = C_det; S.next = K_det
6      return S.cost
7  S.cost = ∞; S.next = K_∞ // K_∞: a placeholder
8  for each K_j ∈ (K_I \ S) do
9      Pr = CONDPR(S, K_j)
10     tmpCost = C_j + Pr[0] × EXPAND(S ∪ {K_j}) +
           Σ_{ℓ=0}^{k-1} (Pr[ℓ+1] × EXPAND'(S, I_ℓ, ∅, K_j))
11     if (tmpCost < S.cost) then
12         S.cost = tmpCost; S.next = K_j
13 for each K_j ∈ (K_φ \ S) do
14     Pr = CONDPR(S, K_j)
15     tmpCost = C_j + Pr[0] × EXPAND(S ∪ {K_j})
16     if (tmpCost < S.cost) then
17         S.cost = tmpCost; S.next = K_j
18 if (C_det < S.cost) then
19     S.cost = C_det; S.next = K_det
20 return S.cost

21 EXPAND'(S, I_ℓ, T, K_h)
22 if (S, I_ℓ, T, K_h).cost has already been computed then
23     return (S, I_ℓ, T, K_h).cost
24 if ((S ∪ T) == (K_ℓ ∪ K_φ)) then
25     (S, I_ℓ, T, K_h).cost = C_det; (S, I_ℓ, T, K_h).next = K_det
26     return cost
27 (S, I_ℓ, T, K_h).cost = ∞; (S, I_ℓ, T, K_h).next = K_∞
28 for each K_j ∈ ((K_ℓ ∪ K_φ) \ (S ∪ T)) do
29     Pr = CONDPR'(S, I_ℓ, T, h, K_j)
30     if (K_j ∈ K_φ) then
31         tmpCost =
               C_j + (Pr × EXPAND'(S ∪ {K_j}, I_ℓ, T, K_h))
32     else //Must be K_j ∈ K_ℓ
33         tmpCost =
               C_j + (Pr × EXPAND'(S, I_ℓ, T ∪ {K_j}, K_h))
34     if ((S, ℓ, T, h).cost > tmpCost) then
35         (S, I_ℓ, T, K_h).cost = tmpCost;
           (S, I_ℓ, T, K_h).next = K_j
36 if ((S, I_ℓ, T, K_h).cost > C_det) then
37     (S, I_ℓ, T, K_h).cost = C_det; (S, I_ℓ, T, K_h).next = K_det
38 return (S, I_ℓ, T, K_h).cost
```

- *Denominator* is the sum of the probabilities of all those rows of the initial probability table in which all the columns corresponding to classifiers in S are labeled IDK; and
- *Numerator* is the sum of the probabilities of all those rows of the initial probability table in which all the columns corresponding to classifiers in S are labeled IDK, and the column corresponding to K_j is labeled with that particular output.[5]

This computation is implemented by a function CONDPR(S, K_j) (pseudo-code omitted, since it does exactly what is described above) that is called at Lines 9 and 14. Let $Pr\{\text{IDK}\}$ denote the probability that this outcome is IDK, and $Pr\{I_\ell\}$ that this outcome is intermediate class I_ℓ (and hence the probability of a base class being returned equals $\left[1.0 - Pr\{\text{IDK}\} - \left(\sum_{\ell=0}^{k-1} Pr\{I_\ell\}\right)\right]$).

Lines 10 and 15 compute the average remaining duration to successful classification of the input as follows:

$$C_j + \left(Pr\{\text{IDK}\} \times \text{EXPAND}(S \cup \{K_j\})\right) + \left(\sum_{\ell=0}^{k-1} (Pr\{I_\ell\} \times \text{EXPAND}'(S, I_\ell, \emptyset, K_j))\right)$$

Here, the first term represents the execution duration of classifier K_j, the second, the remaining expected duration if K_j returns IDK, and each individual term within the summation in the third, the remaining expected duration if K_j returns a particular intermediate class I_ℓ.

Finally, the if-condition at Line 18 checks whether it would be faster to simply directly execute the deterministic classifier.

