Evaluating the Impact of Pushing Voice-Driven Interaction Pipelines to the Edge

Smruthi Sridhar
Intelligent Platforms & Architecture Lab
University of Washington
Tacoma, Washington
smruthi@uw.edu

Matthew E. Tolentino
Intelligent Platforms & Architecture Lab
University of Washington
Tacoma, Washington
metolent@uw.edu

ABSTRACT

With the releases of Alexa Voice Services and Google Home, voice-driven interactive computing has quickly become commonplace. Voice interactive applications incorporate multiple components including complex speech recognition and translation algorithms, natural language understanding and generation capabilities, as well as custom compute functions commonly referred to as skills. Voice-driven interactive systems are composed of software pipelines using these components. These pipelines are typically resource intensive and must be executed quickly to maintain dialogue-consistent latency; consequently, voice interaction pipelines are usually computed in the cloud. However, for many cases, cloud connectivity may not be practical and thus require these voice interactive pipelines be executed at the edge.

In this paper, we evaluate the feasibility of pushing voice interaction pipelines to resource constrained edge devices. Driven by the goal of enabling voice-driven interfaces for first responders during emergencies when connectivity to the cloud is impractical, we characterize the end-to-end performance of a complete open source voice interaction pipeline for four different configurations ranging from entirely cloud-based to completely edge-based. We then identify and evaluate several optimizations, such as caching and customized acoustic models that enable voice-driven interaction pipelines to be fully executed at computationally-weak edge devices at lower response latencies than using high-performance cloud resources.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; **Real-time systems**; • **Hardware → Analysis and design of emerging devices and systems**;

KEYWORDS

Edge Computing, IoT, Systems

CF '18, May 8–10, 2018, Ischia, Italy

ACM ISBN 978-1-4503-5761-6/18/05…$15.00
https://doi.org/10.1145/3203217.3203242

ACM Reference Format:
Smruthi Sridhar and Matthew E. Tolentino. 2018. Evaluating the Impact of Pushing Voice-Driven Interaction Pipelines to the Edge. In *CF '18: CF '18: Computing Frontiers Conference, May 8–10, 2018, Ischia, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3203217.3203242

1 INTRODUCTION

Voice-driven interfaces have gained significant traction in recent years. The introduction of Alexa Voice Services from Amazon [1] and Google Home [3] has transformed devices at the edge of the network into dialogue-driven gateways to access data and services. While voice interactivity initially supported simple commands, these systems continue to grow in sophistication. However, the edge devices users interact with consist primarily of microphones, speakers, and a wireless network controller. Support or complex voice interactivity, which constitute the heart of these voice-driven devices, is provided via remote invocations across the network to cloud-based servers with sufficient computational capacity [14][15][11]. Given sufficient compute power and memory capacity, these pipelines can operate at latencies that approach traditional human-to-human dialogue.

Voice interaction pipelines are resource intensive. For every simple voice command, a software pipeline is invoked and executed. This pipeline starts by detecting a command and then translating the audio-based command to text [18][17][19]. As part of this translation, the audio signal is filtered for noise using compute-bound digital signal processing techniques. The filtered signal is then converted to text using a trained acoustic model that requires significant memory. Once translated to text, the pipeline decodes the query to determine which action, or skill, to invoke that executes in-memory or worse, relies on additional calls to remote servers across the network. Finally, to render verbal responses to the user, generated textual responses are converted back to an audio signal, using a second acoustic model, and rendered by the edge device.

Due to the computational resources required, most voice interaction pipelines are executed on the cloud [1][3][2][20][15][11]. While this alleviates the need for computational resources at edge devices, it also creates a hard dependency that prevents proper operation of voice-driven services or capabilities without a direct connection to the cloud. For use cases in which network connections are unreliable or unavailable, this precludes the use of voice-driven interfaces. As an example, there has been recent work with fire departments to integrate Internet of Things (IoT) sensing platforms, indoor positioning systems, and augmented reality displays into fire fighter gear to provide real-time information during emergency events. Because navigating touch-based displays while fighting fires

is impractical, voice-driven systems enable hands-free interactivity. However, given that cloud connectivity can not be guaranteed during these scenarios, current cloud-based voice interaction platforms are impractical.

The goal in this work is to evaluate the feasibility of pushing voice interactive pipelines to edge devices while maintaining response latencies consistent with cloud-hosted pipelines. We start by analyzing the performance characteristics of the different stages of typical voice interactive pipelines. Given that many voice-interactive platforms, such as Alexa Voice Services, are proprietary, these are difficult to instrument. Thus, we leveraged several widely used open source tools such as pocketsphinx [17] as well as components from the Mycroft platform [9] and assembled a complete, functional voice interaction pipeline. We then instrumented every stage of this pipeline from audio capture to response rendering and characterized the response latency.

In addition to comparing the performance of executing the pipeline solely on an edge device relative to the cloud, we compared the performance of leveraging a local fog server [13] [22] [10] [21]. The idea was to characterize the response latency using a local server with higher performance and greater resources than the edge, but at lower network latency than the cloud. We also explored techniques to minimize needed resources such as using alternative acoustic models, ranging from general models to tuned acoustic models without compromising translation accuracy. Finally, we investigated techniques to leverage caching in an effort to further reduce the computational load for use on edge devices.

