Interactive Planning-based Cognitive Assistance on the Edge

Zhiming Hu Maayan Shvo Allan Jepson Iqbal Mohomed
Samsung AI Center-Toronto

Abstract

Real-time cognitive assistance is one of the most exciting applications in the age of Augmented Reality (AR). Several research groups have explored the use of cognitive assistants, embodied within smartphones or wearable AR glasses, to guide users through unfamiliar tasks (e.g., assembling a piece of furniture or following a recipe). These systems generally consist of two high-level modules: a perceptual module (e.g., a deep-learning based vision system) and a cognitive module (implemented via a rule-engine or state machine), and must operate in near real-time. As such, cognitive assistants are illustrative use-cases for edge computing. While prior work has focused on pushing the frontier of what is possible, it suffers from some defects that hinder practical deployment. First, much research on cognitive assistants has assumed an accurate visual perception system, which may not be true in practice. Second, while some work has explored user errors in performance of tasks, the manner in which this is done is not scalable (i.e., possible errors are explicitly specified in a state machine representation apriori). To address these limitations, in this paper, we propose (i) to involve users in resolving the ambiguity/uncertainty of visual inputs and (ii) to employ automated planning tools as well as execution monitoring techniques to keep track of the task states, as well as to generate new plans to recover from users' mistakes if necessary. To verify the feasibility of our system, we implemented and tested it on both an Android phone and HoloLens 2, supported by an edge server for off-loading computation.

1 Introduction

Cognitive assistance is an exciting emerging application of Augmented Reality (AR) enabled wearable devices. A cognitive assistant is envisioned to observe the user as he/she carries on her tasks, but can intervene if it detects the user might need assistance. Several research groups [9, 16] have explored the frontiers of this problem, and the resulting prototypes typically consist of two high level modules. The first component (perceptual module) perceives the state of the world, while the other component (cognitive module) tracks progress as the user performs a task, and can intervene with assistance if needed.

To realize a practical cognitive assistant, we first need a robust perceptual module, whose purpose is to understand the visual surrounding world and compute the current task state. Fortunately, with advances in deep learning, both the accuracy and the efficiency of many computer vision tasks such as object detection and image classification, have been substantially improved. Moreover, models for these tasks can also meet tight latency requirements even when they are deployed in the edge [9, 14].

However, in practice, the current task state could be ambiguous for the following reasons. First, the perceptual module may provide ambiguous detection results in many different cases [13, 21]. For instance, a detector may produce a wrong label for an object if it is too far or too close to the camera. Second, for efficiency, many systems that target practical deployment only take a single frame as input to obtain the current task state and do not consider temporal relations across frames. Therefore, even with accurate object detection results on a single frame, the system may still be confused about the corresponding task state because different task states may look exactly the same.

Besides the robust visual perception module, we need an efficient state tracker to keep track of the task states and provide real-time suggestions to end users. On the one hand, a few systems built on top of the Gabriel [5] system adopt finite state machines (FSM) to track the state transitions, which is very efficient. However, it can only recover from a few predefined user errors, which are hard-coded in the state machine. Furthermore, it is often infeasible to exhaustively list all the possible state transitions in a predefined state machine given dynamic conditions. On the other hand, Jarvis [16] proposes a planner to dynamically generate the next step, which in theory is able to recover from any number of incorrect steps. However, Jarvis [16] triggers the planner every time it needs to provide suggestions for the next step, even if the user is follow-

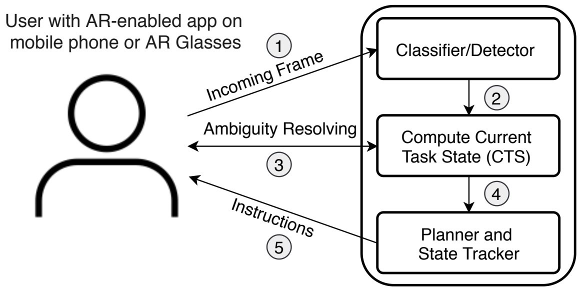

User with AR-enabled app on mobile phone or AR Glasses
1
Incoming Frame
Classifier/Detector
2
Ambiguity Resolving
3
Compute Current Task State (CTS)
4
Instructions
5
Planner and State Tracker

Figure 1: High level system design.

ing the plan. This could be too expensive for time-sensitive applications and wastes computational resources.