The function EXPAND$'(S, I_\ell, T, K_h)$ is essentially understood in much the same manner as EXPAND(S) above. Let S, I_ℓ, T and K_h have the interpretations discussed earlier: S denotes the set of classifiers in $(S \subset (\mathcal{K}_I \cup \mathcal{K}_\phi))$ that have returned IDK while $K_h \in \mathcal{K}_I$ has returned the intermediate class I_ℓ; T (initially $\emptyset$ when EXPAND$'(\,)$ is first called – see Line 10) denotes the set of classifiers in $\mathcal{K}_\ell$ that have returned IDK. If we were to now execute classifier $K_j \in ((K_\ell \setminus T) \cup (K_\phi \setminus S))$, two outcomes are possible: IDK or some base class. The probability of K_j returning IDK can be computed as the ratio $(\textit{Numerator}/\textit{Denominator})$, where

- *Denominator* is the sum of the probabilities of all those rows of the probability table for intermediate class I_ℓ in which all the columns corresponding to classifiers in $(S \cup T)$ are labeled IDK while the column corresponding to the classifier K_h is labeled with a designation indicating that it returns the correct intermediate class (which is, of course, I_ℓ); and
- *Numerator* is the sum of the probabilities of all those rows of the probability table for intermediate class I_ℓ that satisfy all the conditions above (for *Denominator*), and the column corresponding to K_j is labeled with IDK.

[5]Equivalently, it is the sum of the probabilities of all those rows that satisfy the conditions for *Denominator* above, that additionally have the column corresponding to K_j labeled with that particular output.

Algorithm 2: Synthesizing the cascades

```
1  Call EXPAND(∅)
2  // Synthesizing the initial cascade
3  Initialize the cascade to be empty
4  S = ∅
5  repeat
6  |   K_tmp = S.next
7  |   Append K_tmp to the end of the cascade
8  |   S = S ∪ {K_tmp} // Assume K_tmp returns IDK
9  until (K_tmp == K_det);

10 // Synthesizing specialized_cascade(K_i, I_ℓ)
11 Initialize the cascade to be empty
12 S = the classifiers preceding K_i in the initial cascade
13 T = ∅
14 repeat
15 |   K_tmp = (S, I_ℓ, T, K_ℓ).next
16 |   Append K_tmp to the end of the cascade
17 |   if K_tmp ∈ K_φ then
18 |   |   S = S ∪ {K_tmp}
19 |   else
20 |   |   T = T ∪ {K_tmp}
21 until (K_tmp == K_det);
```

As before, let $Pr\{\text{IDK}\}$ denote the probability that the outcome is IDK (and so the probability of a base class being returned is $(1 - Pr\{\text{IDK}\})$).

We can compute the average remaining duration to successful classification of the input as

$$C_j + \left(Pr\{\text{IDK}\} \times \text{EXPAND}'(S \cup \{K_j\}, I_\ell, T, K_h)\right)$$

if $K_j \in \mathcal{K}_\phi$, and

$$C_j + \left(Pr\{\text{IDK}\} \times \text{EXPAND}'(S, I_\ell, T \cup \{K_j\}, K_h)\right)$$

if $K_j \in \mathcal{K}_\ell$.

Synthesizing the cascades. A call to EXPAND$(\emptyset)$ will make the needed recursive calls to EXPAND$(\,)$ and EXPAND$'(\,)$ with the appropriate parameter settings, which in turn will result in the `cost` and `next` variables being assigned appropriate values. The manner in which the cascades are then synthesized is straightforward, and is presented in pseudo-code form in Algorithm 2.

C. Incorporating Hard Deadlines

If a hard deadline $\mathcal{D}$ is additionally specified with the interpretation that classification must always complete within $\mathcal{D}$ time units, the procedures EXPAND$(\,)$ and EXPAND$'(\,)$ of Algorithm 1 are modified to each accept an additional parameter D, denoting the remaining duration to deadline. When EXPAND$(\,)$ is called for the first time (Line 1 of Algorithm 2), this parameter is set equal to $\mathcal{D}$ (i.e., Line 1 of Algorithm 2 is modified to "EXPAND$(\emptyset, \mathcal{D})$"). Within EXPAND$(\,)$ and EXPAND$'(\,)$, IDK classifier K_i is only considered for possible execution if its execution duration $C_i \geq D + C_{\text{det}}$; if it is considered, then the corresponding recursive call (in

Lines 10, 15, 31, or 33) is made with the deadline parameter set to $(D - C_i)$.

