EchoSense: An Attempt at Reinforcement Learning Approach to Personalized Information Prioritization for Visually Impaired Users

Nahom Tamene

December 16, 2025

Abstract

This document reveals EchoSense, a system reliant on AI, which helps in the processing of huge volumes of data in camera-based applications for blind users. The system uses a Deep Q-Network (DQN) reinforcement learning agent that learns how to prioritize information according to each user's feedback. What is more, EchoSense does not only take the feedback given by the user but also the feedback from the user's behavior and combines these. The system employs the combination of three pre-trained computer vision models (YOLOv8, CLIP, EasyOCR) as feature extractors and adds a trainable decision layer which tries to figure out the best possible prioritization policies. The project has opened new avenues in learning about reinforcement for human-in-the-loop systems, designing state representation, and the issues that arise in applying ML systems in real-time applications.

1 Introduction

1.1 Problem Statement

People with visual impairment have a hard time understanding and moving around in the world. Although camera-based devices have come up as new solutions to this problem, they all have a common drawback: people get overwhelmed with too much information. Usually, these gadgets give detailed descriptions of everything around the user like objects, text, and people, making it very hard for the user to get a clear insight due to the large amount of irrelevant information.

Users of old-style systems were provided with even the most minute details about the detected elements, thus being bombarded with irrelevant information. For instance, an application could say: "I see a person, a chair, a table, a laptop, a sign that says 'Exit'..." where the important exit sign is completely missed and is now among the non-important facts.

1.2 Motivation

The primary difficulty is not the precision of detection but rather the smart information prioritization. Users are different in terms of what they need: a user focused on navigation will give priority to signs and landmarks, whereas a user who is mainly concerned with social interaction will give priority to people and conversations. Furthermore, variables like time of day, type of location, and current goals are very important in determining what information is relevant.

This is the problem that this project intends to solve through the innovation of EchoSense, a system that uses reinforcement learning to master the art of the personalized information delivery. The system learns from both explicit feedback (user ratings) and implicit signals (response times, patterns of interaction) to adapt to individual user preferences, which results in a personalized experience that lessens cognitive load but at the same time ensures that safety-critical information is available.

1.3 Contributions

The primary contributions of this work are:

- A novel application of reinforcement learning to assistive technology, demonstrating how RL can learn personalized information prioritization policies
- A comprehensive state representation that combines visual features, contextual information, and user history
- A multi-component reward function that balances explicit feedback, response time, contextual relevance, and information diversity
- An end-to-end system implementation integrating perception models, RL agent, and real-time audio feedback
- Empirical evaluation comparing RL-based prioritization against rule-based baselines

2 Background and Related Work

Assistive technologies that rely on cameras such as Seeing AI [1], Envision AI [2], and Be My Eyes [3] use computer vision to analyze surroundings and thus require users to accept their fixed prioritization rules. However, reinforcement learning techniques, especially Deep Q-Networks (DQN) [4], are now viewed as promising solutions for human-in-the-loop systems since they can learn from sparse feedback. Among the key innovations of DQN are experience replay [5] and target networks [4] which provide stable training. In this project, we will employ frozen pre-trained models as feature extractors while at the same time learning the decision policy, thus following a hybrid approach which is common in vision-based RL [6, 7].

3 System Design and Methodology

3.1 Architecture Overview

EchoSense employs a three-layer architecture:

Perception Layer (Frozen): The computer vision models that are pre-trained and already trained at the top of the pipeline to the data such as YOLOv8 [8] for object detection, CLIP [9] for scene embeddings, and EasyOCR [10] for text recognition provide the features of the camera frames as the frozen models that facilitate training by unfreezing them to be the very ones models in the stage of feature extraction.

Decision Layer (Trainable): A DQN agent gradually learns to instruct the information that the state representations encoded perceive as the most important to be presented to the user. The agent selects the items that have been detected for the user and learns the suitable policies through the interaction.

Interface Layer: A GUI crafted in PyQt6 that shows the incoming video feed from the camera and receives user input is accompanied by an audio output produced by a text-to-speech system.

3.2 State Space Design

The state representation combines three components:

Visual Features (526 dimensions): The feature set consists of object detection (class embeddings, normalized bounding boxes, and confidence scores), 512-dimensional CLIP scene embedding, and text attributes (presence, count, and confidence).

Context Features (10 dimensions): The context features include time of the day (sin/cos), day of the week (one-hot encoding), session length, and the last action performed.

User History Features (64 dimensions): User history features comprise the last 10 feedback scores, learned preference vector (size 10), average satisfaction, and frequency of interaction.

The overall state dimension is 600 after the normalization procedure for the purpose of numerical stability.