In this paper, we argue that obtaining accurate current task state from the perceptual module and dynamic state tracking are the foundations for a cognitive assistant to provide meaningful feedback to end users. To this end, we propose a cognitive assistance framework that will interact with users to resolve ambiguity if there is any when deciding the current task state. Moreover, to be able to handle unpredictable user errors, we employ execution monitoring techniques in a dynamic state tracking system, which tracks the current task state based on the perceptual module and recovers from dynamic user errors. More specifically, the dynamic state tracking system consists of *a planner and a state machine*. In the beginning, the state machine is generated from the initial plan. If the user is following the plan, the state transitions will be efficiently handled by the state machine. If not, the planner will try to generate a new plan and a new state machine is built according to the new plan. By combining the planner with a state machine, we could both benefit from the flexibility of the planner for recovering from different user errors and the efficiency of state machines for fast state transitions.

The contributions of our paper are two-fold. First, we propose to involve users in disambiguating ambiguous current task state that could not be distinguished solely based on the perceptual module. Second, by building upon techniques from both automated planning and execution monitoring and combining a planner with a state machine, we are able to incorporate a state tracking system in our approach to dynamically track the current task state and to recover from user errors in different conditions.

2 Background and Motivation

In this section, we show two key challenges of building a practical cognitive assistant on the edge.

2.1 Ambiguous Current Task State

A computationally efficient solution for the perception system to obtain the current task state could be an object detector or a classifier. After that, the cognitive assistant will try to infer the current task state based on detection results [16].

However, the current task state could be ambiguous for the following reasons. First, the perceptual module may fail to detect the objects in the view accurately. For instance, the object detection algorithm may miss some objects or provide false detection results. Second, inferring the current task state on a single frame could be ambiguous as it ignores temporal relations across frames. For instance, given a frame where a user has placed one object on top of another, with the new object occluding the previous one, it is uncertain whether additional objects have been added or the previous top object has been removed.

Therefore, we argue that clearly identifying those ambiguous cases when deciding the current task state and resolving the ambiguity are very important. Otherwise, it may lead to incorrect instructions towards the task goal. Indeed, the planner used in this work assumes a fully observable setting and cannot handle uncertainty stemming from state ambiguity. While there exist planning paradigms that handle uncertainty (e.g., contingent and conformant planning [12]), state ambiguity can create computational challenges and even cause the planning system to generate plans that are inappropriate in the context of the actual state of the world.

2.2 Dynamic User Errors

After we get the current task state, the next step is to figure out the step-by-step instructions towards the task goal. To achieve this, we may apply a finite state machine (FSM) for this task, as in related applications built on top of Gabriel [5].

However, a FSM is limited for many cognitive assistance applications for the following reasons. First, it is impractical to list all the possible failure cases that users may encounter when completing the tasks. Moreover, in real applications, users may not follow instructions for multiple steps, which may result in a large number of undefined states in the predefined state machine. Furthermore, it is often infeasible to update the sequence of states required to achieve the original goal since normally some sort of corrective action would be required to either repair or work around the previous mistakes.

Instead, a planner [16] can resolve these issues because the dynamic planner does not need to fix the states and state transitions beforehand and it can always generate new plans if necessary. However, running the planner is time consuming and triggering the planner for every state transition is inappropriate for latency-sensitive cognitive assistance applications.

3 System Design

The overview of our system is shown in Fig. 1. In this figure, we can see that live video frames are coming from end devices like smartphones or AR glasses. The frames will be serialized and sent to the edge server for further processing.

