The Safe and Effective Use of Low-Assurance Predictions in Safety-Critical Systems

Kunal Agrawal
Washington University in Saint Louis, MO, USA

Sanjoy Baruah
Washington University in Saint Louis, MO, USA

Michael A. Bender
Stony Brook University, NY, USA

Alberto Marchetti-Spaccamela
Sapienza Università di Roma, Italy

Abstract

The algorithm-design paradigm of *algorithms using predictions* is explored as a means of incorporating the computations of lower-assurance components (such as machine-learning based ones) into safety-critical systems that must have their correctness validated to very high levels of assurance. The paradigm is applied to two simple example applications that are relevant to the real-time systems community: energy-aware scheduling, and classification using ML-based classifiers in conjunction with more reliable but slower deterministic classifiers. It is shown how algorithms using predictions achieve much-improved performance when the low-assurance computations are correct, at a cost of no more than a slight performance degradation even when they turn out to be completely wrong.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Scheduling

Keywords and phrases Algorithms using predictions, robust scheduling, energy minimization, classification, on-line scheduling

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2023.3

Funding *Sanjoy Baruah*: US National Science Foundation (Grants CPS- 1932530, CNS-2141256, and CNS-2229290)
Michael A. Bender: US National Science Foundation (Grants CCF-2247577 and CCF-2106827)
Alberto Marchetti-Spaccamela: ERC 788893 AMDROMA, Sapienza Res. Proj. RM11816436B2F886

1 Introduction

The increasing use of Learning-Enabled Components (LECs) in safety-critical applications such as autonomous driving present unique challenges to the discipline of safety-critical systems engineering. On the one hand, the availability of such LECs dramatically enhances the capabilities of autonomous cyber-physical systems – it is hard to conceive of, e.g., self-driving cars that do not make extensive use of LECs such as Deep Neural Networks for perception. But on the other hand, most widely used LECs cannot provide performance guarantees at high enough levels of assurance that they may be used in the verification and validation processes that are so fundamental to safety-critical systems engineering.

Context. A recent line of work in the algorithms community, *algorithms using predictions*, offers a promising approach for incorporating low-assurance predictions of the kind that are typically provided by LECs into algorithms designed for use in safety-critical systems. Algorithms using predictions are intended to perform very well when the predictor is accurate (this property is called *consistency* – see Sec. 3), while simultaneously guaranteeing to provide acceptable performance regardless of the quality of the predictions (called *robustness* – also formally defined in Sec. 3).

In this paper, we investigate the applicability of this framework to safety-critical systems. We have found that the framework needs to be adapted for this purpose due, amongst other reasons discussed in Sec. 3, to the presence of hard constraints (e.g., deadlines that must be met) in many safety-critical systems. Most prior work on algorithms using predictions has focused upon pure optimization problems that do not include such hard constraints[1] – indeed, amongst the 28 papers tagged with the keyword "scheduling" in the comprehensive list of publications on the topic that is maintained on the Algorithms With Predictions web-site,[2] we found only two [2, 3] that consider workload models with hard deadlines. (We will discuss these two papers further in Sec. 2.)

Our Contributions. In this paper, we *adapt* the algorithms using predictions framework for use in safety-critical systems and *provide an introductory explanation* of the adapted framework, specifically written to be comprehensible with moderate effort by members of the real-time systems community who may have had no prior exposure to algorithms using predictions. This explanation emphasizes the main features of the framework, and makes the case that this framework can contribute to the ongoing efforts of our community at incorporating LECs into safety-critical CPS's. In addition, we *illustrate the applicability* of this framework to real-time systems by considering the following two problems that are of interest to the real-time systems community:

1. As a first example illustration, we examine a problem that has been studied extensively in our community: energy-efficient hard-real-time scheduling. One of the main challenges encountered in this problem is what is commonly referred to as "the WCET problem" [16] – the non-determinism and variation in the execution duration of individual pieces of code upon modern computing platforms. In hard-real-time systems, the WCET problem requires that computing capacity be provisioned to accommodate a conservative worst-case upper bound on the needed amount of computation; we will see that this can lead to excessive energy being consumed during run time. If a reasonably good (although not guaranteed to be correct) prediction on the actual execution duration of the code is also available, then we will see that the algorithms using predictions framework can realise considerable energy savings.
2. Our second example illustration considers a form of LECs called IDK classifiers that has recently garnered some attention in the real-time systems community (see, e.g., [1, 5, 6]). Classifiers are used for classifying sensor readings into one of a set of a classes (e.g., a camera input to "pedestrian", "stop sign", etc.); LEC-based classifiers are generally not able to guarantee to do so at a high enough level of assurance and so should be backed up by deterministic – i.e., not learning-enabled – classifiers (including, perhaps human intervention). We will examine how the framework of algorithms using predictions can nevertheless use LEC-based classifiers to provide improved performance in the form of reduced expected response times.

[1] E.g., a paper at RTSS last year [18] proposed an algorithm using predictions for soft real-time scheduling to minimize the average response time – it is not at all obvious how one would adapt this algorithm to handle hard deadlines.
[2] `https://algorithms-with-predictions.github.io` (Accessed on 25/02/2023.)

Organization. The remainder of this manuscript is arranged as follows. In Section 2 we step through the process of designing an algorithm using predictions for a very simple energy-aware scheduling application, emphasizing, at each point in the presentation, the role that the predictions play in the algorithm design and the consequent implications on algorithm performance. This example application will give us an intuitive understanding of the algorithm-design paradigm of algorithms using predictions; we formalize this intuition in Section 3 and explicitly articulate the elements of the paradigm. We provide a second, somewhat more sophisticated, illustration of the application of this paradigm in Section 4, where we design an algorithm using predictions for minimizing the expected duration needed to successfully complete a classification operation. We close in Section 5 by placing this work within a larger context of safety-critical systems design, and some suggested directions of follow-up research.

2 Illustrative Example: Energy-Efficient Scheduling

In this section we apply the algorithms using predictions framework, suitably adapted as needed, to a problem that has been widely studied in the real-time systems community: scheduling to minimize energy consumption (see, e.g., the surveys [4, 9] and the references cited therein). We will show how predictions of run-time behavior, when available, can be used to significantly reduce energy consumption when the predictions are somewhat accurate while not requiring too much additional energy when the predictions are completely wrong.

This section is organized as follows. We first briefly describe (Sec. 2.1) the system model we will use, in which a job is characterized by its worst-case execution time, which is guaranteed to be a safe upper bound on its actual execution duration, as well as a prediction (which may be inaccurate) of the actual duration for which it will execute. We then propose (Sec. 2.2) a baseline prediction-oblivious algorithm that is fairly straightforward – what a real-time systems developer would probably come up with if no prediction of the actual execution duration had been provided. Next (in Sec. 2.3) we explain how the prediction can be incorporated to obtain an algorithm that is more energy-efficient than the prediction-oblivious one when the prediction is more-or-less correct, whilst consuming not much more energy than the prediction-oblivious algorithm even when the prediction is completely off the mark. We then (Sec. 2.4) address the issue of learning better predictions during run-time, and finally (Sec. 2.5) apply these concepts upon a simple example system. We close out the section by discussing related work and listing some open questions (Sec. 2.6).

2.1 System model

Since our primary purpose is to illustrate the algorithms using predictions framework, we will make a number of simplifying assumptions in both our workload model and our energy model for the purposes of ease of presentation (and so that we can focus on emphasizing the use of prediction in the algorithms we develop). We will point out, where relevant, how our results are easily modified to account for the factors that we abstract away in our simplifications.