Using our instrumented voice interaction pipeline, we performed an in-depth investigation into the viability of the pushing voice interaction pipelines to the edge. Based on these experiments, this paper makes the following contributions:

(1) We instrument and characterize the performance of an end-to-end voice interaction pipeline and compare the performance impact of three distinct configurations ranging from edge-only to cloud-based.
(2) The acoustic model matters. Using general acoustic models at the edge impacts response latency on even simple voice-driven requests. Additionally, using general acoustic models can significantly impact translation quality. In our experiments the default acoustic model only properly translated the requests properly less than 25% of the time. After generating new, tuned acoustic models, we increased the recognition rate from 33% to 87%.
(3) Through tuning, voice interaction pipelines can be executed on commodity edge devices at lower response latencies than using the cloud.

2 ARCHITECTURE OF VOICE INTERACTION PIPELINES

Voice interaction platforms provide a dialogue-based interface for responding to user-generated requests and queries. Responding to queries requires several specific stages as shown in figure 1. First, speech audio is captured by a microphone and preserved in memory or as a file. The recorded audio is then processed through a series of filters to minimize ambient noise. The filtered audio is then translated to text using a predefined acoustic model which can vary significantly based on the extent of the vocabulary, language, and parts of speech. Once the recorded speech is translated to text, the request is decoded and mapped to a specific skill. A skill consists of a function that completes the request and generates a response which then translated into natural language response. The natural language generated response is then translated to an audio response and transmitted to the user via speakers. The following sections describe the details of these pipeline stages.

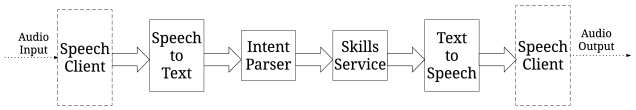


Figure 1: Architecture of a Voice Interaction Pipeline

2.1 Speech Client

The speech client constitutes the first stage of the pipeline and initiates the interaction process. As shown in figure 1, all requests and responses flow through the speech client. Using a microphone and a speaker, the speech client is responsible for constantly listening and detecting audio cues. The speech client consists of three components: a listener, a producer, and a consumer which use a shared queue to manage the processing of the speech input. The listener is activated on a *wake word* and records the spoken instructions based on phrase heuristics and noise levels. The producer pushes this recording into a processing queue and the consumer retrieves the audio data chunks off the queue and passes on to the next stage of the pipeline, the speech recognition service.

2.2 Speech to Text (STT)

This speech recognition service picks up the audio bytes sent by the speech client and converts the audio into text. The speech segments are interpreted as feature vectors and passed to a decoder. The decoder, composed of an acoustic model and a phonetic dictionary, work in conjunction to derive phones and identify utterances. The key to identifying utterances lies in a mapping process. A naive approach is to try all possible combination of words from a vocabulary. However, this puts a heavy load on the acoustic model for large vocabularies. This is undesirable as searching large acoustic models increases the overall response latency - a critical metric for dialogue-based interfaces. Hence it is crucial to use a tuned acoustic model that restricts the word search space and improves the mapping process. The accuracy of the speech recognition relies on an optimized acoustic model.

2.3 Intent Parser (Natural Language Understanding)

The rule-based natural language parser is an intent definition and determination framework. The text from the STT engine is sent to the parser to determine the appropriate intent. An entity is a named value that specifies a requirement against an intent. For instance, the command 'Get current weather in Seattle' refers to a weather intent which requires the entities' location and weather keyword. The

parser interprets the natural language text and translates this into a JSON-based data structure containing the intent, probability match and a list of entities. The intent determination engine uses this parsing information to generate tagged parses containing tagged entities. Each tagged entity is assigned a confidence score that represents a percentage match between the entity and the utterance. A valid parse is thus determined by a high confidence score without containing any overlapping entities.

2.4 Skills

Skills are sets of functions to be performed when users issue voice commands. Skill services abstract the underlying selection logic and trigger skills based on the identified intent. Skills are user-defined and can range from responding to a simple date/time request to directing complex robotic actions. Skills typically produce textual dialog that must be then rendered as verbal responses.

2.5 Text to Speech (TTS)

To complete an interaction, textual responses generated by the skills service must be converted to audio responses. The text to speech (TTS) client generates a synthesized audio response based on the skill response. As opposed to the STT decoder, the synthesizer is composed of a acoustic model as well as lexicon and voice definitions, including prosody models. The acoustic model within the text to speech stage is not to be confused with the one referred to in the speech recognition process. This model adapts a set of tokenization rules and textual analysis techniques to synthesize the audio output.

3 VOICE INTERACTION PIPELINE EXECUTION

The need for low-latency compute has led most voice interactive pipelines to be hosted in the cloud. This enables simple devices such as Amazon's Echo and Google Home to be built inexpensively with limited local compute capacity; however, these devices must be connected to the cloud to function. Unfortunately, this dependence on cloud-based compute resources limits the use of voice interactivity when network connectivity is unavailable. For example, during emergency events, first responders have limited access to the internet. For such cases, enabling these pipelines to be hosted and executed within local edge devices is necessary.

In this paper, we evaluate the impact of deploying voice interaction pipelines on three types of resources: 1) cloud, 2) edge devices, and 3) fog servers. These three deployment options constitute a wide range. At one extreme, we evaluate our voice interactive pipeline hosted within the cloud where all computation occurs on remote, network-accessible resources. At the other extreme, we evaluate the same pipeline hosted entirely on an inexpensive, compute-limited edge device, namely a Raspberry Pi. Finally, we also evaluate the impact of leveraging local servers proximate to our location, which we refer to as a Fog or Edge Server. This server has greater computational capacity than our edge device and fewer resources than the cloud, but at lower network latency than the cloud. Our goal is to identify the feasibility of executing these pipelines on computationally weaker devices without compromising dialogue-consistent response latency.

