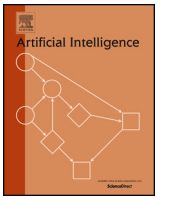
Artificial Intelligence 345 (2025) 104336

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Coltrane: A domain-independent system for characterizing and planning in novel situations

Bryan Loyall [a,*], Avi Pfeffer [a], James Niehaus [a], Michael Harradon [a], Paola Rizzo [b], Alex Gee [a], Joe Campolongo [a], Tyler Mayer [a], John Steigerwald [a]

[a] *Charles River Analytics, United States of America*
[b] *Interagens, Italy*

ABSTRACT

AI systems operating in open-world environments must be able to adapt to impactful changes in the world, immediately when they occur, and be able to do this across the many types of changes that can occur. We are seeking to create methods to extend traditional AI systems so that they can (1) immediately recognize changes in how the world works that are impactful to task accomplishment; (2) rapidly characterize the nature of the change using the limited observations that are available when the change is first detected; (3) adapt to the change as well as feasible to accomplish the system's tasks given the available observations; and (4) continue to improve the characterization and adaptation as additional observations are available. In this paper, we describe Coltrane, a domain-independent system for characterizing and planning in novel situations that uses only natural domain descriptions to generate its novelty-handling behavior, without any domain-specific anticipation of the novelty. Coltrane's characterization method is based on probabilistic program synthesis of perturbations to programs expressed in a traditional programming language describing domain transition models. Its planning method is based on incorporating novel domain models in an MCTS search algorithm and on automatically adapting the heuristics used. Both a formal external evaluation and our own demonstrations show that Coltrane is capable of accurately characterizing interesting forms of novelty and of adapting its behavior to restore its performance to pre-novelty levels and even beyond.

1. Introduction

AI systems today are often brittle to changes in their environment that were not previously anticipated by their designers. This brittleness can be a source of both reduced performance and (at times, catastrophic) failure. Avoiding this brittleness when interacting in an open world often requires an AI system to be able to adapt to changes in the world, immediately, when they occur, and being able to do this across the many types of changes that can occur in an open world [7,21]. In many situations it is not feasible to take the system offline for re-engineering or re-training. The system needs to immediately recognize the change in the world, and adapt its behavior to the change to accomplish its task.

For example, imagine a future world with a self-driving ambulance in which the brakes start requiring 1.5 times as long to stop while it was carrying a patient (due to either, e.g., a change in the road surface, or a change in the physical properties of the ambulance). It would be advantageous if the system were able to identify that the world had changed, characterize that it was a change in stopping power of the brakes, and allow additional space and time for each stop, safely navigating to the hospital in time for the patient to receive effective treatment. After getting the patient safely to the hospital, the ambulance could then be taken offline, allowing it to be repaired, re-engineered, or its AI system to be re-trained as needed. (Note that which of these is required depends on whether the cause was internal to the agent, or external—such as a new substance on the roadway that it was unprepared for.)

* Corresponding author.
E-mail address: bloyall@cra.com (B. Loyall).

https://doi.org/10.1016/j.artint.2025.104336
Received 28 February 2023; Received in revised form 13 April 2025; Accepted 16 April 2025

Available online 24 April 2025

This example illustrates the focus of our research. We are seeking to create methods to extend traditional AI systems so that they can (1) immediately recognize changes in how the world works that are impactful to task accomplishment; (2) rapidly characterize the nature of the change using the limited observations that are available when the change is first detected; (3) adapt to the change as well as feasible to accomplish the system's tasks given the available observations; and (4) continue to improve the characterization and adaptation as additional observations are available. This research is intended to augment re-engineering and retraining technologies which are pursued in other research approaches.

This paper makes three main contributions. (1) A method for extending traditional non-novelty aware systems to be immediately adaptable to novelty when it occurs, and then to incrementally improve their adaptation over time as the novelty is further understood. (2) A domain-independent method for characterizing the novelty through probabilistic program synthesis founded on the pre-novelty domain model which builds on the inductive biases inherent in the domain. (3) Methods to extend traditional planning systems to effectively use the characterized information in their reasoning, including mechanisms to deal with uncertainty and computational complexity that arises due to the novel changes.

Our approach is founded on a representation of the pre-novelty domain as a transition model expressed in a traditional programming language (including native probabilistic modeling),[1] with an MCTS-style planner with heuristics used to plan in the domain. Novelty characterization searches for perturbations of this pre-novelty transition model using probabilistic program synthesis techniques. These techniques use an inductive bias that favors smaller perturbations to find explanations of the novelty that balance simplicity with fit to data. Once one or more new transition models have been provided, the novelty-aware planner integrates these models into its MCTS search, including reasoning under model uncertainty and recommending exploration actions. In addition, the planner automatically removes or modifies its heuristics.

We have demonstrated and evaluated Coltrane both internally and externally. Externally, single-blind evaluations through the SAIL-ON program[2] show that Coltrane is capable of adapting under novelty in diverse domains and across several types of novelty. Our internal demonstrations show details of Coltrane's characterization and adaptation performance over time. These demonstrations show that not only can Coltrane adapt to novelty, but its automated adaptation can perform better than programmed adaptations of other agents.

2. Related work

Enabling AI systems that can recognize, characterize and adapt to changes in an open world is related to and extends several research directions.

2.1. Open world assumptions

The open world assumption (OWA) was formalized by Reiter in the late '70s [36], to indicate worlds in which gaps in one's knowledge about the domain are permitted, in contrast to the closed world assumption (CWA), in which the implicit representation of negative facts presumes total knowledge about the domain being represented. Since then, more and more AI approaches have gradually evolved to abandon the CWA and embrace the OWA, which is much better suited to deploy systems in real world environments (e.g., for a review of such trend in machine learning, see [31]). Given the importance of the OWA, the term "open world" is also often used to generally indicate an environment that is not confined to the lab and resembles the real world.

2.2. Novelty definitions

Open worlds are naturally subject to novelty. The concept of novelty has been defined, operationalized, or formalized in several ways in various AI communities. In classification systems, novelty is considered "new or unknown data or signal that a machine learning system is not aware of during training" [22]; in robotics based on reinforcement learning, Huang & Weng (2002) define novelty as "the normalized distance between the selected primed sensation ... and the actual sensation ... in next time"; in artificial creative systems, Wiggings (2006) defines novelty as "The property of an artefact ... output by a creative system which arises from prior non-existence of like or identical artefacts in the context in which the artefact is produced." Novelty is also related to the concept of surprise in AI systems, that has been used as a learning mechanism in agent and robot architectures (e.g., [34], [23]), and as a computational modeling tool (e.g., [14], [47]).

Given the variety of definitions and uses of the concept, Boult et al. (2021) have proposed a unifying formal framework to create theories of novelty for agents that move from the lab to the open world. The framework is based on the formal definitions of several aspects of the interaction between agent and environment: world / observation / agent state spaces, perceptual operator, agent

[1] The choice of addressing transition models that are represented in general purpose programming language code rather than a declarative formal language such as PDDL deserves some explanation. This choice has five advantages: (1) it is fully capable of expressing declarative models as are common in PDDL, frame-based representations, etc.; (2) using advanced language mechanisms, such as reflection, for manipulating code as data, we are able to treat them declaratively (as described in Section 4); (3) in addition to fully enabling declarative models, code enables a wider range of expressive models to be compactly represented; (4) many deployed and research system express their models in code; and (5) by reasoning over the code as a whole we can infer both where and when an open-world change occurred in all of the levels of abstraction in the domain model using a unified inference process (see Section 4). Recent advances in declarative formal languages convey some of these advantages, e.g., [3], making them potential targets for future representations.

[2] Science of Artificial Intelligence and Learning for Open-world Novelty [7].

task, state recognition and transition functions (to select the next agent's action and represent the new resulting state, respectively), experience tensor (to represent history), dissimilarity functions and thresholds (measuring and using, respectively, the difference between world and observation space), and regret functions (assigning a regret score to a new state after the agent executes an action). The framework enables the creation of implicit theories of novelty through the functions used in the definitions: in other words, researchers can define multiple theories of novelty by providing their own functions to compute, among others, task-based state transitions, dissimilarity, and regret. Our work is consistent with such framework, but also addresses two aspects missing from it, which are the characterization of novelty, and the concept of opportunity. Characterization we believe is an exceedingly useful for effective adaptation, while recognizing that adaptation to novelty without characterization is also an interesting research direction. Regarding the latter additional aspect, the concept of opportunity, a novelty might produce a regret score of zero because it does not negatively affect the agent's performance but, as better described later, might provide an opportunity to increase performance if the agent discovers the novelty and adapts its behavior to leverage the opportunity.

Several types of novelty have been proposed in the literature. For instance, the SAIL-ON Novelty Working Group, in which we participated, developed a novelty hierarchy (described in [9]) based on several features of the environment, such as objects, agents, and actions. Boult et al. (2021) have proposed various novelty types based on the combinations of three primary types (world, observation and agent novelty) with the world or perceptual regret, thereby identifying, e.g., "imperceptible", "ignored", or "nuisance" novelty types, focused on whether and how those types ultimately affect the agent's own performance. Other researchers, e.g., ([24], [25]), have proposed different formalizations of novelty, focused on how an environment can be described and transformed in ways that indicate the kinds of challenges encountered by agents of different types, and on how the agents' performances can be compared. In this paper, we focus on a categorization of novelty into four types (Section 3.4) that we have found most relevant for actively reasoning about effective adaptation in open worlds.

2.3. Novelty detection

Novelty detection in data science, and its synonyms of anomaly and outlier detection, concerns the detection of observations in the data that do not conform to the expected, normal behavior [33]. While this is related to what is needed for an action system that can recognize changes in an open world, it is narrower than what is required for this task. In an action system that is accomplishing a task, the detection of changes needs to include any change, at multiple levels of abstraction, that are impactful to the reasoning needed to accomplish the task, and cannot be limited to anomalies at a single level of abstraction in the data as is typically targeted in anomaly detection. Further, anomaly detection is typically concerned only with detecting novelty, not with characterizing it or using the characterization to act effectively in the face of novelty. As a consequence, traditional approaches to anomaly detection will detect any novelty in the domain, regardless of whether the change is related to the task to be accomplished by the agent. In contrast, because we are concerned with characterization and planning, our work focuses on detecting changes in the model of the world that are impactful to the task. This task-oriented focus is a key element of our approach.

2.4. Novelty characterization

Characterization of novel world dynamics from observations using probabilistic program synthesis has yielded promising results in multiple domains [20][38][37][10]. Coltrane's characterization algorithm builds on this research, with a difference from typical probabilistic program synthesis in that it learns a new program that makes targeted modifications to the pre-novelty domain model, rather than learn a new program from scratch. This extension is key to the goal of continuing to act effectively in the unchanged aspects of the domain, while rapidly adapting to changed aspects.

One of the advantages of probabilistic program synthesis for characterization is that it can be sample efficient. E.g. Lake et al. [20] enabled one-shot learning to recognize novel characters by learning programs with characteristic patterns of pen strokes by using a powerful prior of program primitives for brush stroke types. Rapid adaptation to novelty requires this kind of sample efficiency, so Coltrane similarly leverages an informative prior to enable one-shot or few-shot characterization, by using all of the information in the pre-novelty domain model as the prior for synthesis.

One goal of using the code in the full pre-novelty domain model code is the ability to use abstractions that naturally occur in the code for efficient synthesis. This use of program abstraction to speed inference is analogous to that described in Ellis et al. [10] which speeds synthesis by using functional abstraction. Coltrane's approach is in a similar tradition, with a difference that it starts with a large set of abstractions on which to build because of its use of the full pre-novelty domain model code, rather than building the abstractions from the bottom up, with benefits for rapid characterization of novelty and maintaining fit with the pre-novelty domain.

2.5. Novelty adaptation

The ability to tackle novel inputs has been indicated as crucial for AI robustness [8], and that of quickly adapting to novelties in open worlds has been indicated by Langley [21] as needed to develop radically autonomous systems, that can rapidly revise their expertise when they encounter novel cases produced by reasonably stable environmental changes, which require some form of long-term learning rather than short-term adaptation.

The need for rapid expertise revision indicated by Langley [21] significantly narrows the range of AI system related to the proposed work. Re-training in data-driven machine learning systems such as deep reinforcement learning systems can adapt to open worlds if

the system can be taken offline, and if sufficient observations are available for the novelty. But significant retraining is not applicable when the AI system needs to adapt in an online fashion, and continue to accomplish its task. Further, with the central need to continue to accomplish the task, in the face of the open world changes, sample efficiency is a key requirement for detecting, characterizing and adapting to novelty, which is typically not the case in data-driven training and re-training applications that in general requires orders of magnitude more samples than would be available in the required online adaptation during task execution. The ideal agent would have to use one-shot or few-shot learning [44] to characterize the novelty and adapt to it. The long-term learning advocated by Langley is studied in lifelong machine learning and related paradigms, such as transfer, multi-task, and online learning [6]: however, such approaches are mostly based on a clear separation between training tasks concerning different data (e.g., [30]), while our work aims to enable an agent to learn how to characterize novelties and adapt to them in the context of the same task it currently pursues.

Another field of research that is outside the range of AI systems related to our work concerns meta-learning and generalist agents (e.g., [1], [35]), that are trained in ways that let them perform well in a variety of different tasks. In fact, the purpose of our work, and of other researchers working on novelty adaptation, is not to realize agents that are flexible with respect to many tasks, but that are able to adapt to novelties that can occur in tasks carried out in open worlds that were not anticipated by the designers, and that potentially disrupt performance. Doing well in multiple tasks does not guarantee that the agent will keep its performance, or be able to improve its performance, when negative or positive novelties, respectively, occur in those tasks.

Research in automated planning includes several successful techniques to let agents deal with unknown or probabilistic states, unpredictable facts, or unexpected circumstances occurring during plan execution. For instance, contingent planning creates plans that adapt to states that are unknown at planning time by using sensing actions at execution time [1]; probabilistic planning produces plans that are sufficiently likely to succeed when the classic assumptions of complete and deterministic information do not hold (e.g., [19]); online replanning or plan repair can solve unexpected issues occurring at execution time (e.g., [12]), and can be much more efficient than probabilistic planning by constructing a deterministic variant of the planning problem and selecting actions according to the plan until an unexpected effect is observed, upon which replanning is executed (e.g., [48]); conformant planning can produce plans in the presence of uncertainty about the initial state and action effects without using sensing actions during execution (e.g., [15]); continual planning interleaves planning, execution and monitoring to avoid the computational complexity of the other approaches and improve efficiency and scalability (e.g., [4]), and can be improved by limiting the need of replanning through action modalities reconfiguration (e.g., [40]); case-based planning can reuse previous plans for new situations instead of planning from scratch, and can be used in combination with other planning techniques from those mentioned so far to achieve good performance in open worlds such as video games (e.g., [29]).

