Pokemon Showdown Voice Controller: Extracting in-domain actions from speech-to-text systems

Aditya Pandey Nitish Mallick Savya Sachi Pandey Vivek Kumar

1. Introduction:

Pokemon Showdown[1] is a popular open source online multiplayer game simulating the world of the *Pokemon* saga, where players battle it out in a world of illustratively named Pokemon beasts (such as "Pikachu") with correspondingly illustrative attacks (such as "hyper beam"). The game pits two players against each other, each equipped with 6 unique pokemon, each pokemon having a set of 4 moves each. Each Pokemon has unique moves and stats, and each type of Pokemon has its own strengths and weaknesses.

Typically, the game is operated via a traditional system of on-screen buttons and clicks, unlike the fantasy anime world of *Pokemon* where "trainers" give verbal commands in a conversational manner for their companion Pokemon to follow (For example, Ash Ketchum, the shows protagonist, can be commonly heard yelling out "Pikachu, use thunderbolt"). To emulate this fantasy experience in the video game realm hasn't been attempted meaningfully, thus creating the impetus for this project. We've proposed and built a speech to text interface for the *Pokemon Showdown* game, that allows users to mimic the verbal commands used in the show, in order to play moves in the video game.

2. System Design:

The proposed solution to creating this interface today requires a system with the following functionalities to work effectively:

1. A listener and denoising layer, which implements a standard set of noise reduction functions for the interface, but importantly which also activates the system based on a keyword (ie. activating upon hearing "Pikachu", when the command "Pikachu, use thunderbolt" is given).
2. A speech to text layer, which implements a set of state of the art speech recognition models to convert user audio inputs to text. In an ideal case, a speech

[1] Game Link: https://pokemonshowdown.com/

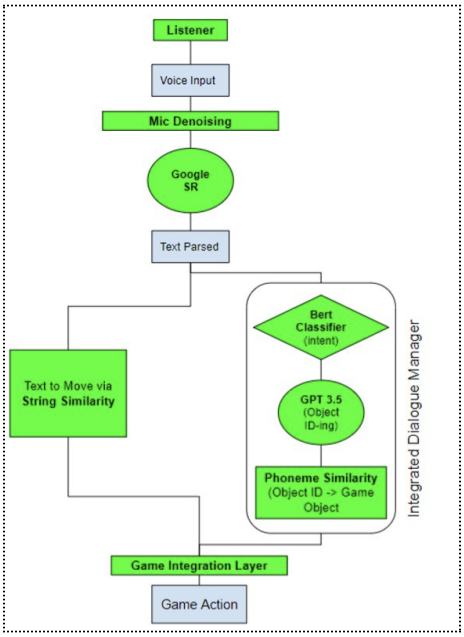
recognition model trained on a large set of samples of Pokemon models would be the best system for implementing the functionality of this project, but due to limited resources, standard pre-trained models were the ones considered.

3. A Dialogue Manager, which converts the text parsed from the audio into in game actions. For the scope of this project, the action space in *Pokemon Showdown* can be classified into two types of actions: An attack action, where the currently active Pokemon will make one of its attack moves, or a switch action, where the currently active Pokemon swaps out for another Pokemon in the user's party. Not only is identifying the action itself important, but it is also key to identify the specific "object" the action semantically refers to (Ex: "Pikachu use thunderbolt" refers to the move "thunderbolt" as the object, similarly "Pikachu come back, go out Geodude" refers to "Geodude" as the object).
4. A game integration layer, which finally takes the parsed "intent" ie. intended action of the user, along with the action object, and converts that into the requisite in game move/provides system feedback to incomplete moves.

Complementing this system design, a large variety of functions are required for a complete product, such as robust handling for user errors, a small visualizing interface, and web audio integration layers. These were not considered in the scope of this project thus far.

3. Solution Overview:

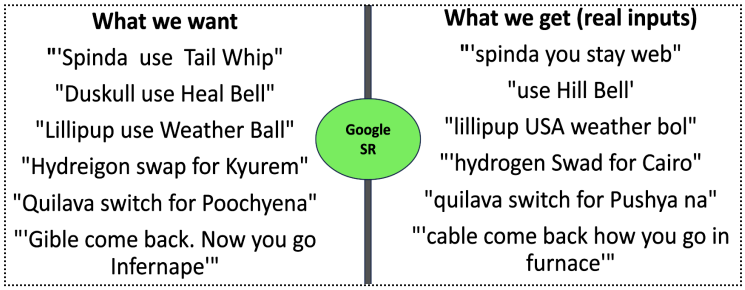
Over two versions of iteration, we built the following solution (as shown on the right).



