Hybrid Planning Using Learning and Model Checking for Autonomous Systems

Ashutosh Pandey, Ivan Ruchkin, Bradley Schmerl, David Garlan
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA, USA
{iruchkin,ashutosp}@alumni.cmu.edu, {garlan, schmerl}@cs.cmu.edu

***Abstract*—Self-adaptive software systems rely on planning to make adaptation decisions autonomously. Planning is required to produce high-quality adaptation plans in a timely manner; however, quality and timeliness of planning are conflicting in nature. This conflict can be reconciled with *hybrid planning*, which can combine reactive planning (to quickly provide an emergency response) with deliberative planning that take time but determine a higher-quality plan. While often effective, reactive planning sometimes risks making the situation worse. Hence, a challenge in hybrid planning is to decide whether to invoke reactive planning until the deliberative planning is ready with a high-quality plan. To make this decision, this paper proposes a novel *learning-based approach*. We demonstrate that this learning-based approach outperforms existing techniques that are based on specifying fixed conditions to invoke reactive planning in two domains: enterprise cloud systems and unmanned aerial vehicles.**

***Index Terms*—automated planning; machine learning; probabilistic model checking**

I. Introduction

Modern enterprise software systems have to provide high availability and optimal performance in spite of changing environments, faults, and attacks. Managing these systems manually can be costly, error prone, and difficult to scale; therefore, these systems are becoming increasingly autonomous. For autonomous systems, the ability to plan is one of the key requirements to decide how to adapt. To this end, researchers have proposed various planning techniques, including rule-based adaptation, case-based reasoning, fuzzy-logic, reinforcement learning, stochastic search (using genetic algorithms), and optimization on probabilistic models (e.g., Markov Decision Processes (MDPs)). We use the term "planning" in a broad sense, referring to any decision-making approach that could be used to determine adaptation plans. The goal of planning is to solve *planning problems* — descriptions of a planning context (a system and its environment) and a goal (a state or an objective function) for planning purposes.

For many autonomous systems, *quality* and *timeliness* are two particularly important requirements to be considered when planning. Here the "quality" of planning refers to the likelihood of a plan meeting the (predefined) adaptation goals. In many domains, a poor quality plan can lead the system into an irreparable failure that endangers lives (e.g., in safety critical systems), or loss of revenue and failure of business goals (e.g., enterprise systems). A high quality plan needs to be ready in time to achieve its adaptation goals. For instance, if an enterprise system fails to produce a timely defense plan in response to an attack trying to extract sensitive data, the system risks being compromised even if the plan itself was high quality (e.g., because assets may have been exfiltrated while the plan was being constructed) [1].

Fundamentally, *quality* and *timeliness* are conflicting requirements: producing higher-quality plans is likely to take more time. Furthermore, the time to produce quality plans increases significantly in larger search spaces resulting from complex environments, large numbers of components, adaptation options, and qualities of interest. To address this quality-timeliness tradeoff, one approach, referred to as *hybrid planning*, combines multiple planning components with different quality-time tradeoffs [2]. When a time-critical adaptation is needed, "fast" (*reactive*) planning determines a quick (but potentially sub-optimal) plan, while "slow" (*deliberative*) planning computes a better plan that takes over once it is ready.

A key obstacle to the adoption of hybrid planning is deciding when to invoke reactive planning. It may be a high-stakes decision: waiting may lead to system failure, whereas reactive planning may make a quick but bad decision that makes the situation worse – not better – and may lead to failure as well.

To avoid an inappropriate invocation of reactive planning, one might specify up-front conditions under which it should be invoked. For example, Netflix used reactive planning only in emergency situations when waiting for a deliberative plan is not advisable (e.g., unexpected surge in workload) [3]. This approach might reduce the risks of inappropriate quick decisions, but suffers from two drawbacks: (a) it requires deep domain expertise to identify the conditions; and (b) it relies on error-prone human judgment to identify the right and comprehensive conditions, which can be difficult for a complex system with multiple conflicting requirements. For instance, such rules may be conservative and avoid reactive planning when they might, in fact, be useful.

To overcome these drawbacks, as the first contribution, this paper proposes a supervised *machine learning-based* (LB) approach to decide whether to invoke reactive planning in combination with deliberative planning or wait (i.e., no adaptation) until a deliberative plan is ready. In the training phase, using planning problems similar to the ones expected at run time, a classifier is trained to choose between invoking reactive planning and waiting. At run time, depending on how the current situation (i.e., the planning problem at hand) relates to problems in the training set, the classifier decides whether to invoke reactive planning or wait. As we will see, this approach

978-1-7281-7277-4/20/$31.00 ©2020 IEEE
DOI 10.1109/ACSOS49614.2020.00026

overcomes disadvantages of condition-based (CB) invocation of reactive planning by removing the need for humans to determine the specific conditions at design time, and by being applicable to a broad range of systems/domains.

To train a classifier, one needs a set of labeled training problems such that the label of a problem indicates whether reactive planning, in combination with deliberative planning will provide a higher performance compared to just waiting until a deliberative plan is ready. Labeling a planning problem requires one to evaluate and compare the performance of (a) the reactive plan (determined by reactive planning for the problem) followed by the deliberative plan, and (b) waiting followed by the deliberative plan.

Obtaining correctly labelled training data for a classifier is challenging for real systems. To label one planning problem, one would have to repeatedly put the system and its environment in the same state to test the two combinations as mentioned above. This is non-trivial, particularly in domains with uncertain dynamics, which is often the case in modern systems. In such domains, the environment evolution and the outcomes of the system's actions may change between attempts, so one would have to perform multiple trials of the same combination to determine the best average outcome. For example, suppose a self-adaptive cloud-based system proactively adds a server, anticipating an increase in future workload. However, if the workload suddenly decreases, adding the server is counterproductive. For such a system, an approach is needed that can take uncertainty (e.g., possible changes in workload) into account, preferably, in a single run (to save time/effort) when evaluating a combination.

To this end, as the second contribution, we employ *probabilistic model checking* to estimate the performance of reactive planning and waiting in combination with deliberative planning over all possible execution paths in a planning problem with a single run of a model checker. For the estimation, we encode a combination and the problem in a probabilistic model checker specification and use it to calculate the expected performance of the combination. By comparing the performance of the two combinations, one can choose the best combination for the problem and label it with "use reactive" or "wait". The probabilistic nature of model checking helps to account for uncertainty when evaluating a combination of reactive and deliberative planning. We expect existing probabilistic model checkers to ease adoption, automation, and reuse of the learning-based approach.

The paper validates the learning-based (LB) approach (and implicitly the labeling process) using simulations of two realistic systems: a cloud-based self-adaptive system [4] and a team of unmanned aerial vehicles [5]; as detailed later, these systems differ in their ability to recover from poor/delayed actions. In addition, the systems use a different set of reactive and deliberative approaches to instantiate hybrid planning. In both cases, we compared the performance of the LB approach against condition-based approach and found that, on average, the former is preferable to the latter. The better performance of the LB approach also indicates that model checking was able to appropriately label sample problems. Moreover, an empirical analysis of the data revealed that the performance of hybrid planning is correlated to the performance of (i) deliberative planning, and (ii) the relatively better-performing approaches amongst the reactive ones. The details to replicate the evaluation process are provided in the supplementary material [6]. These findings can inform engineers who need to prioritize their investment of resources in planners, instead of exploring many possible combinations.