While our work does not use traditional contingent planning, because the assumption that the set of possible states or belief states are known to the agent does not hold in an open world, it shares with the planning approaches above the use of online planning, in that the decision-making of the agent is continuously updated after the execution of each action, so as to ensure the best possible adaptation to novelties. Also, because acting under novelty requires online planning and execution with uncertainty over hypotheses, our approach does draw on and adapt contingent planning concepts in a novel way—for planning over model uncertainty rather than world state uncertainty (See Section 5.2.3). It also shares a probabilistic approach, but in a very different way, by using probabilistic program synthesis to characterize the novelties, as mentioned above and better described later. In fact, the main difference between the planning techniques described above and our work is that, rather than independently solving local unexpected differences between expected and actual states every time they occur, our work aims to recognize changes in how the world works in general, so as to predict how the world will be different in the future under a certain set of actions, and to gradually characterize the nature of the change in a way that continues to improve the adaptation as additional observations are available by identifying latent, unobserved reasons for the observed changes.

Recent work on handling novelty in open worlds combines reinforcement learning with symbolic representations (e.g., [39], [32]), or logical inference methods with classical planning (e.g., [26]) or Markov Decision Processes (e.g., [42]). Another work that handles novelty in a planning and execution architecture is Musliner et al. ([27]) which uses five general domain-independent heuristics to modify planning goals or operators. Our work is complementary to these approaches, exploring general model-based detection, hypothesis generation for characterization, and adaptation with planning model updates.

Recently, a number of researchers have made advances in rapidly recognizing and adapting to novelty. Researchers at Tufts have created a system that uses a PDDL model of the domain, recognizes when operators result in different outcomes than predicted in the PDDL model, and then use deep learning to learn new operators that can be used in the PDDL model [11]. PARC's research system HYDRA [17] uses similar expectation violation of PDDL+ operators compared to a simulated world with model repair mechanisms to modify the model's operators to match the observations.

Our work extends these research approaches by enabling recognition and characterization in a wider range of models than those expressible in PDDL or PDDL+, and also by including model deviations to any aspect of the model, in addition to operators, as described in Section 4. We also extend the reasoning over the characterized model to include additional planning mechanisms for performant reasoning in the modified model that were undermined by the change, as described in Section 5. Musliner et al.'s OpenMind system uses a set of domain-independent heuristics to recognize novelty and adapt to it without characterizing what had changed [27]. This set of techniques is orthogonal and complementary to the approaches we describe herein. A promising direction of future research would be to add these heuristic techniques to our system for additional robustness and task accomplishment ability (perhaps by working around the open-world change) before the change is fully characterized and adapted to.

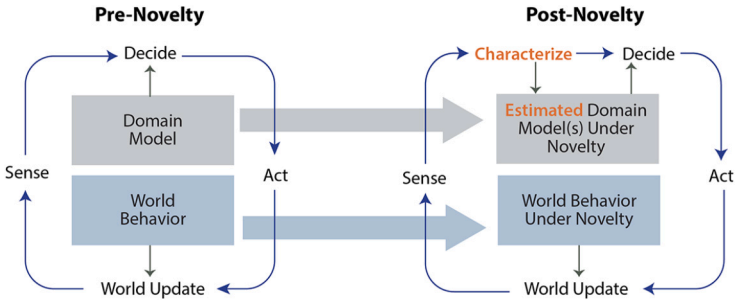


Fig. 1. A general architecture for novelty-aware agents.

3. The Coltrane framework

3.1. Overview of the problem

Coltrane is a framework for designing agents that operate in a dynamic, open-world environment. In an open-world environment, an agent cannot assume that it knows everything about the behavior of the world, and it may encounter aspects of the world that are novel to it. When an agent encounters novelty, it must first detect the novelty, then characterize it and describe how the world has changed, and then plan to act under the newly characterized world. Crucially, the agent cannot go offline while it tries to figure out how to operate under novelty. It must continue to interact with the world, even before it is sure how the novelty is characterized, or it has determined how best to operate under the new characterization. This makes rapid adaptation, together with the ability to improve performance over time as more experience with the novelty is obtained, a vital capability for agents in an open-world environment.

In general, novelty will not be a completely arbitrary change to the world, but will relate to what the agent already knows about the world. This assumption about novelty is crucial to making novelty handling possible. The assumption enables Coltrane to maintain an inductive bias about the novelties it sees. Any given novel experience can have numerous explanations, and the inductive bias enables Coltrane to make sense of the novelty and select a small number of explanations as being plausible and likely.

It is important that the inductive bias does not anticipate what the novelty will be. An approach that represented a domain and explicitly modeled the novel experiences that are to be expected in that domain would be guilty of anticipating the novelty. Although it might be more robust than an agent that did not model the novelty at all, an agent that anticipates the novelty would be viewed as operating in a closed, albeit larger, world. Therefore, the inductive bias must be specified in a domain-independent way.

It is not feasible to create an agent that can handle all arbitrary types of novelty. Furthermore, the requirement for an inductive bias means that there will be some examples on which the agent performs very poorly, because they run counter to its inductive bias.

Given these considerations, we can define the novelty handling problem as involving three tasks:

Detection of a novel situation.

Characterization of the novel situation by generating appropriate hypotheses and selecting one or more of them to describe the domain under novelty. (Note that because the problem of novelty handling is an online process, the characterization typically will be limited to a partial characterization, based on available observations at the time, and need to improved incrementally as additional observations become available.)

Planning to act in the novel situation by using these new hypotheses to attempt to perform well.

Although an agent cannot hope to characterize and plan well in all possible situations, our goal is to perform well in a variety of natural situations. We have focused mainly on characterization and planning, discussing detection briefly in Section 3.5.

3.2. The Coltrane architecture

We first present a general architecture for novelty-aware agents, and then show how this architecture is realized in Coltrane. Fig. 1 shows the general architecture in two frames, pre-novelty and post-novelty. The pre-novelty frame shows a Sense-Decide-Act agent interacting with the world. After every agent action, the state of the world is updated according to some external *World Behavior*. The agent uses an internal *Domain Model* that is assumed to be a reasonably accurate representation of *World Behavior* as relevant to the agent's decision-making and the tasks for which it was designed.

In the post-novelty frame, the World Behavior is replaced by a changed *World Behavior Under Novelty*. Although the agent's Domain Model pre-novelty is reasonably accurate, this is not the case post-novelty. Instead, the agent must create and use one or more hypotheses about *Estimated Domain Model(s) Under Novelty*. In some cases, the agent will use a single *Estimated Domain Model*. In other cases, the agent will consider multiple *Estimated Domain Models* to be plausible. The general architecture allows for both options.

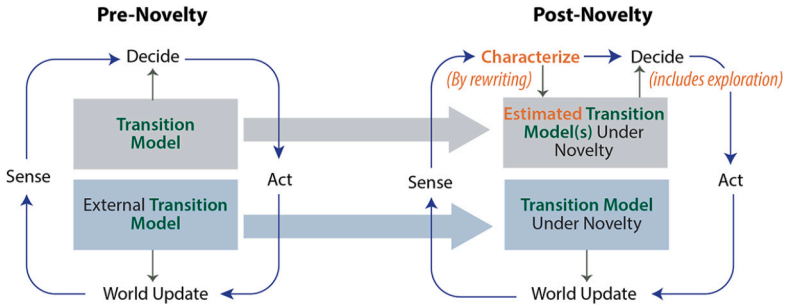


Fig. 2. The Coltrane-specific architecture for novelty-aware agents.

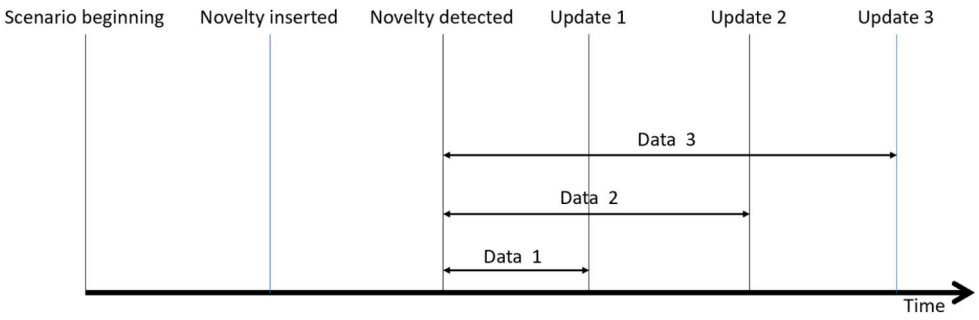


Fig. 3. The dynamic novelty characterization and accommodation process.

To accomplish this change, the agent's process includes an extra step: *Characterize*, which is responsible for creating the *Estimated Domain Model(s) Under Novelty*. It achieves this by considering the original *Domain Model* as a starting point, as well as the newly sensed data. The Decide step then uses the new *Estimated Domain Model(s)* to reason and plan how to act. If there are multiple *Estimated Domain Models*, the agent must reason under model uncertainty when it plans. Detection is not shown explicitly in the architecture as it might not happen in every iteration of the loop. Instead, the responsibility for detecting novelty falls to the Characterize component.

This is a general architecture. It makes no commitment as to the representation of domain models or world behavior. Nor does it specify how the Characterize and Decide steps should work. A particular realization of the architecture must define these elements. Fig. 2 depicts how the general architecture is realized in Coltrane.

In Coltrane, both the *Domain Model* and *World Behavior* are represented by respective *Transition Models*, which are stochastic functions from the current state of the world and the agent's action to the new state of the world. By stochastic function, we mean that the function may be non-deterministic and include random elements. The agent uses its *Transition Model* explicitly in its planning, by considering the effects of its possible actions. In the pre-novelty frame, the agent's *Transition Model* is typically an appropriately abstract representation of the world *Transition Model* that includes the elements required for its decision-making.

Post-novelty, the agent's *Estimated Transition Model(s) Under Novelty* may diverge from the true *External Transition Model Under Novelty*. This may lead the agent to make sub-optimal decisions, because it incorrectly predicts the effects of its actions. The task of the Characterize step is to make the *Estimated Transition Model(s) Under Novelty* as close as possible to the true *Transition Model Under Novelty*. In Coltrane, this is achieved by an algorithm that rewrites the transition model to express the inferred novelty. This algorithm is described in Section 4. The Decide step uses a novelty-aware planning algorithm, which is described in Section 5. An important feature of the novelty-aware planning algorithm is that it uses exploration to learn more about the novelty and to help the Characterize step produce better *Estimated Transition Model(s) Under Novelty*.

3.3. Dynamic operation

Characterization and accommodation of novelty is an ongoing, dynamic process. The agent must operate in the novel world immediately and cannot go off-line or take a long time to characterize the world. As it receives more data, it can update and improve its characterization of the world.

The process is depicted in Fig. 3. At some point in a scenario, novelty is inserted that changes the world behavior. After this change, the novelty must first be detected in order for the agent to be able to accommodate the novelty. After novelty is detected, the agent may repeatedly update its domain model. Each update involves a characterization step that uses all the data since novelty

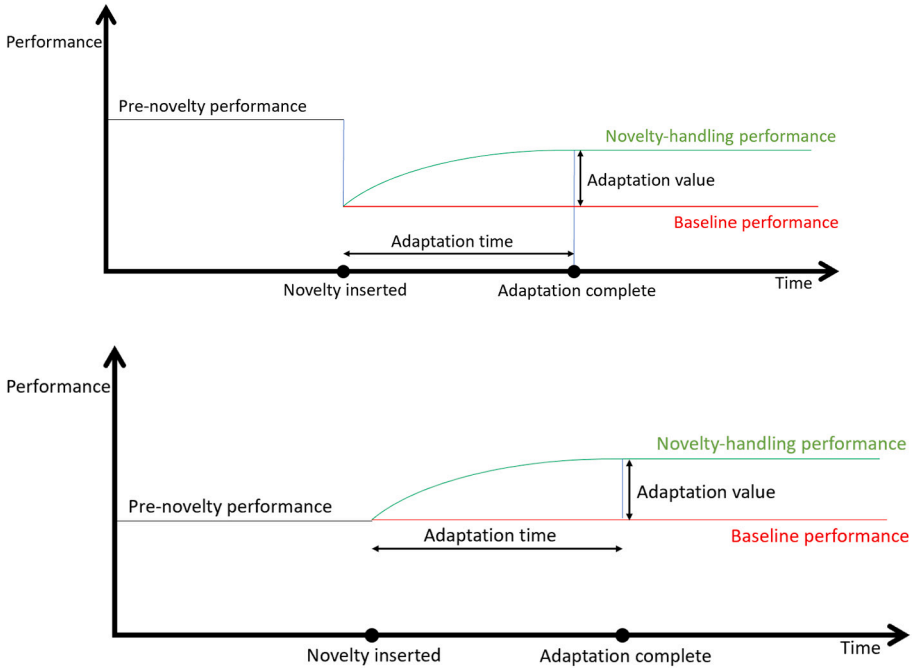


Fig. 4. Performance impact and adaptation under novelty. Top: hazard; bottom: opportunity.

detection until the time of the update.[3] After the update, the novelty-aware planner uses the most recently characterized version of the domain model.

3.4. Classes of novelty

We identify four classes of novelty. Each of these novelties results in a different performance and strategy impact, and understanding these novelty types enables further analysis of the performance of novelty-aware systems.

Hazard A hazard situation is one in which the novelty causes a decrease in performance unless accommodated properly. This kind of situation is shown in the top of Fig. 4. A baseline agent that is not novelty-aware will have reduced performance from the time that novelty is inserted onwards. In contrast, a novelty-aware agent will gradually recover performance. It might not be possible to achieve the original level of performance after novelty, even with an optimal strategy.

Opportunity An opportunity situation, shown in the bottom of Fig. 4, is one in which the novelty does not reduce performance but instead provides an opportunity to increase performance if it is discovered and the strategy is appropriately adapted. Here, the baseline agent will maintain its pre-novelty performance, while the novelty aware agent will gradually increase its performance to a new level. In some cases, the same novelty poses both a hazard and an opportunity. Performance might be negatively impacted for an agent that does not adapt, but an agent that does adapt might manage to even increase its performance beyond the baseline.

Inconsequential novelty In an inconsequential novelty, the novelty affects the agent's performance but has no implications for strategy. The agent's best strategy post-novelty is the same as pre-novelty. Therefore, there is no potential for adaptation and the performance of the baseline agent and the novelty-aware agent is the same. A novelty-aware agent might succeed in detecting and characterizing this novelty, but its overall performance will not be affected.

Nuisance novelty A nuisance novelty is one that does not affect the agent's performance at all over its tasks. In Coltrane, nuisance novelties are typically automatically ignored because Coltrane focuses its characterization on the pre-novelty transition model that was created to accomplish the agent's tasks. This builds on the natural engineering practice of creating abstractions within the domain that are relevant to the task, and not modeling elements that are irrelevant.