3.1. The Listener and Speech Recognition: From the top down, this solution implements a listener with mic muting and ambient noise reduction, built on top of Google's Speech Recognition API. Broadly, this API implements a state of the art *Universal Speech Model*, trained extensively to parse audio from 100+ languages. It's architecture follows a standard encoder-decoder process, implementing the breakthrough "conformer" [1] ie. convolution augmented transformer, giving its recognition a combination of the global awareness that comes from transformers as well as robust understanding of local

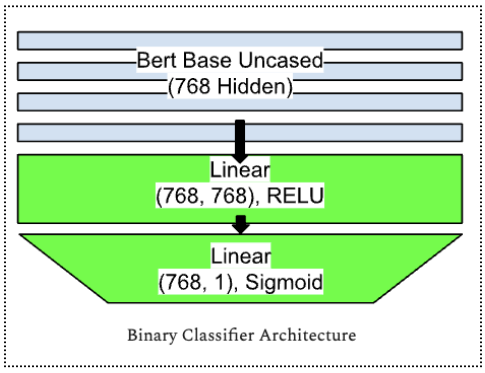
features well captured in CNNs. Other considerations for the speech recognition included Open AI's Whisper model, but due to its current inability to handle streamed audio data (it requires audio files), we decided to stick with Google's Speech Recognition. A future consideration of this work will be to integrate Whisper when it comes, as it does have a greater knowledge of niche words specific to the Pokemon domain.

3.2 The Base Case/Text to Move via String Similarity: The simplest version of implementing the voice controller involves parsing text from a speech to text model, and comparing that text with all possible in game moves via a string matching algorithm (in our case, using Sequence Matcher) to find the most overlapping in game state. We implemented this as a baseline to test our performance since, among many other things, it fails at one core problem: Speech Recognition Models are weak at parsing out exact text representations, especially when the words and phrases are niche and/ or outside of their training domains. Simple fixes like string similarity are thus highly likely to be wrong, and the problem requires somewhat intelligent systems to parse out both the action intent of the user as well as the in-domain words/phrases such as "Spinda" and "Tail Whip".



3.3 The Integrated Dialogue Manager: To overcome the challenge above, we designed a dialogue manager to handle two broad tasks: First, the detection of intent ("attack" or "switch"), and second, to convert the noisy text parsed from SR back into the relevant in game action ("you stay web" to "use Tail Whip"). Each of these sub problems had a broad space of solutions which could work for it, as discussed below:

3.3.1 The intent classifier: When it came to figuring out the user intent, we evaluated two viable options: First, prompting an LLM such as GPT 3.5 to identify what the users action is; Second, using a Binary Classifier built on top of the BERT base uncased language model, fine tuned on real voice-to-text data. In theory, just doing GPT prompting should've sufficed (and it did work fairly well), but it had a few



pitfalls. First, the noisy inputs given to GPT 3.5 led to a fair share of response failures despite three iterations of prompt tuning. Second, when it came to instances where it did parse out an output, its accuracy was still lower compared to the Binary Classifier itself, due to the latters fine tuning on 80 noisy voice to text samples collected from 4 different speakers, combined with 110 proper samples that we generated. Overall, the classifier achieved an accuracy of 96.315% on the entire dataset.

Trial	Samples	BERT Binary Classifier	GPT 3.5 for intent
1	40	.95	.825
2	40	1.0	.925
3	40	.95	.925

Accuracy Scores Comparison

3.3.2 Identifying Objects from Texts: After figuring out intents, the dialogue manager must then convert the text parsed into an in-game action by identifying precisely what that action object is from the text. This posed a challenge since input text contained many different "parts of speech" (relevant for our use case), and at the same time was highly noisy (see the table below). To overcome this, we implemented a two part system: One identifying the relevant portion of the text containing the noisy action object, and the other converting that noisy object to its in domain text analogue.