3.1 Cloud

The cloud configuration captures how most voice interactive systems,including Alexa Voice Servies and Google Home work today. This configuration executes the entire voice interactive pipeline we described earlier using cloud resources and constitutes the other extreme configuration relative to the edge. Consequently, when a user issues a command, all processing to decipher and respond to queries are processed in the cloud. The edge device in this case simply captures the audio command in the speech client, passes the audio to the cloud via REST APIs across the network, and renders the audio response generated by the text-to-speech client in the cloud. This configuration benefits from the use of high performance compute resources and high capacity memory resources, but suffers from unpredictable and potentially high network latencies.

3.2 Edge

The Edge configuration executes the entire voice interaction pipeline locally on the edge device. All pipeline stages are local to the device and communication between stages operate at memory latency given there is no need to invoke pipeline stages across the network. However, the edge configuration has limited computational power and memory capacity. While the edge configuration does not suffer from unpredictable network latencies, the limited compute and memory resources can be problematic when using generalized acoustic models or complex interactions that require significant processing.

3.3 Fog

For the Fog configuration, we evaluate the impact of moving the execution of the voice interaction pipeline *closer* in terms of network distance and hence latency. Fog servers are often used to localize computation that would otherwise be performed on a cloud. In our case, we assume we have a high-performance fog server on which we can execute the voice interactive pipeline. Consequently, for performance-constrained edge devices, the cost of leveraging fog servers could potentially outweigh the round trip delay. For this study, we offload the voice interactive pipeline to a server on the same local network.

Although not a cloud invocation, the components still depend on a remote call for executing pipeline stages. Similar to the cloud topology, this adds latency for communication. However, given the components are closer in proximity to the edge than cloud services, this can significantly reduce transmission time.

4 EVALUATION METHODOLOGY

In this section we describe the experiments we performed and the configurations we used to evaluate the impact of pushing voice interaction to the edge. The experiments were designed to compare the impact across different levels of interactions, ranging from those that require simple responses to others that involve multiple requests to generate an appropriate response. First, we developed and evaluated a prototype demonstrating voice interaction. We started by evaluating the total execution time for executing a full end-to-end voice interaction pipeline across the cloud, edge, and fog configurations. We then instrumented the end-to-end voice interaction pipeline to characterize the performance of each stage

for the three configurations. Even though overall response latency is critical, any improvement is inconsequential if the user must repeat the request several times. Consequently we also evaluated the quality of speech to text translation using alternative acoustic models.

4.1 Evaluation Systems

For all of our experiments, we used a Raspberry Pi3 coupled with a microphone and speaker as our edge device. The Raspberry Pi3 consists of a quad-core, 1.2Ghz ARM Cortex A53 CPU, 1GB DRAM, and 802.11n wireless network controller. To ensure consistency with our use case, we used the Wifi network interface for all of our experiments. Our goal was to characterize whether a resource-constrained edge device could adequately perform the operations of the voice interaction pipeline. For the edge configuration, we executed the entire voice interaction pipeline on the Raspberry Pi. For the other three configurations, the Raspberry Pi served as the end-point device users would interact with similar to the Amazon Echo. The difference between our evaluation of the configurations was where each of the components, or pipeline stages, were executed.

For the cloud configuration, we employed an AWS EC2 instance type t2.micro consisting of 1 GB memory coupled with a Intel Xeon processor that bursts up to a turbo frequency of 3.3Ghz. Based on our initial calibration experiments, the compute time required was notably reduced relative to hosting the complete interaction pipeline on the Raspberry Pi, even when the round trip delay across the network was included. For the fog server, we used a laptop that consisted of a 2.3Ghz Intel i5 CPU, 8GB of DRAM, and 802.11n wireless controller. Similar to the edge configuration, we used the Wifi network interface for all configurations that used the fog server. For the voice interaction pipeline, we leveraged open source components including PocketSphinx as well as several from the open source voice interaction platform, Mycroft [9]. Using open source components enabled us to instrument the entire pipeline to characterize the performance across all three configurations.

4.2 Pipeline Configuration

Another crucial aspect of the set up was to ensure that we executed the exact same pipeline components for every edge, cloud, and fog configuration. We found that may voice interactive pipelines, such as Mycroft, use a cloud-based service (e.g. Google) for the Speech to Text stage and local library for the Text to Speech stage. Unfortunately, using cloud-based services prevents an accurate characterization and analysis. Consequently, for our experiments we used the same components for every stage of our voice interactive pipeline to enable a reasonable comparison between the cloud, edge, and fog configurations. More specifically, we used the following components within our pipeline:

Speech To Text. We used PocketSphinx [17], a widely known open source project for speech recognition from CMU. PocketSphinx comes with a default language and acoustic model. For cloud and fog configurations, a simple REST endpoint was created to make PocketSphinx accessible as a service.

Intent Parser and Skills Engine. We used an open source intent parse library called Adapt [7] and Mycroft Skills Engine executed locally on the Raspberry Pi for all three configurations. This means this stage is not technically cloud-based or fog-based. However, during our initial experiments we observed that neither the Adapt component nor the Skills Engine consume significant time or resources given our queries.