3.3 Action Space

The action space is discrete and consists of 50 actions each of which indicates a distinct method of information item prioritization. Actions are assigned to the various combinations of objects that have been detected, regions of text, and descriptions of the scene. Action masking is used to limit the selection of actions to only those that are valid according to currently available detections.

3.4 Reward Function

The reward function combines multiple signals:

$$R = \alpha \cdot r_{\text{explicit}} + \beta \cdot r_{\text{time}} + \gamma \cdot r_{\text{context}} + \delta \cdot r_{\text{diversity}} \tag{1}$$

where:

- $\alpha = 0.5$: Explicit user feedback (+1 for helpful, −1 for not helpful)
- $\beta = 0.2$: Response time reward, encouraging quick user acknowledgment
- $\gamma = 0.2$: Contextual relevance, rewarding contextually appropriate information
- $\delta = 0.1$: Information diversity, preventing repetitive announcements

Contextual relevance considers scene type, time of day, and item importance (e.g., navigation signs prioritized in navigation contexts, people/vehicles during daytime).

3.5 Training Strategy

The DQN agent uses:

- **Experience Replay:** Buffer size of 10,000 transitions, batch size of 32
- **Target Network:** Updated every 10 steps to stabilize Q-value estimates
- **Epsilon-Greedy Exploration:** Starts at $\epsilon = 1.0$, decays to 0.1 over 10,000 steps
- **Network Architecture:** Fully connected layers [256, 256, 128] with ReLU activations
- **Optimization:** Adam optimizer with learning rate 0.001, gradient clipping at 1.0

4 Implementation Details

The system was constructed primarily with Python 3.9 or later as the main programming language and PyTorch [11] for neural networks, YOLOv8 [8] for object detection, Transformers [12] for CLIP, EasyOCR [10] for text recognition, Gymnasium [13] for RL environments, PyQt6 [14] for GUI, and OpenCV [15] for image processing as the main application programming interface (API) and libraries.

Perception Manager: Controls the various perception models and features vector extraction while also handling the errors.

State Encoder: Produces 600-dimensional normalized state vectors from the outputs of perception, the context, and user history.

DQN Agent: Combines neural networks, experience replay, epsilon-greedy policy, and training with checkpointing.

Feedback Handler: Takes feedback, calculates rewards, and records user preference history.

Baseline System: Comparison through rule-based prioritization (text > people > vehicles > other).

The system has a frame processing rate of 2 FPS along with rate limiting and audio deduplication to avoid overload and repetitive announcements.

5 Project Outcomes and Current Limitations

The project, though it achieved the integration of reinforcement learning and computer vision in assistive technology, did not produce the results that were anticipated. The performance of the RL agent which was based on improvements over the baseline systems in the place of virtual evaluations, has not reached the level of efficiency that was hoped for. Learning and prioritization were inconsistent for the agent especially in situations where there was little user feedback and the context was complex.

On the other hand, the system's perception layer functions efficiently. The object detection facilities that are supported by YOLOv8, keep on identifying and localizing objects with a high level of accuracy. The system is capable of detecting a large number of different objects such as humans, cars, signs and various other items that people use daily. Likewise, the text recognition part (EasyOCR) accurately retrieves text from the images, and the scene comprehension based on CLIP gives valuable contextual embeddings. This is an indication that although the decision-making layer needs more polishing, the perception infrastructure that supports it is already robust and has the capability of being instrumental in the improvement of the future.

The discrepancy between the achievement of the perception layer and the RL agent's limitations at present has revealed the most important lesson that using pre-trained models as feature extractors is simple but deriving good policies from restricted and noisy feedback is still a big issue. This case has given deep understanding of the practical obstacles that come with the use of reinforcement learning in real life, where the theoretical advantages of adaptive learning must fight against the challenges of sparse data, limited computational resources, and high demand for reliable performance.

6 Reflection and Learning

6.1 Key Challenges Encountered

State Representation: One of the most difficult problems to solve in this project was to come up with a good state representation. The first idea of using raw pixels was ruled out due to dimensionality and therefore the second idea of using pre-trained models as feature extractors came in; the agent could then concentrate on decision policies.

Reward Function: The exclusive use of explicit feedback resulted in sparse rewards and learning speed reductions. The acceptance of implicit signals (response time, contextual relevance) demanded precise balancing. The ultimate multi-component function was found through repetitive testing.

Real-Time Performance: The combination of RL inference into the system put a demand on optimization. Among the solutions were limiting the rate (2 FPS), processing asynchronously, and caching. This indicated the significance of profiling in machine learning systems that are close to production.

Exploration vs Exploitation: It is quite difficult to manage the exploration factor in systems over human-in-the-loop. Epsilon-greedy with gradual decay ensured a good balance, however, adaptive strategies could be more effective.

