Explainable AI for System Failures: Generating Explanations that Improve Human Assistance in Fault Recovery

Devleena Das, Siddhartha Banerjee, Sonia Chernova
Georgia Institute of Technology, Atlanta, Georgia
Institute for Robotics and Intelligent Machines
{ddas41,siddhartha.banerjee, chernova}@gatech.edu

Abstract

With the growing capabilities of intelligent systems, the integration of artificial intelligence (AI) and robots in everyday life is increasing. However, when interacting in such complex human environments, the failure of intelligent systems, such as robots, can be inevitable, requiring recovery assistance from users. In this work, we develop automated, *natural language* explanations for failures encountered during an AI agents' plan execution. These explanations are developed with a focus of helping *non-expert* users understand different point of failures to better provide recovery assistance. Specifically, we introduce a context-based information type for explanations that can both help non-expert users understand the underlying cause of a system failure, and select proper failure recoveries. Additionally, we extend an existing sequence-to-sequence methodology to automatically generate our context-based explanations. By doing so, we are able develop a model that can generalize context-based explanations over both different failure types and failure scenarios.

Introduction

In homes, hospitals, and manufacturing plants, robots are increasingly being tested for deployment alongside non-roboticists to perform goal-directed tasks, such as folding laundry (Yang et al. 2016), delivering laboratory specimens (Bloss 2011; Hu et al. 2011), and moving inventory goods (Hägele et al. 2016; Lawton 2016). When interacting in such complex human environments, robot failures are inevitable and assistance in failure recovery can be necessary (Bauer, Wollherr, and Buss 2008). An example of such a scenario is that of a consumer interacting with technology in their own home, such as determining why a robot tasked with retrieving a beverage is stopped in the middle of the kitchen, or a scenario where a production line worker wonders why a robot who was picking up boxes from a conveyor belt moments ago, suddenly stopped. Prior work in the Explainable Planning (XAIP) community has explored closely related problems, such as establishing methods for explaining an agent's chosen plan for a particular task, and explaining unsolvable plans to end-users (Chakraborti, Sreedharan, and Kambhampati 2020). However, providing justifications for points of failures that occur *during* an agent's plan execution has not yet been studied.

In this work, we aim to expand upon the existing set of explanations available in the XAIP community. We propose an additional type of explanation called *error explanations*, in the context of sequential-decision making and planning. These error explanations focus on explaining failures that may occur while executing a chosen plan. We seek to develop automated, *natural language* error explanations that can explain encountered failures in a manner that is understandable by non-expert users. The goal of these explanations is to not only help non-expert users understand the system's point of failure, but also help them determine an appropriate solution required to resume normal operation of a task. Specifically, our core research questions are:

- What type of information constitutes a meaningful explanation of an agent's failure that can aid in a *non-expert's* ability to understand the cause of a failure, and provide accurate fault recovery assistance?
- How can we develop a model that can automatically generate natural language explanations so that these explanations can be generalized across varying failure scenarios and failure types?

Through these fundamental questions, we i) introduce a context-based information type that explanations should include to effectively help users understand the fault diagnoses and in turn provide accurate recovery assistance, and ii) adapt an existing sequence-to-sequence methodology from (Ehsan et al. 2019) to generate automated explanations that can generalize over varying failure types and scenarios.

We validate our approach through a user study, comparing two different types of explanations, action-based and context-based, applied to a pick-and-place robot manipulation task. Through this user study, we measure non-experts' accuracy in understanding the provided fault diagnoses and accuracy in identifying correct recovery solutions. We also measure users' self-reported confidence and difficulty scores for each decision. We observe that context-based explanations significantly improves users' recovery selection over both the baseline and action-based explanations. Additionally, self-reported ratings show that the presence of any ex-

planations allows for higher perceived confidence and lower difficulty scores than having no explanations. Furthermore, the confusion matrix of our automated explanation generating model shows that our model can generalize over different failure scenarios with a 89.7% overall accuracy.

Related Works

In prior work, the XAI community has primarily focused on developing interpretability methodologies for expert users familiar with the domain of AI or ML (Adadi and Berrada 2018; Ribeiro, Singh, and Guestrin 2016). Many of these approaches have focused on model-agnostic implementations, designed to increase understanding of deep learning (DL) outputs for classification-based tasks by leveraging inherently interpetable models, such as, decision trees (Zhang et al. 2019), or visual attributes, such as, heatmaps (Selvaraju et al. 2017). While these approaches are applied to more complex models, the complexity of such classification tasks do not include the complexity of sequential decision-making, long-term interactions, or changing environments (Chakraborti, Sreedharan, and Kambhampati 2020).