Text to Speech. Although there are many open source text to speech options, we used a lightweight version of Text to Speech called Mimic [8] which is based on CMU's FLITE [12].

4.3 Performance Instrumentation

To characterize performance we instrumented each stage by adding time stamps on the entry and exit paths for every component to record the execution latency of each stage. During our initial experiments, we recorded the latency of the prototype pipeline without making any significant configuration changes. In other words, we used the default components within the speech pipeline as well as the default acoustic model. Listed below are key parameters captured during our experiments and used within our performance characterization and analysis:

Component Execution Time. This metric is used to measure the time taken to execute a component, or pipeline stage. This is recorded by calculating the difference in time stamps, before and after the execution of every component: STT, Intent Parser, Skill Engine and execution, and TTS.

Transmission Time. This is measured as the time taken to trigger each component through event emissions, in most cases across the network. This is recorded as the time delay before component invocation.

Round Trip Time. Round trip time is the total time taken by the system to record voice commands, process incoming audio requests, execute skills and generate an audio response. This is calculated by recording the time difference between the moment the user stops speaking and the moment the speech client starts the response play back. The round trip time does not include the audio recording and the audio play back, as the input and the output are dependant on the user query and its corresponding response.

Given these performance metrics, we ran experiments for each of the three configurations described earlier. These experiments consisted of issuing several voice commands to the system and then waiting for the relevant audio response. An experiment is considered successful only if we receive the correct, or relevant, audio response for every query triggered by the user. For each invocation, our instrumentation records the time stamps for each stage for analysis.

Table 1: Examples of different skill complexities

Skill	Complexity
What time is it	Small
What is my IP Address	Medium
What is the weather now	Large

For our initial experiments we used a common skill that could be executed across the different pipeline configurations. This ensured a fair performance measure and enabled us to evaluate how each of the four configurations handled the same level of skill complexity. Having said that, different queries require different levels of computation depending on the skill executed. Consequently, we also

ran experiments using skills of varying complexities to compare overall performance in terms of execution latency.

Table 1 shows examples of different skills categorized by complexity. We categorize "small" complexity skills as those that perform basic operations like fetching the current date and time; these do not have external dependencies on libraries or external, network-accessible services and are executed locally on the edge device. A "medium" level skill requires additional execution time relative to the small complexity skills, but do not perform compute-intensive operations. We classify "large" skills as those that require external service calls to perform the necessary task and may require more extensive memory references and compute resources. For instance, the weather skill relies on external services to capture current weather information.

Since the skill and intent parser are tightly coupled, we use the same intent parser for all configurations. Each experiment is carried out independently on each configuration. All results described are recorded as the average of five experiments for each configuration.

4.4 Quality of Speech Recognition

For voice interaction systems, speech recognition quality is critical to provide a relevant response. PocketSphinx comes with an extensive, generic language and acoustic model to interpret diverse types of user queries. Given our focus on use cases that field a bounded set of requests, using a generalized acoustic model with a large volume of vocabulary is unnecessary.

Our initial experiments used the default vocabularies and general acoustic models; however, we also experimented with reducing the scope to be domain specific with a limited vocabulary. To do so, we trained new acoustic models to be more adaptive to users' accents by recording condition and audio transmission channel. To quantify the relevancy of the results, we designed 15 exclusive phrases used by emergency responders and compared the speech recognition rate against both the models. The recognition rate is the number of accurate STT conversion against the the total number of recognitions.

4.5 Response Caching & Skill Complexity

Once the performance of each configuration was characterized, we experimented with response caching. To minimize latency, we wanted to determine if caching responses could be used for repeated requests. We experimented with caching the audio response from the text to speech module to enable reuse on subsequent queries. We populated a simple key value store with previous textual and audio responses. For every skill interaction, each textual response is compared against entries within the key value store before regenerating new audio responses.

For each skill, we compared execution latency for non-cached execution and cached execution. A non-cached pipeline execution constitutes the worst case latency for a skill as the voice interaction pipeline must be fully executed to render a audio response. A cached response on the other hand eliminates the execution of the TTS conversion and retrieves the response from a key value store to render a speech response. Finally, we also further evaluate the configurations based on skill complexities by comparing their performance over a diverse set of six different queries.

5 RESULTS

5.1 Initial Performance Characterization

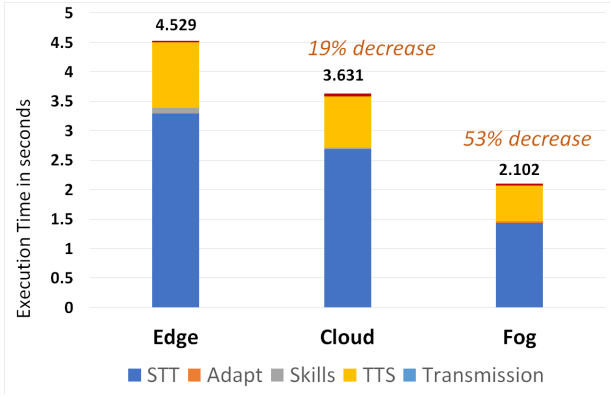
The results of our initial performance experiments are shown in Figure 2a. This chart compares the total execution time for each of the pipeline configurations including edge, cloud, and fog for a skill level of small complexity. The x-axis represents the configurations used in our evaluation and the y-axis describes the execution time in seconds for each configuration. The results are showcased as a bar chart representing the overall round trip time for each pipeline configuration and each stacked column for each configuration, shows the execution time of individual system components and their total transmission time.