II. Background and Related Work

For many self-adaptive systems, using a single (i.e., either a reactive or a deliberative) planning approach can be problematic [7]. Hybrid planning (HP) seems to be a promising way to balance quality and timeliness of planning. However, our formal analysis of hybrid planning [8] highlights two fundamental challenges:

Planning Coordination (PlnCrd): Hybrid planning requires a smooth transition from a reactive plan to a possibly higher-quality deliberative plan. Suppose a system observes an emergency situation, and, as a result, invokes reactive planning to provide a quick response. For a seamless transition from a reactive plan to a deliberative plan, two conditions need to be met [2]: (1) *timing* – the deliberative plan should be ready at the moment of transition; and (2) *preemption* – that the deliberative plan should contain state of the system at the point of the transition. Satisfying these two conditions is challenging for two reasons: (a) uncertainty about deliberative planning time makes it difficult to predict when the deliberative plan will be ready, and (b) uncertainty in the system's environment makes it difficult to predict the expected system and environment state after executing the reactive plan.

Planning Selection (PlnSel): Assume that deliberative planning provides better plans compared to reactive planning. Formally, given set Ξ of all planning problems for a system and set $\mathcal{F}$ of reactive planning approaches, solving the PlnSel problem means approximating function $\mathcal{G} : \Xi \rightarrow \mathcal{F}$ suggesting which reactive approach should be invoked for a planning problem $\xi \in \Xi$.[1] Set $\mathcal{F}$ has a special element (i.e., ρ_{wait}) that, for any planning problem, always suggests to wait until the deliberative plan is ready; therefore, using ρ_{wait} in combination with deliberative planning is equivalent to using deliberative planning alone. ρ_{wait} is required to ensure that hybrid planning does not underperform deliberative planning in cases when none of the other reactive approaches (in $\mathcal{F}$) provide a better plan than just waiting for the deliberative plan to be ready; as supported by our evaluation results (cf., Section V-C), in certain situations, it is prudent to just wait. This paper focuses on choosing between a reactive approach and waiting until a deliberative plan is ready, therefore set $\mathcal{F}$ has only two elements: the reactive approach and ρ_{wait}.

To solve PlnCrd, we adopt the approach suggested in prior work [9]. The approach has two distinguishing characteristics: (a) deliberative planning generates a high-quality universal plan/policy (one containing state-action pairs for all the reachable states from the initial state), where a mapping from a state (say s) to an action (say a) suggests a be executed in

[1]The choice of using deliberative planning followed by reactive planning is not considered since if a deliberative plan is ready to take over, it will provide a higher utility compared to a plan determined by reactive planning.

s; and (b) the operating domain is assumed to be *Markovian*: the state after a transition depends only on the current state — not on the sequence of states that preceded it. These two characteristics increase the chances of successful preemption if reactive and deliberative planning use the same initial state. That is, once the deliberative plan is ready, it can take over plan execution from the reactive plan because any state resulting from executing the reactive plan will be found in the deliberative plan. However, in reality, there is still a possibility that transition between the plans might fail due to violating the timing or preemption condition, thus affecting the quality of adaptation (for more details, see Section V).

To solve PLNSEL, which is the focus of this paper, Mausam et al. [10] proposed to always use reactive planning and, if required, revise the plan using a more deliberative planning approach. They assume that reactive planning will always improve the current situation. This assumption worked because their hybrid planning instantiation is limited to either a specific combination of planners or a particular domain. However, this assumption might not always hold since, as the formalism of hybrid planning suggests, it depends on the quality of a reactive planner and the nature of an operating domain [2]. Our proposed learning-based (LB) based approach can be applied to different domains and different combinations of planners as demonstrated in Section V.

Researchers have proposed various condition-based (CB) approaches. For instance, Pandey et al. [9], and Ali-Eldin et al. [11] proposed hybrid controllers in the context of self-adaptive cloud systems; these instantiations of hybrid planning use threshold-based rules to invoke reactive planning. Bauer et al. [7] extended this idea with more sophisticated conditions. While CB approaches have shown to be effective, as noted earlier identifying these conditions at design time can be difficult, particularly, for complex systems having multiple interacting dimensions of concern (e.g., cost, performance, security). We overcome these drawbacks with the LB approach.

III. LEARNING-BASED PLANNING SELECTION

This section introduces a novel approach to PLNSEL – deciding which reactive approach to invoke for a given problem. Our approach has two phases: *offline* and *online*. During the offline phase, the first step is to collect/identify a training set of planning problems similar to the ones expected at run time. In the second offline step, using a probabilistic model checker, these problems are labelled with the preferred reactive approach to provide an instantaneous response. The third and last offline step is to decide appropriate features in the training set and use them to train a machine learning classifier, which will determine the best reactive planner at each moment. In the online phase, when facing a planning problem ξ (representing the current situation) at run time, the system invokes the classifier on the features of ξ. The classifier picks a reactive planner, which is used by the system until a deliberative plan is ready.

1) The offline phase: In the offline phase, a classifier is built on planning problems that the system expects to observe at run time. The three offline steps are (a) identify sample planning problems to profile the hybrid planner; (b) profile the hybrid planner on these problems to know the label (i.e., which reactive approach is likely to outperform others); and (c) select features and hyper-parameter values to train a classifier.

(1a) Identifying Sample Problems: To select reactive planners effectively, it is crucial to cover the planning problem space comprehensively. However, identifying a set of representative problems is challenging due to a potentially infinite problem space and its unknown structure. No single selection strategy fits all systems and domains, and we suggest tailoring the sample set to the system's context and requirements. Fortunately, modern systems produce large amounts of data that are available to train a classifier. For instance, in our evaluation systems, we mine sample planning problems from the available traces containing the typical system load patterns [12] (for the cloud-based system) and randomly sample the space of missions (for the UAV domain).

(1b) Labeling the Sample Problems: This step determines the reactive approach $\rho_r^i \in \mathcal{F}$ that performs best in combination with deliberative planning for a sample planning problem ξ, and labels it accordingly (i.e., ρ_r^i). At the end of this step, we obtain a set of labelled training data, which is critical to (supervised) learning in our LB approach. However, in the presence of environment uncertainty (which is often the case for realistic systems), it is difficult to evaluate a combination given that its performance may vary across plan executions (for the same problem) because of different possible outcomes leading to different plan execution paths. For example, suppose a self-adaptive cloud-based system proactively adds a server anticipating an increase in the future workload (i.e., the number of requests received by clients). However, if the workload increases or decreases further, adding the server might not have the desired effect. Therefore, an approach is needed that can take uncertainty into account when evaluating the combination.