We assume that we have a single job that is released at time zero, has a hard deadline at time D, and is characterized by a worst-case execution time parameter (WCET) W, that is to be executed upon a uniprocessor platform. The work done by the processor over any time-interval $[t_1, t_2]$ is equal to $\left(\int_{t_1}^{t_2} s(t)\,dt\right)$, where $s(t)$ denotes the processor speed at time-instant t. The speed may be changed by the run-time scheduler at any instant to take on any non-negative real value; there is no cost or penalty associated with changing the

Table 1 Summarizing the system model: $\langle W, D, P, \alpha, \gamma_{\text{ub}} \rangle$.

symbol	interpretation
W	worst-case execution time
D	deadline
P	predicted execution time
α	exponent of power function (power = speed$^\alpha$)
γ_{ub}	robustness bound (maximum ratio of energy consumed)

speed. The *power* required to run the processor at some speed s is equal to s^α, where α is a positive real value that is larger than 1.[3] Hence, the *energy* consumed by the processor over any time-interval $[t_1, t_2]$ is given by $\left(\int_{t_1}^{t_2} \big(s(t)\big)^\alpha \, dt\right)$.

A predicted execution duration. As is widely known in the real-time systems community, in safety-critical systems the values that are assigned to WCET parameters tend to be conservative over-approximations; hence it is unlikely that our example job will actually need to execute for a duration W under the overwhelming majority of run-time circumstances that arise in practice. Let us assume that we additionally have a prediction, P, on the *actual execution time* (AET) of the job, where P is a non-negative real-valued number that is no larger than the WCET W.

Recall that our goal is to develop an algorithm that performs very well when the prediction is correct, while simultaneously guaranteeing to not perform too poorly even when the prediction is incorrect. We assume that we are provided with an additional parameter that is a real number > 1, γ_{ub}, specifying the maximum multiplicative factor by which the energy consumed by an algorithm using predictions may exceed the energy consumed by a reasonable prediction-oblivious algorithm that ignores the prediction (which we will describe in Sec. 2.2), regardless of how poor the prediction turns out to be.

Generalizing the model. As stated earlier, we are assuming a very simple system model in order to keep the focus on the algorithms using predictions framework. Here, we briefly discuss possible extensions to our model that render it more realistic.

- We are considering the problem of scheduling a single job within a specified time-interval $[0, D]$. Such problems could arise in application systems where a more complex workload is being scheduled non-preemptively, with the non-preemptive schedule generated using heuristic tools that seek to maximize the *slack* for each job. Energy-efficient implementation of the schedule so generated would require us to schedule each job within a specified time-interval, yielding (for each job) the problem addressed in our model.
- Our energy model (that the power needed to execute the processor at speed s is equal to s^α) is idealized in that it ignores, e.g., leakage energy. But as we will see later in this section, the methodology we develop allows for "black-box" models to be substituted for this simple energy model – the overall methodology remains unchanged, the only difference being that numerical computation (rather than the solving of closed-form expressions) may be needed.
- In a similar vein, adding the restriction that the processor speed may only take on one of a few pre-specified permissible values does not change the essential structure of our methodology (although the combinatorial explosion resulting from having to choose from amongst a set of discrete speeds may slow down the running times).

[3] Observe that energy-efficient scheduling is trivial if $\alpha \leq 1$: it is optimal to simply run the processor at maximum speed until the job completes.

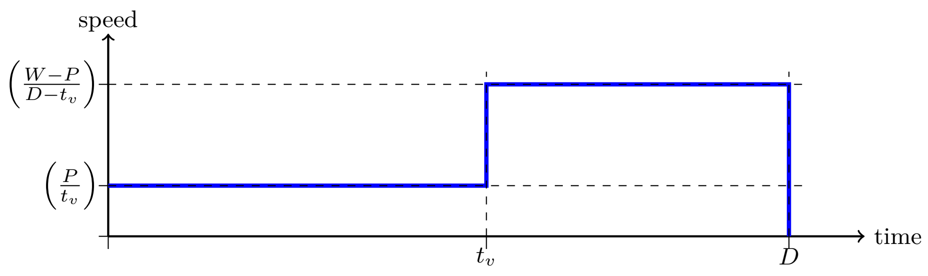

Figure 1 Speed profile for Algorithm ALG (Sec. 2.3): A job is executed according to this profile until completion (upon which the speed is set to zero).

2.2 A Prediction-Oblivious Algorithm

If we were to choose to not make use of the prediction, a reasonable run-time algorithm would be to set the processor speed to (W/D) and then execute the job until completion. (This follows from the well-understood phenomenon – see, e.g., [17] – that for $\alpha > 1$, the energy required to complete a known amount of work is minimized by having the CPU run at a constant speed for the duration of execution of the work. Thus, setting the processor speed equal to W/D would result in the minimum energy consumption in the worst case – i.e., when the job requires maximum execution.) Since this algorithm has the least worst-case energy consumption, let us refer to it as WC. Note that the duration for which WC would execute a job whose realized (i.e., actual) execution time turns out to be equal to A is $\frac{A}{\text{speed}} = \frac{A}{W/D} = \frac{AD}{W}$ time units. Hence, the energy consumed in executing a job with actual execution time A, denoted $E_{\text{WC}}(A)$, is given by the following expression.

$$E_{\text{WC}}(A) \overset{\text{def}}{=} \left[\left(\frac{W}{D}\right)^{\alpha} \times \left(\frac{AD}{W}\right)\right] = \left[\left(\frac{W}{D}\right)^{\alpha} \times A \cdot \left(\frac{D}{W}\right)\right] = \left(\frac{W}{D}\right)^{\alpha-1} \times A \qquad (1)$$

2.3 Incorporating The Prediction

We now develop an algorithm, ALG, that makes use of the predicted value P of the actual execution time of the job. Recall that the goal in so doing is to have ALG consume less energy than WC if the prediction is correct, i.e., if the actual execution time happens to be equal (or close) to P, while consuming no more than γ_{ub} times the amount of energy consumed by WC even if the prediction is completely off.

Our prediction-oblivious algorithm of Section 2.2 had set the processor speed to equal W/D. Observe that it is unsafe for ALG to analogously replace the WCET W by the predicted execution duration P and simply set the speed equal to P/D, since it would then have completed exactly P units of execution by the deadline D. Hence if the prediction turns out to be an underestimation and the actual execution time exceeds P, ALG would cause a deadline miss to occur. Therefore ALG instead sets itself a *virtual deadline* t_v that is strictly before the actual deadline at D (i.e., $t_v < D$ – see Figure 1), and seeks to ensure that the job will complete at this virtual deadline if the prediction is correct (i.e., the job needs exactly P units of execution). Analogously to Algorithm WC in Section 2.2, this goal is achieved by setting the initial processor speed equal to exactly P/t_v. If during some execution the job completes prior to time-instant t_v, then it follows, by a reasoning analogous to that used in deriving Expression 1, that the energy consumed in ALG is equal to $\left((P/t_v)^{\alpha-1} \times A\right)$, where A again denotes the actual execution time (the AET) of the job on this particular execution. If however the job has <u>not</u> completed execution by its virtual deadline, ALG concludes that the prediction is incorrect and adjusts processor speed to ensure that the maximum amount of execution that could remain, $(W-P)$, will complete within the duration remaining prior to the actual deadline, $(D-t_v)$; it does this by increasing the processor speed to equal $(W-P)/(D-t_v)$. In this case the energy consumed over the interval $[0, t_v]$ equals $\Big((P/t_v)^{\alpha-1} \times P\Big)$ (since P units of work was done during this interval), while $(A-P)$ units of work is done for the remainder of the execution and so the energy consumed between time-instant t_v and the job's completion is equal to $\Big(((W-P)/(D-t_v))^{\alpha-1} \times (A-P)\Big)$. Putting the pieces together, we conclude that the energy consumed by ALG on executing an invocation of the job that has AET A is given by

$$E_{\text{ALG}}(A) = \begin{cases} \left(\frac{P}{t_v}\right)^{\alpha-1} \times A, & \text{if } A \leq P \\ \left(\frac{P}{t_v}\right)^{\alpha-1} \times P + \left(\frac{W-P}{D-t_v}\right)^{\alpha-1} \times (A-P), & \text{otherwise} \end{cases} \tag{2}$$