The values highlighted above each column group indicate the absolute execution time taken by the system for that specific configuration. For instance, the edge configuration has an overall round trip time of 4.53 seconds. Since our objective is to evaluate the performance of the edge model, we also compared all other configurations against the edge and showed the speed-up or slow-down of the three other configurations relative to the edge configuration. We found that the execution time using the cloud configuration decreased by 19% compared to the edge configuration. This means that for this simple query, the overall response latency was lower using Cloud resources than executing locally. Surprisingly, the fog configuration realized improved performance, resulting in a 53% decrease in execution time compared to the edge configuration. To better understand where the bottlenecks were for each configuration, Figures 2b and 2c further break down the difference between the compute times and the transmission times respectively. For all configurations, the total compute time dominated the total response time, ranging from 98.6% for the cloud configuration to 95% for the fog configuration. These results show the best case scenario for hosting voice interaction pipelines on the cloud. Moreover, these results also show how the limited compute capabilities at the edge impact response latency when executed solely on resource-limited edge devices. As expected, the intermediate fog-based configurations did incur additional transmission delays and consequently suffered from network latencies.

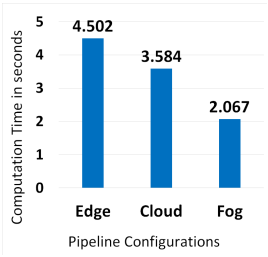
5.2 Quality of Speech Recognition

The results observed in the first performance characterization (figure 2a) constitute the best case scenario, when every single user request is accurately captured and processed by the system on the first request. In other words, the voice interaction pipeline was able to decipher our requests on the first try. Since we are evaluating a speech interaction pipeline for first responders, time is critical, but accuracy is also crucial to avoid the need for repeating requests. Accurately providing the correct response depends on many factors including hardware components, recording environment, user accents, and acoustic models. A misinterpreted request could cost time and can turn disastrous. Thus, an improper speech recognition could affect the overall performance of the system.

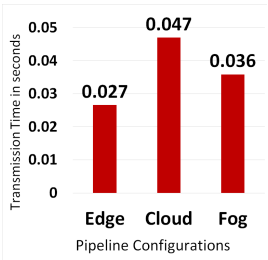
In order to assess the accuracy of the entire speech interaction pipeline at the edge, we evaluated the recognition rate of the speech to text component. We ran an experiment of 15 different phrases for five iterations. Table 2 lists the 15 phrases used as part of this



(a)



(b)



(c)

Figure 2: Performance Characterization of Pipeline Configurations: Edge, Cloud and Fog. (a) compares the total execution time for each configuration, (b) shows the total compute time for each configuration, and (c) shows the total communication time for each configuration. These results indicate the edge configuration has the lowest latency of the configurations, while the cloud configuration requires nearly double the latency.

experiment along with the results of the experiment. The column labelled "Old acoustic model" shows the number of successful phrase recognitions out of 5 attempts. For instance, the first phrase, "What time is it" was interpreted accurately 4 out of 5 attempts, whereas the third phrase, "What location is this" was interpreted accurately only 1 out of 5 attempts. Averaged across all requests, we observed a relatively poor recognition rate of only 33%. This means that on average, users would have to repeat the requests nearly three times, further increasing the latency by factor of three. In the case of our edge configuration this would further motivate the use of the cloud configuration.

Table 2: Speech Recognition Rate

Phrase Number	Phrase	Number of successful outcomes	
		Old acoustic model	New acoustic model
1	What time is it	4	5
2	Where am I	4	5
3	What location is this	1	4
4	What is my oxygen level	2	5
5	What is the pump pressure	2	4
6	What is the temperature	3	4
7	Call control room	2	5
8	Request ambulance	1	3
9	Report a casualty	0	4
10	Register an incident	3	4
11	Turn off the screen	1	4
12	Send distress call	0	4
13	Broadcast message	1	5
14	Locate fire hydrant	0	4
15	Share my location	1	5
Recognition Rate		**33%**	**87%**

Speech recognition primarily relies on the n-gram model which interprets speech by observing word sequences in a text corpus or a dictionary. However, for the given prototype, an exceptionally large vocabulary of words constitutes an additional computational overhead on resource-constrained edge devices. For the specific firefighter use case discussed earlier, such a corpus is not essential as the objective is not to build a general purpose intelligent personal assistant, but rather a device that serves domain-specific queries. Hence, we optimized the vocabulary to contain only a set of 20 phrases(15 phrases specific to the firefighter use case and 5 generic phrases) and retrained the acoustic model for better adaptation.