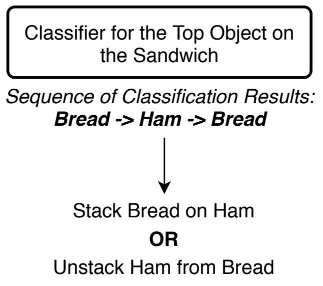

Classifier for the Top Object on the Sandwich
Sequence of Classification Results:
Bread -> Ham -> Bread
Stack Bread on Ham
OR
Unstack Ham from Bread

Figure 2: A case of ambiguity when deciding the current task state based on the information from the perceptual module.

In the edge server, the frames will first go through a perceptual module, which could be a classifier or an object detector. The classification/detection results will then be used to compute the current task state. If there is any ambiguity in this step, our system will initiate a dialogue with the user to resolve the ambiguity. Finally, given the up-to-date current task state, the planner and the state tracker will generate the corresponding next step instruction to the user.

We will use a sandwich assembly task as an example for the rest of the paper. This is the same task used in the Gabriel cognitive assistant and is a good toy example to consider the key aspects of the system. The user is given a set of ingredients (e.g. tomato, lettuce, bread) and given explicit instructions to assemble a particular sandwich.

3.1 Dialogue-based Ambiguity Resolving

We may not be able to compute the current task state because of partial observability in some applications. For instance, as shown in Fig. 2, if a user is stacking a sandwich, because of occlusion, the perceptual module can only detect the top object on the sandwich. Initially, the bread is detected followed by a piece of ham. In this case, the system will assume that a piece of ham has been put on the bread. However, after that, if the top object then changes to bread again then the system could conclude that either a new slice of bread added or perhaps the ham was taken away. (Many other possibilities exist, but we will assume the user is cooperative and is only adding or removing one object at a time.)

In the above case, the system can resolve the ambiguity in current task state by requesting for clarification from the user. Previous work [20] suggests that a clarification request should list all uncertain options when there exist no more than two options (e.g., "Did you <add/remove> <item1> or <item2>?". Otherwise, humans prefer to be asked a Wh-question (e.g., "What item did you <add/remove>?").

3.2 Planner

To address the aforementioned shortcomings of FSMs we turn to AI planning (e.g., [8, 11]), where the objective is to generate a partially or totally ordered sequence of actions – a plan – which transforms some initially specified state of the world to a desired (goal) state. Research in the field, spanning many decades, has strived to produce general solutions to this problem, leading to planners which are not domain-specific. That is, general-purpose planners have been created, which are agnostic to problem and domain specific peculiarities and return a solution, a plan, given some input specified in a generic and standard format (e.g., the Planning Domain Definition Language (PDDL) [15]).

Definition 1 (Planning Problem). *A planning problem is a tuple of the form $\mathcal{P} = (F, A, I, G)$, where F is a finite set of fluent symbols, A is a set of actions, $I \subseteq F$ defines the initial state, and $G \subseteq F$ defines the goal state. Each action $a \in A$ is associated with a precondition, Pre_a, add effects, eff_a^+, delete effects, eff_a^-, and non-negative action costs, COST(a).*

A state, s, is a set of fluents that are true (a fluent is a condition that can change over time). An action $a \in A$ is *executable* in a state s if $Pre_a \subseteq s$. The successor state is defined as $\delta(a,s) = ((s \setminus eff_a^-) \cup eff_a^+)$ for the executable actions. The sequence of actions $\pi = [a_1, ..., a_n]$ is executable in s if the state $s' = \delta(a_n, \delta(a_{n-1}, \ldots, \delta(a_1, s)))$ is defined. Moreover, π is the solution to a planning problem $\mathcal{P}$ if it is executable from I and $G \subseteq \delta(a_n, \delta(a_{n-1}, \ldots, \delta(a_1, I)))$.