It remains to specify how the value of t_v is to be determined. Informally speaking, the slower the processor speed the less the energy consumed. Since our goal in designing algorithms that use predictions is to have excellent performance (in this example, minimal energy consumption) in the event that the prediction is correct, we would like the initial processor speed to be as small as possible – this is achieved by setting t_v to be as large as we possibly can. (Note from Expression 2 that $E_{\text{ALG}}(P)$ is smaller for larger t_v.) However in the event of the prediction turning out to be *in*correct, there should be adequate time remaining between the virtual deadline and the actual one to allow the job to be completed by increasing the processor speed. We have seen that increasing the processor speed requires an exponential increase in the power needed (and hence the energy consumed) – we must ensure that the total amount of energy consumed when the prediction is incorrect does not break the specified robustness bound (i.e., does not exceed γ_{ub} times the amount of energy that would be consumed by Algorithm WC to execute the same job). It is evident that since ALG executes at a greater speed (and hence less energy-efficiently) than WC for $A > P$, the ratio of $E_{\text{ALG}}(A)/E_{\text{WC}}(A)$ *increases* with increasing A, taking its maximal value when the job actually executes for a duration equal to its WCET (i.e., A takes on its largest possible value of W). The virtual deadline is therefore assigned the largest value of t for which

$$\begin{aligned} & E_{\text{ALG}}(W) \leq \gamma_{\text{ub}} \times E_{\text{WC}}(W) \\ \equiv \quad & \left(\frac{P}{t}\right)^{\alpha-1} \times P + \left(\frac{W-P}{D-t}\right)^{\alpha-1} \times (W-P) \leq \gamma_{\text{ub}} \times \left(\frac{W}{D}\right)^{\alpha-1} \times W \end{aligned} \tag{3}$$

Computing the virtual deadline t_v. We now discuss how the largest value of t satisfying Expression 3 is determined. For the special case where α, the exponent defining the relationship between the processor's speed and its power consumption, is equal to two (i.e., power equals speed-squared), Expression 3 may be algebraically rewritten as the quadratic inequality

$$\gamma_{\text{ub}} \times t^2 - \overbrace{\left((\gamma_{\text{ub}}-1) \times D + \frac{2PD}{W}\right)}^{-B} \times t + \overbrace{\left(\frac{PD}{W}\right)^2}^{C} \leq 0 \tag{4}$$

and hence, using the well-known formula for finding the roots of a quadratic equation, we get the following closed-form expression for t_v (with B and C denoting the expressions indicated in Eqn 4 above):

$$t_v = \frac{-B + \sqrt{B^2 - 4\gamma_{\text{ub}} C}}{2\gamma_{\text{ub}}} \tag{5}$$

For values of α other than two, we do not have a closed-form expression for t_v Instead we can obtain, via numerical methods, an arbitrarily close approximation of the true value of t_v in the following manner.

1. Since ALG should have its initial speed be no larger than the worst-case algorithm WC's initial speed, we should have $(P/t_v) \leq (W/D)$ or $t_v \geq (P/W) \times D$. This defines a lower bound on the value of t_v; D is obviously an upper bound.
2. Within this range, observe the monotonicity property of t satisfying Expression 3: if t_o satisfies Expression 3 than so do all $t < t_o$; if t_o does not satisfy Expression 3 then neither does any $t > t_o$.
3. We can therefore do binary search between $\big((P/W)\times D\big)$ and D to determine an arbitrarily close approximation of the largest value of t satisfying Expression 3.

We have assumed here the simplified power model: (power = speed$^\alpha$) to derive the value to be assigned to t_v. We point out that more complex models may be used – simply replace the one occurrence in the derivation of Eqn. 1 and the two occurrences in the derivation of Eqn. 2 of (speed$^\alpha$) by calls to a black-box implementation of the appropriate power function.

2.4 Learning an Improved Prediction

Thus far, we have assumed that our goal is to schedule the execution of a single job in an energy-efficient manner. It may be the case for certain application scenarios that this single job is invoked repeatedly in a periodic or sporadic manner. In such applications it may make sense to update the predicted execution duration (the parameter P) based on the observed actual execution times of previously-executed invocations. (Standard techniques, such as a PID controller or some appropriate Reinforcement Learning [15] method, may be used for this purpose.) But observe that P appears in Expression 3 which is used in computing the virtual deadline; hence, updating the value of P will require that the value assigned to the virtual deadline parameter t_v also be updated. That is, if the value of P is not a priori fixed then the value assigned to the virtual deadline can be looked upon as a function of P – this dependence of the virtual deadline on the value of P may be made explicit by referring to the virtual deadline as a function of P: $t_v(P)$. Computing $t_v(P)$ every time the value of P is updated may be computationally too expensive to do during run-time; hence we propose to precompute the values of $t_v(P)$ within a reasonable range of values for P, and store these values (alongwith the corresponding values of the speeds at which the processor should be run before and after t_v) in a lookup table for use during run-time.

2.5 An Example

We will now illustrate the concepts introduced above upon an example problem instance $\langle W, D, P, \alpha, \gamma_{\text{ub}}\rangle = \langle 8, 10, 5, 2, 1.1\rangle$. That is, we have a job with WCET $W = 8$, relative deadline $D = 10$, and a predicted execution duration $P = 5$, that is to be executed upon a processor with the power exponent $\alpha = 2$ (recall that this means that executing the processor at speed s requires power s^α). A robustness bound $\gamma_{\text{ub}} = 1.1$ is also specified: in using the prediction we may not exceed 1.1 times the energy consumed by the prediction-oblivious algorithm WC regardless of how inaccurate the prediction turns out to be.

Computing the virtual deadline. Since α happens to have value 2 in our example instance, we may solve for t_v exactly using Equation 5 to compute that $t_v = \mathbf{7.6}$.

Comparing the two algorithms. We have plotted the ratio of the energy that is consumed by the algorithm using predictions to that consumed by the prediction-oblivious algorithm in Figure 2. This plot reveals that ALG consumes approximately 82.2% of the energy that WC

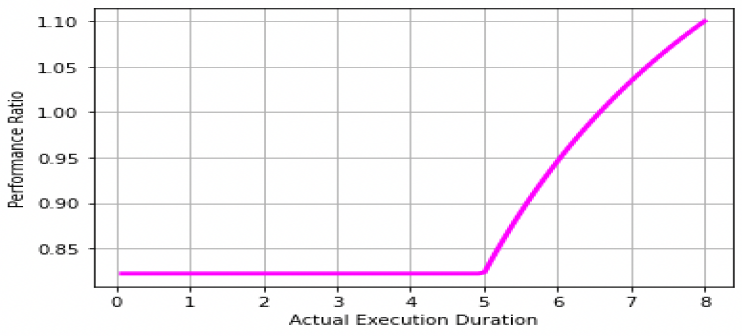

Figure 2 Ratio of energy consumed by ALG to the energy consumed by WC as a function of the actual execution time (AET), as AET ranges from 0 to $W = 8$.

consumes when the AET of the job is at (or below) its predicted execution duration. Thus, in this example *a perfect prediction results in an approximately* 17.8% *savings in consumed energy.* This same level of savings is also obtained with predictions that over-estimate the AET (i.e., when AET $< P$). And while the relative savings decreases as the AET increases beyond the predicted value, notice that the energy consumed by ALG remains below that consumed by WC as long as the AET is no larger than ≈ 6.575. Algorithm WC is more energy-efficient for values of AET that fall in the range $(6.575, 8]$; note, however, that the ratio never exceeds the specified bound of $\gamma_{\text{ub}} = 1.1$.