We reevaluated the recognition rate of the system for the new acoustic model for the same set of phrases. Column "New acoustic model" in table 2 lists the results of our reevaluation. The new model greatly reduced perplexity and we observed a significant improvement in the recognition rate from a 33% to 87%. This comparison is further detailed in figure 3 which illustrates the execution times taken to convert a speech instruction into textual representation by the speech-to-text component for a set of 15 phrases that were specific to the firefighter use case. The x-axis on the chart indicates the phrase numbers from the table 2 and the y-axis indicates the corresponding execution time in seconds for each phrase. Each phrase plots two columns that highlight the value of completion time using the generic old acoustic model and the optimized new acoustic model. The percentages shown above each set of columns quantify the performance improvement in using the modified acoustic model relative to the original acoustic model. Although we did not achieve 100% recognition accuracy, the optimized model reduces

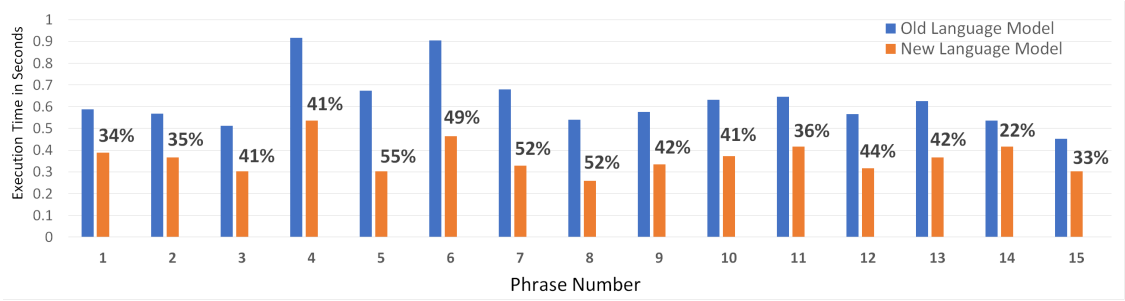


Figure 3: Speech Recognition Performance

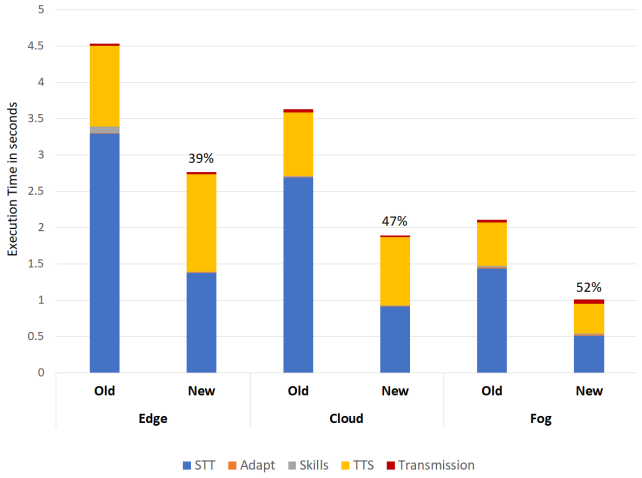


Figure 4: Performance Comparison of Acoustic model across Pipeline Configurations : Edge, Cloud, Fog

memory requirements and look-up time, further reducing latency for resource-constrained edge devices. Moreover, on average, we observed a 41% increase in accuracy for the same set of 15 different phrases.

5.3 Performance Characterization Revisited

After adapting the acoustic model, we re-evaluated the performance of the three configurations and repeated the same experiments that led to the performance results in Figure 2a. The new results summarized in Figure 4, compares the round trip time between the old and the new model for each of the three pipeline configurations. Execution time results are grouped based on the pipeline setup and each column in the group reflects the absolute execution time needed to execute the same skill. The stacked column (on the left) highlights the individual component execution times and transmission times for the old model. The right column highlights the same for the new model. The new acoustic model improved response latency for all 3 configurations with an average of 46% reduction in execution time compared to earlier experiments. The absolute execution time of the edge configuration was reduced from 4.5 seconds to approximately 2.7 seconds, nearly a 39% reduction. Similarly we observed a 47% decrease in the cloud configuration, which was reduced from 3.6 seconds to 1.9 seconds. The biggest improvement was observed using the fog configuration, where the computation time is reduced from an earlier result of 2.1 seconds to 1 second. The fog configuration definitely benefits from the fact that the computational capacity of the fog server is greater than the edge with lower network latency compared to the cloud.

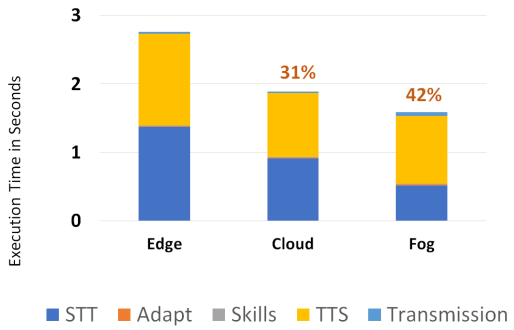


Figure 5: Overall Performance Characterization with Optimized acoustic model across Pipeline Configurations : Edge, Cloud, Fog

While the new acoustic model did improve performance, its also gives us an opportunity to further investigate each configuration performance based on the new model. Figure 5 shows the overall performance of the configurations based on the new acoustic model. Unlike our earlier results using the original acoustic model, the cloud configuration round trip latency is 31% lower than the edge configuration. Recall in our earlier initial experiments, the cloud configuration was only 19% lower than the edge configuration. This new improvement in performance is due to the integration of the optimized acoustic model, which reduces the compute and memory requirements. On the other hand, the fog model performed 42% better than the edge which is still consistent although it falls 10% short.

An important take away from the above observations is the reduction in compute time of the speech to text module as additional compute resources are utilized. The blue bars in figure 5 captures the execution time of the STT component, revealing the source of

the improvement. The compute time for different between the three configurations: 1.37 seconds for the edge, 0.9 seconds for the cloud, and 0.5 seconds for the fog. The fog configuration is 63% better than edge due to the increased compute capacity. This suggests that integrating acceleration hardware or edge-optimized speech to text libraries may prove beneficial.