To overcome this problem, we propose using *probabilistic model checking*, which considers stochastic uncertainty when evaluating a combination of reactive and deliberative plans. By constructing a model for a model checker, one can encode the combined execution of a reactive plan followed by the deliberative plan (produced by the respective approaches) for some planning problem; the model checker returns expected utility of this execution. Given a planning problem, a finite set of reactive approaches, and a deliberative approach, this process is repeated for each reactive approach to evaluate its combination with the deliberative approach for the problem. Model checking helps label training problems by evaluating plan combinations under probabilistic uncertainty, by considering all possible execution paths weighted by their probabilities. Here, we assume that different conflicting quality attributes for a self-adaptive system can be represented as a multi-dimensional utility function such as Equations 1 and 2 in Section IV, and that the planning goal is to maximize expected utility. In other words, the quality of a (combined) plan can be assessed based on the utility it is expected to provide.

Figure 1 shows how a model checker can be used to evaluate the combination of reactive (ρ_r^i, producing plans π_r^i) and deliberative (ρ_d, producing plan π_d in time t_d) planning. The outcomes of executing actions from each plan are uncertain,

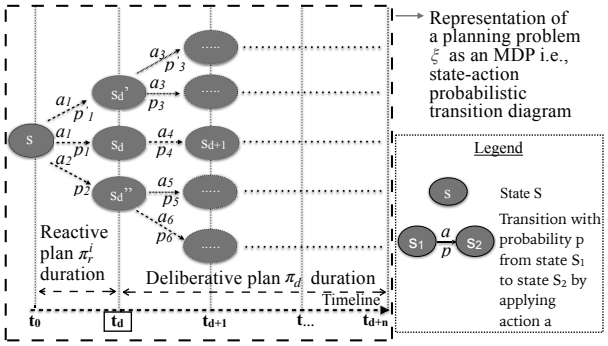

Figure 1: Evaluating reactive approach ρ_r^i by calculating the utility for a reactive plan followed by a deliberative plan.

and a model checker handles this uncertainty by aggregating the quality of possible outcomes as expected utility, denoted U_r^i. Since planning time t_d is difficult to predict upfront, and not guaranteed to remain fixed for different sample problems (or even the same one in different runs), in practice, one can configure the worst-case planning time (t_d) chosen as an over-approximation after a large number of trial runs for sample problems as we did in our experiments (cf., Section V-B).

To compute U_r^i for ξ, the model checker calculates the expected utility for the combination of plans π_r^i (until time step t_d) and then π_d. If set $\mathcal{F}$ has *N* reactive planners, then each sample problem ξ requires *N* evaluations corresponding to each $\rho_r \in \mathcal{F}$. For probabilistic model-checking, our evaluation uses PRISM [13]; it can model check MDP-based planning problems, which is the case for the two evaluation systems (cf., Section IV). However, our approach is not limited to any specific model checker.

Finally, we need to compare expected utilities for each combination. For problem ξ, suppose the plan determined by $\rho_r' \in \mathcal{F}$ (in combination with the deliberative plan) provides the highest utility, ξ is assigned the label corresponding to ρ_r'. In cases where more than one reactive approach provides the highest utility, any one approach can be chosen. Thus, each sample problem is labeled with one of the *N* labels, given *N* reactive approaches. This approach can be naturally extended to also support any number of deliberative approaches (rather than one); basically, the labeling process can help in deciding the best pair of reactive and deliberative approaches.

(1c) Training a Classifier: The first step to train a classifier on the labeled problems is to identify relevant features of planning problems that help separate the *N* classes. We use two complementary sets of features: ones representing the current state of the system, and ones describing how the system will evolve in the future. As an example, for a cloud-based system, state variables such as the number of active servers can be used to capture the current state. To capture future evolution, one can use real-time predicted request arrival rates for the future within the planning horizon [14]. These features reasonably capture a planning problem, which has current (i.e., initial) state and transitions as the fundamental elements. Once features are identified, we use cross-validation on the sample problems to train and test different classifiers; we pick the classifier that provides the best performance during cross-validation.

2) The online phase: When a self-adaptive system requires planning (e.g., periodically or in response to a fault), it formulates a planning problem ξ. The offline-trained classifier is used on ξ to assign the label corresponding to an appropriate $\rho_r \in \mathcal{F}$. It is necessary that the classifier is near-instantaneous — otherwise the classification delay makes the profiling scenario too dissimilar from the online scenario.

The proposed LB approach overcomes the limits of relying on predefined conditions to choose among reactive approaches. Now, domain expertise is not necessary to decide which reactive approach needs to be invoked. Instead, engineers can rely on planning problems encountered in the past to answer the same question, without committing to specific up-front conditions. Moreover, full/partial automation is possible for the LB approach, which can relieve designers from the painstaking and error-prone process of identifying the conditions.

IV. Systems for Evaluation

Section V evaluates the LB approach and, implicitly, the labeling process using model checking. The key question investigated in the evaluation is, *"how effective is the LB approach compared to the CB?"* Effectiveness is a measure of a system's ability to meet its adaptation goal, which is encoded in a multidimensional utility function as presented later in Formulas 1 and 2. To compare the approaches, we conducted controlled experiments by keeping all the experimental parameters constant except the planning approach (LB, CB, ...) and the traces (cf., Section IV-A) or missions (cf., Section IV-B) used as inputs for the two systems. We controlled the parameter values to isolate the effects of the planning approach (independent categorical variable) on the utility (dependent ordinal variable).

This evaluation is done using two different systems: a cloud-based load balancing system and a team of UAVs on a reconnaissance mission; the differences are discussed in Section VI. These systems are used because balancing timeliness and quality of planning is critical to their success, and developing a single planning approach from scratch can be challenging. The systems let us investigate different compositions of constituent planners, which vary in their action sets, planning horizons, and treatment of uncertainty.

A. The Cloud-based Load Balancing System

As the first system, we adopted a cloud-based load balancing system that we used to evaluate CB hybrid planning in our previous work [9]. We hope that in the future, cloud-based systems will facilitate comparisons among potential solutions to PLNSEL since they have become a de facto benchmark for researchers in the self-adaptive community [15]. As an implementation of a cloud-based system, we used SWIM, which is a well accepted artifact in self-adaptive research community [4]. The system is hosted on a heterogeneous set of servers of varying capacity and per-minute usage cost, which increases with the capacity. The request arrival rate varies in an uncertain manner, leading to variance in the system's workload. The goal of self-adaptation is to optimize profitability by maximizing revenue

(dependent on the number of processed requests) and minimizing operating cost (dependent on the server usage) using various adaptation tactics. To maximize revenue, it is desirable to maintain the response time for user requests below a certain threshold (T), since higher response times lead to revenue loss. To reduce the response time caused by increased arrival rate, the system can add more servers (using tactic `addServer<type>`) and pay their costs. To reduce costs, the system can deactivate servers with an adaptation tactic `removeServer<type>`.