Current work in XAIP aim to address the need for interpretable explanations for complex planning problems which expand beyond single-classification tasks. In a recent survey paper, (Chakraborti, Sreedharan, and Kambhampati 2020) highlight some of the key components of plan explanations studied by the community: contrastive question-answering, explaining unsolvable plans, and achieving explicable justifications for a chosen plan. In the realm of answering contrastive questions, (Krarup et al. 2019) describe a framework to transfer domain-independent user questions into constraints that can be added to a planning model, while (Hoffmann and Magazzeni 2019) describe how to utilize common properties within a set of correct plans as an explanation for unmet properties in incorrect plans. In order to explain unsolvable plans, (Sreedharan et al. 2019) abstract the unsolvable plan into a simpler example through which explanations can be formulated. Additionally, (Zhang et al. 2017) describe the need for explanations to be "explicable" by end-users. The authors of this work implement explicability by using conditional random fields (CRFs) to model humans' labelling schemes to agent plans, and use such model to develop explicable explanations for new plans. Additionally, to minimize the constraints on an agent's plan, (Chakraborti et al. 2019; 2017) describes a particular process of achieving explicability, known as model reconciliation. The authors produce explanations by considering the difference between an agent's and end user's mental model. In all these cases, a chosen plan, or lack thereof, is explained. In our work, instead of explaining a particular plan, we aim to explain possible faults within a plan that consequently halt its execution.

Outside the scope of XAIP and in the context of reinforcement learning systems, (Ehsan et al. 2018; 2019) also describe the need for humanly understandable explanations. The authors coined the usage of *rationales* as a way of generating explanations in language that is understandable by everyday people. They developed an automated rationale generating system, studied within the context of the game Frogger, that can translate game state representations into humanly understandable explanations. However these explanations are generated within the domain of discrete-action state space and not continuous-action state space which are commonly found in sequential decision-making, planning problems.

Furthermore, within the realm of fault recovery in robotics, (Knepper et al. 2015) studies how robots can utilize natural language to generate assistance requests during a point of error. Their natural language framework is trained to generate assistance requests with accurate multi-object disambiguation ('table leg under the table' vs. 'table leg near the table') in efforts shorten idle times during assembly. Instead of focusing on object disambiguation or asking for a specific assistance, we utilize natural language to generate explanations that can explain a robot's failure in a manner that allow non-expert users to deduce a plausible recovery assistance.

Problem Definition

Building on the definition presented by (Chakraborti, Sreedharan, and Kambhampati 2020), we define a planning problem Π in terms of a transition function $\delta_\Pi : A \times S \rightarrow S \times \mathbb{R}$, where A is the set of actions available to the agent, S is the set of states it can be in, and the real number denotes the cost of making the transition. A planning algorithm $\mathbb{A}$ solves Π subject to a desired property τ to produce a plan or policy π, i.e. $\mathbb{A} : \Pi \times \tau \mapsto \pi$. Here, τ may represent different properties such as soundness, optimality, etc. The solution to this problem is defined as a *plan* $\pi = \langle a_1, a_2, ..., a_n \rangle, a_i \in A$, which transforms the current state $I \in S$ of the agent to its goal $G \in S$, i.e. $\delta_\Pi(\pi, I) = \langle G, \Sigma_{a_i \in \pi} c_i \rangle$. The second term in the output denotes the plan cost $c(\pi)$.

In this context, we argue that there are (at least) two categories of explanations that are useful to a user. The first was included in the survey by (Chakraborti, Sreedharan, and Kambhampati 2020), and the second we introduce here:

- $\mathcal{E}_\pi$**:** This explanation serves to justify to a human user that solution π satisfies property τ for a given planning problem Π. For example, the user may ask "Why π and not π'?". In response to this question, $\mathcal{E}_\pi$ must enable the user to compute $\mathbb{A} : \Pi \times \tau \mapsto \pi$ and verify that either $\mathbb{A} : \Pi \times \tau \not\mapsto \pi'$, or that $\mathbb{A} : \Pi \times \tau \mapsto \pi'$ but $\pi \equiv \pi'$ or $\pi > \pi'$ with respect to some criteria. $\mathcal{E}_\pi$ applies to the plan solution as a whole and can be elicited at any time. Approaches that address $\mathcal{E}_\pi$ are listed in the Related Works section.
- $\mathcal{E}_{err}$**:** This explanation applies in the event that an unexpected failure state $f \in \mathcal{F}$, triggered by a failed action in $\langle a_1, a_2, ..., a_n \rangle$, halts the execution of π. For example, the user may ask "The robot is at the table, but why did it not pick up my beverage?" In response to this question, $\mathcal{E}_{err}$ must allow the user to understand the cause of error in order to help the system recover.