6.2 Design Decisions and Trade-offs

Frozen vs Fine-tuned Models: I preferred to freeze the perception models instead of fine-tuning, thus trading off possible incremental advantages for quicker research and development. The models come with a lot of features already, and the main obstacle is setting the right order of priority.

Discrete vs Continuous Actions: I opted for discrete actions (50) for the reasons of keeping the system simple and stable. Continuous actions bring the benefit of finer control but discrete actions make Q-learning easier and fit the discrete nature of item selection.

State Dimension: Through a series of empirical tests, the 600-dimensional state was established as the right point between information richness and efficiency.

6.3 Mistakes and Lessons Learned

Error Handling: The first releases of the system crashed whenever models could not be loaded. This incident emphasized the need for the application of thorough error handling and also the permitting of systems to degrade.

State Normalization: The concatenation of raw features without normalization in the beginning led to the training process being unstable as the scales were different. Normalization has been confirmed to be indispensable for the training process to be stable.

Reward Function Complexity: An initially over-engineered reward function having a lot of components made the debugging process very difficult.

6.4 Learning Outcomes

This project has grown one's knowledge about: (1) **Reinforcement Learning:** DQN hands-on training, experience replay, target networks, and theoretical foundations (Bellman equations, Q-learning); (2) **State Representation:** Feature engineering in RL and iterative refinement are of critical importance; (3) **System Integration**:

Key resources included PyTorch [11] and Gymnasium [13] documentation, the DQN paper [4], YOLOv8 [8] and CLIP [9] repositories, and PyQt6 tutorials.

7 Conclusion

EchoSense proves a potential use case wherein reinforcement learning is applied to personalize information delivery in assistive technologies. Even though the system's performance during simulation evaluations is better than that of the rule-based systems, the RL agent's effectiveness has not yet reached the original target. The agent encounters problems with slow learning, unpredictable prioritization, and non-negligible real-world factors like sparse feedback and context complexity.

On the other hand, the project is able to show the successful operation of the perception layer. The YOLOv8-powered object detection capabilities never fail to spot and locate objects with good accuracy, capturing a great variety of items from humans to cars, signs, and a lot of other objects used in daily life. The same goes for the text recognition module (EasyOCR), which consistently extracts text from pictures and, at the same time, the CLIP-based scene understanding offers expressive contextual embeddings. This shows that while the decision-making layer still needs some polishing, the perception part is already strong enough and reliable to support further enhancement in the future.

This project provided invaluable experience in reinforcement learning, system integration, and the practical challenges of deploying ML systems. The challenges encountered—from state representation design

to real-time performance, and particularly the gap between theoretical promise and practical implementation—offered deep learning opportunities that extend beyond the specific application domain. The experience of integrating pre-trained models as feature extractors proved straightforward, while learning effective policies from limited and noisy feedback remains a significant challenge that requires further investigation.

Future work should focus on: (1) conducting real user studies with visually impaired participants to validate the approach with authentic feedback, (2) exploring more sophisticated state representations and reward functions to improve RL agent learning, (3) implementing online learning for continuous adaptation, and (4) refining the RL training process to achieve more reliable and consistent performance. Additionally, the robust perception layer could be leveraged independently or integrated with alternative decision-making approaches while RL techniques are further developed.

Acknowledgments

I would like to acknowledge the open-source communities that developed the tools and libraries used in this project, particularly the PyTorch, Ultralytics, and Hugging Face teams for their excellent frameworks and documentation.

References

[1] Microsoft Seeing AI. *Seeing AI - Talking Camera for the Blind*. https://www.microsoft.com/en-us/ai/seeing-ai

[2] Envision AI. *Envision AI - Visual Assistant*. https://www.letsenvision.com/

[3] Be My Eyes. *Be My Eyes - See the world together*. https://www.bemyeyes.com/

[4] Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533.

[5] Lin, L. J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4), 293-321.

[6] Zhuang, F., et al. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1), 43-76.

[7] Ngiam, J., et al. (2011). Multimodal deep learning. *Proceedings of the 28th international conference on machine learning*.

[8] Ultralytics. (2023). *YOLOv8 Documentation*. https://docs.ultralytics.com/

[9] Radford, A., et al. (2021). Learning transferable visual models from natural language supervision. *International Conference on Machine Learning*.

[10] Jaided AI. (2020). *EasyOCR*. https://github.com/JaidedAI/EasyOCR

[11] Paszke, A., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

[12] Wolf, T., et al. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*.

[13] Towers, M., et al. (2023). Gymnasium. https://gymnasium.farama.org/

[14] The Qt Company. (2023). *PyQt6 Documentation*. https://www.riverbankcomputing.com/static/Docs/PyQt6/

[15] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.