Another way to control response time is by reducing the amount of optional content (e.g., advertisements or product recommendations) that is provided in each response — sometimes called *brownout*. The optional content can increase revenue, but requires more bandwidth for each request, thus increasing response time. Tactics `increaseDimmer` and `decreaseDimmer` can control brownout by, respectively, raising or lowering the probability that a request will be served with optional content. Higher dimmer values lead to higher proportions of requests served with optional content.

To summarize, the system needs to maximize revenue, keep response time below the threshold to avoid penalties, and minimize the number of active servers to reduce cost. These objectives are captured in a multidimensional utility function shown in Formula 1. The adaptation goal of the system is to maximize the utility calculated using this formula. If the system runs for duration L, its utility function is defined as:

$$U = R_O x_O + R_M x_M - P x_T - \sum_{i=1}^{n} C_i \int_0^L s_i(t)dt, \quad (1)$$

where R_O and R_M is revenue generated by a response with optional and mandatory content respectively; x_O, x_M, and x_T are the number of requests with optional content, only mandatory content, and having response time above the threshold; C_i is cost of server type i, and s_i is number of active servers of type i; n is number of different types of servers.

Both the timeliness and quality of planning is needed to maximize utility for this system. A timely (i.e, quick) response is needed to minimize penalty P in case of the response time constraint violation. Simultaneously, a quality plan is needed for the long term utility gains by considering factors such as the current state of the system and its environment, predicted values of request arrival rate, and timing of tactic latency [14].

To construct a realistic environment of users accessing the cloud-based system, we used a research dataset with online traffic common in web analytics — the daily traces of user requests from the FIFA WorldCup website [16]. Each day's trace contains time stamps representing inter-arrival time between two client requests, abstracting away the details of user requests to focus on their frequency. We picked these traces because they contain the patterns for high-demand cloud systems as classified by Ghandhi et al. [12]. Moreover, this trace set is considered as a benchmark for traffic in web analytics [7], [17]. We mined training data and performed experiments on 87 traces (out of 92), ignoring 5 empty/partial ones; the illustrating plots are available in the supplementary material [6].

For hybrid planning, we use one reactive deterministic planning approach (ρ_{det}) in addition to waiting (ρ_{wait}), i.e., $\mathcal{F} = \{\rho_{det},\ \rho_{wait}\}$, along with deliberative planning (ρ_{mdp}) — as discussed in Section II. The uncertainty in request arrival rate is ignored by ρ_{det} by assuming it to be constant at the current value. This reduction in the search space greatly reduces the planning time for ρ_{det}, making it practically instantaneous in the context of the this system. When using the CB approach, ρ_{det} is invoked when response time is above the threshold; therefore, the intent behind using ρ_{det} is to avoid penalty P. In contrast to ρ_{det}, ρ_{mdp} considers predicted (but uncertain) values of the request arrival rates. We use a time-series predictor to anticipate the future workload on the system, similar to others [14]. The goal of both reactive and deliberative planning is to maximize utility (Formula 1) for their look-ahead horizon (parameter values described in Section V-B). In addition to labeling sample planning problems (Section III), we use PRISM both as a deterministic and MDP planner as we did in our previous work [9].

Action	Description	Survival/Detection Chance
`IncAlt`	Climb one altitude level	increases/decreases
`DecAlt`	Descend one altitude level	decreases/increases
`IncAlt2`	Climb two altitude levels	increases/decreases
`DecAlt2`	Descend two altitude levels	decreases/increases
`GoTight`	Change to tight formation	increases/decreases
`GoLoose`	Change to loose formation	decreases/increases
`EcmOn`	Turn ECM on	increases/decreases
`EcmOff`	Turn ECM off	decreases/increases

TABLE I. Adaptation actions for the UAV team.

B. A Team of Unmanned Aerial Vehicles

As the second evaluation system, we used a simulated team of unmanned aerial vehicles (UAVs) performing a reconnaissance mission in a hostile environment; as an implementation of a team of UAVs, we used DARTSim [5]. The predefined route of the team is a straight line, divided into equal segments of fixed length. Each segment can have threats and detection targets depending on how they are randomly placed in the route. The locations of targets and threats depends on a random seed, which is an input parameter to DARTSim. The mission of the team is to maximize the number of targets detected and avoid being shot down by the threats, which would lead to the mission failure (no more targets can be detected further). However, it is difficult to meet the two requirements simultaneously since there is no action available that increases the chances of both target detection and survival for the team (see Table I). If the team chooses to execute an action, then all of its UAVs in the team execute the same action.

The team has different sensors to detect targets and threats as it flies a route at constant speed. For each route segment within the range, the sensor reports whether it detects a target or threat, depending on the sensor type. However, due to sensing errors, these reports may include false positives and false negatives. An adaptation manager can get multiple observations to construct a probability distribution of threat or target presence in a cell.

A threat can destroy the team only if both are in the same segment. However, a threat has range r_T, and its effectiveness is inversely proportional to the altitude of the team, denoted by $\mathcal{A}$. In addition, the formation of the team affects the probability of it being destroyed. The team can be in two

different formations: loose (ϕ = 0), and tight (ϕ = 1). The latter reduces the probability of being destroyed by a factor of ψ. When the team uses (E = 1) electronic countermeasures (ECM), the probability of being destroyed is reduced by a factor of α. The probability of detecting a target with the downward-looking sensor, given that the target is in the segment being traversed by the UAVs, is inversely proportional to the altitude of the team [5]. Furthermore, flying in tight formation reduces the detection probability due to sensor occlusion or overlap, and the use of ECM also affects target detection, reducing the probability of detection by a factor of β. The probability of the team being destroyed d and detecting a target g depends on factors such as altitude, formation, and the use of ECM as formulated by Moreno et al. [5].

Given constants μ and λ, and the number of segments survived and target detected for a mission is S and T respectively, the utility of the mission is calculated as

$$U = \mu S + \lambda T. \tag{2}$$

Both the timeliness and quality of planning are needed to maximize utility for this system. A timely (i.e, quick) response is needed in response to threats, which could lead to mission failure. Simultaneously, a quality plan is needed for the long term utility gains that requires not only surviving but also detecting targets; this requires considering factors such as uncertainty in the threat and target locations.

To instantiate hybrid planning here, we use waiting(ρ_{wait}) and short-horizon planning (ρ_{srt}) as the reactive approaches (so, $\mathcal{F} = \{\rho_{srt},\ \rho_{wait}\}$), along with long-horizon deliberative planning (ρ_{lng}). Both ρ_{srt} and ρ_{lng} use MDP planning over similar models but with different planning horizons. Moreover, when planning, ρ_{srt} does not consider adaptation actions `IncAlt`, `DecAlt`, and `EcmOn`, and `EcmOff`; although less precise, the effect of these actions on survival/detection chances can also be achieved by the actions considered by ρ_{srt}. For instance, by using actions `IncAlt2` and `DecAlt2`, ρ_{srt} can plan to increase/decrease two altitudes levels in response to a threat or an opportunity to detect a target. Using a shorter horizon in combination with a subset of actions results in a smaller state space in ρ_{srt} compared to ρ_{lng}. The goal for both reactive and deliberative planning is to detect targets on the ground and avoid being shot down by threats. When using a CB approach, this instantiation invokes ρ_{srt} if $\mathcal{A} < r_T$ i.e., the team is in the threats' range, else ρ_{wait} is used; therefore, ρ_{srt} is used to provide a quick response when the team is in danger.