5.4 Impact of Caching

As discussed in the earlier sections, the other biggest bottleneck in the pipeline was the text to speech conversion for the audio response. The results after employing caching strategies are showcased in figure 6, which compares the latency impact of response caching versus non-caching. More specifically, the chart shows the 3 configurations across the X-axis and the execution times in seconds on the y-axis. The yellow stack highlighted on the left columns(non-cached) exposes the execution time of the text to speech (TTS) module. When employing caching techniques, the compute time is nearly eliminated, having been reduced from 1.34 seconds to 0.004 seconds. For the edge configuration this is a 99.7% reduction. By minimizing the TTS time, this has a dramatic impact on the overall response latency, resulting in a 53% reduction in execution time. By using simple response caching techniques, the average execution time for all 3 configuration is reduced by 45%-53%.

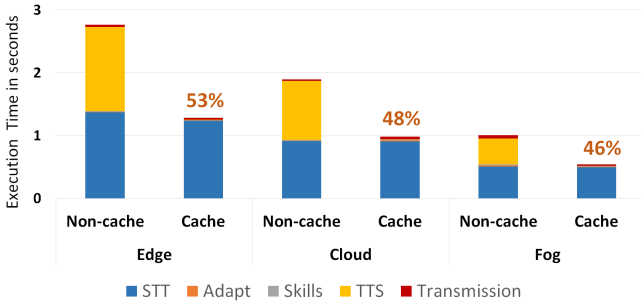


Figure 6: Performance Characterization of the Pipeline configurations based on Caching

During these caching experiments, we observed the performance improvements were consistent across all three configurations. The response latency was highest on the edge minimized using the fog. In fact, after integrating an optimized acoustic model as well as caching strategies, the fog configuration outperforms the cloud both in terms of computation and latency. As the edge and fog configurations constitute the book-ends of our performance characterization, going forward we focus the remainder of our comparisons between the fog and edge configurations.

5.5 Performance Characterization of Skill Complexity

We further evaluated the fog and edge configurations for varying skill complexities. Figure 7 shows the performance distribution of the edge and the fog configurations over six different skill complexities. The goal of this comparison is to understand how the configurations perform under a diverse set of queries. We start by evaluating the configurations for 6 different queries under two conditions: cached interaction and non-cached interaction. The values observed against each experiment is shown in terms of absolute execution time.

The first four queries are of small to medium complexity where the skill resides locally on the device. Hence it doesn't account for transmission time. The total execution time of these skills range is minimal, between 0.004 seconds to 0.009 seconds. The last two queries in the set signifies a complex skill that requires an external REST call to cloud resources, adding transmission time. The execution time ranged from 1.3 seconds for the weather skill and 0.5 seconds for the news skill. In these cases, the network latency of accessing remote resources for executing pipeline stages introduces additional overhead relative to executing simple skills locally. This is particularly important when the acoustic model is considered, which reduces the memory and compute overhead for look-ups. In this case, the fog configuration outperforms the edge irrespective of the complexity skill despite the additional transmission time for the large complexity skills. On average across these 6 skills, the fog configuration performed 38.72% better than the edge. Under a highly complex skill condition, the fog still outperformed the edge case by 36%.

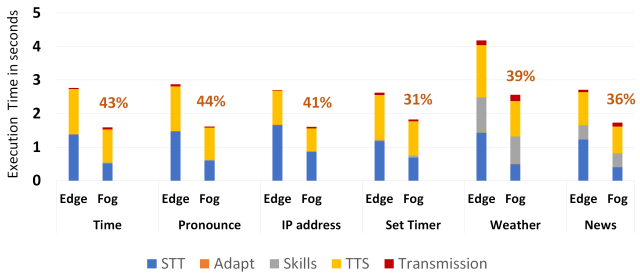


Figure 7: Performance Characterization across diverse query set - Non cached(Worst case)

Based on our observations over the six queries, caching consistently reduced response latency across the set of queries. On average, the fog configuration further decreased by 50%. While the large queries still consume extra time for the skill execution, the overall performance considerable declined by a minimum of 44% .

6 RELATED WORK

There has been considerable work related to Intelligent Personal Assistants. NASA's Jet Propulsion Laboratory has been working on an AI-powered solution called Audrey[6] that is designed to help local law enforcement, firefighters, and other first responders to augment situational awareness and personal safety. Audrey works in combination with other wearable sensors to collect information on temperature, GPS location and other surrounding environmental conditions. NASA claims that Audrey would be able to guide and provide a more personalized response based on predictive learning.

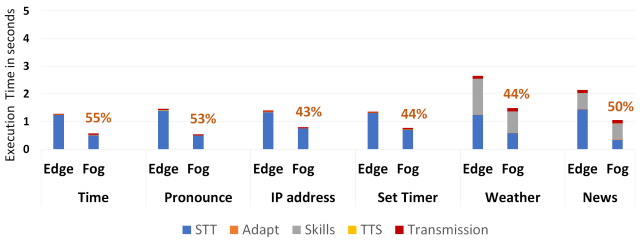


Figure 8: Performance Characterization across diverse query set - Cached(Best case)

In contrast to our objective, Audrey is a cloud-based model and is focused on deep learning and prediction strategies.

Lucida, previously known as Sirius, is an open-source end-to-end intelligent personal assistant (IPA) service from the University of Michigan [5]. Lucida integrates several well-established open-source platforms including for speech recognition, computer vision, natural language processing and a question-and-answer system. It was built to investigate the cost and performance implications of speech and visual recognition systems on large-scale server systems and identify server-based acceleration opportunities.