[3] More precisely, it uses all the data since the inferred time of novelty insertion, if that is different from the novelty detection time. In Coltrane, these two times are the same.

3.5. Novelty detection

As we mentioned earlier, the agent must first detect the novelty to accommodate it. If the world behavior is deterministic and the agent fully senses the world, detection is straightforward. The agent can simply check whether its prediction of the new world state after an update is correct. All the domains we have worked with are fully observed, with any non-determinism isolated to a small number of mechanisms like die rolls, so detection was not a primary focus of the presented research.

However, we emphasize that although novelty detection is trivial in fully observed, deterministic domains, characterization can be difficult even in these domains. The agent needs to detect not only how the world is different now, but also to predict how the world will be different in the future under a certain set of actions, which requires characterizing latent, unobserved reasons for the observed changes (as opposed to the observed states).

4. Novelty characterization

The two key elements of Coltrane are (1) mechanisms for detecting and characterizing novelty after it is introduced, and (2) mechanisms to plan over the changed world to accomplish its tasks using the learned characterizations. We describe each of these in turn in the following sections.

After novelty is introduced, the Coltrane system works to create a modified domain model that captures the way the world works post-novelty. As described above, Coltrane seeks to immediately create one or more best guesses, using the limited observations that it has at that point, and then as additional observations are available, create improved characterizations with each subsequent observation, until the novelty is fully characterized.

The aim of Coltrane is to rapidly create high-quality, general characterizations of the novelty using few observations, in the same way that people are able to, but do that using domain-independent mechanisms. Coltrane does this by leveraging domain- and task-specific information in the system's pre-novelty domain model to inform its modifications. It does this by using a *domain-independent* hypothesis generation method that applies a set of transforms to the *domain-specific* information naturally present in the pre-novelty domain model, and then searching for the hypotheses that are preferred under its inductive bias. Coltrane's inductive bias is based on Occam's razor, where the simplicity principle is applied to the transformations rather than the domain representation. The inductive bias is implemented in two ways: by a scoring function that assigns higher score to simpler hypotheses, and by a search algorithm that makes it likely simpler hypotheses will be found before more complex ones.

Coltrane's characterization algorithm can be viewed as a probabilistic program synthesis method [28,37,38] that differs from typical probabilistic program synthesis in that it learns a new program that makes targeted modifications to the pre-novelty domain model, rather than learn a new program from scratch. This difference makes the synthesis more achievable. Others have used a somewhat similar approach, where a prior probability distribution over suitable programs, meta-learned from similar kinds of programs, enables efficient search for new programs that characterize the data. This approach enabled one-shot learning to recognize novel characters by learning programs with characteristic patterns of pen strokes [20]. Our approach is similar in that a probability distribution over transformations constraints the programs to be considered and enables one-shot or few-shot learning.

Our domain-independent characterization method relies on the domain-specific pre-novelty domain representation to describe the kinds of abstractions that are used in the search for novel hypotheses. In other words, Coltrane thinks about novelty using the same terms and constructs that the original domain model programmer used. This programmer may have built the domain model without even considering the potential of the agent becoming novelty-aware. Therefore, the programmer will not have written the model to anticipate the novelty, but instead used the natural concepts and relationships to model the domain. Coltrane's approach of using the natural description of the domain to guide the search for novelty is inspired, informally, by the way we think people approach novelty, using their model of the domain to guide possible extensions. For representational generality, when creating characterizations of the pre-novelty domain model, the hypothesis space includes probabilistic extensions of the pre-novelty domain, both for generality and to facilitate the probabilistic program synthesis.

Using an Occam's razor principle over transformations has several beneficial properties:

1. **Optimizing for generality by reusing of domain and task abstractions present in the base model:** By optimizing for minimal changes to the code model that explain the full range of observations, reuse of abstractions present in the base model are preferred over a larger set of (less abstract) changes across the model that explain the same set of observations. Where this bias captures generality appropriate to the introduced novelty, it will be maintained. If subsequent observations are inconsistent with this generality, the natural search will replace it with fewer changes to the code in less abstract locations that focus on the observed novelty. For example, if tax payments and prices of rents in Monopoly are both increased in a similar way, Coltrane will prefer a more abstract hypothesis that explains the changes through a general pricing mechanism rather than a less abstract explanation involving changes to both taxes and rents, even though both hypotheses explain the data equally well. This is achieved as long as the pre-novelty transition model representation already contains the abstraction that taxes and rents are both instances of monetary amounts. This example illustrates how our domain-independent characterization method interacts with a domain-specific pre-novelty domain representation.
2. **Minimal, targeted changes to the model:** By optimizing for minimal changes that explain the full set of non-novel and novel observations, Coltrane's characterization mechanisms seeks to only change targeted elements of the domain model, while leaving all of the elements unrelated to the novelty unchanged.

3. **Multiple hypotheses are created when observations are inconclusive:** When there are very few observations of the novelty, even with bias toward simplest explanations, there can be multiple characterizations that are equally supported by the evidence. Coltrane creates a distribution of modified domain models that capture these hypotheses, both for subsequent refinement as new observations arise that inform each hypothesis, and for direct use by reasoning for accomplishing its task in the face of uncertainty over the accurate explanation of the novelty. This explicit representation of model uncertainty also motivates and guides Coltrane's exploration.

In the following subsections, we present details of the characterization algorithm. We begin with an overview of the algorithm, and continue with the hypothesis space, the scoring function, and the search algorithm.

4.1. Overview of characterization algorithm

The core of our characterization approach is to recognize deviations of pre-novelty domain models using observations of the world, and adapt the domain models to the changes in the environment.

We consider a highly general class of domain models describing state transitions via probabilistic programs $T(\Phi_{t+1}|\Phi_t)$.

In the course of execution one observes sequences of states $\Phi_0, \Phi_1, ..., \Phi_n$, but the true domain model is unobservable. We can then treat inference of a novel domain model as a Bayesian filtering problem where a distribution over the novel domain model T' is inferred from the previous sequence of observations:

$$P(T_{novel}|\Phi_0, \Phi_1, ..., \Phi_t) \propto P(\Phi_0, \Phi_1, ..., \Phi_t|T_{novel})P(T_{novel}) \tag{1}$$

Note here the need for a prior distribution over programs $P(T_{novel})$. In practice there are combinatorially many programs consistent with any finite sequence of states, so having a narrow prior for plausible transition models is critical to producing useful results (i.e. domain models that are predictive for future impacts of a given novelty). The main idea is that we use the original domain model as the basis for specifying this prior, and assign higher probability to prefer transition models that deviate in small ways from the original domain model. Directly specifying such a prior over domain models is difficult, however, as one needs to somehow define a distance between model programs that reflects semantically plausible novelties.

To enable inference of a novel domain model, our approach includes three elements:

Specify a hypothesis space of program perturbations that are considered for the novelty.
Specify a prior distribution that represents the prior probability of a program perturbation based on its size.
Implement a search algorithm to search for a hypothesis in the hypothesis space that has high prior probability and also fits the data.

We now explain each of these elements in turn.

4.2. The hypothesis space

Since we are permuting a transition model that is represented in code, Coltrane's hypothesis space consists of four elements:

1. The places in the program that can be perturbed
2. The elements in the program in those places that can participate in the perturbations
3. The variables that the perturbations depend on
4. The representation of the perturbations

We illustrate the hypothesis space through a running example[4]: Consider the following function that computes the rental price of a property that does not have houses. In standard Monopoly, this price is doubled if the player owns all properties of the same color. This *rental_price* function may be perturbed to match output from a novelty that changes rental prices owed.

```
1 function rental_price(property)
2   c = color(property)
3   o = owner(property)
4   if !has_monopoly(o, c)
5     return base_rental(property)
6   else
7     return base_rental(property) * 2
8   end
9 end
```

[4] The specific implementation we are currently using for this research is based on reflection capabilities in the Julia language. The examples are grounded in its intermediate representation (IR). Application to languages without internal reflection support could be done using various source-to-source translation and compilation mechanisms.

Julia compiles this to an intermediate representation (IR) that we operate on directly:

```
1 -      c = Main.color(property)
|        o = Main.owner(property)
|   %3 = Main.has_monopoly(o, c)
|--      goto #3 if %3
2 - %5 = Main.base_rental(property)
|--      return %5
3 - %7 = Main.base_rental(property)
|   %8 = %7 * 2
|--      return %8
```

What places can be perturbed?

In principle, every function call site in the transition model could be replaced by a different function. We consider a static single assignment representation of the code, so that the result of every function call is assigned to a variable. In our example, the following sites are considered:

- The assignment to `c` from `color(property)` in line 2
- The assignment to `o`
- Temporary variable assignment `has_monopoly`
- Return value assignment by `base_rental` line 5
- Temporary assignment `base_rental` in line 7 to variable `_x`
- Return value assignment by `*` in line 7

Changes within called functions can also be considered. For example, call sites within `color`, `owner`, `has_monopoly`, `base_rental` are also candidate call sites.

What elements in those places can be perturbed?

A perturbation is a replacement of the function at each of these call sites by a different function. For a hypothesis of a perturbation at a particular site, Coltrane synthesizes a new function to replace the original one. As a result, the value produced at any callsite can be perturbed to describe a novel model. Although mutations are allowed in general in the transition model, we do not consider novel functions that directly mutate variables that have been defined previously, meaning that the perturbations cannot change the values of variables defined previously. However, existing mutating calls can themselves be perturbed to change the effect of the mutations.

What can the perturbations depend on?

If a call site has been chosen for a perturbation, that call site is originally associated with a function that takes a certain set of arguments and returns a value. In the simplest perturbations, we replace the original function with a new function that takes the same arguments. For example:

- Replacing the `base_rental` function in line 7 to return a different set of base rental prices;
- Changing the implementation of multiplication (e.g., to add 1 to the result);
- Changing the definition of `has_monopoly` to always return true, so that the base rental price is always doubled

A much wider, more powerful set of hypotheses can be obtained by changing the set of arguments that can be passed to a function. This allows for new program calling context to be introduced in the definition of a function under novelty, as well as further abstractions, generalizations, and simplifications. For example, consider a novelty where $20 is added to the rent, but only for red properties. To represent this, we must make the color of the property, represented by the variable `c` from line 2 in the program, an extra calling context variable.

To enable this, any value that was the result of a function call previously in the program can be used. In order to be considered, this value must be generated before the function call under consideration in every run in the data. To achieve this, we introduce an auxiliary function call after the existing assignment that takes the result of the assignment, along with other values that are to be considered. The possibilities for this new function call are the same as for the replacement function call described above. For example, a perturbation that the rental price of a property depends on its color can be achieved by replacing `base_rental(property)` in line 5 with the lines

```
_x = base_rental(property)
return f(_x, c)
```

Here, `_x` is a temporary variable to hold the base rental price while `f` is a new function created to process the base price, in a way that is dependent on the color `c`, and return the result.

How are the perturbations represented?

The key representational structure for the domain model characterizations is one or more functions at individual call sites in the single state assignment representation of the pre-novelty transition model. Since single state assignment IR is a decomposed,

linearization of the logical flow of the domain model code (as shown in the `rental_price` function example above), this is equivalent to making any source-code level change in the logical computation of the pre-novelty transition function program, with the change ranging from changing a constant to introducing modified dynamics at any point in the logical flow. For a simple novelty, this might entail changing a single line in the IR. More complex novelties will typically require multiple points in the logic to be changed, requiring changes to multiple lines of the IR by fitting a function to each using the call site and value information as described in Section 4.4.2. Note that because of normal functional abstraction in code, and the ability to introduce code changes e.g. both within a function, within a helper function, or at every call site to a function, this allows for both general modifications (e.g. by placing the change inside a function or helper function), and situation-specific modifications (e.g. by placing the change at multiple call sites to the function). Because of Coltrane's Occams razor principal, the synthesis mechanism will prefer single, more universal positions in the code over multiple, less-universal ones if both equally explain the observations, as described below.

Using the call site, chosen argument, and value information for the functions, we need to determine the contents of the function. Our approach is general and can allow any representation of functions, provided there is a learning method to learn that representation from input-output data. The ideal representation and learning method are subject for further research. In Coltrane, specifically, we represent the function as a random forest, associating each partition of the input space with an output value of the appropriate type.

Note that perturbation models are specific to the call site for a given method, so, e.g., if a particular invocation of `has_monopoly` is perturbed then the function will retain its meaning in other places. If the correct novelty is defined by universal redefinition of `has_monopoly` then perturbations can be made inside of its implementation directly instead. In this way, perturbation models enable us to capture the appropriate degree of generality for the novelty, neither being too specific and missing functions that the novelty changes, nor being too general and changing functions that the novelty does not change.

The hypothesis space we have described is enormous. The search method, using the inductive bias towards simplicity, enables Coltrane to find a simple hypothesis that explains the data, provided such a hypothesis exists. However, if the data is such that none of the simple hypotheses are sufficient, Coltrane is potentially able to discover a more complex hypothesis. Therefore, Coltrane is able, in principle, to characterize large changes to the domain, provided there is enough data to do so.

4.3. The prior probability

Now that we have defined the hypothesis space, we must specify how we evaluate one perturbation over another. This is done in a Bayesian way, combining a prior distribution over perturbations $P(\Pi)$ and a likelihood function for the data given the perturbation.

Since we want to characterize the length of the perturbation Π, a natural heuristic is to measure the length (in bits) $L(\Pi)$ and use it in the construction of the prior:

$$P(\Pi) \propto 2^{-L(\pi)} \tag{2}$$

Π is a representation of a perturbation. It consists of:

- The call sites being perturbed.
- The change to the specification of the call sites, e.g. through additional arguments.
- The new implementation of the call sites themselves.

In Coltrane, this information is represented in a data structure in Julia. Julia provides a mechanism to compute the length in bits of any data structure, which is used for the prior. This prior is then plugged into Equation (1) to obtain the scoring function.

The approach we have defined is domain-independent and relies only on the specification of the program. We expect that a good programmer will introduce useful abstractions into the specification of the program that can also be used to characterize novelty. For example, a programmer will write a generalized function for calculating Monopoly property rent from a table of property data, and this function will rely on other functions that compute whether a player owns all properties of a group and how many houses and hotels the property has. Poor abstraction, on the other hand, would be to encode individual functions for each property, duplicating code and not sharing concepts among the properties. If the program contains useful abstractions, novelty hypotheses that match these abstractions will also have short representations as perturbations. In other words, we rely on the human programmer's description of the baseline domain as a universal inductive bias for novelty hypotheses. The extent to which Coltrane will be successful using this inductive bias depends both on the degree to which the programmer actually captures the important aspects of the domain and on the degree to which novelty in the world is organized along the structures represented in the program. We believe that for practical applications, with a well-written original program, small perturbations to the program correspond to naturally occurring novelties. This amounts to a fundamental belief in the regularity of the world. Coltrane might therefore perform less well under adversarial novelties that are specifically created to alter large portions of how the program works.