In this work, we address the second variant of explanations, $\mathcal{E}_{err}$. We assume that both the algorithm $\mathbb{A}$ and the

Failure Type	Scenario	Action-Based	Context-Based
motion-planning	obj. too far away	Could not move its arm to the desired object	Could not move its arm to the desired object because the desired object is too far away
motion-planning	obj. close to other objs.	Could not move its arm to the desired object	Could not move its arm to the desired object because the desired object is too close to other objects
detection	obj. not present	Could not detect the desired object	Could not detect the desired object because the desired object is not present where the robot is looking
detection	obj. occluded	Could not detect the object	Could not detect the desired object because the desired object is occluded
navigation	mis-localization	Could not navigate to the desired object	Could not navigate to the desired object because the robot is lost
navigation	controller	Could not navigate to the desired object	Could not navigate to the desired object because the robot's motors are malfunctioning

Table 1: Example explanations for each failure type and failure scenario that are provided to the AB and CB study conditions.

policy π are sound, and that the cause of error is triggered by a failure state $f \in \mathcal{F}$ from which it cannot recover without user assistance. Our objective is to find $\mathcal{E}_{err}$ such that the user correctly understands the cause failure, and can help the agent recover from an error. We introduce a set of *information types* Λ that evaluate varying characteristics of an explanation $\mathcal{E}_{err}$ in order to find a meaningful $\lambda^{'} \in \Lambda$ for non-expert users. To generalize and automate an explanation $\mathcal{E}_{err}$ for different failure scenarios, we take inspiration from (Ehsan et al. 2019)'s work to translate the state of the agent, S, into natural language explanations that fit $\lambda^{'}$.

Information Types for $\mathcal{E}_{err}$

The first question we have to answer is: given an error while executing π, what format should explanation $\mathcal{E}_{err}$ take?

(Ehsan et al. 2019)'s work establishes that explanations for everyday users should take the form of rationales that justify a reasoning in layperson's terms while being representative of a particular scenario, as opposed to revealing the true decision making process of an agent. Thus, to provide an effective and meaningful $\mathcal{E}_{err}$ to non-experts, we first evaluate a set of information types Λ to find the best information type $\lambda^{'}$ that $\mathcal{E}_{err}$ should encompass. For this, we conducted a three-way between-subjects user study where participants were asked to identify and suggest fixes to a set of failure states $\mathcal{F}$ that a robot encounters while performing π. In this study design, Λ consists of the following three study conditions that differ the information type of $\mathcal{E}_{err}$:

- **None (Baseline)**: Participants receives no explanations on the cause of error.
- **Action-Based (AB)**: Participants receive $\mathcal{E}_{err}$ that use the failed action as the cause of error, seen in Table 1.
- **Context-Based (CB)**: Participants receive $\mathcal{E}_{err}$ that use the failed action as well as a contextualized reasoning deduced from the environment as the cause of error, seen in Table 1.

To validate which type of $\mathcal{E}_{err}$ is most meaningful, we conducted an experiment using simulated robot errors and scripted explanations. In the subsections below, we present our experimental framework, the study design, and the results. This evaluation serves to inform the $\lambda^{'}$ that will of focus when developing an automated generation of $\mathcal{E}_{err}$.

Metrics

We use the following metrics to evaluate the effectiveness of a particular $\mathcal{E}_{err}$:

- **Solution Percentage (Sol%)**: measures how accurately participants select solutions to recover the encountered failure. The average solution percentage is calculated as:

$$Sol\% = \frac{correctSolution}{correctSolution + incorrectSolution} \quad (1)$$

- **Action Identification Percentage (*AId %)***: measures how accurately participants identify the action on which plan π fails. The average action identification percentage is calculated as:

$$AId\% = \frac{correctAction}{correctAction + incorrectAction} \quad (2)$$

- **Action Confidence (*ActConf*)**: measures self-reported confidence in determining a failed action in π. Action confidence is measured using a 5-Point Likert Scale rating based on the question "How confident are you in determining the failed action?" (1= Not Confident, 5=Very Confident).
- **Difficulty Rating (*DiffRate*)**: measures self-reported difficulty in determining a plausible solution to the encountered failure. Difficulty rating is measured using a 5-Point Likert Scale rating based on the question "How difficult was it to determine a solution to the encountered failure?" (1 = Not Difficult, 5=Very Difficult).

We hypothesize that the presence of AB or CB explanations will lead to high action identification scores *(AId%)*,

compared to no explanations. However, we believe that in determining a plausible solution to an encountered failure *(Sol%)*, those with CB explanations will perform better due to the additional contextual reasoning they are provided. We also believe that both confidence and difficulty ratings will correlate highly with respect to each conditions' action identification and solution percentages. That is, CB and AB will have comparable confidence *(ActConf)*, but CB participants will have lower perceived difficulty *(DiffRate)* than AB participants.