Updating the prediction. For application scenarios where the same job is invoked repeatedly at run-time, we had discussed in Section 2.4 how one could use on-line learning to *update* the value of the prediction P. We had stated that the value of the virtual deadline parameter t_v changes as P changes – Figure 3 depicts t_v as a function of P for our example. Prior to run-time a table of values for t_v as a function of P should be pre-computed and stored for use during run-time.

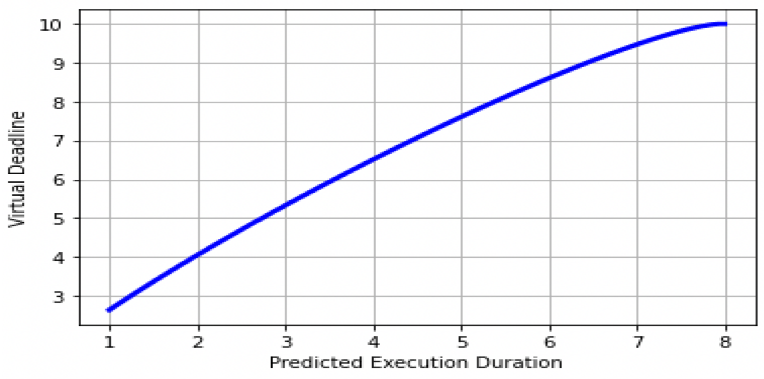

Figure 3 The virtual deadline $t_v(P)$ as a function of the predicted execution duration P.

2.6 Related Work and Future Directions

Although (as mentioned earlier) there is an extensive body of work dealing with energy-efficient scheduling, most of it does not consider predictions. We found only three papers [2, 3, 10] dealing with energy from amongst the 28 papers tagged with the keyword "scheduling" in the list of publications maintained on the Algorithms With Predictions web-site[4] ("energy" is not a searchable keyword there). In contrast to our model where we have just a single job, these papers all allow for multiple jobs; however they make simplifying assumptions regarding AETs that appear unrealizable in practice. Antoniadis et al. [2] assume that the exact execution duration of a job is known beforehand (and it is the arrival times and deadlines that are unknown and the subject of prediction), while Bamas et al. [3] assume that a job's AET becomes known at the instant of job arrival. These assumptions do not model the reality of the vast majority of embedded systems where the AET only becomes known by actually executing a job to completion. Lee et al. [10] consider a very different problem (that does not involve deadlines): minimize energy cost in data centres where the cost of energy depends upon both the volume and the peak usage, and where energy can be either locally generated or drawn from the grid.

Future work. In our opinion, the most obvious generalization of our model would be to allow for the specification of multiple jobs (eventually, even recurrent tasks). However exploiting predictions for energy-efficient scheduling becomes considerably more challenging when multiple jobs are to be considered – the interested reader is encouraged to glance through [3] to get a flavor for the complexities that arise even under the unrealistic assumption that predictions of a job's AET are revealed to hold or not upon job arrival (despite the other simplifying assumptions there, such as that all jobs have the same relative deadline, processor sharing is permitted, etc.).

3 Algorithms Using Predictions: The Foundations

The idea of "better than worst case" analysis has recently enjoyed much attention in theoretical computer science [14]. The motivation is that worst case instances are not common, and so we should design algorithms to also perform well in the common case while simultaneously maintaining worst case guarantees. One approach is to use predictions to guide an algorithm – these predictions may be generated via machine learning, human intuition, or other low assurance methods. Such predictions may be incorrect, therefore algorithms should not trust them entirely. The goal is to get the best of both worlds by improving performance when the prediction holds, without degrading performance by too much when the prediction fails. More specifically, these algorithms are characterized by three properties (formally defined a bit later in this section):

1. *Consistency* (Definition 4): When the predictions are correct, the algorithm provides very good performance for the instance.
2. *Robustness* (Definition 3): When the predictions are incorrect, the algorithm is not much worse than a good algorithm that does not use predictions.
3. *Smoothness* (Definition 5): The algorithm's performance degrades smoothly with the error in the prediction.

[4] `https://algorithms-with-predictions.github.io` (Accessed on 25/02/2023.)

In other words, the *consistency* of an algorithm using predictions characterizes its performance when the predictor is perfectly accurate, and its *robustness* its performance guarantee regardless of the quality of the predictions.

▶ **Example 1.** Let us revisit the example application of Sec. 2 – Fig 2 denotes the performance ratio of the algorithm using predictions, $\textsc{alg}$, to that of the prediction-oblivious algorithm WC. Note that the performance ratio is ≈ 0.822 when the prediction is exact – this is a quantitative measure of $\textsc{alg}$'s consistency. $\textsc{alg}$'s robustness follows from the observation that the performance ratio never exceeds the specified bound of $\gamma_{\text{ub}} = 1.1$. $\textsc{alg}$'s smoothness property is visually evident from Fig 2: the performance ratio remains ≈ 0.822 for AETs $\leq$ prediction, and degrades smoothly with increasing error for AETs $>$ the prediction. ⌟

The standard manner of characterizing the performance of any algorithm (not just those using predictions) designed to operate in an online setting is via *competitive analysis.* Consider some relevant performance objective that we want to minimize [5] – in the example in Section 2, this metric is energy.

▶ **Definition 2** (Competitive Ratio). *For any input X, let $\textsc{opt}(X)$ denote the cost achieved by an optimal clairvoyant algorithm and let $\textsc{alg}(X)$ denote the cost of the online algorithm $\textsc{alg}$ on this input. The competitive ratio of $\textsc{alg}$ is the ratio $(\textsc{alg}(X)/\textsc{opt}(X))$ maximized over all possible inputs X.*

One can think of the algorithms using predictions framework as a generalization of competitive analysis. In this framework, the online algorithm gets a prediction, say P. This prediction should be something that allows the algorithm to make better decisions than it would be able to in the absence of this prediction – for instance, in the example of Sec. 2, the prediction is the actual execution time (AET) of the job. We want the algorithm to perform well even with incorrect predictions; therefore, robustness is simply the competitive ratio of an algorithm which uses predictions.

▶ **Definition 3** (Robustness). *Given an online input X and prediction P for X, let $\textsc{alg}(X, P)$ denote the cost of the online algorithm using this prediction and $\textsc{opt}(X)$ the cost of the optimal clairvoyant algorithm. $\textsc{alg}$'s robustness is defined as the ratio $(\textsc{alg}(X, P)/\textsc{opt}(X))$ maximized over all values of X and P.*

The algorithms using predictions framework seeks to ensure that robustness of an algorithm is as close as possible to the best competitive ratio that can be achieved by any online algorithm for the problem.

Informally, *consistency* is the competitive ratio of the algorithm when the prediction is accurate – it measures how well the algorithm uses predictions to improve performance. In particular,

▶ **Definition 4** (Consistency). *Given an online input X and an exact prediction P for X, let $\textsc{alg}(X, P)$ denote the cost of the online algorithm using this prediction and $\textsc{opt}(X)$ the cost of the optimal clairvoyant algorithm. The consistency for the algorithm is defined as the ratio $\textsc{alg}(X, P)/\textsc{opt}(X)$ maximized over all values of X assuming P exactly matches the predicted value.*

[5] We can define this for maximization objectives in a similar manner.