V. Evaluation

To compare the effectiveness of learning-based (LB) and condition-based (CB) approaches we proceeded as follows: We used the two systems as discussed in Section IV. Then we collected sample planning problems, labeled them with a model checker, and trained a classifier for the LB approach (Section V-A). To compare the performances, we conducted experiments in different planning modes (Section V-B) – our findings are presented in Section V-C.

A. Learning-based Approach Implementation

This section explains the implementation of the LB approach.

1) The Offline Phase: As explained in Section III-1, the offline phase involves three steps: identifying sample problems, labeling the sample problems, and training a classifier.

(1a) Identifying Sample Problems: To generate sample problems for the two systems, our goal was to create a set of problems similar to the ones expected at run time. For the cloud-based system, we executed each trace in a mode where ρ_{det} was always invoked in combination with ρ_{mdp}. This mode is different from using a learned classifier since the later switches between ρ_{det} and ρ_{wait} depending on a planning problem. Therefore, the training data is less likely to include the exact problems that the system would observe at run time, thus providing us with data similar to what can often be mined from system execution logs. We generated 1651 planning problems from 87 traces. For the UAVs, we simulated 630 missions (using 630 different seeds) in the mode similar to the cloud-based system i.e., always invoke ρ_{srt} in combination with ρ_{lng}. In total, 16822 planning problems were generated.

(1b) Labeling the Sample Problems: Since in both the systems set $\mathcal{F}$ has two elements, the offline phase of the LB approach labels each sample problem (say, ξ) with one of three classes (i.e., *UseReactive*, *UseWait*, or *UseEither*). Suppose the expected utility (after model checking) for the combination ρ_{det}/ρ_{srt} (depending on the system) and deliberative planning is U_R, and for the combination of ρ_{wait} and deliberative planning is U_w. If $U_r > U_w$, then the problem is labeled to invoke the reactive planning (i.e., CLASSIFY(ξ) = *UseReactive*); if $U_r < U_w$, then the problem is labeled to wait for the deliberative plan to be ready (i.e., CLASSIFY(ξ) = *UseWait*). Finally, if $U_r = U_w$, then the choice between reacting and waiting does not matter (i.e., CLASSIFY(ξ) = *UseEither*). One can also include a small margin (δ such that $U_r > U_w + \delta$, or vice versa) when comparing U_r and U_w. For the cloud-based system, 111, 253, and 1287 problems were labeled as *UseWait*, *UseReactive*, and *UseEither*, respectively. The UAV team had 358, 8391, and 8073 problems labeled as *UseWait*, *UseReactive*, and *UseEither*, respectively. For the two systems, the significant number of *UseEither* labels indicate that for large parts of state space wait and non-wait reactive planners agreed on the action.

(1c) Training a Classifier: Next we choose and train a classifier by separating train/test data via cross-validation. In the cloud system we used leave-one-out cross-validation. First, we left out a test trace (iterating through all 87 traces) on which the LB approach would later be evaluated. On the problems from the remaining 86, we trained via 10-fold cross-validation. Classifier performances are then averaged over all validation folds, and the best one is picked for evaluation on the test trace (i.e., the one not used for training). Similarly for the UAV team, we used 630 mission seeds for 10-fold cross-validation. The best classifier is evaluated as part of the LB approach on a different 70 missions. For the systems, each fold had the same proportion of classes as the whole dataset.

To find the "best" classifier in cross-validation, we used *recall, precision, and F1 score* based on dataset characteristics. For both the systems, it was challenging to discover situations when ρ_{wait} is the best choice because the data is skewed against *UseWait*. Thus, we maximized the recall value for *UseWait*.

For this criterion, an ensemble classifier known as *extremely randomized trees* achieved the best performance in the both systems. For the cloud, this classifier had recall/precision for *UseWait* above 0.8, and above 0.9 for *UseReactive* and *UseEither*. For UAVs, the best *UseWait* recall was 0.70, and precision — 0.72. Both recall and precision for *UseReactive* and *UseEither* were between 0.8 and 0.85.

2) The Online Phase: Both the systems periodically evaluate if adaptation is needed: When the systems observe a problem (ξ) at run time, they execute the *hybrid planning algorithm* we developed in [9]; the algorithm is formalized in the supplement [6], and we summarize it here. To find a plan for ξ, the algorithm first refers to the previous deliberative plan. If this plan exists and contains the current state, it is applied to the ξ — otherwise, a new deliberative plan is computed. In the meantime, the algorithm needs to decide its instantaneous response, which requires choosing a reactive planner $\rho_r \in \mathcal{F}$ (i.e., either invoke ρ_{det}/ρ_{srt} or ρ_{wait}) until the deliberative plan is ready. Regardless of the above decision, deliberative planning is started simultaneously in order to eventually arrive at a plan that is expected to yield higher utility than any reactive approach. As discussed earlier, the structure of the plan enables a smooth transition from a reactive to a deliberative plan, thus taking care of PLNCRD. In the algorithm, the logic to pick an appropriate reactive approach can be implemented by checking predefined conditions (i.e., CB) on ξ, or by learning (i.e., LB) which reactive approach is most suitable.

In the LB approach, the offline-trained classifier is used on ξ to assign it to one of the three classes discussed above in Section V-A1. If the returned class is *UseWait* or *UseReactive*, the system invokes ρ_{wait} or ρ_{det}/ρ_{srt}, respectively. However, if the class is *UseEither*, then the choice is not fully defined by the profiling information. To deal with this ambiguity, we consider two variants of the LB approach: LB-W chooses to wait in the case of *UseEither*, and LB-R chooses *UseReactive*. Both variants are studied in the evaluation.

B. Experimental Setup

The systems evaluates the need for adaptation at each minute, and determines an action, if adaptation is needed. For the profiling process, we configured the worst-case planning time (t_d) for ρ_{mdp} as 1 minute, chosen as an over-approximation after a large number of trial runs (which took between 35 and 55 seconds). For UAVs, the horizon for ρ_{srt} and ρ_{lng} is 2 and 5 minutes, respectively. Except aggregate utility (based on Formulas 1, and 2), all the parameters (e.g, choice of reactive/deliberative planning, instantiation of CB, LB-W, and LB-R) are independent. For each trace/mission, we define higher effectiveness of a planning approach as greater utility accrued over the trace/mission. The objectives for the evaluation is to investigate: if (a) using the LB/CB approach improves the effectiveness of HP compared to its constituent approaches, and (b) the LB approach to solve PLNSEL is more effective compared to the CB approach.