Experimental Setup

Our experimental setup uses a Gazebo simulation of a Fetch robot in a household setting performing a pick-and-place task (Figure 1). Similar to prior work in robotics (Banerjee et al. 2020), the robot's action set $A = \{move, segment, detect, findgrasp, grasp, lift, place\}$, where $move$ navigates the robot to a specified location, $segment$ is a perception action performed by the robot to identify which pixels in its sensory space correspond to objects, $detect$ performs object detection to obtain a label for a given object, $findgrasp$ executes grasp sampling to identify possible grasp poses for the gripper, $grasp$ moves the robot arm into a grasp pose and closes the gripper, $lift$ raises the arm, and $place$ places a held object at a specified location.

The robot's state at each time step t is defined as $s_t \in S$, where $S = S_e \cup S_l \cup S_i \cup S_k$ describe the entities in the environment, the location of each entity, the agent's internal states and the task states, respectively. S_e denotes the set of names for all entities in the environment, and does not change during the execution of π. We additionally define $S_o \subset S_e$ as the specific objects of interest to our agent, and $S_p \subset S_e$ as the semantic places of interest to the agent. S_o is defined as: *milk, coke can, ice cream, bottle, cup*, and S_p is defined as: *dining table, left kitchen counter*. $s_l(t) \in S_l$ is a vector of $\langle x, y, z \rangle$ locations of each entity $s_e \in S_e$ at a given time step t. $s_i(t) \in S_i$ is defined by three tuples $\langle x_{avel}, y_{avel}, z_{avel} \rangle$, $\langle x_{lvel}, y_{lvel}, z_{lvel} \rangle$,$\langle x_{pos}, y_{pos}, z_{pos} \rangle$ that describe the angular velocity, linear velocity and position of the agent at t. Finally, $S_k = \{k_{grasp}, k_{findgrasp}, k_{move}, k_{pick}, k_{detect}, k_{seg}\}$ where $s_k(t) \in S_k$ describes the status of each $a \in A$ at t, and whether each action is: active (0), completed (1) or errored (-1). Therefore, at all time steps, the number of elements in $s_k(t)$ is equal to the number of actions in A.

Simulating Failures

In this work, the agent's initial state is defined as $s_0 = \{\langle 0,0,0 \rangle, \langle 0,0,0 \rangle, \{null\}\}$, where the position tuple and the velocity tuples are set to zero, and the tasks states $s_k(0)$ are not defined. The agent's final state is defined as $s_T = \{\langle x_T, y_T, z_T \rangle, \langle 0,0,0 \rangle, \{1,1,...,1\}\}$, where the position tuple is set to the goal location, the velocity tuple is zero and the each task state in $s_k(T)$ is 1. With these assumptions, we define a failure f in plan π when any task state in s_k has a value -1.

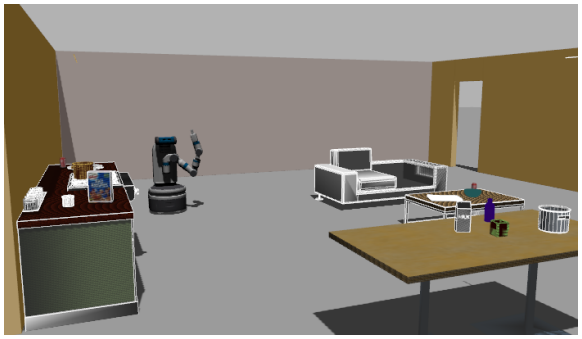

Figure 1: Home and kitchen environment in which Fetch performs a pick-and-place manipulation task.

Failure Type	Failed action	Failure Scenario
motion planning	grasp	obj. too far away
motion planning	grasp	obj. close to other objs.
detection	detect	obj. not present
detection	detect	obj. occluded
navigation	segment	mis-localization
navigation	move	controller

Table 2: Description of the associated failure types, and the robot's failed action for each specific failure scenario.

Previous work in fault diagnosis has summarized possible categories of faults that may occur in a given π. We specifically focus on *Component Faults* and *Contextual Faults*. While the former describe hardware or software module failures, the latter describe failures caused by changes in the environment (Banerjee and Chernova 2019). Table 2 lists the type of failures F_t, the scenarios F_s that can cause each type of failure, and the action on which π fails. For the purposes of our experimentation, we simulate the navigation errors as *Component Faults* caused by an error in the navigation software module, and the motion-planning and detection errors as *Contextual Faults*. We define two failure scenarios per failure type, reflecting on the fact that a given failure type may have multiple causes. We denote $\mathcal{F} = size(S_o) \times size(F_s)$ to be the set of all possible failure states, where S_o is the objects of interest and F_s are the failure scenarios.