There typically is a trade-off between consistency and robustness since one can think of robustness as hedging against an inaccurate prediction. In the energy example of Section 2, one can adjust this trade-off by making different decisions about the virtual deadline – a later virtual deadline may improve consistency while potentially being less robust and vice versa.

In addition to consistency and robustness, we want the algorithms to be able to gracefully handle small errors in predictions – this property is called *smoothness*. Smoothness is harder to define precisely and generally. Given an input X, say the perfect prediction for X is $\mathcal{P}_X$ – this is the prediction which one would generate if we knew X precisely. Say the actual prediction given the algorithm $\textsc{alg}$ is P. The error $\textsc{error}$ is some function of $\mathcal{P}_X$ and P – this function can be problem and algorithm specific, but informally is some difference function between the two parameters which is 0 when $P = \mathcal{P}_X$ and increases as the difference between the two increases. Ideally, we want algorithms whose competitive ratio relative to the optimal increases slowly as the error increases.

▶ **Definition 5** (Smoothness). *Given an online input X and a prediction P for X, let $\textsc{alg}(X, P)$ denote the cost of the online algorithm using this prediction and $\textsc{opt}(X)$ the cost of the optimal clairvoyant algorithm Let $\textsc{error}$ denote the error between the prediction P and the perfect prediction $\mathcal{P}_x$ for X. We say that the algorithm is $f(\textsc{error})$-smooth if $\textsc{alg}(X, P)/\textsc{opt}(X) \leq \beta + f(\textsc{error})$ for all $\textsc{error}$ and all X.*

We want the smoothness function $f(\textsc{error})$ to grow smoothly as $\textsc{error}$ moves away from 0. Therefore, smoothness is simply a generalization of consistency – most definitions of consistency are also in terms of the competitive ratio when the error is 0.

Adapting the framework. Above we have described the algorithms using predictions framework as it is currently defined (and used) by the optimization algorithms community. We now briefly discuss two ways in which this framework needs adaptation in order to render it more suitable for safety-critical systems.

1. Whereas the framework focuses on the trade-off between robustness and consistency, *we look upon robustness as a constraint* that must be met, with consistency (and smoothness) being metrics that we can optimize for/ trade off amongst. Although this is in one sense merely a restriction of the algorithms using predictions framework, both our examples (in Sec. 2 and the upcoming one in Sec. 4) show that this restriction fundamentally changes the process of applying the framework to specific problems.
2. The framework, as currently defined, primarily compares the performance of $\textsc{alg}$ with that of an optimal clairvoyant algorithm. But such comparison may not always be appropriate: consider, for instance, our energy-efficient scheduling application of Sec. 2. It is evident that no on-line algorithm can have a competitive ratio (Definition 2) smaller than infinity[6], and hence the framework as currently defined is not applicable to this application. But we saw in Sec. 2 that applying our adaptation, which *compares the performance of* $\textsc{alg}$ *with that of a reasonable prediction-oblivious algorithm*, yields significant energy savings.

[6] To see this, let s_o denote the initial speed at which $\textsc{alg}$ runs the processor; for very (small) AET A, it consumes $\left(s_o^{\alpha-1} \times A\right)$ units of energy. Meanwhile the clairvoyant $\textsc{opt}$, knowing the value of A, would execute at speed (A/D) and so consume $\left((A/D)^{\alpha-1} \times A\right)$ units of energy. The competitive ratio is thus $\left(\frac{s_o D}{A}\right)^{\alpha-1}$, which, for any fixed $s_o > 0$ (i.e., any initial speed chosen by $\textsc{alg}$), approaches ∞ as $A \to 0$.

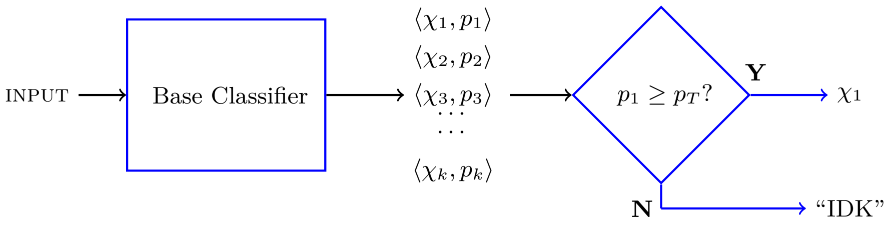
INPUT
Base Classifier
$\langle \chi_1, p_1 \rangle$
$\langle \chi_2, p_2 \rangle$
$\langle \chi_3, p_3 \rangle$
$\langle \chi_k, p_k \rangle$
$p_1 \geq p_T$?
Y
χ_1
N
"IDK"

Figure 4 Obtaining an IDK classifier from a base classifier. For a given input, the base classifier outputs up to k ordered pairs $\langle \chi_i, p_i \rangle$, indicating that it believes that the input belongs to the class χ_i with confidence score p_i. (It is assumed that $p_1 \geq p_2 \geq \cdots \geq p_k$, i.e., $\chi_1, \chi_2, \ldots, \chi_k$ are the k most likely classes, in order.) The threshold confidence parameter for the IDK classifier is set at p_T.

We point out that while the first adaption above can be looked upon as a restriction of the algorithms using predictions framework, the second adaptation *expands* the typical use-case for this framework and thereby enables its use in situations where the original framework is not directly applicable. This expanded applicability was in evidence in Sec. 2; next in Sec. 4, we will however not make use of this second adaptation, instead going back to the original framework and comparing with an optimal clairvoyant algorithm.

4 Illustrative Example: IDK Classifiers

A classifier is a software component that categorizes each input provided to it into one of a fixed set of classes. IDK classifiers, also known as classifiers that *defer* [7, 8, 12], are obtained from base classifiers that use deep learning and related AI technologies in the following manner (see Figure 4): if the base classifier is unable to arrive at a classification decision at an adequately high level of confidence, then a dummy class, IDK (for "I Don't Know") is output instead. In scenarios where classification is safety-critical, IDK classifiers should be used in conjunction with more traditional ("deterministic") classifiers for the same classification problem: if an IDK classifier outputs IDK upon some input, then the deterministic classifier is called upon that input to provide an authoritative classification. (Of course, it is only meaningful to use IDK classifiers in this manner if doing so is more efficient, in some sense or the other, than directly calling the deterministic classifier in the first place.)

As part of ongoing efforts to provide a scheduling-theoretic framework that allows for the use of LECs in hard-real-time safety-critical systems, the real-time scheduling theory community has recently (see, e.g., [1, 5, 6]) begun studying IDK classifiers. The three cited papers [1, 5, 6] consider variants of this problem: *given one or more IDK classifiers and a deterministic classifier for the same classification problem, how does one determine which of the classifiers to execute, and in what order, to minimize the expected duration needed to obtain a successful classification (perhaps within a specified hard deadline)?*

Abdelzaher et al. [1] describe how the training phase of IDK classifiers should be modified in order to estimate the probabilities that they will succeed in classifying an arbitrary input.[7] In this paper, we look upon the probability values so obtained as predictions of the true (unknown – perhaps unknowable) underlying probabilities, and apply the algorithms using predictions paradigm to deal with the possibility that these predictions are incorrect.

[7] Dependencies amongst classifiers can also be estimated by the method in [1]; e.g., what is the conditional probability that IDK classifier K_i returns "IDK" upon inputs for which classifier K_j has returned "IDK"?