4.4. The search algorithm

The goal of the search algorithm is to find a perturbation that has a high combined prior probability and data likelihood. We define a perturbation to be a higher-order, function of the transition model T. In general, T is a function from a complete state Φ_t

to a new state Φ_{t+1}. A perturbation Π changes T to be a different function on Φ. Ideally, we would like Π to make this new function match the data as well as possible, besides having a high prior. In other words, we would like:

$$\Phi_{t+1} \approx \Pi(T)(\Phi_t) \quad \forall t \in [0, n] \tag{3}$$

Coltrane's characterization search proceeds in two steps. First, for each time point at which we have data, we find a set of candidate perturbations that explain the data in that time point. Second, we unify these instances to create overall hypotheses to explain all the time points at which we have data. For example, if only a single call site is involved in the perturbations for individual time points, then that call site is likely to be the target for the entire program perturbation for all time points. However, if multiple call sites are involved in the individual perturbations, then a more complex unification method is needed to identify the call sites to cover all time points. These two steps are described in detail in the next two subsections.

4.4.1. Search for perturbations that fit a single time point

The first step of our algorithm constructs what we call *instance perturbations (IPs)*. An IP is a least general perturbation that fits a single training instance. IPs accomplish this by specifying value changes at particular points in the program; they are least general because they specify value changes rather than more general functions. Later, we will generalize the specific values into functions by considering the IPs for multiple training instances.

In general, a given training instance may have multiple IPs that correct it, as multiple changes in the program may produce the desired result. We introduce the *FitIPs* algorithm to produce a covering set of perturbations that can contribute to full, unified perturbations across training instances downstream. Precisely, a call to FitIPs(Φ, Φ') constructs a set of IPs, such that each IP_i consists of value changes $\text{IP}_i[\alpha_j] = y_{\alpha_j}$ where α_j is one of a set of call sites in the program whose value is changed and y_{α_j} is the value that it is changed to in that call site. Ideally, the FitIPs algorithm identifies the complete set of IPs such that applying any IP_i corrects the program prediction for a single observed transition $\Phi \rightarrow \Phi'$. We can then think of each IP_i as a perturbation that can be applied to correct a single call to the transition model:

$$\Phi' = \text{IP}_i(T)(\Phi) \tag{4}$$

In practice, it is not feasible to compute a complete set of perfect IPs. The total number of potential IPs may be unbounded. Even if the call sites in an IP are known, finding a correct parameterization that fits the data perfectly is still equivalent to solving some sort of black-box optimization problem (while forgoing potentially intractable program analysis of the model, at least). Therefore, FitIPs uses an optimization algorithm that makes two approximations. First, it does not attempt to find the complete set of IPs, but rather attempts to find as many small IPs as is possible in some bounded time. Second, it does not require the IPs it finds to perfectly explain the data, but rather correct the transition model output to as large a degree as possible.

For efficient optimization, it is desirable to have a distance function that is tailored to the specifics of the domain, so that perturbations that have similarly semantic impact in the domain, are calculated to have a similar distance, and e.g. accidental differences in the order of magnitude of constants used don't pollute the distance measurement. For example, in monopoly, a permutation that changed which square the player landed on might be a constant change of 1 or 2 in the location data structure, while a change in the rent paid would typically be in the 100s. It would not be desirable for efficiency of the search to have the second judged as a 100x larger change than the first.

To have a tailored, informative distance metric that provides these semantic properties, we again rely on a domain independent mechanism, that then leverages the knowledge already encoded in the pre-novelty domain model for the specifics. In this case, to measure the degree to which an IP corrects the model output, we use a natural distance metric d on pairs of states Φ, Φ' that is constructed such that the domain dynamics appear like a diffusion process under d. Specifically, the distance metric is defined by a kernel $d(x, y)^2 = \mathcal{K}(x, x) + \mathcal{K}(y, y) - 2\mathcal{K}(x, y)$ where $\mathcal{K}$ is fit to domain dynamics s.t. $E[(\mathcal{K}(\Phi', \Theta) - \mathcal{K}(\Phi, \Theta))^2] = \mathcal{K}(\Theta, \Theta)$ for all states Θ.

With this distance function, and after novelty is detected, Coltrane synthesizes program perturbations on the transition function program to characterize the change in the world, using all observations perceived up to that time. As new observations are later received, this process is repeated to further refine the characterization.

To accomplish the approximate optimization, FitIPs first traces the execution of the transition model and records all method invocations as well as their results. It identifies call sites where the returned value is different from the expected value as candidates. Then it implements a random search to select candidates and replace the call with a constant representing the observed returned value. This search uses some basic heuristics informed by stack depth, method module, and other factors as a prior for what call sites are likely to change. We then evaluate the result of the transition model with the new value applied and determine if it makes the result closer or further from the observed result. This is employed in a Markov-Chain Monte-Carlo algorithm to find a distribution of IPs that improves the result accuracy. The posterior we solve for is defined as follows:

$$P(\text{IP}|\Phi, \Phi') \propto P(\text{IP}) \exp\{-\frac{d(\text{IP}(T)(\Phi), \Phi')^2}{2d(T(\Phi), \Phi')^2}\} \tag{5}$$

Consider again the `rental_price` function above. Presume that one observes after landing on a property that the amount subtracted from a player's wallet and added to the owner's is different from expectation. For instance, suppose the normal rent is \$40 and it is instead observed to be \$60. The FitIPs search algorithm would identify that an IP that replaces the returned value of `base_rental` for the specific property with \$60 causes a perturbation of the original program Φ, that produces the expected

observation of $40, to be replaced with a new program Φ' that produces the actually observed value of $60. One might also produce an IP that adds $20 to the value returned by `base_rental`. This illustrates the fact that individual latent changes are often ambiguous. In this example, the IP that adds $20 is general and applies to all properties, while the one that modifies the return value of `base_rental` applies only to the specific property. This ambiguity can be resolved in the next step by comparing observations of the rental value of other properties. On the very first observation, the ambiguity cannot be resolved.

4.4.2. Creating full hypotheses from the training instances

Each of the IPs generated by the previous step satisfy (approximately) Equation (4) for a single training point, but may predict very poorly at other training points. Therefore, the next step of the algorithm unifies the IPs for individual training points into a single *universal perturbation UP* that explains, as well as possible, all the time points. To motivate the process, consider that the previous set has proposed multiple candidate IPs for each instance. We expect one of those IPs to be correct for that instance, but we do not know which one. The question is resolved by considering how well each IP for this instance fits with candidate IPs for other instances to explain all the data. So we must consider a complete selection of IPs, one for each instance, create a candidate UP, and evaluate it. Creating the candidate UP consists of forming the structure of the UP, including the calling context variables that each function in the UP uses, and then fitting those functions. Evaluating the UP considers the prior probability and fit to data.

The algorithm to create UPs from the set of IPs is accomplished in several steps:

1. Proposing a set of IPs, one for each training instance. This is done by an importance sampling algorithm that constructs the set one IP at a time. At each step of the algorithm, it randomly selects a training instance i that has not yet had an IP selected from it. It then selects an IP for that training instance according to a categorical distribution. The probability of each IP considers (a) the prior probability of the perturbation from Equation (2); (b) its accuracy in correcting the output; and (c) its distance from each of the previously selected IPs under some naturally defined distance q for IPs. Specifically, the probability $p_k(\text{IP})$ at round k, where i is the selected training set in round k and IP is a candidate selection for that training set, is given by:

$$p_k(\text{IP}) \propto P(\text{IP}) \exp\{-\frac{d(\text{IP}(T)(\Phi_k), \Phi_{k+1})^2}{2d(\Phi_k, \Phi_{k+1})^2}\} \prod_{i=1}^{k} \exp\{-\frac{q(\text{IP}, \text{IP}_i)^2}{2}\} \tag{6}$$

   Intuitively this creates sets of candidate instance permutations that score higher the more they fit all of the training observations, correct all of the novel training observations, are compatible with each other (making it more likely that the choice of IPs in the set lead to accurate generalizations), and fit the prior including being consistent with the pre-novelty domain model. In the larger inference process, this joint probabilistic scoring steers the Bayesian inference process toward better choices for where to place the perturbations in the model to provide good characterizations of the novelty.
2. Identifying the set of call sites that are changed in the UP. Different IPs may have different call sites, so we take the union of the call sites in each of the IPs, and extend the IPs by adding any missing call sites with the true value from the corresponding execution trace.
3. Selecting a random set of calling context variables x_j for each call site α_j. This is done by independently deciding, for each candidate variable (i.e., it has been defined previously to this call site), whether it is in the set of inputs.
4. Fitting a function f_j for each call site α_j in the UP. For the call site α_j and instance i, let x_j^i denote the values of the selected calling context variables to the fit function. Let y_j^i denote the value associated with α_j in IP_i. The goal is to fit f_j such that $f_j(x_j^i)$ is as close as possible to y_j^i for each training instance i. As we have discussed, our approach is general and can accommodate different function representations and fitting methods. In our approach, as mentioned, we use random forests to represent and learn the function.
5. Steps 3 and 4 are repeated until an accuracy threshold is reached under cross-validation (or until timeout).

This search process when embedded inside the larger Bayesian filtering process steers the probability distribution of permutations to the pre-novelty domain model to ones that are both generalized, while also specific to fit both the novel observations and the unchanged observations.

Again consider the `rental_price` function defined above. Consider a case where rent at all red-colored properties had increased by $20. So, in one example, a player landing on Kentucky Avenue in turn t, which is a red-colored property, might pay $38 in rent instead of $18. For any turn where rent was paid on a red property there would be a set of IPs produced, likely including changes to the result of `base_rental`. For this turn t in particular one would expect an IP changing the result of `base_rental` to $38 (or something close by). Another possibility is that the property Kentucky Avenue has been replaced by Park Place, which has a rent of $35 that is also close to $38. As a result, FitIPs might return the following pair of IPs for this observation.

$$V_t = \begin{cases} \ldots \\ \{\texttt{rental\_price @ IR[32]} : 37.9\} \\ \\ \{\texttt{property @ IR[29]} : \text{“Park Place”}\} \\ \ldots \end{cases} \tag{7}$$

On a different turn s the player lands on Illinois Avenue and pays \$40 instead of \$20. FitIPs might return:

$$V_s = \begin{cases} \ldots \\ \{\texttt{rental\_price @ IR[32]} : 41.5\} \\ \{\texttt{rental\_price @ IR[32]} : 39.9\} \\ \ldots \end{cases} \tag{8}$$

Next we would select a IP for each turn biased by accuracy and joint similarity. Suppose that the agent also visits two properties with non-novel rents: a purple property with rent \$4 and a pink property with rent \$12. One likely set of IPs, with one for each training instance, looks as follows:

$$y = \begin{cases} \ldots \\ \{\texttt{rental\_price @ IR[32]} : 37.9\} \\ \ldots \\ \{\texttt{rental\_price @ IR[32]} : 4\} \\ \ldots \\ \{\texttt{rental\_price @ IR[32]} : 12\} \\ \ldots \\ \{\texttt{rental\_price @ IR[32]} : 39.9\} \\ \ldots \end{cases} \tag{9}$$

Here IPs generated for novel transitions are interspersed with IPs filled in on non-novel transitions that are populated with the default value from execution.

To fit a UP we randomly sample calling context variables either nearby in scope or from the top-level transition function input as arguments to a perturbed function. Here, for instance, we would like the color of the property and the nominal rent value to be an argument. Given that both of these are nearby in scope, they are likely to be selected quickly. For any selection one would produce a function fitting problem with inputs as follows:

$$x = \begin{cases} \ldots \\ \{\texttt{color @ IR[29]:"red", \_rental\_price @ IR[32]} : 18\} \\ \ldots \\ \{\texttt{color @ IR[29]:"purple", \_rental\_price @ IR[32]} : 4\} \\ \ldots \\ \{\texttt{color @ IR[29]:"pink", \_rental\_price @ IR[32]} : 12\} \\ \ldots \\ \{\texttt{color @ IR[29]:"red", \_rental\_price @ IR[32]} : 20\} \\ \ldots \end{cases} \tag{10}$$

Having selected inputs, a function fitting approach is used to try to predict the IP value from the conditioned inputs on each observed transition instance. This can return the following function:

```
if(color == "red")
    return _rental_price + 20
else
    return _rental_price
end
```

5. Planning under novelty

After characterizing or partially characterizing the change in the world, Coltrane needs to be able to plan and act effectively using the new information to accomplish its goals. As novelty is characterized, Coltrane currently uses the hypothesis that is estimated to be most likely as the planning model.

5.1. Challenges in planning under novelty

Planning with novelty presents a unique set of challenges for current planning systems.

First, the decision-making of the agent must be updated online; the agent cannot pause or stop to retrain on new data or be re-coded with new insights. Therefore the agent cannot operate with only a fixed, pre-learned or pre-encoded policy. When the novelty

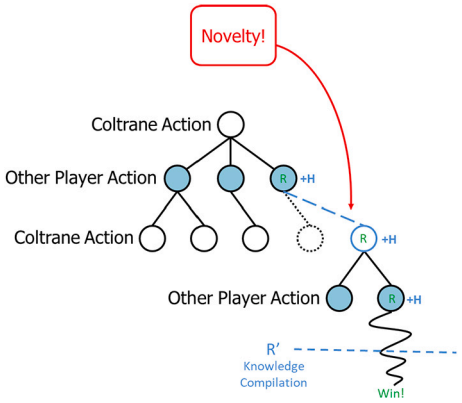


Fig. 5. A MCTS with novelty injected.

becomes partially or fully characterized the agent must use this new characterization in its reasoning. Solely reinforcement learning approaches, such as AlphaStar [43], and hybrid approaches, such as AlphaGo [41], do not have this capacity to adapt to novelty online without modification because the pre-learned models do not take into account the novelty.

Second, the agent operates in large domains that are typical of real-world problems; the agent cannot feasibly search or effectively sample a significant portion of the plan space in the time that it needs to act. Naive search-based approaches fail to find solutions in time due to significant branching factors. Sampling-based approaches such as vanilla Monte-Carlo Tree Search [5] cannot sample the domain effectively to make any but the most local of plans. Instead, pre-learned or pre-encoded heuristics are needed to reduce the search space and identify the most promising samples. These heuristics must also adapt to novelty.

Third, the agent must continue to plan and act when the novelty has been detected and may only be partially characterized, introducing uncertainty in the agent's transition model. This model uncertainty is qualitatively different from underlying uncertainty in the world (e.g., the resulting value from rolling dice). Observing examples of the novelty provides additional information to the agent about the nature of the novelty, and the agent can plan to act and observe to reduce this uncertainty. Other treatments have examined Markov Decision Process model uncertainty from perspectives such as reinforcement learning [45], which is not applicable to our online planning needs.