A note: Recall (footnote 3) that we've been using a simplified model for IDK classifiers thus far: the model given in [11, Sec. 3] actually characterizes each classifier's execution duration via a pair of parameters, $\tilde{C}_i$ and C_i, with $\tilde{C}_i$ denoting the average execution duration and C_i the worst-case duration. Thus it is the $\tilde{C}_i$ values (rather than the C_i's) that are used in Algorithms 1 and 2 for minimizing average classification duration; however, the C_i values –WCET characterizations– are used in the tests ("if $C_i \geq D + C_{\text{det}}$") that check whether it is safe to execute a classifier K_i.

D. Run-time Complexity

There are two aspects to the computational complexity of our approach. *Pre-runtime complexity* accounts for the time required to (i) train the classifiers; (ii) populate the initial and the specialized probability tables; and (iii) synthesize the cascades. *Classification complexity* concerns the time taken to classify a single input during runtime.

The pre-runtime complexity of our approach is high, as is the norm in most deep-learning based computational approaches. Effective training of the IDK classifiers requires the collection of large amounts of representative training data, upon which each classifier must be trained. Populating the probability tables also requires that large amounts of representative data be classified by the different classifiers. The cascade-synthesis algorithm has running time exponential in the number of classifiers – Section V-C provides some example measurements. This is not surprising: earlier cascade synthesis algorithms [11]–[15] that did not exploit hierarchical structure already had exponential running time.

When it comes to classification complexity, however, our approach is very efficient : assuming that all the classifiers are pre-loaded and indexed in some manner that allows for constant-time retrieval, the time to classify any input is linear in the number of classifiers needed to classify it optimally (i.e., with minimum average execution time). Thus our complexity tradeoff is entirely consistent with the paradigm in DL-based edge AI: trade off considerable pre-runtime complexity for very efficient (in fact, optimal) runtime use.

E. Generalizing to Deeper Hierarchies

Both the IDK model of Section III and the algorithms discussed earlier in this section readily generalize to deeper hierarchies – generalizations of Figure 1 with the hierarchy represented as a tree with more levels, with each leaf denoting a base class and each non-leaf node representing an intermediate class. Different IDK classifiers are trained for different purposes: top-level classifiers classify any input into one of the intermediate classes immediately below the root; intermediate classifiers for each non-leaf node are trained to classify inputs determined to belong to the intermediate class associated with that node into the classes immediately below that node.[6] Probability tables are associated with each non-leaf node, specifying the probability space of classification outcomes for the classifiers that are specialized to that class, conditioned on the outcomes of classifiers associated with each intermediate class associated with nodes between the node and the root of the classification tree.

V. Evaluation

We evaluate our algorithm on two real-world applications: vehicle detection using multi-modal sensory inputs, and human activity recognition (HAR) using wearable sensors. This section presents our experience and findings; the data and code used have been uploaded as Supplementary Material (and will be offered for Artifact Evaluation if this paper is accepted).

A. Vehicle Detection Case Study

Data collection and pre-processing. We use the M3N-VC dataset [16], a large-scale vehicle monitoring dataset with acoustic and seismic signals recorded from multiple moving vehicles. As described in [16], sensor nodes equipped with a microphone (acoustic, 1.6 kHz), geophone (seismic, 200 Hz), and GPS (1 Hz) were deployed in six distinct environments, with 6–8 nodes per scene. Vehicles were driven around the area while the sensor nodes recorded signals and synchronized their clocks via GPS. Only the acoustic and seismic signals are used in this study.

For this study, we considered four base vehicle types: MUSTANG, MX 5, CX 30, and GLE 350, organized into the hierarchy of Figure 1. We use a 3.43-hour subset of data from six nodes and segment it into 2-second windows, while segments with missing sensor readings are discarded. The segmented data is transformed into spectrograms using Short-Time Fourier Transform (STFT). By combining three complexity levels of DeepSense [4] classifiers (varying convolutional/recurrent layer width and depth) with two sensor modalities, we generate 12 model variants. Using these model variants, we train three types of classifiers: (i) *global classifiers* that predict one of the four base vehicle classes or BACKGROUND, (ii) *intermediate classifiers* that classify samples as either SUV, COUPE, or BACKGROUND, and (iii) *specialized classifiers* that classify samples into base classes within the SUV or the COUPE group. Each classifier outputs a predicted label and confidence score. We set a required confidence threshold of 0.90 for global classifiers, and 0.95 for both intermediate and specialized classifiers (so that the cumulative error at the end of both stages is at most 10%, similar to global). If the confidence of a classifier exceeds the threshold, a label is returned; otherwise, the classifier outputs IDK.