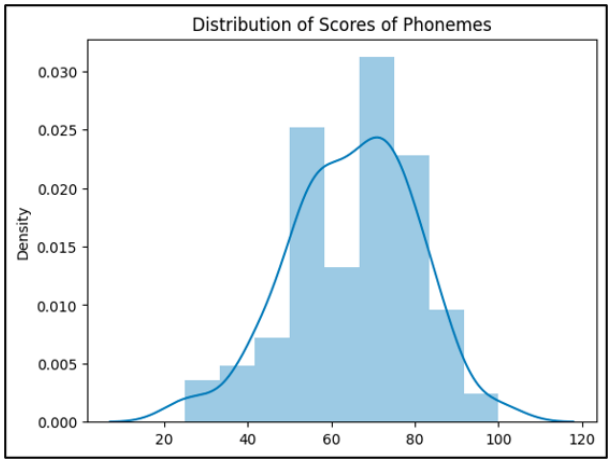
"Pikachu,	*destroy him with*	*solar beam"*
"Peek at you,	*Tolstoy him id*	*mortar bheem"*
Subject: Pikachu	**Action Intent:** Attack	**Action Object:** solar beam

3.3.2.1 Object Iding using GPT 3.5: To implement the core object identifier, we prompted gpt 3.5 to tag and return the relevant part of the objects, providing it three few-shot examples as a reference. We then tested this mechanism with an annotated dataset of our 80 voice-to-text prompts collected, prior to implementation, using the phonemic similarity discussed below. An alternative system we briefly considered was training our own POS tagging system/text identification system, but due to the high variability in outputs we observed from our Speech Recognition, we figured that would need a lot of good data.

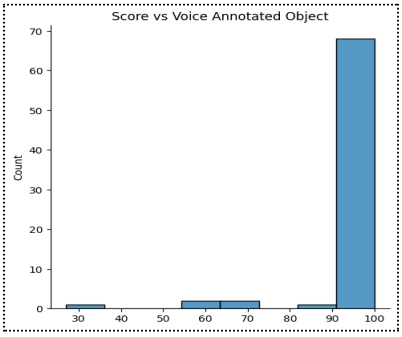
3.3.2.2 Converting Objects to In Game States via Phoneme Similarity: Now that we could isolate the relevant game object, we rested the lynchpin of our project on a key hypothesis:

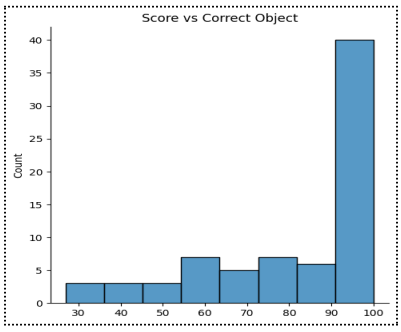
Speech Recognition Models capture the syllabic analogues of our domain specific words (also known as phonemes).

The impetus for this hypothesis came from the data itself, as we observed phrases that sounded similar to our domain specific ones when spoken out loud, being the ones that the SR model was parsing. We thus proceeded to encode both the identified action object phrase as well as all the in-game moves in their syllabic representations (Using the NYSIIS algorithm [2] developed decades ago), and observed very similar phonemic representations. We measured a score (from 0-100) using the Levenshtein distance between the syllabic representations, and then analytically found an optimum threshold around 50 for this similarity via studying the distance distributions between multiple phonemes of 25 Pokemon names. This phonemic similarity was also used in the listener's keyword detection, listening and dynamically scoring audio to match the active pokemon.



3.3.2.3 Evaluation of the Object IDing system: Via phoneme similarity scoring, we could observe the performance of both the hypothesis and the GPT prompt based identification. These results were hugely impressive: looking at the object text identification alone, GPT was successful at identifying the correct phrases compared to the annotations 90.1% of the time, with the average phoneme score being 96.82. Moreover, when it came to evaluating the phoneme scores of the identified objects versus their real in-game counterparts, 92.2% of the data scored above the Levenshtein score threshold, with an average syllabic similarity score of 86.07, thereby giving us a strong confidence in our hypothesis of syllabic encodings. This effectively meant that 9/10 times we could expect to capture exactly the right object from the voice.





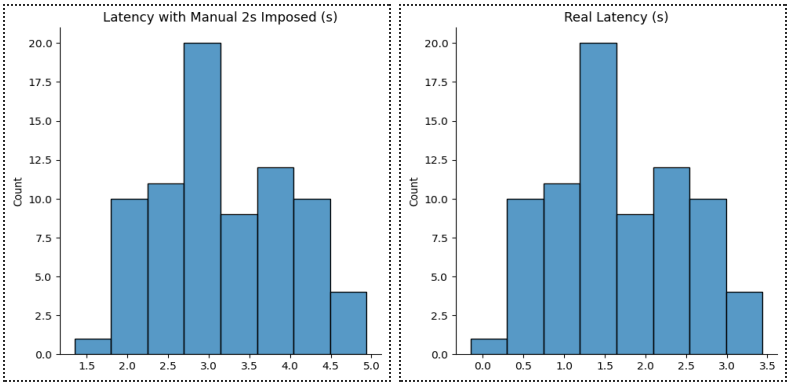
3.4 Game Integration: Having created the various subsystems, we could then integrate back to the game via the "poke-env" [3] open source APIs and packages. This allowed us to both receive and send information via requests to the game, allowing us to use the game's battle states in a dynamic fashion. We subsequently cloned the game to simulate a server locally, before making further evaluations.