Fourth, the agent must consider taking actions to explore novelty and reduce uncertainty around novelty, not just act to achieve it's immediate goal. Beyond using the partial characterizations of novelty, the agent may need to explore off-policy actions and spaces to find suspected novelty in its environment. Agents that only pursue primary goals may miss beneficial novelties that are off the usual path, and they may fail to fully characterize harmful novelties and discover improved plans and strategies.

5.2. Coltrane approaches to planning under novelty

5.2.1. Online planning with novelty using Monte-Carlo tree search

To provide online planning, Coltrane uses Monte-Carlo tree search (MCTS) [5] with a transition model of the underlying domain, a reward function for the outcomes of the domain, and domain specific heuristics to enable efficient search. Each current state and hypothesis pair presents a planning problem: A planning problem P is a tuple (s_0, T, A_g, R) where s_0 is the current state; T is the transition model of the form state, action, probability distribution over new state $f(s,a) \rightarrow s'$; A_g denotes the other agents with their (perhaps partially known) policies $\pi(s) \rightarrow a$; and R denotes the reward vector for each agent over the states $R(s) \rightarrow [R_0, R_1, \ldots, R_n]$. Because we assume full observability of the domain in this paper, we do not include an observation model.

For stochastic adversarial domains, such as Monopoly, Coltrane uses MCTS-UCT [5], which enables compact state representation and online weighted sampling of possible actions to maximize rewards. MCTS-UCT attempts to find the expected reward Q value of an action such that:

$$Q(s,a) = \frac{1}{(N(s,a))} \sum_{i=1}^{N(s)} I_i(s,a) z_i$$

where $N(s,a)$ is the number of times action a has been selected from state s, $N(s)$ is the number of times a game has been played out through state s, z_i is the result of the ith simulation played out from s, and $I(s,a)$ is 1 if action a was selected from state s on the ith play-out from state s or 0 otherwise [5]. MCTS approximates this function by sampling from the tree of potential actions and states. UCT uses a balancing term to manage tradeoffs between searching seemingly good actions and searching unknown actions. MCTS-UCT applies directly to the Coltrane planning problem by using (s_0, T, A_g, R) to form the search space and reward functions [13]. Heuristics are added to the UCT term during search and state evaluation to weight the search to more favorable states. For example, in Monopoly it is generally better to have more net worth than less.

Fig. 5 shows an example of MCTS game-tree planning under novelty. The root of the tree represents the current state in which Coltrane is choosing its action. In this tree, the actions alternate between Coltrane and another player. MCTS (1) selects a node that represents a state, (2) expands the tree with the possible moves under that state, (3) uses a default policy to simulate the game to a

conclusion or natural scoring location where the state value is functionally estimated, and (4) propagates the updated reward of that state to parent nodes. In this diagram, Coltrane has characterized a novelty as changing the resulting state of one of the possible future actions, on the right. The updated transition model from characterization generates a different state than the pre-novelty transition model, and this state is used in the search tree.

The use of the updated transition model from characterization provides Coltrane its first method to adapt to novelty. A key insight is that the MCTS search can accommodate a novel transition function that does not break any of the major assumptions about the search space, including the structure of the reward signal, the approximate average branching factor, and the necessary number of MCTS rollouts, search depth, or simulations to provide a useful estimate of a state's true reward. In practice in our testing, this seems to be a significant set of novelties for games like Monopoly. Once the MCTS hyperparameters are tuned to play the base game, novel transition functions are often close enough that they do not require changing the hyperparameters to play well. For example, a novelty that adds $20 to the rent values of the red properties in Monopoly affects the relative value of owning these properties, but it does not change the search depth needed to estimate the value of projected states. Since each action is planned with a new search from the current state, a new MCTS tree is created when novelties are characterized.

Coltrane is able to plan with a single updated transition function and, in many cases, adapt online without further modifications to the search processes. This updated transition function is a single hypothesis from characterization. With domains and novelties in which the characterization process converges on a single transition function relatively quickly, Coltrane can simply plan with a random transition function from the candidate sets until the novelty is fully characterized. In cases in which characterization is slow to converge or periods of poor planning are risky to overall performance, explore/exploit reasoning is needed (see Section 5.2.3).

5.2.2. Adapting heuristics to novelty

To increase the effectiveness of MCTS planning in large domains with deep search trees and high branching factors, Coltrane uses heuristics to augment the reward signal from MCTS simulation. Heuristics are often essential to make search computationally practical in moderate sized domains (such as non-novel Monopoly, Minecraft, and extensions to real world tasks). Heuristics make underlying assumptions about the structure of the domain to aid search efficiency. They may be soft preferences (e.g., having more money in non-novel Monopoly is often better than having less), hard constraints (e.g., current opponents will never trade away a monopoly, so do not add that possibility to the search space), or simplified simulations that can run exceptionally fast (e.g., estimating the expected current cash flow to a Monopoly player if all players simply rolled the dice, paid rent, and passed their turns).

A key consideration is that novelty may break the underlying assumptions made by a heuristic, rendering it less effective, void, or counterproductive. For example, a heuristic may indicate that in non-novel Monopoly, owning more expensive-to-buy properties is often better than owning less expensive properties, but a novelty that changes the rent structures across the properties may cause the less expensive properties to produce higher rent and become more valuable to own. Moves such as mortgaging properties that should only be considered as a last resort pre-novelty may then become profitable and always a viable option under novelty. And usually important factors such as expected cash flow may become less important when novelty prioritizes immediate rewards.

Adapting heuristics to novelty is a different problem than general purpose automatic heuristic formulation for entirely new domains. Because novelty represents a relatively small change to a domain, most of the heuristics may still be applicable post-novelty, improving search results. Coltrane uses and adapts the pre-novelty heuristics to provide useful estimates on the post-novelty action rewards. This includes directly deleting selected heuristics that were affected by novelty, rebalancing and learning heuristics (which can include additional implicit deletions), and functionally adapting heuristics. Additional methods of repairing or generating new heuristics online, cf., [18], in novel situations are the subject of future research.

To rebalance heuristics (including deleting heuristics that are no longer relevant) and learn new heuristics, Coltrane includes mechanisms to use the sampling of MCTS search to learn an updated heuristic and feature weighting function post-novelty. Intuitively, this leverages the knowledge of how the way the world now works that has been created by characterization, to update the heuristics combinations. Coltrane must still take actions online, without significant retraining time, and it therefore cannot retrain to discover detailed policies from massive amounts of data. However, the MCTS sampling of the domain provides a set of data to tune an overall heuristic function from component parts with e.g. regression or polynomial function learning. For example in Monopoly, if a novelty makes the owning of high rent properties suddenly more valuable than in the pre-novelty game, the heuristic contribution of average property rent can be increased to match the win/loss reward signal sampled to MCTS. These functions can include the pre-novelty heuristics (e.g., property value, cash value, expected cash flow, board position in Monopoly) as well as individual game features (e.g., number of properties owned). When the sampled weight reduces to zero, the re-balancing effectively deletes the newly-invalid heuristic. While in cases where the heuristics have enough structure where they could be rebalanced, this is approach is promising, in our experiments we did not have many situations where heuristic rebalancing of this sort resulted in significant performance gains, and Coltrane's other heuristic adaptation techniques accounted for the majority of the improvements.

Coltrane's final heuristic adaptation technique is in the case of functionally defined heuristics: where the pre-novelty heuristics are functionally defined using the transition model, these heuristics automatically adapt when characterization updates the transition model to more accurately match the world model. For example, the strategic value of owning a property on the current rent structure and likelihood of landing on it on the current board is computed online as a heuristic of the original planning process. When the novel transition function changes the rent structures and probability of landing on properties, the heuristic automatically recomputes the relative value of each properties as part of its normal operation, that now benefits from the changed (and unchanged) knowledge provided by the updated transition function. For example, if Coltrane has characterized the novelty as rents on red properties being increased by $20, the effect of this change on the strategic value of owning these properties will be automatically incorporated into the planning.

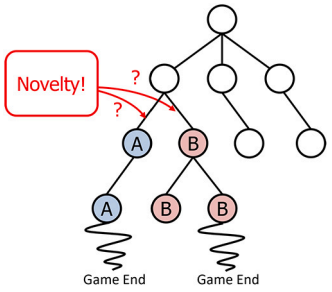


Fig. 6. Coltrane searches over the hypothesis space, branching and maintaining hypothesis states as the search tree goes deeper.

5.2.3. Planning under model uncertainty using explore / exploit reasoning

Because novelty must be handled online, Coltrane must continue to plan and act after novelty has been detected and before novelty has been fully characterized. This creates a challenge for the agent to reason under uncertainty about the transition model. In this situation, Coltrane has multiple hypothesized transition models created via characterization. Combining these transition models into the same search space enables the agent to balance exploring novelty with exploiting known properties of the domain. It enables the agent to manage the risk and reward profiles of uncertain eventualities.

Fig. 6 shows a branch point at which either the mutually exclusive transition model A or transition model B invokes the outcome. At the time of search, Coltrane is unable to tell which of A or B is the correct transition outcome on a single action. The search splits the space between A and B and marks the search node with the new possible set of hypothesized transition models, either ruling out B and leaving A as the only option in this case or vice versa. Coltrane must keep up with this information so that when it comes to a further point at which transition model A differs from B, the result of the relevant action is consistent with the path chosen higher in the search space. For example, if A states that rent paid is 200 percent and B states that rent paid is 50 percent, exploring the A branch will always return 200 percent of the rent.

In general, each node in the search space maintains a list of possible transition models that may be active at that point. When transition models differ on the results of a resulting actions, the search space is split on the results and the consistent transition models for each sub-branch are kept. When rolling up the utility of the sub-branches, the choice of transition model is treated as a random event and the sub-values can be combined with average, minimum, maximum, or other function that meets the agent's policy towards unknown events.

6. Evaluation and demonstration

In this paper, we have presented the difficult task of developing a domain-independent method for agents that characterize and plan under novelty. Our general approach applies to a wide range of systems and provides mechanisms others can use, including augmenting our specific methods, e.g., for probabilistic program synthesis, with other methods. Here, we first present demonstrations that show in detail how Coltrane operates end-to-end in interesting situations and then present the results of a formal single-blind evaluation of Coltrane by an external team.

6.1. Demonstration and evaluation

We illustrate the operational behavior of Coltrane with a set of demonstrations in diverse domains. First, we demonstrate four novelties in a small maze pathfinding grid world domain to inspect the approach in a well controlled single agent environment. Second, we demonstrate two novelties in a Monopoly game with intelligent adversaries to inspect the approach in a relatively complex adversarial turn based environment. Third, we demonstrate a novelty in a Doom game simulation to inspect the approach in a real-time adversarial environment.

6.1.1. Maze pathfinding with novelty

In this domain, an agent is tasked to explore a grid maze of discrete tiles and find the least costly path from a start position to a goal position by making moves along a grid in the four cardinal directions. The grid is filled with symbols that represent open space, walls, and the goal position. The agent is not allowed to move into or across walls. In these experiments we generated mazes of size 25x25 with the outer border as wall tiles. Each movement through open space costs ten (10), and each maze is generated with at least two paths to the goal. Fig. 7 shows a depiction of one maze problem. The agent uses A* search as the planner to find the minimally costed path to the goal tile. Without novelty, A* is able to find the minimally costed path every time.

We introduce 4 novelties into this environment. First, the Hill novelty alters the cost of movement. A new symbol representing a hill is placed on hill tiles. Landing on a hill costs the agent one hundred (100) instead of ten (10). The novelty randomly places at most four (4) hills along the shortest path (which is the best path without novelty) without intersecting the longest path. The agent is not given the meaning of the new hill symbol, nor the additional cost function of moving onto a hill.

To solve the maze problem, we developed a pre-novelty model of the domain that exposes the following functions to the planner: (1) can_move: returns whether or not a particular movement is allowed, (2) get_cost: returns the cost of a particular movement, (3) find_next_terrain: returns the terrain type of a particular grid tile. These functions are exposed to Coltrane novelty characterization.

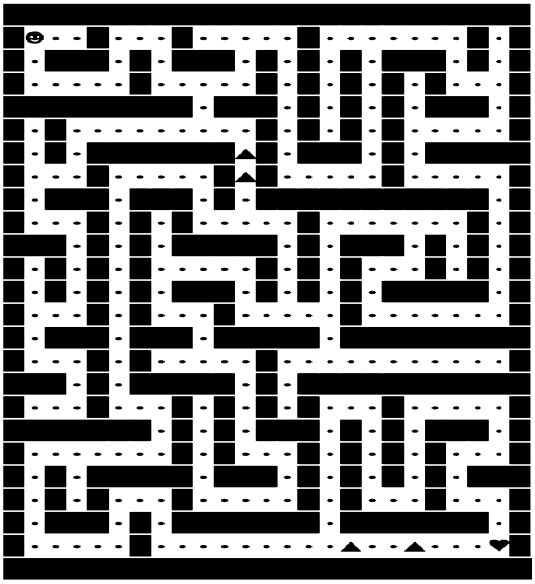


Fig. 7. Maze Pathfinding maze example.

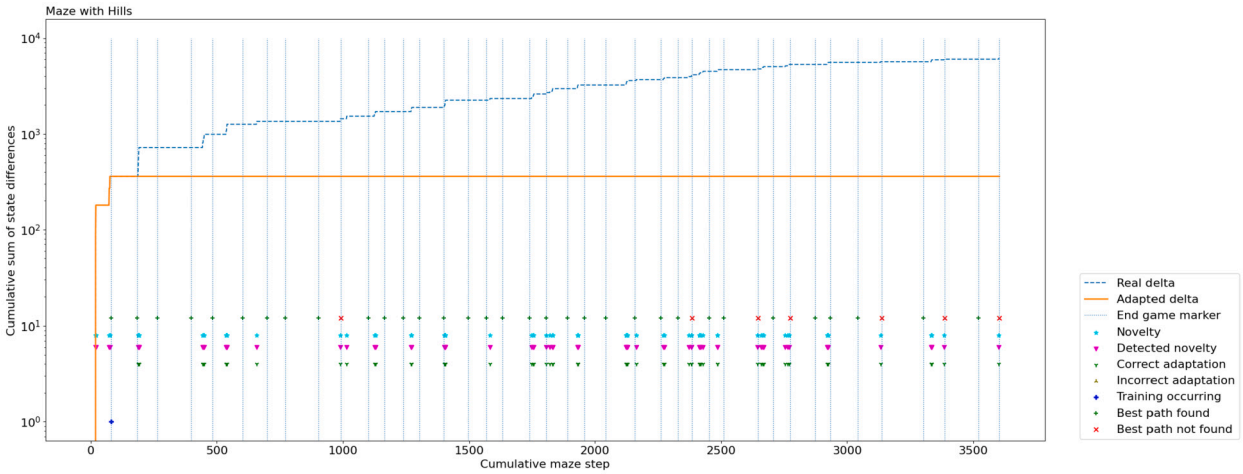


Fig. 8. Coltrane performance on the maze with hills novelty.