Profiling. By examining the Pareto front of classifiers considering both the success rate and execution time, we select two intermediate classifiers (K_0 and K_1), two global classifiers (K_2, K_3), one specialized classifier for the SUV class (K_4),

[6]The idea of the *global classifiers* (those in $\mathcal{K}_\phi$) generalizes as follows: such a classifier may be associated with any node of the classification tree that is above the two levels levels, and classifies an input into any class, intermediate or base, in the subtree that is rooted at that node.

Classifiers	Intermediate		Global		SUV	COUPE		
	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_{det}
Modality, params	Both, 129698	Both, 356610	Both, 130469	Both, 1217109	Acoustic, 80355	Acoustic, 80355	Both, 129955	-
Success rate	76.1	87.6	66.8	99.9	1	91.1	94.9	1
Execution time	80.8	317.0	104.7	869.9	80.9	80.9	104.5	10000

TABLE IV: Classifiers for the vehicle detection case study (Section V-A).

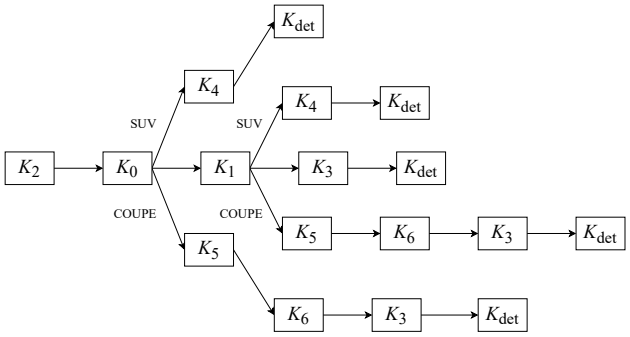
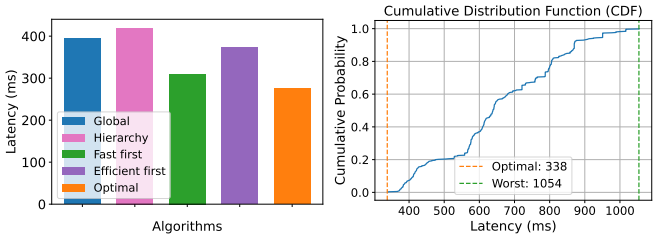


Fig. 3: Optimal cascade for the vehicle detection case Study (Section V-A).



(a) Latency comparison of the algorithms. (b) Latency distribution of all cascades.

Fig. 4: Vehicle detection case study results (Section V-A).

and two specialized classifiers for the COUPE class (K_5 and K_6). These selected classifiers are applied to the test data to obtain class predictions and confidence scores. Using these, we construct empirical probability tables as described in Section III, and record average and worst-case execution times. A summary of the classifiers with the number of parameters and modality choices can be found in Table IV.

Findings. We compare the optimal cascade generated by the proposed algorithm (labeled *Optimal*) to four baselines: (1) *Global*: Best cascade using only global classifiers. (2) *Hierarchy*: Best cascade using only intermediate and specialized classifiers. (3) *Fast first*: Classifiers ordered by increasing execution time. (4) *Efficient first*: Classifiers ordered by increasing (Non-IDK rate)/(execution time).

The expected execution duration of all algorithms is compared in Figure 4a. The proposed algorithm finds the cascade with the minimum expected execution duration. Compared with the best baseline algorithm, it reduces latency by 11.2%. A cumulative latency distribution of all possible cascades in a reduced setting (with fewer classifiers) is shown in Figure 4b, with the latency of the optimal cascade our algorithm found and the latency of the worst cascade highlighted. Compared with the other possible cascades, the optimal one has significantly shorter expected execution time. The worst cascade takes significantly longer to execute, showing that organizing the classifiers in the wrong way can lead to poor performance. A reduced setting is used here because the total number of possible cascades grows exponentially with the number of classifiers, making exhaustive evaluation infeasible in the full setting. The optimal classification strategy determined by our algorithm for this case study is shown in Figure 3. This is not a strategy one would intuitively have considered: a global classifier (K_2) is used before any intermediate classifier. While the first intermediate classifier is the fastest and most efficient, it is not selected first. The optimal cascade also differs from both the *Fast first* policy and the *Efficient first*, where the selected cascades are $K_0 \to K_2 \to K_1 \to K_3$ and $K_0 \to K_2 \to K_3 \to K_1$, respectively.