Presenting Explanations to Users

For each study condition, participants were shown both failure scenarios from $\mathcal{F}$ as well as successful executions of π for the given task objective. Participants watched three videos of Fetch successfully executing π with randomly selected objects from S_o. The motivation for showing successful iterations of π was to show participants that the plan π itself was complete and executable. The remainder of the study consisted of identifying failure scenarios. Participants

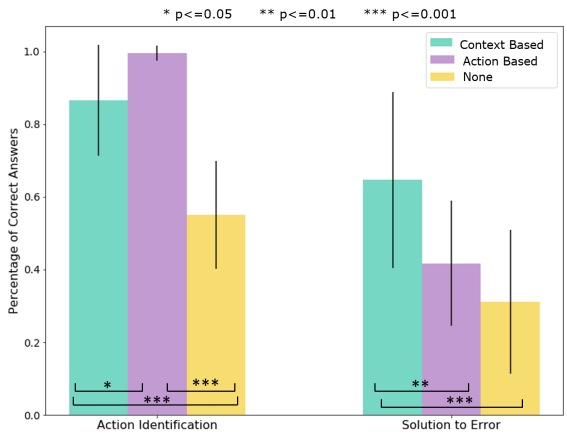

Figure 2: Average *AId%* and *Sol%* across the conditions, where AB and CB participants were presented with an $\mathcal{E}_{err}$.

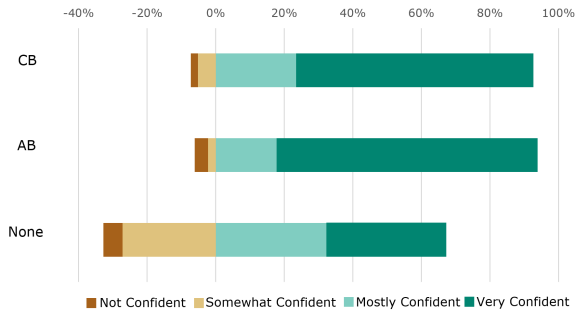

Figure 3: Participant's *ActConf* across the conditions where AB and CB participants were presented with an $\mathcal{E}_{err}$.

watched twelve videos, corresponding to twelve randomly chosen failure scenarios from $\mathcal{F}$. After each video, participants were presented with questions asking them to identify: the action a that prompted the failure in π, a solution to the encountered failure, and their perceived difficulty of the questions and perceived confidence of their answers.

Participants

We recruited 45 individuals from Amazon's Mechanical Turk, who were split into the three experimental groups. Our participants included 27 male and 18 female, who were all 25 or older. Specifically, 24 between 25- 34 years, 7 between 35-44 years, 8 between 45-54 years, and 6 who were 55 years or older. The task took participants approximately 20-25 minutes on average and they were compensated with $2.50.

Experimental Results

Since the participants' assessment data followed a normal distribution, we used ANOVA with a Tukey HSD post-hoc test to evaluate statistical significance for the *AId%* and

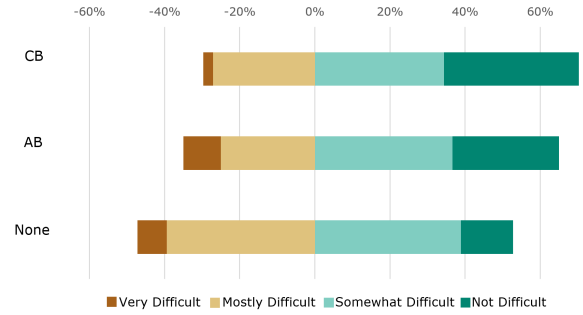

Figure 4: Participant's *DiffRate* across the conditions where AB and CB participants were presented with an $\mathcal{E}_{err}$.

Conditions	*ActConf*	*DiffRate*
CB vs. AB	NS	NS
CB vs. None	$p \leq 0.001$	$p \leq 0.001$
AB vs None	$p \leq 0.001$	NS

Table 3: Mann-Whitney U test significance values for the *ActConf* and *DiffRate* metrics with a Bonferroni p-value correction.

Sol% metrics. To evaluate the statistical significance for the self-reported rating metrics, *ActConf* and *DiffRate*, we used Kruskal-Wallis with a Mann-Whitney U post-hoc test and a Bonferroni correction.

Figure 2 presents the average percentage of correctly identifying the failed action (*AId%*), and correctly identifying a solution to the encountered failure (*Sol%*) for each study condition. We observe a significant difference in performance between the baseline (None) condition and the other conditions AB and CB. In other words, the presence of *any* explanation $\mathcal{E}_{err}$ helped participants better understand the failed action in π and deduce possible solutions to errors than those who were provided with no explanations. Additionally, we see that the inclusion of environmental context within an explanation (CB) significantly increased the accuracy of solutions to errors than explanations that only described the failed action (AB) in the plan π. This supports the idea that CB explanations help participants better understand the underlying cause of *why* an error has occurred and therefore how to provide recovery assistance, as opposed to only knowing *what* action caused the error within the system.