There are many emerging personal assistant platforms. Jasper is a personal assistant service that offers voice-interactive features [4]. The Deep Speech project leverages deep neural networks for speech recognition [16]. We leveraged part of the open-source Mycroft platform for some of our experiments as we found its architecture to be modular and simpler to decouple [9].

Unlike previous work which has focused on leveraging large-scale, server or cloud based resources for providing dialogue-quality voice interaction capabilities, our goal is to push these pipelines to resource-constrained edge devices. Additionally, this work focuses on characterizing the performance of full voice interaction pipelines and identifying optimization opportunities to enable, full interaction without requiring cloud connectivity.

7 CONCLUSION

In this paper, we identified an important use case where voice-driven applications can play a vital role if we can push the required pipeline execution onto edge devices. We also analyzed various pipeline configurations that can accommodate such a voice interaction system to function effectively based on skill complexities. Our results showed that resource-limited edge devices can perform well when aligned with the right skill complexity and an optimized acoustic model. Using an optimized acoustic model, we were able to push the full voice interaction pipeline onto a computationally-constrained edge device and execute with lower latency than using a compute-rich cloud configuration. We've also shown that embedding additional compute capabilities near the edge can further reduce latency relative to cloud configurations, for voice interactions with higher resources requirements.

Although these results show promise for pushing resource intensive pipelines to current edge computing devices, there are many opportunities to further improve edge computing. We've shown that by simply tuning the acoustic model we were able to compute at the edge faster than using cloud resources. However, for more complex pipelines, we intend to explore the use of specific acceleration functions within the edge such as including the use of FPGAs. Additionally, our results using fog-server configuration variants opens up the possibility of distributing complex pipeline stages amongst multiple nodes within a local environment. We plan to further explore such configurations in the future.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-#1742899.

REFERENCES

[1] [n. d.]. Alexa Voice Services. ([n. d.]). https://developer.amazon.com/alexa-voice-service
[2] [n. d.]. Apple Siri. ([n. d.]). http://www.apple.com/ios/siri
[3] [n. d.]. Google Home. ([n. d.]). https://madeby.google.com/home/
[4] [n. d.]. Jasper. ([n. d.]). http://jasperproject.github.io
[5] [n. d.]. Lucida AI. ([n. d.]). http://lucida.ai
[6] [n. d.]. NASA AI, Audrey. ([n. d.]). https://www.nasa.gov/feature/jpl/ai-could-be-a-firefighter-s-guardian-angel
[7] 2016. Adapt. (2016). http://adapt.mycroft.ai
[8] 2016. Mimic. (2016). http://mimic.mycroft.ai
[9] 2016. MyCroft Core. (2016). http://mycroft.ai
[10] Mohammad Aazam and Eui-Nam Huh. 2014. Fog computing and smart gateway based communication for cloud of things. In *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. IEEE, 464–470.
[11] Paramvir Bahl, Richard Y Han, Li Erran Li, and Mahadev Satyanarayanan. 2012. Advancing the state of mobile cloud computing. In *Proceedings of the third ACM workshop on Mobile cloud computing and services*. ACM, 21–28.
[12] Alan W Black and Kevin A K. A. Lenzo. 2001. Flite: a small fast run-time synthesis engine. *ISCA Tutorial and Research Workshop (ITRW)on Speech Synthesis, 2001* (2001).
[13] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 13–16.
[14] Yu-Shuo Chang, Shih-Hao Hung, Nick JC Wang, and Bor-Shen Lin. 2011. CSR: A Cloud-assisted speech recognition service for personal mobile device. In *Parallel Processing (ICPP), 2011 International Conference on*. IEEE, 305–314.
[15] Heidi Christensen, Iñigo Casanueva, Stuart Cunningham, Phil Green, and Thomas Hain. 2013. homeService: Voice-enabled assistive technology in the home using cloud-based automatic speech recognition. In *4th Workshop on Speech and Language Processing for Assistive Technologies*. 29–34.
[16] et.al. Dario Amodei. 2016. Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 173–182. http://proceedings.mlr.press/v48/amodei16.html
[17] David Huggins-Daines, Mohit Kumar, Arthur Chan, Alan W Black, Mosur Ravishankar, and Alex I. Rudnicky. 2006. Pocketsphinx: A free real-time continuous speech recognition system for hand-held devices. *Proceedings of IEEE International Conference on Acoustics Speech and Signal Processing, 2006* (2006).
[18] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. 2011. The Kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society.
[19] David Rybach, Stefan Hahn, Patrick Lehnen, David Nolden, Martin Sundermeyer, Zoltan Tüske, Siemon Wiesler, Ralf Schlüter, and Hermann Ney. 2011. Rasr-the rwth aachen university open source speech recognition toolkit. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop*.
[20] Johan Schalkwyk, Doug Beeferman, Françoise Beaufays, Bill Byrne, Ciprian Chelba, Mike Cohen, Maryam Kamvar, and Brian Strope. 2010. âĂIJYour Word is my CommandâĂİ: Google Search by Voice: A Case Study. In *Advances in Speech Recognition*. Springer, 61–90.
[21] Ivan Stojmenovic and Sheng Wen. 2014. The fog computing paradigm: Scenarios and security issues. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. IEEE, 1–8.
[22] Luis M Vaquero and Luis Rodero-Merino. 2014. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review* 44, 5 (2014), 27–32.