Given a user goal, G, we instantiate a planning problem $\mathcal{P} = (F, A, I, G)$ where I, A and F are predefined. We make assumptions pertaining to the initial state I that may or may not hold in the world. To illustrate, we partially model our sandwich example as a planning problem:

- *stack(x,y)* $\in A$
  - $Pre_{stack} = \{clear(x), clear(y), ontable(x)\}$
  - $eff^+_{stack} = \{on(x,y)\}$ (note: x is on y)
  - $eff^-_{stack} = \{clear(y)\}$
- $G = \{$*onTable*(bread1),*on*(ham,bread1),*on*(lettuce,ham), *on*(bread2,lettuce),*on*(tomato,bread2),*on*(bread3,tomato)$\}$

Given that initially all of the ingredients are clear and on the table, one possible solution to the above problem is a plan $\pi =$ *stack*(ham,bread1),*stack*(lettuce,ham),*stack*(bread2,lettuce), *stack*(tomato,bread2),*stack*(bread3,tomato).

3.2.1 Interactive State Tracking System

The system guides the user through the execution of π by providing her with the plan incrementally, one step at a time. In the beginning, we generate the initial plan π and build a state machine based on the plan. After that, the vision system is assumed to observe actions performed by the user (e.g., *stack*(lettuce,ham)) which are then used to update the current state of the world. More specifically, when the vision system observes some action a in some state s, the new state will be

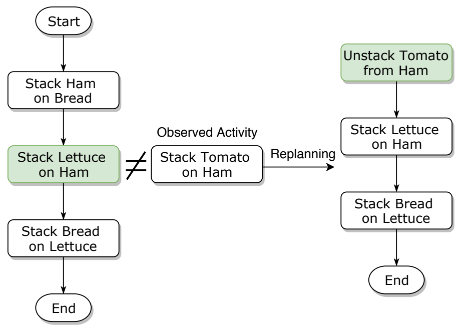


Figure 3: State tracking with a planner and state machines. The green box shows the current expected action.

the successor state $\delta(a,s)$. Following the observation of a user action, we check whether the observed action a matches the current step of the plan that was given to the user. If yes, we will provide the next instruction directly based on the state machine and change the current state in the state machine. However, if not, we generate a new planning problem where I is the updated state $\delta(a,s)$ and solve it to obtain π'. For instance, if the user made a mistake such as stacking tomato on ham instead of stacking lettuce on ham, π' will correct the user's mistake and instruct her to remove the tomato and place the lettuce instead. An illustrative figure is shown in Fig. 3. With the new π', a new state machine is built based on the new list of instructions. The approach of integrating the planner with a state machine is advantageous since the system will not trigger the planner every time as long as the user is following the plan. Note that we employ here a simple approach to plan execution and monitoring; the rich body of related work offers a myriad of solutions (e.g., [7, 8]).

4 Implementation

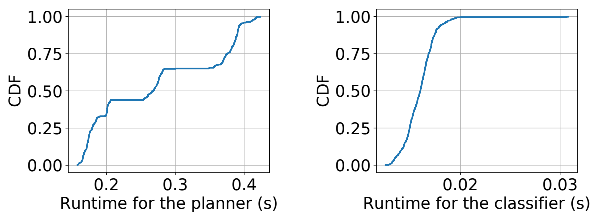
We built our system on top of the gabriel-sandwich project [9] with a edge server architecture and adopted the WebSocket to transfer frames and resulting instructions between the client and edge server. We have implemented a new Hololens 2 client and added our image classification module, the activity recognition module and the interactive state tracking module in the edge server. A video demonstration is available [2].

For the client, as we can see in Fig. 1, it also acts as the camera source. In our system, we only fetch new frames from the camera source if there are less than two frames that are under processing in the edge server. All the other frames will be dropped at the camera source directly.

In the edge server, for the image classification module, we adopt the transfer learning method and build our model based on a lightweight MobileNet v2 backbone network [1], which could be deployed on a lot of edge devices. We have five classes in our case, which are bread, tomato, lettuce, ham and a label for the background class. To fine tuning the network, we take pictures on the objects in a sandwich toy and prepare around 60 images for each label. It achieves the accuracy of more than 99% for both the training set and the validation set after 20 iterations. In the edge server, we also smooth the detection results and only report the detection of an object if it has been detected for four consecutive frames.