Fig. 8 shows the results of Coltrane detection, characterization, and adaptation on the Hill novelty. The dashed blue line shows the cumulative error between the unadapted internal model's prediction of chosen path cost and the actual path cost with the hill tiles. The solid orange line shows the cumulative error of the adapted Coltrane model. The horizontal axis shows each step in the maze, and the vertical blue lines mark the end of each path through the maze. The markers along the bottom show the timeline of novelty introduction (ground truth), Coltrane novelty detection, characterization, adaptation, and success in finding the minimal cost path.

As shown in the hills novelty graph, both the unadapted and Coltrane adapted model accumulate initial error during the first episode. Coltrane detects and reports the novelty, characterizes the novelty, and then uses the characterized model in the second episode. After this first episode, the unadapted model accumulates additional error and the Coltrane adapted model accumulates no additional error, predicting the movement costs perfectly for the rest of the 40 episodes.

Second, the valley novelty alters the cost of movement. A new symbol representing a valley is placed on valley tiles. Landing on a valley costs the agent one (1) instead of ten (10). Fig. 9 shows the Coltrane performance on this novelty. The valley novelty is slower to characterize than the hill novelty. Coltrane's novelty characterization is sensitive to thresholds on the relative costs of actions, and stepping on a valley has a difference in cost of 9 (10 to 1) instead of the difference in stepping on a hill is 99 (100 to 1). In this case, Coltrane accurately characterizes the valley novelty costs by episode seven while the unadapted model error continues to grow with each encounter of the novelty.

Third, the shimmering walls novelty adds a new symbol that also represents a wall, preventing movement. Fig. 10 shows the Coltrane performance on this novelty. In this case, when the agent attempts to move through a shimmering wall, we immediately

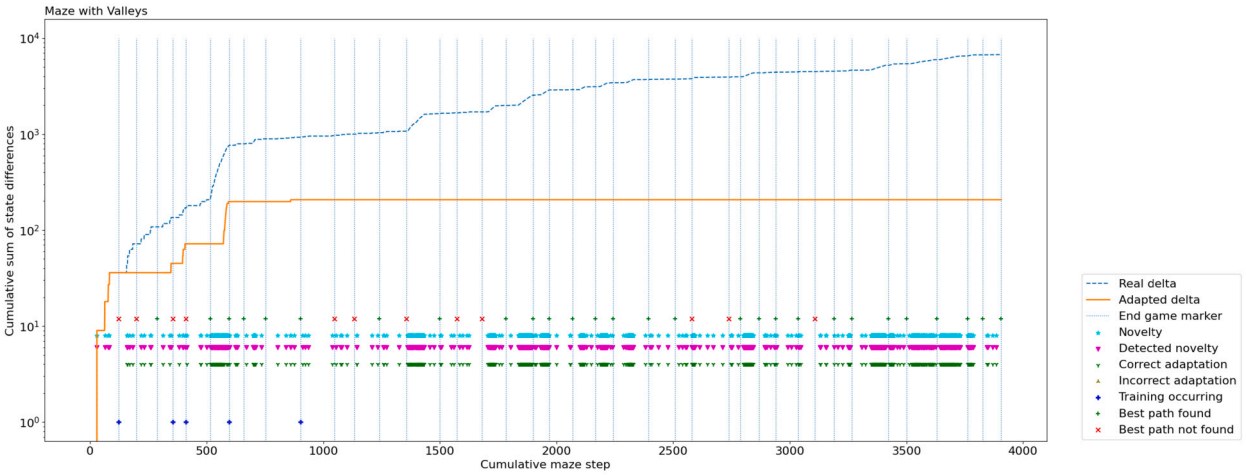


Fig. 9. Coltrane performance on the maze with valleys novelty.

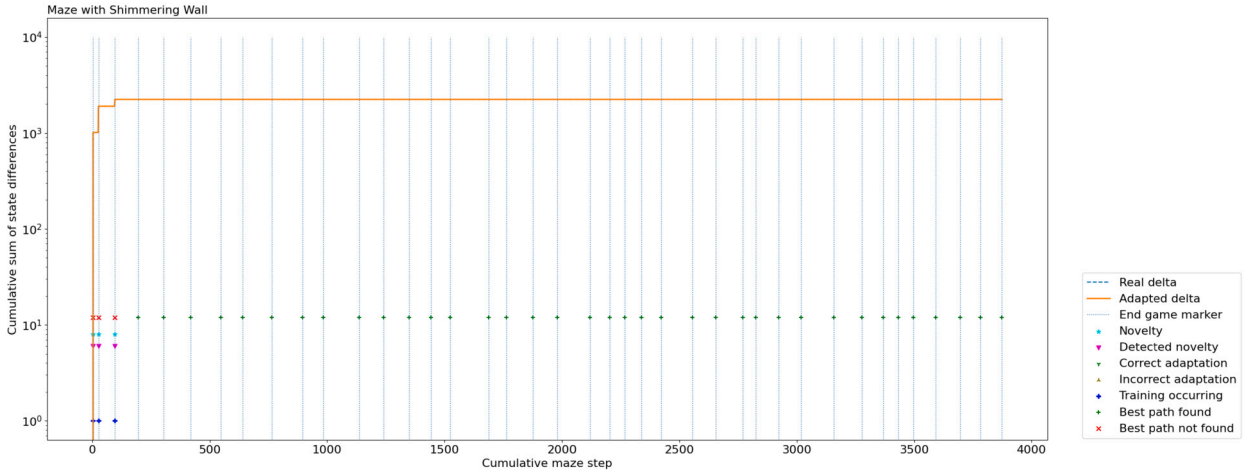


Fig. 10. Coltrane performance on the maze with shimmering walls novelty.

end the episode, since the search will never suggest a different path. In this case, we do not score an unadapted model error and just mark the episode as failure, so there is no dotted blue line in this graph. The shimmering walls are characterized after episode 3, and the minimally costed path is found for the rest of the 40 episodes.

Fourth, the teleporter novelty adds new teleporter entry and teleporter exit symbols. When the agent steps onto the teleporter entry symbol, it is moved instantly with no cost to the teleporter exit symbol. Fig. 11 shows the Coltrane performance on this novelty. Here, the agent must characterize the relationship between the two symbols in the movement function, which takes longer than the single symbol novelty of the shimmering walls. Coltrane characterizes this novelty after episode 8 and uses the teleporters to find the minimally costed path to the goal.

6.1.2. Monopoly with novelty

In Monopoly, there are three ways to acquire a property:

1. By landing on an unowned property and paying the purchase price.
2. By trading with another player.
3. By winning an auction, which happens if the person who landed on the unowned property declined to buy it for the stated price.

The novel rule is that option 1 has a significant tax imposed on it. This novelty has a strong impact on strategy. In the original game, it is usually best to purchase an property for the stated price upon landing on it. Under the novelty, however, this is almost never the right strategy. It is better to wait for an auction or to obtain the property in trade. In our experiments, Coltrane plays against three other agents that are, once the novel rule is introduced, aware of the novelty and handle it appropriately. If Coltrane is unable to characterize the novelty and adapt its strategy, it will lose badly.

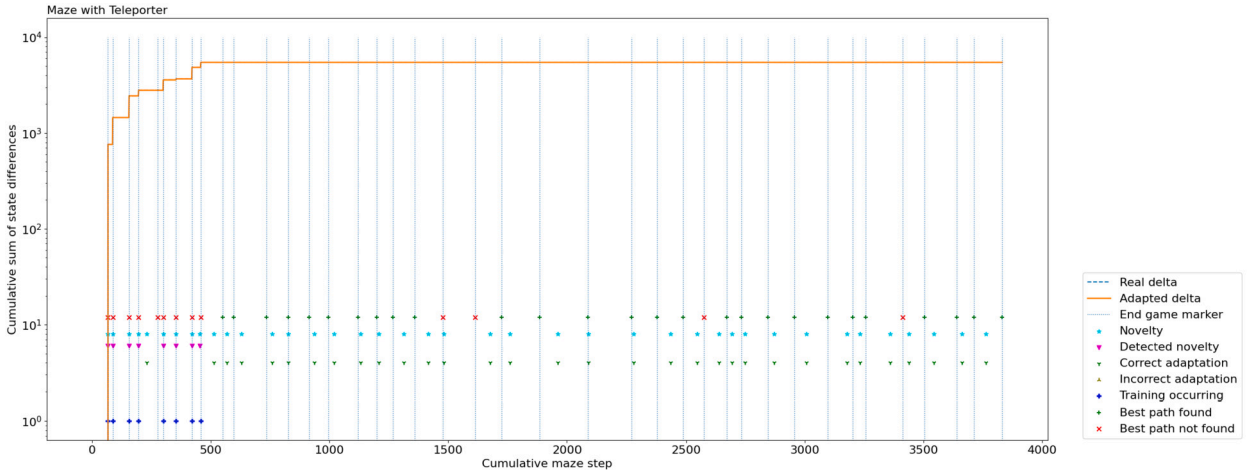


Fig. 11. Coltrane performance on the maze with teleporters novelty.

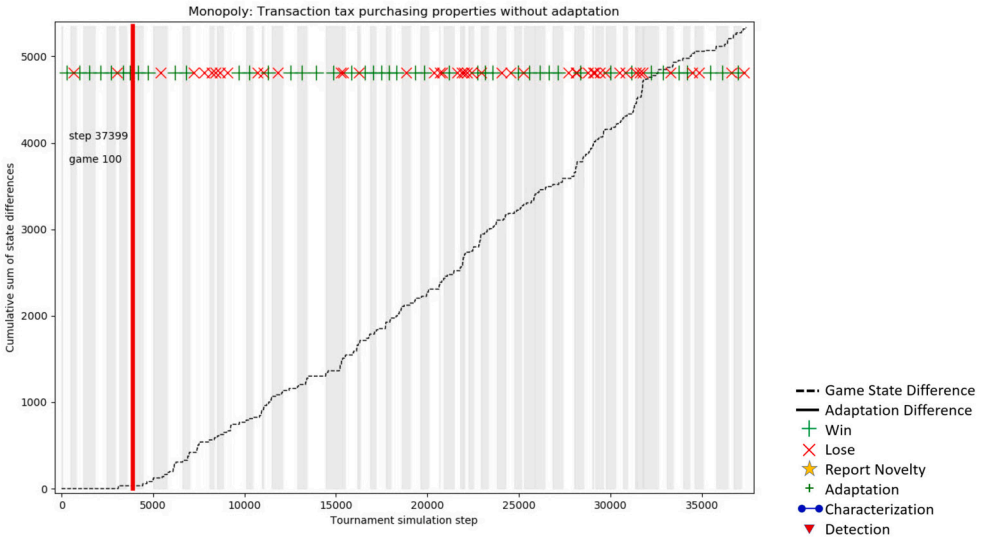


Fig. 12. Tax on purchases novelty, without adaptation.

To demonstrate Coltrane operating under a novel rule, we experiment with an adaptive versus non-adaptive agent playing a sequence that may consist of hundreds of games. At some point in the sequence, the novel rule is introduced into the game model. Soon after that, the agent detects novelty and, in the case of the adaptive Coltrane agent, begins online characterization and adaptation. In the experiment, we track both the ability of the two agents to predict game states, which tests characterization, as well as their win frequency, which tests both characterization and planning.

Fig. 12 shows what happens. There is a lot of information in this figure, so we present it step-by-step. Along the x axis we denote the number of steps taken by the agent, accumulated over multiple games. The y axis shows sum of state differences between the predicted state and the actually observed state, accumulated over time. At each time step, the state difference is computed using the natural diffusion-based distance metric d from Section 4. The vertical red bar shows the time point at which the novel rule is inserted. The jagged line graph shows the cumulative state difference over time for the non-adaptive agent. We can immediately see that until the novelty is inserted, the curve is approximately zero, although there are slight differences due to domain stochasticity. After the novelty is detected, the curve is approximately linear, showing that there is no improvement in characterization over time, as we would expect for a non-adaptive agent.

Fig. 12 also shows the agent performance. The top row shows win (green cross) and loss (red x) events. The pre-novelty win rate of our Monopoly-playing agent is about 75%, measured across many games. As can be seen, after novelty the green crosses become much sparser, with the win rate going down to only about 39%. This demonstrates that this novelty poses a significant hazard, with the non-adaptive agent losing badly against other agents that handle the novelty.

Fig. 13 shows the behavior of the adaptive Coltrane agent under the same purchase tax novelty. The dark black line shows the cumulative sum of state differences over time for the adaptive agent. For reference, the dotted line shows the same quantity for the

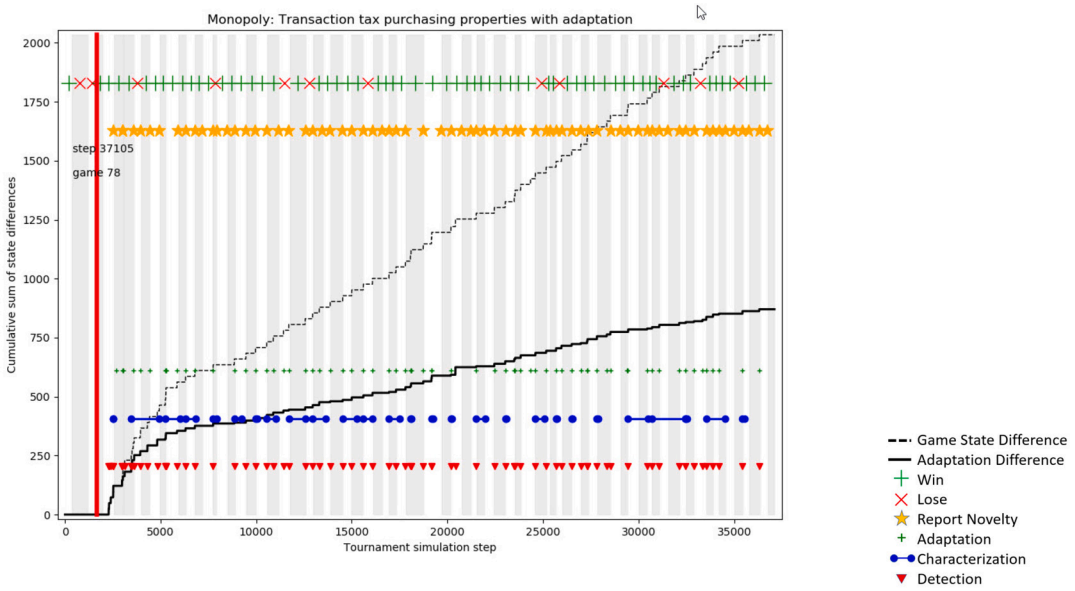


Fig. 13. Tax on purchases novelty, with adaptation.

non-adaptive agent, described earlier. It is immediately clear that the adaptive agent is able to predict future states much better than the non-adaptive agent, albeit not perfectly. In fact, by the end of the tournament, the slope of the solid curve is approximately $1/3$ of that of the dotted curve. This means that new states are about three times as surprising for the non-adaptive agent as for the adaptive agent.