To meet these objectives, each trace/mission was evaluated in seven modes: (i) non-wait reactive — only ρ_{det}/ρ_{srt} is used (i.e., used ρ_{det} for the cloud and ρ_{srt} for the UAVs); (ii) wait — only ρ_{wait} is used, which essentially means the system does not adapt; (iii) deliberative — the system invokes ρ_{mdp}/ρ_{lng} (i.e., used only deliberative planning ρ_{mdp} for the cloud and ρ_{lng} for the team), and waits until a deliberative plan is available; (iv) non-wait hybrid planning (NW-HP) — when ρ_{det}/ρ_{srt} is always invoked until a deliberative plan is ready; (v) condition-based HP — when a deliberative plan is not available, ρ_{det} and ρ_{srt} is invoked only when the predefined conditions are met as described in Section IV-A and Section IV-B, respectively; (vi) LB-W HP — the LB approach solves PLNSEL and invokes ρ_{wait} if classification is uncertain; and (vii) LB-R HP — the same LB approach solves PLNSEL, but invokes ρ_{det}/ρ_{srt} if classification is uncertain. Non-wait reactive and wait modes represent the two possible reactive modes given ρ_{mdp}/ρ_{srt} and ρ_{wait}. The CB approach that calls ρ_{wait} was not considered separately since it is equivalent to the deliberative mode. In both the systems, although the classifier performed well during the cross-validation, comparison of the LB modes (i.e., LB-W and LB-R) with NW-HP and deliberative mode will further indicate whether the learned classifier was able to switch effectively between the reactive approaches (i.e., ρ_{wait} and ρ_{det}/ρ_{srt}); NW-HP and deliberative mode use only one reactive approach.

C. Results

The results of our experiments show that on average (both the LB and CB) hybrid planning outperforms its constituent planners, but LB outperforms CB. Also, to aid software engineers, we characterize the impact of constituent planners on the performance of the hybrid planners.

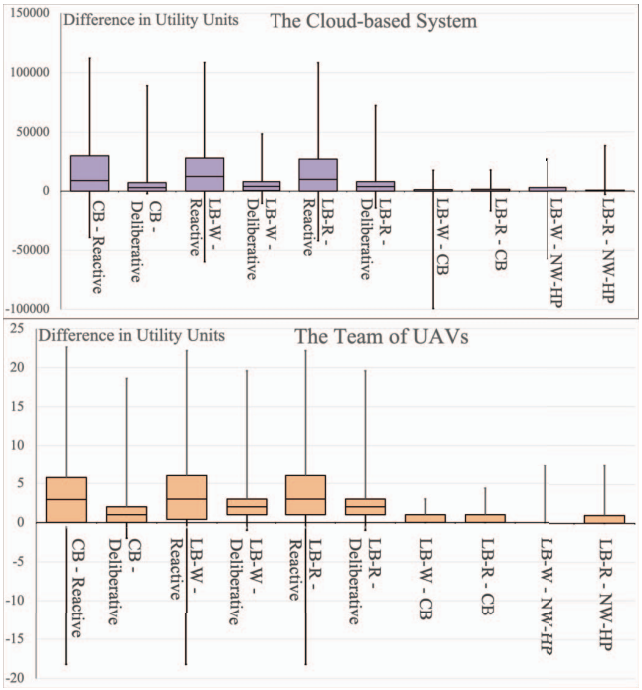

Figure 2: Utility differences per trace/mission added up for all traces/missions. Each bar represents a sum of differences for a pair of planning approaches.

1) Hybrid Planning Outperforms its Constituent Planners: Our experiments in both systems indicate that HP provides

more utility than individual planning, as depicted in Figures 2 and 3. In Figure 2, the box-plots show the differences in accrued utility (per trace/mission) when comparing pairs of planning approaches. The boxes represent the median 50% of traces (in terms of the difference between a pair of planners), with the horizontal lines inside showing the median difference across the traces. The whiskers show the minimum and maximum difference in utilities. For example, the leftmost box compares CB to only using reactive (i.e., ρ_{det}/ρ_{srt}). The lower edges (i.e., the first quartile) of the six leftmost boxes are above zero, indicating that for most of the traces/missions HP provides equal-or-higher utility compared to non-hybrid approaches. Specifically for the cloud-based system, CB, LB-W, and LB-R show equal-or-higher utility than *both* reactive and deliberative planners on 57 (66%), 60 (69%), and 57 (66%) traces respectively (out of 87 total); moreover, for the respective boxes, the positive whisker is longer than the negative one, indicating higher maximum gain than loss when choosing HP.

Based on the estimator of true probability with a confidence level of 95%, the true probability ranges for the three HP approaches to match or improve over both non-hybrid planners are (0.55; 0.76), (0.58; 0.80), and (0.55; 0.76). For the UAVs, CB, LB-W, and LB-R show **higher-or-equal** utility than *both* reactive and deliberative planners on 51 (71%), 55 (78%), and 56 (80%) traces respectively (out of 70 total). The longer negative whiskers for the 1st, 3rd, and 5th box-plot are explained by the team being averse to destruction in reactive mode, which avoids the threats at all costs; therefore, in certain missions the team survives, but in HP modes it gets destroyed (i.e., mission failure), losing significant overall utility. Based on the estimator of true probability with a confidence level of 95%, the true probability ranges for CB, LB-W, and LB-R to match or improve over *both* non-hybrid planners is (0.58; 0.82), (0.65; 0.89), and (0.67; 0.9), respectively.

We also found that it is unlikely that HP performs worse than *both* reactive and deliberative planning; therefore, using HP is **less risky** compared to reactive or deliberative planning. Out of 87 traces, HP does worse than *both* non-hybrid planners only in 1 (1%), 5 (6%), and 5 (6%) traces for CB, LB-W, and LB-R planning, respectively. This leads us to, respectively, $(0, 0.12)$, $(0, 0.16)$, and $(0, 0.16)$ probability ranges of both reactive and deliberative planning outperforming HP according to the estimator of true probability, with 95% confidence. For the UAVs, HP does worse than *both* non-hybrid planners only in 4 (6%), 2 (3%), and 2 (3%) missions respectively (out of 70 total). The true probability ranges for the three HP approaches to match or improve over both non-hybrid planners are (0; 0.17), (0; 0.15), and (0; 0.15). Therefore, when choosing between deliberative, reactive, and HP, the latter is the least risky choice.