As shown in Fig. 2, the top object cannot fully represent the current task state because different actions may lead to the same top object. Therefore, we propose a simple activity recognition method based on the changes of the top objects and the objects that we already stacked in the pile. For instance, if we have stacked A and B in sequence, we will know that C is put on B if C is detected right after B. This method allows us to infer a temporal relation (e.g., cheese stacked on bread) by leveraging a classifier running on a sequence of frames. However, if A is detected after B, we need to involve the user to resolve the ambiguity and figure out whether the user has put another A on B or it has removed the B on top.

Given the user's goal (e.g., make a ham sandwich), we call the fast-downward planning system [10] to generate a plan, which achieves the goal. The user is then provided with the first instruction. Following this, when a user action is observed, we check whether the action matches the current step of the plan. As described previously, we only call the planner if the observed action does not match the current instruction. Otherwise, we directly track the state transitions based on a state machine derived from the plan. Whenever a user action is observed, we update the state of the world by modifying the PDDL file to reflect effects of the action.



(a) Runtime for the planner. (b) Runtime for the classifier

Figure 4: Runtime for the planner and the classifier.

5 Evaluation

In this section, we show the characteristics of running the planner and the classifier in our experiments.

5.1 Case Study: A Planning Example

We show a planning example in Table 4. In this table, we can see that the original plan is to stack bread1, ham1, lettuce1, bread2, tomato1 and bread3 in sequence. However, if we make a mistake and stack the lettuce instead of ham in the second step, the planner automatically generates a new plan and the

Classification Result	Input	Inferred Action	Plan
Bread	-	-	[‘stack ham1 bread1’, ‘stack lettuce1 ham1’, ‘stack bread2 lettuce1’, ‘stack tomato1 bread2’, ‘stack bread3 tomato1’]
Lettuce	-	‘stack lettuce1 bread1’	[‘unstack lettuce1 bread1’, ‘stack ham1 bread1’, ‘stack lettuce1 ham1’, ‘stack bread2 lettuce1’, ‘stack tomato1 bread2’, ‘stack bread3 tomato1’]
Tomato	-	‘stack tomato1 lettuce1’	[‘unstack tomato1 lettuce1’, ‘unstack lettuce1 bread1’, ‘stack ham1 bread1’, ‘stack lettuce1 ham1’, ‘stack bread2 lettuce1’, ‘stack tomato1 bread2’, ‘stack bread3 tomato1’]
Lettuce	‘yes’	‘unstack tomato1 lettuce1’	[‘unstack lettuce1 bread1’, ‘stack ham1 bread1’, ‘stack lettuce1 ham1’, ‘stack bread2 lettuce1’, ‘stack tomato1 bread2’, ‘stack bread3 tomato1’]
Bread	‘yes’	‘unstack lettuce1 bread1’	[‘stack ham1 bread1’, ‘stack lettuce1 ham1’, ‘stack bread2 lettuce1’, ‘stack tomato1 bread2’, ‘stack bread3 tomato1’]

Table 1: Recovering from two user errors. The input is the response from the user for the question: have you removed something?

first step in the new plan is to unstack the lettuce. Similarly, in the third step, if we stack the tomato, the planner will remind us to unstack the tomato first before going ahead. In the fourth step, it requires user input because of the ambiguity. After we respond with ‘yes’, the system can confirm that we have removed the tomato on top and the newly generated plan also reflects the changes. As we can see, our planner can recover from arbitrary mistakes without being explicitly coded.

5.2 Runtime of the Planner and the Classifer

The planner and the classifier are running inside an Nvidia-Docker container, on a host that has two GTX 1080 Ti GPUs. The CPU frequency is 3311.00 Mhz and the amount of free CPU memory inside the container is 4.7 GB. The planner runs on CPU and the runtime is shown in Fig. 4(a). After a closer inspection of the runtime, we find out that when the user has made mistakes, the runtime of the planner is around 0.35 to 0.4 seconds. When the planner is generating the initial plan, the runtime is around 0.25 to 0.28 seconds. If the user is following the plan, the searching space is getting smaller and the runtime of the planner is less than 0.2 seconds. Here, our planner will search for the optimal solution, which has the smallest number of steps in the plan and normally takes longer. We may trade the optimality of the solution for lower searching time in a time-sensitive application.