Fig. 13 also shows more information about the novelty detection, characterization, and adaptation of the adaptive Coltrane agent. Red triangles along the bottom of the graph show times at which novelty was detected internally. Gold stars at the top of the graph denote when the agent chose to report novelty. Blue circles along the bottom shows times at which the agent was characterizing novelty, and small green pluses along the bottom denote when the agent used a partially or fully characterized element of novelty in its model during search. An interesting detail is that in some cases, there is a horizontal blue line between two characterization points, indicating that Coltrane is continually improving its characterization during that period.

Focusing on wins and losses, we note that post-novelty, Coltrane wins most of the time. In fact, its win rate has gone up to about 86%, which is higher than the pre-novelty win rate. This is a surprising result, because it demonstrates that Coltrane has adapted automatically to novelty even better than the programmed adaptations of other agents. This is the case even though the characterization is imperfect. The Coltrane heuristics and search prove more adaptive under this novelty than the rule-based agent opponents are able to be with human designers choosing post-novelty strategies.

We have also experimented with comparing adaptive and non-adaptive agents on other novelties. For example, consider a novelty in which the number of houses required to build a hotel is change from four, as in the original game, to three. It turns out that this novelty does not have a significant effect on game strategy, but it is interesting from a characterization point of view because it represents a detail in the precondition of the build hotel action. Fig. 14 compares the cumulative sum of state differences under this novelty between the non-adaptive (dotted line) and adaptive (dark black line) agents. Here, the adaptive agent has far less model error than the non-adaptive agent. Its characterization is mostly accurate almost immediately, as shown by the flatness of the line, once it has seen the other agents build a hotel with only 3 houses.

6.1.3. Doom with novelty

Fig. 15 shows a top down view of the Doom game simulation. We used VizDoom as the underlying simulation engine, but input the game state directly into the Coltrane Doom agent, avoiding the complexity of visual processing. The Coltrane Doom agent controls a player with a gun on a small map that has enemies, health pickups, ammunition pickups, and traps. The player's task is to kill all of the enemies before time runs out by shooting them multiple times while keeping the player's health above 0. The enemies choose an action probabilistically depending on the situation, move left, move right, move forward, move backward, turn left 45 degrees, turn right 45 degrees, and fire. The player's choices of actions are the same. Simulation ticks represent small discrete timesteps. In general, the player must line up and take shots at the enemies, manage health and ammunition, and avoid being in the enemies' line of fire.

The Doom novelty is a constant "wind" that moves the player in one direction every other simulation tick. Without recognizing and adapting to this novelty, the player will be pushed against a wall and unable to target and shoot the enemies, failing by running out of time. Fig. 16 shows the performance of Coltrane recognizing and adapting to this novelty. The novelty affects the game state every other tick, and therefore the cumulative sum of state differences is approximately monotonically increasing. Coltrane is able to recognize this novelty and its play improves once the wind is accounted for. Note that in this graph, Coltrane detects game state

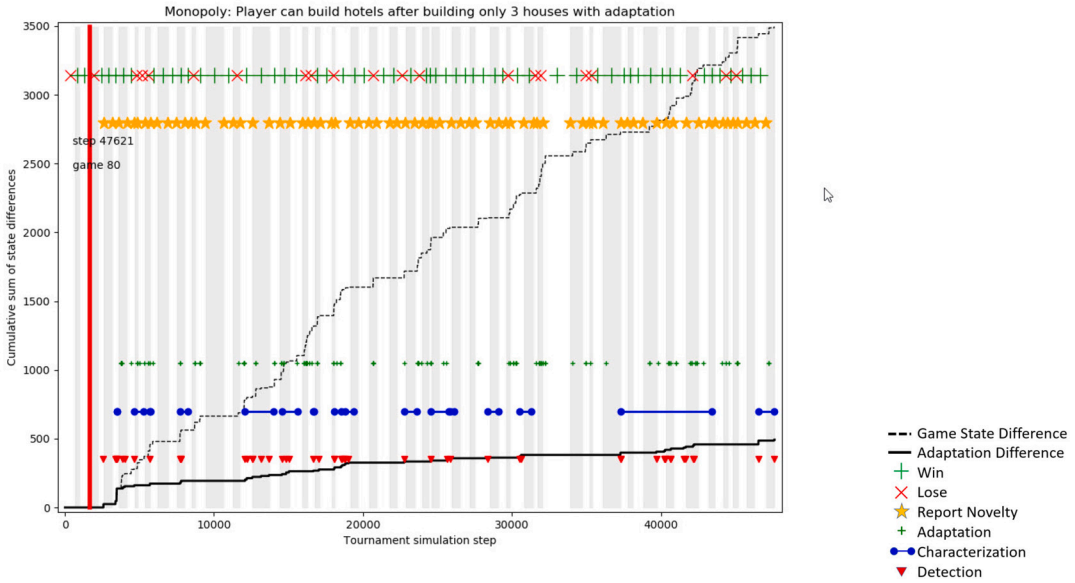


Fig. 14. Number of houses novelty comparison.

Fig. 15. Doom map view.

differences between its internal predications and the states of the simulator prior to the actual novelty being introduced. This indicates small misalignments in Coltrane's internal model and the simulator; we have noticed edge cases for shooting through corners in the simulator, for instance. Even with these misalignments, the game state differences do not reach the threshold for reporting novelty until the novelty is introduced. After this, the expectations are closer to the simulation output, as indicated by the blue line of cumulative game state differences between the simulator and the adapted model being under the red dotted line of the unadapted model.

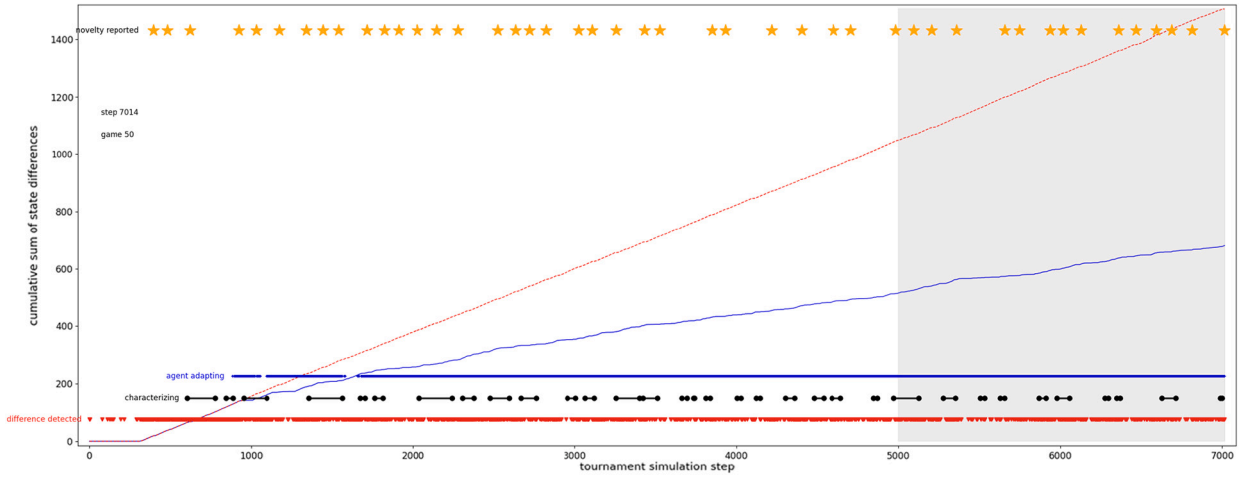


Fig. 16. Doom wind novelty, with adaptation.

Metric	Monopoly	Doom
Correctly detected trials	66.7%	97.0%
False negative instances per correctly detected trial	1.3	13.8
False positives	3.4%	3.0%
Asymptotic task performance improvement	86.6%	73%

Fig. 17. Coltrane performance metrics in DARPA SAIL-ON evaluations, month 36 evaluation of the program. Asymptotic task performance improvement measures the ratio of average Coltrane performance to a baseline agent performance in the last set of trials in each tournament.

6.2. Blind independent evaluation

Coltrane was evaluated against target performance metrics in the DARPA SAIL-ON program by independent evaluators. Evaluation consisted of running Coltrane in simulation domains with injected novelties that were unknown to our team. These domains were Monopoly and Doom, as described above. We developed shared APIs with the evaluation teams that provided a platform to communicate observations and perform actions in each simulated world. Novelties were injected during tournaments of repeated gameplay, and Coltrane's performance was evaluated over the course of these tournaments. False negatives, correctly detected trials, and false positives provide performance on novelty detection. Asymptotic task performance improvement provides information on adaptation performance with measurement of post novelty performance. Because characterization metrics were not recorded in these evaluations, post-novelty performance could be due to a combination of our system's robustness to the novelty without adaptation as compared to the baseline as well as correct characterization and adaptation. These evaluation results demonstrate that Coltrane was able to meet target performance metrics designed by a community of researchers to advance the state of the art on unanticipated novelty detection, characterization, and adaptation.

Novelties that were injected during this tournament were unknown to the Coltrane team prior to evaluation. The evaluators crafted these novelties to conform to a hierarchy of novelty types, including environmental, goal, and event novelties. Sufficient novelties were created to establish the statistical significance of the recorded metrics. Specifics on the set of novelties used in the evaluation are not published here to preserve the ability for those testbeds and novelties to be used in future evaluations of novelty-handling systems.

The evaluation results are shown in Fig. 17. Correctly detected trials, false negative instances per correctly detected trial, and false positives have the standard meanings. Asymptotic task performance improvement measures performance at the end of the trials, and normalizes the score by dividing the agent's performance by the sum of the agent's performance and a baseline agent's performance (e.g. a 50% score would indicate that the agent performed equal to baseline). The evaluation found that this version of Coltrane was able to meet, nearly meet, or exceed metric targets. In Monopoly, 68% of novelties were correctly detected, with false negatives only lasting 1.5 games on average and false positives in only 3.4% of non-novel trials. In Doom, 97% of the trials were correctly detected, false negatives were at a higher 13.8%, and false positives remained at a low 3.0%. Asymptotic task performance improvement in both Monopoly and Doom were significantly above baseline, with 83.6% and 73% respectively.

These results show varying performance in detection, characterization, and adaptation to novelties in different domains. Our pre-novelty models are not exactly the same as the pre-novelty environment and observations can be noisy. For example, a new card may be added to the Monopoly Community Chest card deck, but this card may never be drawn in the game. Or, our pre-novelty model may not track property trades between other players and novelty may alter trading values for some properties. In these cases we include a tolerance for unexpected observations, which enables a tuning parameter for tradeoffs between false positives and false negatives. This parameter and model accuracy is different between these two environments and the resulting metrics reflect these differences. Doom has higher correctly detected trials and also a higher number of false negatives per correctly detected trial.

7. Discussion and analysis

In this paper we have presented a framework for the general problem of enabling agents to detect, characterize, and adapt in novel situations. We have also presented an instantiation of this framework for agents reasoning with transition-function models, and planning-based reasoning. We presented results of the approach's efficacy in a single blind experiment, to demonstrate its ability to adapt to unanticipated novelties that were fully unknown by the researchers, and in detailed demonstrations to illustrate how the mechanisms are able to effectively characterize and adapt to novelties in general.

As described in the introduction, the goal of the framework and instantiation is to provide domain-independent detection, characterization and adaptation to a wide range of novelties, when applied to a pre-novelty reasoning system that was created for a range of tasks without anticipation of the novelties. In particular, the practical target of our research is enabling agents to detect, characterize, and plan in novel situations, when (a) the agent must respond quickly, with few observations, (b) the range of possible novelties are realistic and plausible novelties of the sort that might arise in the real world, (c) the approach to novelty handling cannot anticipate the novelties in a domain-specific way, and (d) the task-oriented pre-novelty system is well designed in the ways typical by software engineers today, e.g. with relevant task-level abstractions present in the code as they are useful to encoding the domain.

While this definition may seem imprecise in (b) and (d), we believe it is an important and meaningful target in practice. In particular, it is meaningful and important because, if one were to accomplish them, they would be useful for deployed systems in the real world, where their meaning is well grounded, and where this ability to respond to changes in the world would have practical benefit in a wide range of applications, including those described in the introduction.

For example it is important for self-driving vehicles to be able to function in novel situations for safety. Consider the well-known example of the Tesla crash in China, where the autonomous vehicle, driving in the left lane, failed to notice or interpret the fact that other vehicles were leaving its lane and merging into the lane to the right. Unfortunately, the vehicle persisted in its lane and eventually crashed into a road sweeper that was going much more slowly than it expected in the left lane. In this example, the novelties consisted of an unusual type of vehicle (the road sweeper) and two unusual behaviors (vehicles leaving the lane for no apparent reason, and a vehicle traveling slower than expected in the left lane). These are exactly the sort of realistic and plausible novelties that might arise in a driving situation, and it is these sorts of novelties that Coltrane is intended to address.

To measure progress against this goal, we presented three complementary types of information to inform different elements of the goal: (1) performance in a single blind experiment, in which the injected novelties were unknown to the researchers, help inform both the ability to perform the goal, and specifically the aim that the system not explicitly or implicitly anticipate the novelties; (2) detailed demonstration in novelties that are fully described, to enable insight into the complexity of novelties that can be characterized and adapted to; and (3) a detailed description of the mechanisms employed so that the range and generality can be analyzed by researchers looking to understand in detail its weaknesses, strengths, and opportunities for improvement.

The goal is that this framework and instantiation be a step toward deployed systems that can robustly adapt to novelties that are inevitable in the real world, and that future research can build upon to accomplish this goal. In the rest of this analysis, we discuss the framework's current strengths, limitations, and opportunities for future research.

7.1. Reuse of pre-novelty domain abstractions for sample efficient, generalized characterization, automatic handling of nuisance novelty, and limitations for novelties outside the natural task abstractions

One of the key advances Coltrane attempts to make is leveraging the abstractions present in the pre-novelty model to enable rapid convergence to generalized characterizations of task-relevant changes in the world. In practice, this yields sample-efficient inferences as described in Section 4. It also has the additional benefit of giving a grounded approach to reasoning about nuisance novelty (modifications to the way the world works that are irrelevant to the task being accomplished). Using a prior that is defined by the pre-novelty domain model used for task-oriented reasoning, abstractions relevant to the task are typically represented in the prior, while those that are superfluous are ignored. When permuting this model based on divergent differences in the world, those that are outside these task-relevant abstractions are typically ignored. In the typical case this enables nuisance novelties to be efficiently and efficaciously ignored.