B. HAR Case Study

Data collection and pre-processing. For this case study, we use the RealWorld-HAR dataset [17], a dataset for human activity recognition with data from sensors attached to 7 positions on the human test subjects, while they perform one of the 8 activities. The dataset provides the accelerometer data (used in our study), as well as GPS, gyroscope, light, magnetic field, and sound level measurements, all collected at 50Hz.

We group the 8 activity types into two categories: intermediate class MOVING including CLIMBING-UP, CLIMBING-DOWN, WALKING, JUMPING, and RUNNING; intermediate class STATIC including STANDING, SITTING, and LYING. We use only the accelerometer data collected from the shin sensor and segment the data into 4-second windows. Similarly, we convert the sensor input into STFT form and train 3 types of DeepSense-based IDK classifiers: (i) *global classifiers* that classify inputs into one of the 8 base activity classes, (ii) *intermediate classifiers* that classify inputs as either STATIC or MOVING, and (iii) *specialized classifiers* that classify inputs within the STATIC or MOVING group.

In addition to the three model sizes used in the vehicle study, we introduce a smaller classifier, observing that it achieves high accuracy with low cost, especially for the intermediate classification task. We assume a required classification accuracy of 0.85 and use required precision of 0.85 for the global classifiers, 0.94 for the intermediate classifiers, and 0.91 for the specialized classifiers to compute per-classifier confidence thresholds, leading to at most 15% error, no matter which classifiers are used.

Profiling. We select two intermediate classifiers (K_0 and K_1), 3 global classifiers (K_2, K_3 and K_4), 3 specialized classifiers (K_5, K_6 and K_7), for the STATIC class, and 4 specialized classifiers, (K_8, K_9, K_{10} and K_{11}), for the MOVING class. Similar to the previous case study, we evaluate all classifiers on the test set, record their predictions and confidence levels, construct the empirical probability tables, and measure both

Classifiers	Intermediate		Global			STATIC			MOVING				
	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{det}
Parameters	5134	22978	77320	360072	1111256	5167	23043	76035	5233	23173	76549	358533	-
Success rate (%)	40.2	99.9	67.2	92.4	1	84.3	92.4	97.1	52.0	69.0	91.7	95.5	1
Execution time (ms)	31.3	47.0	77.0	242.2	571.8	31.1	46.4	69.6	30.9	50.8	69.8	241.3	10000

TABLE V: Classifiers for the HAR case study (Section V-B).

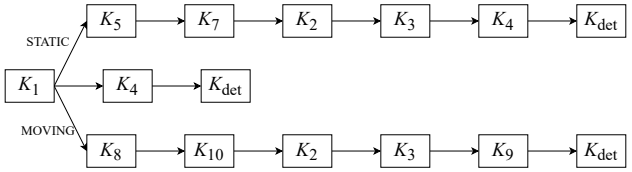
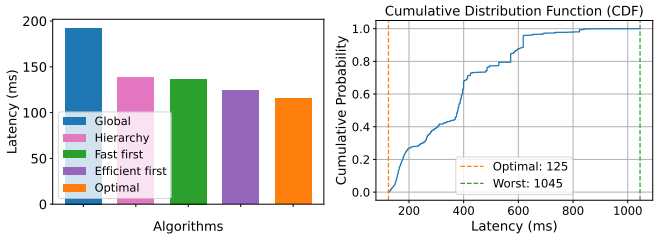


Fig. 5: Optimal cascade for the HAR case study (Section V-B).



(a) Latency comparison of the algorithms. (b) Latency distribution of all cascades.

Fig. 6: HAR case study results (Section V-B).

average and worst-case execution times. A summary of the classifiers can be found in Table V.