4. Overall Performance Evaluation

Having iteratively built and evaluated our subsystems, it was time to put our system to the test. There were two real dimensions for us to test along: 1. Did the system outperform the string similarity baseline?, and 2. Was the system fast enough to work in deployment for a turn based video game?

4.1 Accuracy: We tested it on another set of 80 real voice-to-text generated data points, in comparison with the string similarity baseline, and observed its performance in a real game setting. What this meant was for each row of data, we simulated a real game state, complete with 6 randomized Pokemons and 4 randomized attack moves, with the correct object move or pokemon inserted in correspondingly. The baseline string similarity model had an accuracy of 70.13%[2] when it came to predicting the correct intent and move, while our integrated dialogue manager had an accuracy of 89.61%.

4.2 Latency: We measured the latency as the time difference taken for an action in the game: Premonitiously the largest component of said latency was going to be the Open AI rate limits, due to which we had artificially already throttled the GPT prompting to 2 seconds. Our measured latency was quite good and negligible to the user side, averaging around 3 seconds with the rate limiting imposed, and around 1.2 seconds without.



[2] Note: A Previous Presentation of this data stated an incorrectly lower accuracy value, as it used a flawed test case methodology that was skewed against string similarity.

5. Conclusions

After a rich two weeks spent on this project, we were able to successfully deliver a functioning speech-to-text interface for the pokemon showdown game, which for ~9/10 spoken commands would be successful on its first try (and for those where it isn't, it prompts the user to repeat what they said).

5.1 Key Areas of Improvement: Our implementation of the system was somewhat longer than need be, and could be optimized along various dimensions. First, to optimize for accuracy overall, an approach could have been to train a speech to text model directly, which would require a large amount of data gathering (and it's unclear whether it would perform significantly better than our current accuracy). At a subsystem level, most of our accuracy is impacted directly from the speech to text component, and a refinement of how speech is processed with either tuning/training or even swapping out Google SR for Whisper could create a big improvement. Second though, a subsystem impacting our accuracy as well as latency was the need to send requests to GPT 3.5: A feasible replacement for this would've been training a sequence tagging model with our domain specific parts of speech, with annotated data being required for this. Doing so could achieve comparable accuracy with much faster speeds. This would also have reduced the user complexity since it wouldn't require an Open AI API key.

5.2 Future Work: Given that we have a core functional system, productionizing this into a product for the millions of Pokemon players would require a few augmentations:

1. First, a user interface likely implemented via a chrome extension + audio would provide dynamic system feedback to the user, with information such as their Pokemon team's stats/attributes as well as audio feedback in the case of unidentified commands.
2. Second, using GPT/LLMs for complex prompting by providing context could allow users to make compound verbal commands which require semantic parsing (ex: "Pikachu, use your strongest move" or "Pikachu, come back. Send out the best pokemon I have to defeat the current enemy pokemon"). Note that these augmentations rest on a robust parsing of the speech to text.

5. Acknowledgements

We're sincerely grateful to Professor Monojit Choudhury and Kabir Ahuja at Plaksha for giving us the opportunity and guidance to build this project in his class.

Links

Github Repository containing our Code/Tests/Data Sets/models/etc:
https://github.com/AdityaPandey0901/PokemonShowdownVoice

Video of the Tool in Action:

https://youtu.be/u9tnhb0MlcI

Poke Env Library:

https://github.com/hsahovic/poke-env

Pokemon Showdown Server Code:

https://github.com/smogon/pokemon-showdown

Citations

[1] Gulati, Anmol, et al. “Conformer: Convolution-Augmented Transformer for Speech Recognition.” Interspeech 2020, Oct. 2020. Crossref, https://doi.org/10.21437/interspeech.2020-3015

[2] Robert L. Taft, Name Search Techniques, New York State Identification and Intelligence System, Special Report No. 1, Albany, New York, December 1970.

[3] Haris Sahovic, Poke Env Module Documentation, 2023

https://poke-env.readthedocs.io/en/latest/module_documentation.html