In Figure 3 and Figure 4, we observe the self-reported ratings of how confident participants were in discerning the failed action, *ActConf*, and how difficult it was to know the correct solution to a failure, *DiffRate*. The Likert scale data shows that participants who were given an explanation (CB or AB), were more likely to rate of 'Very Confident' and 'Not Difficult', compared to the those who received no explanations (None). We also observe that AB and None participants had a similar number of 'Very Difficult' ratings

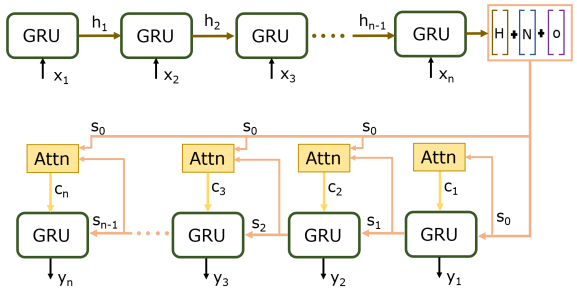

Figure 5: Our sequence-to-sequence model architecture for generating automated explanations.

compared to CB participants, supporting that in the context of deducing a solution, AB explanations were not significantly more helpful than having no explanations (None). Our statistical analyses in Table 3 support these conclusions, showing that *any* explanation significantly improved participants' *ActConf*, but only CB explanations were able to significantly improve participants' *DiffRate*.

Automated Explanation Generation

Our evaluations from above show that CB explanations were the most effective type of $\mathcal{E}_{err}$ that helps users make informed decisions on a failed plan. Therefore, in this section we introduce an automated explanation generation system that generalizes CB natural language explanations over the failure scenarios and failure types enumerated in Table 2.

Neural Translation Model Overview

We adapt a popular encoder-decoder network (Bahdanau, Cho, and Bengio 2015; Bastings 2018) utilized by (Ehsan et al. 2019) to train a model that can generate CB explanations from a set of features obtained from an agent's state. The set of features, U, is comprised of environment features X, raw features N and the desired object of interest o. As seen in Figure 5, the network's input features to the encoder include only the environment features. The decoder uses the output of the encoder, H, appended with the raw features, N, and the object of interest, o, to generate a sequence of target words $Y = \{y_1, y_2 ... y_m\}$, where y_i is a single word and Y is the CB explanation.

The encoder and decoder are comprised of Gated Recurrent Units (GRU). The encoder processes the input semantic feature set $X = \{x_1, x_2 ... x_n\}$, and produces a set of hidden states $H = \{h_1, h_2 ... h_n\}$, where a hidden state $h_i = GRU(x_i, h_{i-1})$. In other words, each hidden state is derived from the previous hidden state h_{i-1} as well as the current input semantic feature embedding x_i. The decoder's input, s_0, is the encoder's output vector concatenated with the raw feature set, N. The decoder then generates hidden states, where a single hidden $s_i = GRU(s_{i-1}, y_{i-1}, c_i)$. In this case, each hidden state s_i is derived from the previous predicted word y_{i-1}, previous hidden state s_{i-1} and a context vector c_i. The context vector represents a weighted attention vector that allows the model to dynamically focus on features from the decoder's previous hidden state,s_{i-1}, and the decoder's input vector,s_0, for producing the current hidden state s_i. To select an output word y_i, we apply a softmax function to s_i to obtain a probability distribution over all possible output words and choose the most probable word.

Feature Set

Recall from above that the agent's state is defined as $S = S_e \cup S_l \cup S_i \cup S_k$. We utilize the agent's state representations to define the model's feature set U. Instead of including the names of all entities S_e from the environment, we include only entities that are present at the agent's final location, denoted as Obj_G. Additionally, instead of including the agent's absolute position, we include it's position relative to the goal location, denoted as Rel_{a-Goal}. Similarly, we include the *minimum* relative distance between objects in Obj_G and the desired object $o \in S_o$, as Rel_{o-Obj_G}, and the relative distance between the desired object o and the agent as Rel_{a-o}. We also include the agent's angular v_{ang} and linear v_{lin} velocity as well all task states in S_k. Recall that S_k is comprised of $\{k_{grasp}, k_{findgrasp}, k_{move}, k_{pick}, k_{detect}, k_{seg}\}$ and describes the status of the agent's actions in A. Furthermore, we define o_p which represents whether $o \in Obj_G$ is true or false. Therefore, our environment and raw feature sets are defined as follows: $X = \{Obj_G\}$, $N = \{Rel_{a-Goal}, Rel_{o-Obj_G}, Rel_{a-o}, v_{ang}, v_{lin}, S_k, o_p\}$.