The specific problem considered. Analogously to our approach in Section 2, we consider here a simplified version of the problem studied in [1, 5, 6] upon which to apply our adaptation of the algorithms using predictions framework. Specifically, we assume that we have available to us a single IDK classifier and a single deterministic classifier for the same classification problem, and must decide whether to (i) directly execute the deterministic classifier; or (ii) first call the IDK classifier, subsequently calling the deterministic classifier in the event that the IDK classifier returns "IDK." Our objective is to minimize the expected execution duration – i.e., the duration needed to successfully classify the input. If the probability of the IDK classifier returning a successful classification were a priori known, this problem would be trivial to solve: call the IDK classifier first if and only if its execution duration, plus the product of the probability it returns "IDK" times the execution duration of the deterministic classifier, is strictly smaller than the execution duration of the deterministic classifier. As stated above we assume, however, that we only have a predicted, rather than the actual, value of this probability. Specifically, let us suppose that the available pair of classifiers is characterized as a three-tuple $I \overset{\text{def}}{=} \langle C, \Pi, D\rangle$ where

- C is the execution duration[8] of the IDK classifier, and D is the execution duration of the deterministic classifier. (It is a reasonable assumption that $C < D$, since if $C \geq D$ there is no benefit, from the perspective of minimizing the expected execution duration, to executing the IDK classifier.)
- Π is the *predicted* probability that the IDK classifier will succeed (i.e., $(1 - \Pi)$ is the predicted probability that it will output "IDK").
  We point out that *the "real" probability that the IDK classifier will succeed, which we will denote as P, is unknown* to a non-clairvoyant algorithm. That is, Π is a prediction of the unknown (perhaps unknowable) value of P.

Given such a three-tuple $I \overset{\text{def}}{=} \langle C, \Pi, D\rangle$, an algorithm must decide whether to call the IDK classifier first (and call the deterministic classifier only if this returns "IDK"), or to directly call the deterministic classifier. The **performance metric** of an algorithm that makes this decision is the expected duration of its classification decision:

$$\begin{cases} C + (1 - P) \times D, & \text{if the algorithm calls the IDK classifier first} \\ D, & \text{if the algorithm directly calls the deterministic classifier} \end{cases}$$

The objective is to obtain a classification with minimum expected duration. We define the **prediction error** (see Sec. 3, the paragraphs between Definitions 4 and 5) to be the difference between the actual and the predicted success probabilities:

$$\text{ERROR}(I) \overset{\text{def}}{=} |\Pi - P| \tag{6}$$

A problem instance. As we have previously pointed out, in safety-critical systems it makes sense to place a bound on the "poorness" of acceptable solutions – in the formalism of algorithms using predictions as laid out in Sec. 3, to specify a bound on the robustness of the algorithm. Hence we specify a problem instance as (I, γ_{ub}) where I denotes the pair of classifiers: $I = \langle C, \Pi, D\rangle$, and γ_{ub} denotes the maximum acceptable robustness, defined here as the ratio of the expected execution duration for the algorithm using predictions

[8] For this application (in contrast to the one in Sec. 2), we make the simplifying assumption that the given parameters C and D represent the actual execution times (AETs) of the classifiers – this allows us to focus on LEC-specific aspects of prediction without getting bogged down revisiting the kinds of issues explored in Sec 2. The model is easily generalized to incorporate uncertainty in (and predictions of) the values of these AET's.

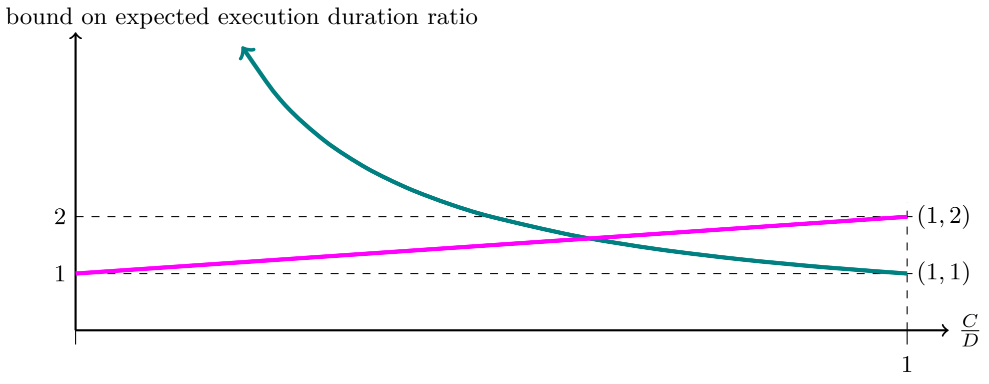

Figure 5 Bound on ratio of expected execution durations as a function of C/D. The straight line: use the IDK classifier. The curve: do not use the IDK classifier.

to the expected execution duration for an optimal algorithm (denoted OPT) that knows the value of P. Note that in contrast to the energy-efficient scheduling example of Sec. 2 where we had compared its performance of an algorithm designed to use predictions against the "reasonable" prediction-oblivious algorithm that we specified in Sec. 2.2, here we are comparing against an idealized clairvoyant optimal algorithm in OPT.

4.1 An Algorithm Using the Prediction

We will now design an algorithm, ALG (see Algorithm 1), that uses the predicted value Π to decide whether to directly execute the deterministic classifier, or to execute the IDK classifier first and the deterministic one only if the IDK classifier fails. We will first explore the constraints placed on ALG due to the robustness requirement: that its performance (expected duration) never exceed that of OPT by more than a factor γ_{ub}. Once we have identified the space of robust designs, we will next see how consistency considerations direct us to a choice within this design space; finally, we will characterize the smoothness characteristics of this choice.

Meeting the robustness constraint. ALG must make one of the two available choices: execute the IDK classifier first, or directly execute the deterministic classifier. Let us study the implications upon robustness of each choice.

1. If ALG executes the IDK classifier first, its expected duration is $C + (1 - P) \times D$. In the worst-case scenario for ALG, the value of P turns out to be equal to zero and OPT, knowing this, would directly execute the deterministic classifier for a duration of D. The performance ratio is therefore equal to $\left(1 + \frac{C}{D}\right)$ – this is plotted as a function of (C/D) in **magenta** as the increasing straight line in Fig. 5.
2. If ALG directly executes the deterministic classifier, then its duration is D. In the worst-case scenario for ALG, $P = 1$ and hence OPT would execute the IDK classifier for a duration of C. The performance ratio is therefore equal to $\left(\frac{D}{C}\right)$ – this is plotted as a function of (C/D) in **teal** as the decreasing curve in Fig. 5.

The plots of Figure 5 help us identify the space of acceptable designs for any given problem instance $(I = \langle C, \Pi, D \rangle, \gamma_{\text{ub}})$. For this instance, locate the position of the point with x-coordinate (C/D) and y-coordinate γ_{ub} in Fig. 5.

1. If this point lies below both the curves, then neither choice can guarantee to meet the specified robustness bound and hence we must **declare failure**.
2. If this point lies between the two curves, the choice of whether to directly call the deterministic classifier or to call the IDK classifier first is forced on ALG by the robustness bound. If the point lies to the left of the intersection of the two curves, then only calling the IDK classifier first can guarantee the robustness bound whereas if it lies to the right of the intersection, then only directly calling the deterministic classifier can guarantee the robustness bound.
3. However if this point lies above both curves, then either choice meets the robustness bound. *This is the only situation where an algorithm is free to use the prediction*; below we will discuss how it does so and makes a choice to achieve consistency.