Our results show that it is infeasible to run a planner on every single observation from the perceptual system. Our approach addresses this by running the planner only when necessary, translating the resulting plan to a FSM representation, and thus saves on computation.

The classifier is built on top of the MobileNet v2 backbone network. The inference time is shown in Fig. 4(b). We can see that most of the inference time is below 0.02 seconds after the system has warmed up.

6 Related Work

In this section, we discuss the most closely related work in cognitive assistance and planning domain.

Cognitive Assistance. Jarvis [16] is the most related work. It has a perceptual module and a planner to generate dynamic plans. However, our approach is different in the following ways. First, we reveal that there are ambiguous cases in the current task state and we propose to involve the user to resolve the ambiguity. Second, we obtain the current task state based on the classification results across multiple frames instead of one frame as in Jarvis. Finally, we only call the planner when necessary. Unlike Jarvis, our state-tracking system can be used to track the progress in cases when the user is following the plan, alleviating the need to spend computational resources on executing the planner in those cases.

Another closely related system is Gabriel [9], which also considers a cognitive assistant supported by the edge. While Gabriel uses an efficient finite state machine to represent task states, we propose an interactive planner along with state machines in order to deal with dynamic conditions (such as unpredictable user errors), which are often infeasible to be explicitly specified in a finite state machine. Also, in our perceptual module, we decoupled classification labels from task states, which could be more flexible.

Interactive Planning. Chakraborti et al. propose a projection-aware task planning and execution architecture [4] where the system can exchange the intents with the collaborating human. Differently, we employ a dialogue module to communicate with the user in order to disambiguate observed actions. Furthermore, while their work focuses on a setting where both the robot and the human are jointly operating in the world, our work focuses on a setting where the system is only an observer and the human is the sole executor.

7 Conclusion

Cognitive assistance is one of the key applications on AR enabled devices. To build such a cognitive assistant, we propose to involve the users to resolve the ambiguity of visual perceptions when necessary. Moreover, we propose to employ automated planning tools as well as execution monitoring techniques to keep track of the task states, as well as to generate new plans to recover from users’ mistakes if necessary. Preliminary results show that the low overhead of our design may be more amenable to practical deployments.

8 Discussion Topics and Future Work

- What are the appropriate metrics that would clearly demonstrate the feasibility of a planner-based cognitive assistant and accurately evaluate the performance under different scenarios? This is a multi-disciplinary problem and touches on Systems, HCI, planning, and knowledge representation.

- Are the infrastructure facilities envisioned in edge cloud infrastructure sufficient to support cognitive assistants for every mobile user? Are there infrastructure elements (e.g. at the level of perceptual detectors or planners) that can be utilized in a multi-tenant fashion? Muise's system is an example of this idea at a small scale [17].

- Specifying a planning domain may be in cases more complex than specifying a state machine. While we have effectively created a planning domain for a particular toy problem, does the approach generalize to other problems that researchers in the field have studied? Relatedly, it is worth mentioning the growing body of work concerned with learning planning domains from textual and visual inputs which can help mitigate for the human-effort required to author planning domains (e.g., [3, 22, 23]). Additionally, there is a body of work concerned with the development of tools that facilitate the authoring of planning domains (e.g., [18, 19]).

- As our activity recognition algorithm is based on the changes of the top objects, we cannot detect the actions if the user has put some objects on the bottom of the pile or in the middle of the pile. For the same reason, we could not detect the activity if the user has put another instance of the top object to the top.

- In this work, the actions in the plan form a total order set. We could also extend our system to support partially ordered actions, which is required by many applications, by adopting the linear temporal logic (LTL).

- In the current system, we trust the user inputs to resolve the ambiguity. However, there might exist some "harmful users", giving on purpose incorrect information to the system. To resolve this issue, we may adopt approaches powered by crowd sourcing [6] to separate "harmful users" from normal users.