Data Collection & Processing

For our data set, we collected 54 videos, representing each failure scenario from Table 2. For each video, we sampled the collected data at 1 Hz to obtain a holistic representation of the agent's state when executing a plan π. In addition to annotating each failure state in π for each video, we annotated all successful states leading up to the failure state. Given our task objective, some examples of successful states included, "robot moving to the dining table","robot has segmented objects in the scene," and "robot has found grasps for the desired object". To differentiate these annotations from $\mathcal{E}_{err}$, we denote explanations of successful actions as $\mathcal{E}_{corr}$. In this work, $\mathcal{E}_{corr}$ explanations were only used in model training and were not a focus in the experimental evaluation above. In regard to task states in S_k, we assumed that any value in a given task state stays valid until a subsequent change overrides the current state. Additionally, any empty features were assigned an 'Empty' token that the model disregarded via masking.

Model Training

Our model is trained using a two-step grouped leave one out cross validation (LOOCV) with 10 folds. Our LOOCV consists of leaving out an entire scenario of data (25-30 data points) from each possible scenario in Table 2. The first LOOCV is utilized to populate the training set, while the second is used to populate the validation set. Based on the validation loss, on average, our model finishes training in 180 epochs. We train with a batch size of 20. Our GRU cells

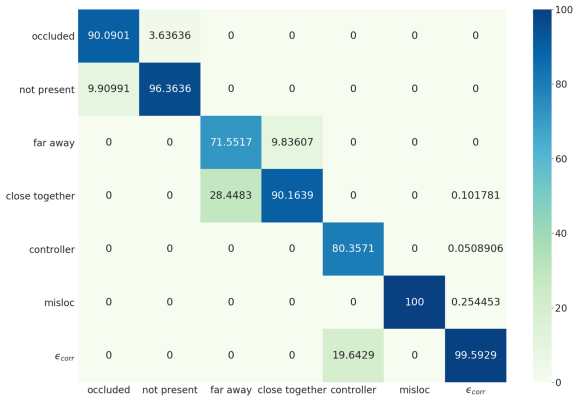

Figure 6: Confusion matrix analysis of our model's performance where the first six columns represent $\mathcal{E}_{err}$ explanations and the last column represents $\mathcal{E}_{corr}$ explanations.

in the encoder have a hidden vector size of 20 and the GRU cells in the decoder have a hidden vector size of 49 which accounts for additional raw features, N and the embedding size of o. We train our model using a Cross Entropy classification loss optimized via Adam with a learning rate of 0.0001.

Quantitative Model Evaluation

Figure 6 presents the performance of our model across both the six failure explanations $\mathcal{E}_{err}$ presented in Table 1 as well as non-error explanations, $\mathcal{E}_{corr}$. In our evaluation, a predicted phrase is only marked correct if it identically matches its target phrase.

On average, our model is able to generalize failure scenarios with a 89.7% accuracy. We observe that for each failure scenario, the model has a much larger true positive percentage than false positive or false negative percentage. Furthermore, we see that for each failure scenario under the failure types 'detection' and 'motion-planning' from Table 2, the false positives are within the same failure type. For example a 'not present' explanation is only wrongly generated as an 'occluded' explanation, both of which are a 'detection' failure type. Similarly, 'far away' is only wrongly generated as a 'close together' explanation and vice versa; both of these failure scenarios fall under the 'motion-planning' failure type. However, the failure scenario 'controller', under the 'navigation' failure type, does not follow this same pattern. Although the 'controller' error does not get wrongly predicted as any of the other failure scenarios, it is incorrectly predicted as a correct navigation explanation 19.6 percent of the time. Additionally, while we do not analyze the false positives within differing $\mathcal{E}_{corr}$, we do observe that the non-error explanations are rarely confused with any of the $\mathcal{E}_{err}$ explanations.

Conclusion

In this work, we have introduced a new format of explanations, context-based explanations, that is meaningful to a non-expert in not only understanding the failed action in a plan, but also in selecting a recovery solution for a failed action. To validate our context-based explanations, we evaluated it in the domain of a pick-and-place manipulation robot task and investigated users' accuracy in failed action identification, correct recovery identification as well as self-reported ratings of confidence and difficulty in selecting an answer. The results from our user study show that for explanations to be effective in aiding non-expert users to select accurate failure recoveries, the explanations need to include environmental context. The CB explanations allowed users to more effectively select the cause of failure, and the correct failure recovery technique than those who received no explanations. Additionally, we have we have adapted an existing neural translation model from (Ehsan et al. 2019) to develop automated, CB explanations. The accuracy scores from the confusion matrix show our model's ability to generalize and generate these CB explanations for varied failure scenarios.