Let $(\hat{x}, \hat{y})$ denote the point where the two curves in Fig. 5 intersect. We have $1 + \hat{x} = 1/\hat{x}$ and $\hat{y} = 1/\hat{x}$, from which it follows that $(\hat{x}, \hat{y}) = (1/\varphi, \varphi)$ where φ denotes the famous constant commonly called the Golden Ratio: $\varphi = (1 + \sqrt{5})/2 \approx 1.618$. Theorem 6 follows:

▶ **Theorem 6.** *(i)* ALG *never returns failure for problem instances with $\gamma_{\text{ub}} \geq \varphi$; and (ii) the prediction is never useful for instances with $\gamma_{\text{ub}} \leq \varphi$, where φ denotes the Golden Ratio: $\varphi = (1 + \sqrt{5})/2$.* ◢

Achieving Consistency. As we have observed above, the prediction is meaningful only for systems $(I = \langle C, \Pi, D \rangle, \gamma_{\text{ub}})$ for which the point $(C/D, \gamma_{\text{ub}})$ lies above both curves in Fig. 5. Recall (Definition 4) that consistency characterizes the performance of an algorithm using predictions when the prediction is exactly correct; for our example, that translates to the case when $\Pi = P$. If $\Pi = P$, it is evident that ALG should use the IDK classifier iff

$$\Big(C + (1 - \Pi) \cdot D < D\Big) \Leftrightarrow \Big(C + D - \Pi\, D < D\Big) \Leftrightarrow \left(\mathbf{\Pi} > \frac{\boldsymbol{C}}{\boldsymbol{D}}\right)$$

An ALG that executes the IDK classifier iff $\big(\Pi > (C/D)\big)$ clearly has consistency equal to one: when the prediction is exact, it has exactly the same expected duration to successful classification as OPT does.

Smoothness Analysis. We now analyze the smooth properties (Definition 5) of the algorithm obtained above (for convenience, also listed in pseudo-code form in Algorithm 1). Since the predictions are only usable when $\gamma_{\text{ub}} > \max(D/C, (1 + C/D))$, we restrict our analysis of smoothness to instances satisfying this property. In Figure 6, we plot the expected duration of both ALG (in blue) and OPT (in red) as a function of the true probability of successful classification by the IDK classifier, P, for both the cases when ALG chooses to first execute the IDK classifier (top left), and when it directly executes the deterministic classifier (bottom left). Let us separately consider the two cases.

1. For ALG to choose to use the IDK classifier (top left in Fig. 6), it must be the case that $\Pi > C/D$. Now when P is also $\geq C/D$, ALG's performance is the same as OPT's. On the other hand when $P < C/D$, the performance ratio is given by

$$\frac{C + (1 - P) \times D}{D} = \frac{C}{D} + (1 - P) = 1 + \left(\frac{C}{D} - P\right)$$

Meanwhile, ERROR $= \Pi - P > \left(\frac{C}{D} - P\right)$; plugging into the above expression, we have performance ratio

$$= 1 + \left(\frac{C}{D} - P\right) < 1 + \text{ERROR}$$

Algorithm 1 The algorithm using predictions ALG. (The prediction finds use in Lines 10-13.)

```
Input: $(I, \gamma_{\text{ub}})$
1  /*$I \overset{\text{def}}{=} \langle C, D, \Pi \rangle$; $\gamma_{\text{ub}}$ is the maximum permissible robustness
2  if $\left(\gamma_{\text{ub}} < \min\left(1 + \frac{C}{D}, \frac{D}{C}\right)\right)$ then /*Below both lines in Figure 5
3      return failure
4  if $\left(\min(1 + \frac{C}{D}, \frac{D}{C}) \leq \gamma_{\text{ub}} \leq \max\left(1 + \frac{C}{D}, \frac{D}{C}\right)\right)$ then /*Between the lines in Figure 5
5      if $\left(1 + \frac{C}{D}\right) \leq \frac{D}{C}$ then
6          return execute the IDK classifier first
7      else
8          return execute the deterministic classifier
9  if $\left(\gamma_{\text{ub}} > \max\left(1 + \frac{C}{D}, \frac{D}{C}\right)\right)$ then /*Above both lines in Figure 5
10     if $\left(\Pi > \frac{D}{C}\right)$ then
11         return execute the IDK classifier first
12     else
13         return execute the deterministic classifier
```

Observe that ALG's performance loss when compared to OPT is *one-sided* – there is no performance loss unless $P < \Pi$ (i.e., Π is an over-estimation of the true probability P). The top right plot of Fig 6 plots the worst-case relative performance as a function of error, as the error ranges over $[0, C/D]$.

2. We can similarly consider the situation where ALG decides against using the IDK classifier (bottom left in Fig. 6). It must then be the case that $\Pi \leq C/D$. If P is also $\leq C/D$, then the performance ratio is one. If $P > C/D$, then the performance ratio is given by

$$\frac{D}{C + (1-P) \times D} = \frac{1}{\left(\frac{C}{D}\right) + (1-P)} = \frac{1}{1 - \left(P - \frac{C}{D}\right)}$$

Meanwhile, $\text{ERROR} = P - \Pi > P - \left(\frac{C}{D}\right)$; plugging into the above expression, we have performance ratio

$$= \frac{1}{1 - \left(P - \frac{C}{D}\right)} < \left(\frac{1}{1 - \text{ERROR}}\right)$$

Once again, ALG's performance loss when compared to OPT is one-sided – there is no loss unless $P > \Pi$ (i.e., Π is an under-estimation of the true probability P). The bottom right plot of Fig 6 plots the worst-case relative performance as a function of error, as the error ranges over $[0, 1 - (C/D)]$.

By visual inspection of the right-hand plots in Fig. 6, it is clear that the relative performance of ALG does indeed degrade smoothly with error. (When ALG chooses to use the IDK classifier – top right in Fig. 6 – the performance degrades linearly with error, but that is not the case when ALG directly uses the deterministic classifier – bottom right in Fig. 6.)

4.2 Related Work and Future Directions (on IDK Classifiers)

While several papers [1, 5, 6] dealing with IDK classifiers have recently appeared in the real-time literature, we are not aware of any prior work that integrates predictions with IDK classifiers. We nevertheless consider the prior papers [1, 5, 6] to be very relevant to our work