2) Learning-based Outperforms Condition-based approach: Our experiments show that LB provides **more utility** than CB on average. In Figure 2, the 7th and 8th box is above zero, indicating that for the majority of traces/missions LB does equal-or-better than CB. Specifically, out of 87 traces, LB-W and LB-R provided higher or equal utility for 70 (80%) and 62 (71%) traces, respectively. The estimator of true probability suggests with a confidence of 95% that the true probability

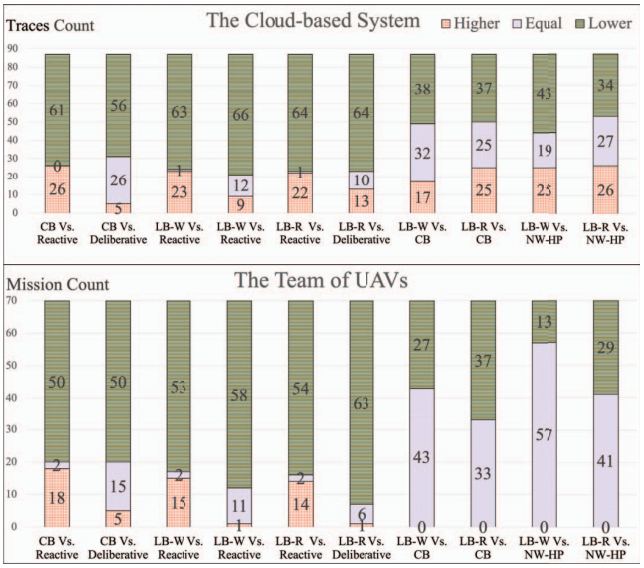

Figure 3: Pairwise performance comparison of planning approaches. Each bar is for a pair of approaches, labeled with the counts (out of the total traces/missions) of traces/missions where the first approach provides higher/equal/lower utility compared to the second approach in the pair.

range for the CB approach yielding higher utility than LB-W and LB-R is $(0.09, 0.3)$ and $(0.18, 0.39)$, respectively. For the UAV team, out of 70 missions, both LB-W and LB-R provided higher or equal utility for all the 70 missions. With a confidence of 95%, the true probability range for the CB yielding higher utility is $(0, 0.12)$. Thus, it is **less risky**, and in many cases advantageous, to use the LB over the CB.

However, the magnitude of the utility difference between CB and LB is smaller than that between HP and its constituent planners. The reason is, in response to CB constraint violations, reactive planners typically propose conservative measures such as `addServer`, `decreaseDimmer`, and `IncAlt2`. These actions decrease the worst-case utility loss, which is particularly high for the second system due to the possibility of destruction. In contrast to CB, despite not falling behind in performance, LB enables the system to (automatically) learn when utility could be gained by using reactive planning, even without violations. Thus, we conclude that CB is more **risk-averse**, whereas the LB is **more opportunistic** since it does not limit the use of non-wait reactive planning to constraint violations.

Similar to CB, the outperformance of LB is less significant compared to the NW-HP mode as shown in the right-most two boxes, because in both systems invoking non-wait reactive planner (i.e., ρ_{det} or ρ_{srt}), in general, was preferred over using ρ_{wait}.[2] However, compared to NW-HP modes, the LB approach was able to automatically learn a classifier that switches effectively between the reactive approaches.

3) Influence of Constituent Planners on Hybrid Planning: Our evaluation shows that the performance of HP depends

[2]This fact is supported by the class imbalance of the labelled data, which is skewed against using ρ_{wait} as presented in Section V-A; this indicates the model checking was able to label the problems reasonably well.

on the performance of the following modes: (i) deliberative planning, and (ii) the (relatively) effective reactive planners. Below is the evidence and implications for software engineers.

Deliberative planning performance has a **consistent positive impact** on the performance of HP. We observe a medium-to-strong correlation ($p < 0.01$) between the deliberative mode and each of the three HP modes. For the cloud system, the Pearson correlation is 0.95 for CB, 0.97 for LB-W, and 0.95 for LB-R. For the UAVs team, the correlation is 0.6 for CB, 0.61 for LB-W, and 0.59 for LB-R. The interpretation of this finding is that, once a deliberative plan is ready, it inevitably takes over from any reactive plan, hence the performances of HP and deliberative planning are tightly coupled.

To further investigate this correlation, we conducted chi-square independence test, which also showed that the **ability of HP to perform better** than or equal to its constituent planners significantly **depends** ($p < 0.01$) on deliberative planning performing better than or equal to reactive ρ_{det}/ρ_{srt}. For the cloud-based system, the χ^2 values for CB, LB-W, and LB-R are 43.79, 32.38, and 19.02, indicating strong-to-moderate dependency. For the UAVs, the χ^2 values for CB, LB-W, and LB-R are 18.97, 22.16, and 20.37 also indicating strong-to-moderate dependency. This finding supports our assumption that an effective deliberative planning approach is a foundation for hybrid planning. As the chi-square test suggests, one should prefer hybrid planning to reactive planning if deliberative planning consistently provides higher or equal utility compared to reactive approaches.

The performance of each reactive planner has a **positive impact** on the performance of HP, **moderated by the relative performance** of the reactive planner. We found that among the reactive approaches, the more effective ones had a stronger influence on the HP performance. To conclude this, we fit a regression (the equation can be found in [6]) to the HP utility, using deliberative and reactive utilities as independent variables. Further, these explanatory variables were weighted with a ratio of deliberative to NW-HP modes (for the deliberative utility) and the inverse of that ratio (for the reactive utility). This ratio characterizes the relative goodness of the wait planner and the reactive planner, since these two planners represent the only difference between the deliberative and NW-HP modes. For both systems and all HP modes, the regression coefficients for the weighed utilities were positive and non-zero with high significance ($p < 0.01$). Our interpretation is that the more effective reactive approaches are used more often, influencing the HP performance more than those used rarely. The above holds assuming that the classifier performs reasonably well (in our evaluation this meant having precision/recall above 0.7 for all classes). Therefore, we suggest identifying the more effective approaches (by comparing their utilities or respective class counts in training data) and focusing the resources on improving them further.

VI. Discussion and Conclusion

This paper proposes a general learning-based approach to PLNSEL where problems can be classified if a domain is predictable enough to do labeling. Its advantage is non-reliance on domain expertise for specific conditions, and on average it performs better than the CB approach. In part this is due to the flexibility of learning to avoid risky invocations of reactive planning that would fit the fixed conditions, and taking advantage of adaptation opportunities that arise outside of fixed conditions. Although finely-tuned conditions in the CB approach can result in performance comparable to the LB approach (as in some scenarios in our evaluation), these qualitative benefits of the LB approach still hold.

We believe the steps to apply our LB approach to PLNSEL can be automated, thereby reducing the burden on engineers. The steps involved are (a) gathering planning problems for training and estimating the deliberative planning time, (b) using probabilistic model checking to label these problems, and (c) training a classifier using these labels. Gathering training problems and planning times is often straightforward for modern-day enterprise systems (e.g., Netflix) and automated vehicles (e.g., UAVs) using their execution logs. Labeling the problems using model checking can also be automated as we did for the two systems; the framework is available as an open-source project (https://github.com/Ashutoshp/ProfileInfra). Finally, machine learning frameworks (e.g., scikit-learn for Python) can be used to automate the training process. Guidelines for challenges beyond solving PLNSEL, such as instantiating constituent planners, can be found in our other work [2].

Using model checking is fundamental to our LB approach as it determines the accurate labels of training problems given the *a priori* uncertainty. Moreover, (multiple) existing model checkers ease adoption, automation, and reuse of the LB approach. Future work includes investigation of other techniques to label problems and using unsupervised learning.