Findings. The expected execution duration of all algorithms is compared in Figure 6a. Our algorithm produces the cascade with the lowest expected execution time, reducing latency by 7.4% compared with the best baseline. Figure 6b shows the latency distribution across all cascades in a reduced-scale setting. Compared with the other possible cascades, the optimal one selected by our algorithm has significantly shorter expected execution time. The optimal cascade is shown in Figure 5. It is different from the one for the vehicle case study, and it is also counterintuitive. Instead of starting with any global classifier, or starting with the smaller intermediate classifier K_0, it directly goes to the larger but more efficient intermediate classifier K_1. Then it skipped classifiers K_2 and K_3, and directly selected K_4. In the STATIC branch, the selected optimal cascade skipped the specialized classifier K_6, which differs from the one generated by *Fast first* and *Efficient first*: $K_5 \rightarrow K_6 \rightarrow K_7 \rightarrow K_2 \rightarrow K_3 \rightarrow K_4$. In the MOVING branch, the cascade generated by the *Fast first* and *Efficient first* is $K_8 \rightarrow K_9 \rightarrow K_{10} \rightarrow K_2 \rightarrow K_{11} \rightarrow K_3 \rightarrow K_4$. In contrast, the optimal cascade delays classifier K_9 until after the slower special classifier K_{10}, and also after the much slower global classifiers K_2 and K_3. It also skips classifier K_{11} altogether, even though it is faster than both K_2 and K_3. This demonstrates that due to the complex probability dependencies in the data, one cannot easily determine the optimal cascade by looking at the classifier performance metrics.

C. Algorithm Execution Times

We profile the time and memory required by the cascade-synthesis algorithm to find the optimal cascade, using the vehicle detection example. Experiments are run on a workstation computer (Lambda Labs Vector with an AMD Threadripper Pro 3975WX at 3.50 GHz, 128GB of RAM, running Ubuntu 20.04). For a problem with 5 intermediate classifiers, 5 global classifiers, and 5 specialized classifiers for each intermediate class, the algorithm takes 155 minutes and a maximum memory of 128 MB to derive the optimal solution.

VI. Conclusions

The research discussed in this paper extends the use of IDK cascades for purposes of real-time classification to the case of class hierarchies. It establishes that hierarchical IDK classification is not decomposable to a sequence of flat IDK cascade classification problems without loss of optimality. Optimal holistic algorithms are therefore derived for exploiting the hierarchical class structure, and evaluated on real classifiers and sensor data from two application domains. Results demonstrate the advantages of holistic optimization.

This work may be extended in several directions. While we have focused on IDK classifiers primarily due to the large body of pre-existing work on them in the real-time community, our ideas and techniques generalize to other situations where hierarchical structure may be exploited to reduce latency. With minor modifications, our algorithms can be adapted to provide optimal classification strategies using *early exit* classifiers [18]–[20] (rather than IDK ones). They can also be extended to handle *mixture-of-experts* [21]–[24] situations, where resource constraints prevent the invocation of all experts at the same time, calling for optimally staged processing, with one (or a bounded number of) experts per stage.

The work in this paper focused on scenarios where a single class label is sufficient to describe the environment at any given time. In reality, many perception problems involve *multi-label* classification (e.g., recognizing all car types in a video frame). One solution to handle such problems might be to decompose them into single-label problems. For example, background subtraction can be used to identify the approximate locations of all moving objects in a frame, then pass each object to a different classifier cascade. Another solution could be to extend this work to cascades of multi-label classifiers.

Finally, domain shift is a key problem with all machine-learning, including IDK classifiers. When input data are from a substantially different distribution than the one used for training, the confidence values returned by the IDK classifiers may be incorrect. Out-of-distribution detection techniques [25]–[27] may be needed to detect domain shift and force classifiers to return an IDK instead. These topics will be investigated in our future work.

ACKNOWLEDGMENTS

Add more here... Research reported in this paper was sponsored in part by DEVCOM ARL under Cooperative Agreement W911NF-172-0196, NSF CNS 20-38817, and the Boeing Company.

S_kB: NSF CNS-2141256, CNS-2229290 (CPS), and CNS-2502855 (CPS)

REFERENCES

[1] Dhiraj Neupane and Jongwon Seok. A review on deep learning-based approaches for automatic sonar target recognition. *Electronics*, 9(11):1972, 2020.

[2] Khalid El-Darymli, Eric W Gill, Peter Mcguire, Desmond Power, and Cecilia Moloney. Automatic target recognition in synthetic aperture radar imagery: A state-of-the-art review. *IEEE access*, 4:6014–6058, 2016.