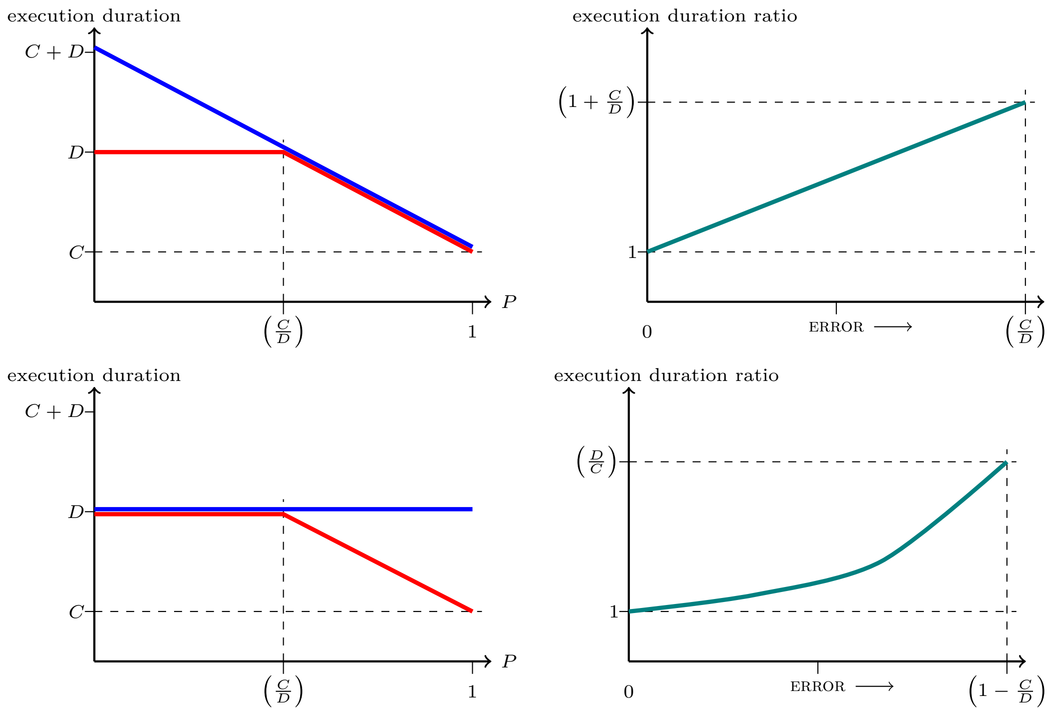

Figure 6 Smoothness. Blue is ALG; red is OPT. **Top**: ALG uses the IDK classifier (and so $\Pi > C/D$). **Bottom**: ALG does not use the IDK classifier ($\Pi \leq C/D$).

(and closely related to it) since they, Abdelzaher et al. [1] in particular, directly address the issue of obtaining a probabilistic characterization of run-time behavior of the classifiers by detailing procedures for determining the probability values – these are the "predictions" of our model. These papers also consider cascades of multiple classifiers (rather than just two, one IDK and one deterministic, as we have done), and show that one can in general obtain smaller expected duration to classification via such longer cascades. As future work, we plan to extend the methodology we have proposed in this manuscript to cascades of > 2 classifiers.

Another direction of future research involves incorporating learning. In our energy-efficient scheduling example of Sec. 2, we saw that if the job under consideration is repeatedly invoked and predictions can therefore be improved via a process of on-line learning, how doing so may allow for further efficiency gains (Sec. 2.4). But incorporating learning is not straightforward for IDK's even if the classification problem is repeatedly invoked: if $\Pi \leq C/D$, then ALG recommends against the use of the IDK classifier and we will never have a chance to update the prediction (the value of Π). Therefore some more sophisticated learning framework must be applied. We may, for example, consider using RL-based learning [15] to occasionally force ALG to call the IDK classifier, but for this we can only guarantee robustness over a sequence of classification decisions rather than on each individual classification.

5 Context and Conclusions

Sources of uncertainty abound in safety-critical real-time CPS's that operate within complex environments – these include traditional sources such as variation in run-time execution duration (the "WCET problem" [16]), and newer ones arising from the use of learning-enabled components in such systems. We believe that the recently proposed algorithm-design paradigm of *Algorithms using Predictions* may be a good fit for dealing with such uncertainty, and should therefore be considered for use in safety-critical real-time systems – this manuscript reports on our efforts at doing so. Most prior work on algorithms using predictions dealt with pure optimization problems, and studied trade-offs between consistency (making good use of accurate predictions) and robustness (not suffering too much performance loss when predictions turn out to be incorrect). Since safety-critical systems are characterized by hard constraints, we had to adapt the framework and accord primacy to robustness: given an acceptable degree of robustness, identify a design space of acceptable algorithms and determine the algorithm offering maximum consistency within this space (and then characterize the smoothness of this algorithm). To demonstrate the usefulness of this framework, we have used it to design algorithms for a pair of applications that are of current interest to the real-time computing community; for both, we have shown that algorithms using predictions achieve improved performance when accurate predictions are available, without being penalized beyond pre-specified robustness bounds if the predictions go wrong.

As stated earlier, there are multiple sources of uncertainty in modern complex safety-critical real-time CPS's in addition to the two we have considered here. For each such source, the availability of good predictions could potentially be used to improve performance. For instance, in many event-triggered systems where triggering events may occur repeatedly, the *period* parameter [11, 13] specifies the minimum duration between successive triggerings. Current scheduling techniques usually allocate computational resources assuming that successive triggerings will occur at the maximum rate (i.e., separated by exactly the period parameter); if predictions are available of a larger inter-triggering duration, the algorithms using predictions framework could perhaps be used to design algorithms that make efficient use of computational resources when such predictions are correct, without catastrophic consequences if they turn out wrong.

References

1 Tarek Abdelzaher, Kunal Agrawal, Sanjoy Baruah, Alan Burns, Robert I. Davis, Zhishan Guo, and Yigong Hu. Scheduling IDK classifiers with arbitrary dependences to minimize the expected time to successful classification. *Real-Time Systems (to appear)*, 2023. URL: `https://www-users.york.ac.uk/~rd17/papers/IDKarbitrary.pdf`.

2 Antonios Antoniadis, Peyman Jabbarzade, and Golnoosh Shahkarami. A Novel Prediction Setup for Online Speed-Scaling. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*, volume 227 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.SWAT.2022.9`.

3 Etienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15350–15359. Curran Associates, Inc., 2020. URL: `https://proceedings.neurips.cc/paper/2020/file/af94ed0d6f5acc95f97170e3685f16c0-Paper.pdf`.

4 Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. Embed. Comput. Syst.*, 15(1), January 2016. `doi:10.1145/2808231`.

5 Sanjoy Baruah, Alan Burns, Robert Davis, and Yue Wu. Optimally ordering IDK classifiers subject to deadlines. *Real Time Syst.*, 2022. `doi:10.1007/s11241-022-09383-w`.

6 Sanjoy Baruah, Alan Burns, and Yue Wu. Optimal synthesis of IDK-cascades. In *Proceedings of the Twenty-Ninth International Conference on Real-Time and Network Systems*, RTNS '21, New York, NY, USA, 2021. ACM.

7 C. K. Chow. An optimum character recognition system using decision functions. *IRE Transactions on Electronic Computers*, EC-6(4):247–254, 1957. doi:10.1109/TEC.1957.5222035.

8 Corinna Cortes, Giulia DeSalvo, and Mehryar Mohri. Learning with rejection. In Ronald Ortner, Hans Ulrich Simon, and Sandra Zilles, editors, *Algorithmic Learning Theory*, pages 67–82. Springer International Publishing, 2016.

9 Bartloomiej Kocot, Pawel Czarnul, and Jerzy Proficz. Energy-aware scheduling for high-performance computing systems: A survey. *Energies*, 16(2), 2023. URL: https://www.mdpi.com/1996-1073/16/2/890.

10 Russell Lee, Jessica Maghakian, Mohammad Hajiesmaili, Jian Li, Ramesh Sitaraman, and Zhenhua Liu. Online peak-aware energy scheduling with untrusted advice. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, e-Energy '21, pages 107–123, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3447555.3464860.

11 C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

12 David Madras, Toniann Pitassi, and Richard Zemel. Predict responsibly: Improving fairness and accuracy by learning to defer. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 6150–6160, Red Hook, NY, USA, 2018. Curran Associates Inc.

13 Aloysius Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

14 Tim Roughgarden, editor. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020. doi:10.1017/9781108637435.

15 Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

16 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36:1–36:53, May 2008.

17 F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In IEEE, editor, *36th Annual Symposium on Foundations of Computer Science: October 23–25, 1995, Milwaukee, Wisconsin*, pages 374–382, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1995. IEEE Computer Society Press.

18 Tianming Zhao, Wei Li, and Albert Y. Zomaya. Real-time scheduling with predictions. In *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*, pages 331–343. IEEE, 2022. doi:10.1109/RTSS55097.2022.00036.