This work is motivated to aid non-expert users understand failures that an AI agent may encounter while executing a plan to in turn provide effective failure recovery solutions. Although it includes important contributions, there are limitations that should be addressed in future work. First, while the CB explanations are significantly more useful for assisting in failure recovery than AB or no explanations, they still are not guaranteed to be useful to *all* non-expert users. Therefore future work entails being able to tailor explanations to individual users using reinforcement learning techniques similar to those found in recommender systems (Wang et al. 2018). Furthermore, our automated explanation generation model can so far generalize over varying failure scenarios. However, a next progression would be to also extend the current model to generalize over varying environments and varying tasks.

Acknowledgments

This material is based upon work supported by the NSF Graduate Research Fellowship under Grant No. DGE-1650044.

References

Adadi, A., and Berrada, M. 2018. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). IEEE Access 6:52138–52160.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In 3rd International Conference on Learning Representations, ICLR 2015.

Banerjee, S., and Chernova, S. 2019. Fault diagnosis in robot task execution. In AAAI Spring Symposium Series.

Banerjee, S.; Daruna, A.; Kent, D.; Liu, W.; Balloch, J.; Jain, A.; Krishnan, A.; Rana, M. A.; Ravichandar, H.; Shah, B.; et al. 2020. Taking recoveries to task: Recovery-driven development for recipe-based robot tasks. arXiv preprint arXiv:2001.10386.

Bastings, J. 2018. The annotated encoder-decoder with attention.

Bauer, A.; Wollherr, D.; and Buss, M. 2008. Human–robot collaboration: a survey. International Journal of Humanoid Robotics 5(01):47–66.

Bloss, R. 2011. Mobile hospital robots cure numerous logistic needs. Industrial Robot: An International Journal.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. arXiv preprint arXiv:1701.08317.

Chakraborti, T.; Kulkarni, A.; Sreedharan, S.; Smith, D. E.; and Kambhampati, S. 2019. Explicability? legibility? predictability? transparency? privacy? security? the emerging landscape of interpretable agent behavior. In Proceedings of the international conference on automated planning and scheduling, volume 29, 86–96.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2020. The emerging landscape of explainable ai planning and decision making. arXiv preprint arXiv:2002.11697.

Ehsan, U.; Harrison, B.; Chan, L.; and Riedl, M. O. 2018. Rationalization: A neural machine translation approach to generating natural language explanations. In Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, 81–87.

Ehsan, U.; Tambwekar, P.; Chan, L.; Harrison, B.; and Riedl, M. O. 2019. Automated rationale generation: a technique for explainable ai and its effects on human perceptions. In Proceedings of the 24th International Conference on Intelligent User Interfaces, 263–274.

Hägele, M.; Nilsson, K.; Pires, J. N.; and Bischoff, R. 2016. Industrial robotics. In Springer handbook of robotics. Springer. 1385–1422.

Hoffmann, J., and Magazzeni, D. 2019. Explainable ai planning (xaip): Overview and the case of contrastive explanation. In Reasoning Web. Explainable Artificial Intelligence. Springer. 277–282.

Hu, J.; Edsinger, A.; Lim, Y.-J.; Donaldson, N.; Solano, M.; Solochek, A.; and Marchessault, R. 2011. An advanced medical robotic system augmenting healthcare capabilities-robotic nursing assistant. In 2011 IEEE international conference on robotics and automation, 6264–6269. IEEE.

Knepper, R. A.; Tellex, S.; Li, A.; Roy, N.; and Rus, D. 2015. Recovering from failure by asking for help. Autonomous Robots 39(3):347–362.

Krarup, B.; Cashmore, M.; Magazzeni, D.; and Miller, T. 2019. Model-based contrastive explanations for explainable planning.

Lawton, J. 2016. Collaborative robots. International Society of Automation 12–14.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. " why should i trust you?" explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 1135–1144.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE international conference on computer vision, 618–626.

Sreedharan, S.; Srivastava, S.; Smith, D. E.; and Kambhampati, S. 2019. Why can't you do that hal? explaining unsolvability of planning tasks. In IJCAI, 1422–1430.

Wang, X.; Chen, Y.; Yang, J.; Wu, L.; Wu, Z.; and Xie, X. 2018. A reinforcement learning framework for explainable recommendation. In 2018 IEEE International Conference on Data Mining (ICDM), 587–596. IEEE.

Yang, P.-C.; Sasaki, K.; Suzuki, K.; Kase, K.; Sugano, S.; and Ogata, T. 2016. Repeatable folding task by humanoid robot worker using deep learning. IEEE Robotics and Automation Letters 2(2):397–403.

Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan explicability and predictability for robot task planning. In 2017 IEEE international conference on robotics and automation (ICRA), 1313–1320. IEEE.

Zhang, Q.; Yang, Y.; Ma, H.; and Wu, Y. N. 2019. Interpreting cnns via decision trees. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 6261–6270.