- We also plan to extend the system by providing personalized instructions to users given the user data or by learning their behaviour models. We could also provide more flexible input modalities to improve the usability of the system.

9 Acknowledgments

We would like to thank Krish Perumal for his help on the design of the dialogue module. We also gratefully acknowledge the insightful comments from the anonymous reviewers.

References

[1] MobileNet. shorturl.at/wABVX, 2020. Accessed: 2020-02-14.

[2] Planning-based Cognitive Assistance. https://youtu.be/RttF2NBBWh8, 2020. Accessed: 2020-02-27.

[3] SRK Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. Learning High-Level Planning from Text. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 126–135. Association for Computational Linguistics, 2012.

[4] Tathagata Chakraborti, Sarath Sreedharan, Anagha Kulkarni, and Subbarao Kambhampati. Projection-Aware Task Planning and Execution for Human-in-the-Loop Operation of Robots in a Mixed-Reality Workspace. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4476–4482. IEEE, 2018.

[5] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, et al. An Empirical Study of Latency in An Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance. In *Proc. of the Second ACM/IEEE Symposium on Edge Computing*, page 14. ACM, 2017.

[6] Florian Daniel, Pavel Kucherbaev, Cinzia Cappiello, Boualem Benatallah, and Mohammad Allahbakhsh. Quality Control in Crowdsourcing: A Survey of Quality Attributes, Assessment Techniques, and Assurance Actions. *ACM Computing Surveys (CSUR)*, 51(1):1–40, 2018.

[7] Christian Fritz and Sheila A. McIlraith. Monitoring Plan Optimality During Execution. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007*, pages 144–151.

[8] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[9] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards Wearable Cognitive Assistance. In *Proc. of the 12th annual international conference on Mobile systems, applications, and services*, pages 68–81. ACM, 2014.

[10] Malte Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[11] James A Hendler, Austin Tate, and Mark Drummond. AI Planning: Systems and Techniques. *AI magazine*, 11(2):61, 1990.

[12] Jörg Hoffmann and Ronen I Brafman. Conformant Planning via Heuristic Forward Search: A New Approach. *Artificial Intelligence*, 170(6-7):507–541, 2006.

[13] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[14] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[15] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL-the Planning Domain Definition Language. 1998.

[16] Shiwali Mohan, Kalai Ramea, Bob Price, Matthew Shreve, Hoda Eldardiry, and Les Nelson. Building Jarvis-A Learner-Aware Conversational Trainer. In *IUI Workshops*, 2019.

[17] Christian Muise. Planning.domains. *ICAPS system demonstration*, 2016.

[18] Christian Muise, Tathagata Chakraborti, Shubham Agarwal, Ondrej Bajgar, Arunima Chaudhary, Luis A Lastras-Montano, Josef Ondrej, Miroslav Vodolan, and Charlie Wiecha. Planning for Goal-Oriented Dialogue Systems. *arXiv preprint arXiv:1910.08137*, 2019.

[19] Shirin Sohrabi, Anton V Riabov, Michael Katz, and Octavian Udrea. An AI Planning Solution to Scenario Generation for Enterprise Risk Management. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[20] Tom Williams and Matthias Scheutz. Resolution of Referential Ambiguity in Human-Robot Dialogue Using Dempster-Shafer Theoretic Pragmatics. 2017.

[21] Xiongwei Wu, Doyen Sahoo, and Steven CH Hoi. Recent Advances in Deep Learning for Object Detection. *Neurocomputing*, 2020.

[22] Kristina Y Yordanova. TextToHBM: A Generalised Approach to Learning Models of Human Behaviour for Activity Recognition from Textual Instructions. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[23] Ziqi Zhang, Philip Webster, Victoria S Uren, Andrea Varga, and Fabio Ciravegna. Automatically Extracting Procedural Knowledge from Instructional Texts using Natural Language Processing. In *LREC*, volume 2012, pages 520–527, 2012.