A. Threats to Validity

The *internal validity* of our study is threatened by three potentially confounding factors. First, our objective function for cross-validation (recall on *UseWait*) could lead to increased performance of the LB approaches. This threat is mitigated by precision and recall for other classes also being high for our chosen classifier, and that the patterns are observed in experiments with a broad range of classifier performances. However, it is possible that the classifier could lead to higher utility than that of LB-W and LB-R.

Second, the relative performances of the CB and LB approaches are due to the specific conditions for triggering reactive planning. Although this condition is tied to the system's utility function, it is possible to fine-tune it further, to approach the theoretical limit of perfectly matching a situation to a reactive approach. However, this fine-tuning is difficult in practice due to the multi-dimensional utility function and uncertainty in the external environment that leads to uncertainty in (reactive) action outcomes. Therefore, we expect this tuning to have a minor effect on the evaluation results.

Third, the performance of the LB and CB approaches may depend on system parameters (e.g., server costs, ECM factors), changing which might affect the penalties for poor quick reactions. This threat is mitigated by two different test-beds and hybrid planners, and a sizable set of traces/missions with substantial variation, which leads to a robust assessment of

planner performance through cross-validation. In the authors' knowledge, this is the largest set of traces ever used for evaluation on a cloud-based system.

To measure effectiveness of planners, we use cumulative utility functions (presented in Section IV). These functions express conflicting goals, and similar functions are used to measure performance of cloud-based systems/UAV teams in related work [15]. Such utility functions are applicable when there is a need to accumulate correct behavior while avoiding undesirable behavior, by performing actions with uncertain outcomes in uncertain environments (modeled as MDPs).

The *external validity* of our conclusions is threatened by the use of only two systems and three reactive planners ($\rho_{det}, \rho_{srt},$ and ρ_{wait}). In theory, the LB approach should apply to any number of reactive approaches in set $\mathcal{F}$, although we evaluate using only two planners at a time. As a sanity check, we compare the LB approach with deliberative only and NW-HP mode; these modes are constrained to use only one of the reactive approaches. The LB approach outperforming them shows that the classifier switched effectively between the reactive approaches. This conclusion is corroborated by the precision/recall values from cross-validation. Furthermore, labeled training data can be used to narrow down the set of constituent planners. The interactions between a hybrid planner and its constituent approaches are dependent on various factors, including the utility function and assumptions behind the approach. We expect these factors to hold for any utility function that is accrued over states of traces/missions and reflects that fast reactions are vital to the system's goals, yet the choice of when to react is not obvious.

We further mitigate the threat to validity by evaluating on two well accepted testbeds (i.e., SWIM [4] and DartSim [5]) for self-adaptive research that differ in three significant ways:

- *The ability to recover from poor/delayed actions:* Even if the cloud-based system fails to maintain the critical response time constraint due to poor/delayed actions, it can still recover back to a desired state later. However in case of the UAV, a failure to avoid a crash (i.e., safety constraint) will lead to a mission failure as illustrated by the negative long whiskers in Figure 2.
- *Different instantiations of hybrid planning:* Reactive planning for the cloud system ignores uncertainty, whereas for UAVs it uses a reduced horizon and set of actions.
- *Different proportions of class labels:* The cloud system has a large proportion of *UseEither*, whereas the UAV team has a similar proportion of *UseReactive* and *UseEither*.

B. Conclusion

In the past, the promising idea of hybrid planning has been studied from theoretical [2] and algorithmic [10], [7] perspectives. In this paper, we improve its engineering aspects by providing (i) a learning-based approach to planning selection, which aims to replace domain-specific hard-coded conditions for invoking reactive planning, and (ii) the correlation between the performance of hybrid planning and its constituent planners. One of the barriers to adopting the learning based approach is the difficulty in having a labeled set of training planning problems. We overcome this in a novel way by using probabilistic model checking to label the training problems. Moreover, this enables the steps (including model checking) of the learning-based approach to be automated. Our evaluation uses two realistic systems from different domains and with different combinations of planners indicating generality of hybrid planning and the learning-based approach.

ACKNOWLEDGMENTS

This work is supported in part by award N00014172899 from the Office of Naval Research and by the NSA under Award No. H9823018D0008. Any views, opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research. We also thank to Christian Kästner for his feedback.

REFERENCES

[1] P. Tucker, "You have 19 minutes to react if the Russians hack your network," https://tinyurl.com/y6s4lvjk, 2019, [Accessed 1-May-2020].

[2] A. Pandey, "Hybrid planning in self-adaptive systems," Ph.D. dissertation, Carnegie Mellon University, 2020. [Online]. Available: https://kilthub.cmu.edu/articles/Hybrid_Planning_in_Self-adaptive_Systems/11926836

[3] D. Jacobson, D. Yuan, and N. Joshi, "Scryer: Netflix's predictive auto scaling engine," https://tinyurl.com/tvtgew7, [Accessed 1-May-2020].

[4] G. A. Moreno, B. Schmerl, and D. Garlan, "Swim: an exemplar for evaluation and comparison of self-adaptation approaches for web applications," in *13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2018, pp. 137–143.

[5] G. A. Moreno, C. Kinneer, A. Pandey, and D. Garlan, "Dartsim: an exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems," in *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Montreal, QC, Canada, May 25-31, 2019*, 2019, pp. 181–187.

[6] "Supplementary materials for hybrid planning using learning and model checking for autonomous systems," "https://tinyurl.com/ycmxo6r5".

[7] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: a hybrid, proactive auto-scaling mechanism on a level-playing field," *Transactions on Parallel and Distributed Systems*, 2018.

[8] A. Pandey, I. Ruchkin, B. Schmerl, and J. Cámara, "Towards a formal framework for hybrid planning in self-adaptation," in *12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2017, pp. 109–115.

[9] A. Pandey, G. A. Moreno, J. Cámara, and D. Garlan, "Hybrid planning for decision making in self-adaptive systems," in *10th International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 2016.

[10] Mausam, P. Bertoli, and D. S. Weld, "A hybridized planner for stochastic domains." in *IJCAI*, 2007, pp. 1972–1978.

[11] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control," in *Proceedings of the 3rd workshop on Scientific Cloud Computing*. ACM, 2012, pp. 31–40.

[12] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "Autoscale: dynamic, robust capacity management for multi-tier data centers," *ACM Transactions on Computer Systems*, vol. 30, p. 14, 2012.

[13] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: verification of probabilistic real-time systems," in *International conference on computer aided verification*. Springer, 2011, pp. 585–591.

[14] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in *10th Joint Meeting on Foundations of Software Engineering*, 2015.

[15] T. Chen, R. Bahsoon, and X. Yao, "A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 61:1–61:40, 2018.

[16] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE network*, vol. 14, no. 3, pp. 30–37, 2000.

[17] S. Imai, "Elastic cloud computing for qos-aware data processing," Ph.D. dissertation, Rensselaer Polytechnic Institute, 2018.