These advantages come with consequent limitations that are important to be aware of as research and application of these techniques are advanced. In particular, if a task-relevant novelty is not well captured by perturbations to the abstractions that are naturally present in the pre-novelty model, those novelties would take more time to characterize (as larger numbers of primitive types have to be permuted to create characterizations), or, in the extreme, might be missed entirely (if there is not representational expressiveness sufficient to capture the novelty, or if the difference is not recognized because it is outside what is observed by the system).[5] For example, in Monopoly, the colors of properties are naturally important to determining their prices and membership in a monopoly, so color is a relevant abstraction in the pre-novelty model that is among those focused on, as in our example of rents changing for red properties. However, when driving a vehicle in a racing simulation, the colors of the other vehicles are typically irrelevant, so Coltrane would have a harder time adapting to a novelty in which color suddenly became relevant, (and in fact color would likely not even be in the set of features observed by the pre-novelty system).

[5] Note that representational expressiveness can be easily addressed by providing sufficient primitives. Lack of perceiving that there is a difference, and speed of characterization are larger practical issues.

7.2. Dependence on correctness of pre-novelty domain model

One of the surprising insights we discovered when doing this research was the discovery by our automatic characterization methods of errors in the pre-novelty domain model. Model errors themselves are not unusual, and are a normal part of the debugging process for a pre-novelty system. The surprising result is the potential presence of inaccuracies in the model that the planning system is robust to, and so may not be corrected as part of normal debugging of a pre-novelty system.

With novelty detection and characterization, however, these model errors cause novelty detection and characterization, and can be a distraction from other novelties, and removing them requires additional work beyond what might be necessary for the pre-novelty system.

There are straightforward mitigations to address this. The easiest include (1) correcting the model errors, or (2) marking these aspects of the model so that they are excluded by the detection and characterization mechanisms. (3) Running the system without any novelties and having it automatically characterize the incorrectly modeled elements prior to deployment is a more ambitious research direction with interesting potential benefits, in addition to reducing manual effort that is needed in the first two options. One down side of this approach is it may mask genuine model errors, that would be beneficial to correct manually. (4) A hybrid approach where automatic detection is performed, with human-on-the-loop choices of whether to automatically characterize and extend the model to correct the model inaccuracies, automatically exclude that aspect of the model, or enable hand-correction of the model (providing benefit to the correctness of the pre-novelty system), we believe is one of the more interesting directions for future work because of its multiple benefits.

7.3. Formal guarantees

Our approach is deliberately very general. In principle, it can work with any novelty that can be expressed in a transition model in code. Given the breadth of our approach, we cannot possibly claim that it is a sound and complete method that will work in all circumstances, which we believe to be impossible.

An alternative, and rich area of future work, is a narrower approach of finding islands of representations and algorithms that are provably correct, but limited in their applicability. Our approach is compatible with this alternative and indeed provides a framework in which these islands can be used within a larger real-world system. A best-of-both-worlds approach would be ideal for real applications; to use specialized, correct methods where available and to fall back on general heuristic methods with no guarantees where necessary. We have started to explore this combined approach in our own work. In an earlier version of the system, we integrated a method to discover the structure of time series (e.g., inflationary trends in Monopoly) using Gaussian process kernels [38]. Future research should proceed along three fronts: (1) to increase the range of these specialized representations and algorithms to cover more real-world situations; (2) to improve a system's ability to determine when and how these specialized methods apply; and (3) building on the ideas in this paper, to improve the effectiveness of the general-purpose methods that must be used when no specialized methods are applicable.

7.4. Additional type-driven perturbation knowledge

In addition to these fully domain-independent mechanisms for efficient characterization and adaptation that leverage domain knowledge in the pre-novelty system, it is also interesting to consider possible relaxations of the fully domain-independent approach that could further increase sample efficiency, and rapid convergence to general characterizations of novelties.

We have worked with adding type annotations to domain models that describe the semantic types of objects, actions, and events in the domain, alongside a general theory of how concepts of these types function. This general theory is used to help novelty handling in all domains in which objects of these types participate.

In this approach, semantic types are hierarchical. For example, a common type of action in a game is selection from a set according to a probability distribution. A subtype might involve a uniformly random selection, and a further subtype might be rolling a die. The approach stipulates that die roll novelty is handled the same way in all games with dice, and similarly for the levels above. According to this approach, characterization of novelty considers the novelty hypotheses associated with both the specific concept and the concepts it inherits from. For example, with random selection, adding or subtracting choices or changing the probability distribution might be novelties considered.

We believe this type-driven novelty is closer to the way people handle novelty than the more abstract program perturbation approach we presented in this paper. People might think about concepts and all their possible variations hierarchically based on prior experience with those concepts. For example, when encountered with a novel vehicle that might be a car, but hops in an unusual way, a person might consider their range of experience with cars, then ground vehicles in general, then all moving vehicles they have encountered. This might lead them to characterize the object as a kind of helicopter/car hybrid. Associating concepts with their semantic types is what enables this past experience to be brought to bear on the present domain. A limitation of the type-driven novelty approach for building agents, however, is that it relies on significant knowledge engineering to obtain the knowledge about objects that humans gather through experience.

In future work, we plan to explore this type-driven approach more deeply, both individually and in conjunction with our program perturbation approach. A relatively simple integration of the approaches is to make the agent initially attempt to characterize the novelty using types, and, if it fails, to fall back on the perturbation approach as a recourse to first principles. A deeper integration would include the expression of types and their behaviors within the program itself, perhaps using the concepts as library functions, so they can participate in the search for alternative programs to characterize the novelty.

CRediT authorship contribution statement

Bryan Loyall: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Avi Pfeffer:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **James Niehaus:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Michael Harradon:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Paola Rizzo:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Conceptualization. **Alex Gee:** Software. **Joe Campolongo:** Software. **Tyler Mayer:** Conceptualization. **John Steigerwald:** Software.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Charles River Analytics reports financial support was provided by Defense Advanced Research Projects Agency. Bryan Loyall serving as an editor for this special issue. Dr. Loyall has recused himself and not been involved in any evaluation or discussion of this article. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) for this research. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contracts No. HR001120C0042 and 140D0422C0045.

The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Distribution Statement “A” (Approved for Public Release, Distribution Unlimited).

Data availability

No data was used for the research described in the article.

References

[1] A.P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z.D. Guo, C. Blundell, Agent57: outperforming the Atari human benchmark, in: International Conference on Machine Learning, 2020.
[2] T. Boult, P. Grabowicz, D. Prijatelj, R. Stern, L. Holder, J. Alspector, S. Walter, Towards a unifying framework for formal theories of novelty, Proc. AAAI Conf. Artif. Intell. (2021).
[3] R.I. Brafman, D. Tolpin, O. Wertheim, Probabilistic programs as an action description language, Proc. AAAI Conf. Artif. Intell. (2023) 15351–15358.
[4] M. Brenner, B. Nebel, Continual planning and acting in dynamic multiagent environments, in: Proceedings of the 2006 International Symposium on Practical Cognitive Agents and Robots, 2006.
[5] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A survey of Monte Carlo tree search methods, IEEE Trans. Comput. Intell. AI Games 4 (2012) 1–43.
[6] Z. Chen, B. Liu, Lifelong Machine Learning, Morgan & Claypool Publishers, 2018.
[7] Defense Advanced Research Projects Agency, Science of artificial intelligence and learning for open-world novelty (SAIL-ON), https://sam.gov/opp/88fdca99de93ddbb74cd8fb51916ceaa/view, 2019. (Accessed 16 February 2025).
[8] T.G. Dietterich, Steps toward robust artificial intelligence, AI Mag. 38 (2017) 3–24.
[9] K. Doctor, C. Task, E. Kildebeck, M. Kejriwal, L. Holder, R. Leong, Toward defining a domain complexity measure across domains, arXiv preprint arXiv:2303.04141, 2023.
[10] K. Ellis, C. Wong, M. Nye, M. Sablé-Meyer, L. Cary, L. Morales, L. Hewitt, A. Solar-Lezama, J.B. Tenenbaum, Dreamcoder: growing generalizable, interpretable knowledge with wake–sleep Bayesian program learning, Philos. Trans. R. Soc. A 381 (2020).
[11] M. Faizan, V. Sarathy, G. Tatiya, S. Goel, S. Gyawali, M.G. Castro, J. Sinapov, M. Scheutz, A novelty-centric agent architecture for changing worlds, in: 20th International Conference on Autonomous Agents and Multiagent Systems AAMAS, Virtual Event, United Kingdom, May 3–7, 2021, ACM, 2021, pp. 925–933.
[12] M. Fox, A. Gerevini, D. Long, I. Serina, et al., Plan stability: replanning versus plan repair, in: ICAPS, 2006, pp. 212–221.
[13] S. Gelly, D. Silver, Monte-Carlo tree search and rapid action value estimation in computer go, Artif. Intell. 175 (2011) 1856–1875.
[14] K. Grace, M.L. Maher, Surprise-triggered reformulation of design goals, Proc. AAAI Conf. Artif. Intell. (2016).
[15] J. Hoffmann, R.I. Brafman, Conformant planning via heuristic forward search: a new approach, Artif. Intell. 170 (2006) 507–541.
[16] X. Huang, J. Weng, Novelty and Reinforcement Learning in the Value System of Developmental Robots, 2002.
[17] M. Klenk, W. Piotrowski, R. Stern, S. Mohan, J. de Kleer, Model-based novelty adaptation for open-world AI, in: International Workshop on Principles of Diagnosis (DX), 2020.
[18] G. Kuhlmann, P. Stone, Automatic heuristic construction in a complete general game player, in: Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence, 2006, pp. 1457–1562.
[19] N. Kushmerick, S. Hanks, D.S. Weld, An algorithm for probabilistic planning, Artif. Intell. 76 (1995) 239–286.
[20] B.M. Lake, R. Salakhutdinov, J.B. Tenenbaum, Human-level concept learning through probabilistic program induction, Science 350 (2015) 1332–1338.
[21] P. Langley, Open-world learning for radically autonomous agents, in: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI Press, New York, NY, 2020.
[22] M. Markou, S. Singh, Novelty detection: a review—part 1: statistical approaches, Signal Process. 83 (2003) 2481–2497.

[23] M. Molineaux, D. Aha, Learning unknown event models, Proc. AAAI Conf. Artif. Intell. 28 (1) (2014).
[24] M. Molineaux, D. Dannenhauer, An environment transformation-based framework for comparison of open-world learning agents, in: Designing Artificial Intelligence for Open Worlds, AAAI 2022, Spring Symposium, 2022.
[25] M. Molineaux, D. Dannenhauer, E. Kildebeck, A framework for characterizing novel environment transformations in general environments, arXiv:2305.04315, 2023.
[26] F. Muhammad, V. Sarathy, G. Tatiya, S. Goel, S. Gyawali, M. Guaman, Scheutz M. others, A novelty-centric agent architecture for changing worlds, in: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, 2021, pp. 925–933.
[27] D.J. Musliner, M.J. Pelican, M. McLure, S. Johnston, R.G. Freedman, C. Knutson, Openmind: planning and adapting in domains with novelty, in: Proceedings of the Ninth Annual Conference on Advances in Cognitive Systems, 2021.
[28] A.V. Nori, S. Ozair, S.K. Rajamani, D. Vijaykeerthy, Efficient synthesis of probabilistic programs, ACM SIGPLAN Not. 50 (2015) 208–217.
[29] S. Ontanón, K. Mishra, N. Sugandh, A. Ram, On-line case-based planning, Comput. Intell. 26 (2010) 84–119.
[30] G.I. Parisi, R. Kemker, J.L. Part, C. Kanan, S. Wermter, Continual lifelong learning with neural networks: a review, Neural Netw. 113 (2019) 54–71.
[31] J. Parmar, S. Chouhan, V. Raychoudhury, S. Rathore, Open-world machine learning: applications, challenges, and opportunities, ACM Comput. Surv. 55 (2023) 1–37.
[32] X. Peng, J.C. Balloch, M.O. Riedl, Detecting and adapting to novelty in games, in: AAAI Workshop on Reinforcement Learning in Games, 2021.
[33] M.A. Pimentel, D.A. Clifton, L. Clifton, L. Tarassenko, A review of novelty detection, Signal Process. 99 (2014) 215–249.
[34] N. Ranasinghe, W.M. Shen, Surprise-based learning for developmental robotics, in: 2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS), IEEE, 2008, pp. 65–70.
[35] S. Reed, K. Zolna, E. Parisotto, S.G. Colmenarejo, A. Novikov, G. Barth-Maron, et al., A generalist agent, arXiv:2205.06175, 2022.
[36] R. Reiter, On closed world data bases, in: Readings in Artificial Intelligence, Morgan Kaufmann, 1981, pp. 119–140.
[37] J. Rulison, Inductive Program Synthesis in BLOG, Technical Report UCB/EECS-2018-107, University of California, Berkeley, 2018.
[38] F.A. Saad, M.F. Cusumano-Towner, U. Schaechtle, M.C. Rinard, V.K. Mansinghka, Bayesian synthesis of probabilistic programs for automatic data modeling, Proc. ACM Princ. Prog. Lang. 3 (2019) 1–32.
[39] V. Sarathy, D. Kasenberg, S. Goel, J. Sinapov, M. Scheutz, Spotter: extending symbolic planning operators through targeted reinforcement learning, in: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, 2021, pp. 1118–1126.
[40] E. Scala, P. Torasso, Proactive and Reactive Reconfiguration for the Robust Execution of Multi Modality Plans, 2014.
[41] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (2016) 484–489.
[42] T. Thai, M. Shen, N. Varshney, S. Gopalakrishnan, U. Soni, C. Baral, Sinapov J. others, An architecture for novelty handling in a multi-agent stochastic environment: case study in open-world monopoly, in: Designing Artificial Intelligence for Open Worlds: Papers from the 2022 Spring Symposium, Virtual, AAAI Press, 2022.
[43] O. Vinyals, I. Babuschkin, W.M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D.H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al., Grandmaster level in StarCraft II using multi-agent reinforcement learning, Nature 575 (2019) 350–354.
[44] Y. Wang, Q. Yao, J.T. Kwok, L.M. Ni, Generalizing from a few examples: a survey on few-shot learning, ACM Comput. Surv. 53 (2020) 1–34.
[45] Y. Wang, S. Zou, Online robust reinforcement learning with model uncertainty, Adv. Neural Inf. Process. Syst. 34 (2021) 7193–7206.
[46] G.A. Wiggins, A preliminary framework for description, analysis and comparison of creative systems, Knowl.-Based Syst. 19 (2006) 449–458.
[47] G.N. Yannakakis, A. Liapis, Searching for surprise, in: Proceedings of the Seventh International Conference on Computational Creativity, 2016.
[48] S.W. Yoon, A. Fern, R. Givan, Ff-replan: a baseline for probabilistic planning, Int. Conf. Autom. Plan. 7 (2007) 352–359.