[3] Kaixuan Chen, Dalin Zhang, Lina Yao, Bin Guo, Zhiwen Yu, and Yunhao Liu. Deep learning for sensor-based human activity recognition: Overview, challenges, and opportunities. *ACM Computing Surveys (CSUR)*, 54(4):1–40, 2021.

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

[5] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, and Joseph E. Gonzalez. IDK cascades: Fast deep learning by learning not to overthink. *CoRR*, abs/1706.00885, 2017.

[6] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, and Joseph E. Gonzalez. IDK cascades: Fast deep learning by learning not to overthink. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, August 2018.

[7] Nicholas Kashani Motlagh, Jim Davis, Tim Anderson, and Jeremy Gwinnup. Learning when to say "I Don't Know". In George Bebis, Bo Li, Angela Yao, Yang Liu, Ye Duan, Manfred Lau, Rajiv Khadka, Ana Crisan, and Remco Chang, editors, *Advances in Visual Computing*, pages 196–210, Cham, 2022. Springer International Publishing.

[8] Roi Cohen, Konstantin Dobler, Eden Biran, and Gerard de Melo. I don't know: Explicit modeling of uncertainty with an IDK token. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 10935–10958. Curran Associates, Inc., 2024.

[9] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts. *arXiv preprint arXiv:2407.06204*, 2024.

[10] Shuochao Yao, Yiran Zhao, Huajie Shao, Aston Zhang, Chao Zhang, Shen Li, and Tarek Abdelzaher. Rdeepsense: Reliable deep mobile computing models with uncertainty estimations. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, 1(4):1–26, 2018.

[11] 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.

[12] Sanjoy Baruah, Alan Burns, Robert Davis, and Yue Wu. Optimally ordering IDK classifiers subject to deadlines. *Real Time Syst.*, 2022.

[13] Tarek Abdelzaher, Sanjoy Baruah, Iain Bate, Alan Burns, Robert I. 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, New York, NY, USA, 2023. Association for Computing Machinery.

[14] 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.

[15] Sanjoy Baruah, Iain Bate, Alan Burns, and Robert Davis. Optimal synthesis of fault-tolerant IDK cascades for real-time classification. In *Proceedings of the 30th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2024)*. IEEE, 2024.

[16] Jinyang Li, Yizhuo Chen, Ruijie Wang, Tomoyoshi Kimura, Tianshi Wang, You Lyu, Hongjue Zhao, Binqi Sun, Shangchen Wu, Yigong Hu, Denizhan Kara, Beitong Tian, Klara Nahrstedt, Suhas Diggavi, Jae H. Kim, Greg Kimberly, Guijun Wang, Maggie Wigness, and Tarek Abdelzaher. RestoreML: Practical unsupervised tuning of deployed intelligent iot systems. In *2025 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE, 2025.

[17] Timo Sztyler and Heiner Stuckenschmidt. On-body localization of wearable devices: An investigation of position-aware activity recognition. In *2016 IEEE international conference on pervasive computing and communications (PerCom)*, pages 1–9. IEEE, 2016.

[18] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, 2016.

[19] Yamin Sepehri, Pedram Pad, Ahmet Caner Yüzügüler, Pascal Frossard, and L. Andrea Dunbar. Hierarchical training of deep neural networks using early exiting. *IEEE Transactions on Neural Networks and Learning Systems*, 36(4):6271–6285, 2025.

[20] Fatih Ilhan, Selim Furkan Tekin, Sihao Hu, Tiansheng Huang, Ka-Ho Chow, and Ling Liu. Hierarchical deep neural network inference for device-edge-cloud systems. In *Companion Proceedings of the ACM Web Conference 2023*, WWW '23 Companion, page 302–305, New York, NY, USA, 2023. Association for Computing Machinery.

[21] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[22] Noam Shazeer, Azade Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V Le, Wei Chen, Mohammad Norouzi, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[23] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(127):1–39, 2022.

[24] Dmitry Lepikhin, Hieu Xu, Yanqi Hoffmann, Yaroslav Aharoni, Dianne Chen, Mia Johnson, Orhan Lee, Adam Metzler, Zhifeng Qi, James Smith, et al. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[25] Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. Energy-based out-of-distribution detection. *Advances in neural information processing systems*, 33:21464–21475, 2020.

[26] Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *Advances in neural information processing systems*, 32, 2019.

[27] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *International Journal of Computer Vision*, 132(12):5635–5662, 2024.