The Case for Node Multi-Versioning in Cognitive Cloud Services: Achieving Responsiveness and Accuracy at Datacenter Scale

Matthew Halpern[1], Todd Mummert[2], Miroslav Novak[2], Evelyn Duesterwald[2], and Vijay Janapa Reddi[1]

[1]The University of Texas at Austin, Austin, TX
[2]IBM T.J. Watson Research Center, Yorktown Heights, NY

Abstract

Cognitive cloud services seek to provide end-users with functionalities that have historically required human intellect to complete. End-users expect these services to be both responsive and accurate, which pose conflicting requirements for service providers. Today's cloud services deployment schemes follow a "one size fits all" scale-out strategy, where multiple instantiations of the same version of the service are used to scale-out and handle all end-users. Meanwhile, many cognitive services are of a statistical nature where deeper exploration yields more accurate results but also requires more processing time. Finding a single service configuration setting that satisfies the latency and accuracy requirements for the largest number of expected end-user requests can be a challenging task. As a result, cognitive cloud service providers are conservatively configured to maximize the number of end-user requests for which a satisfactory latency-accuracy trade-off can be achieved. Using a production-grade Automatic Speech Recognition cloud service as a representative example to study, we demonstrate the inefficiencies of this single version approach and propose a new service node multi-versioning deployment scheme for cognitive services instead. We present an oracle-based limit study where we show that service node multi-versioning can provide a 2.5X reduction in execution time together with a 24% improvement in accuracy over a traditional single version deployment scheme. We also discuss several design considerations to address when implementing service node multi-versioning.

1. Introduction

Lying at the intersection of machine learning, artificial intelligence, big data, and cloud computing, *Cognitive Cloud Services* can compute "human kinds of problems" on behalf of their end-users. Cognitive cloud services represent an emerging computing paradigm, known as Cognitive Computing, that aims to identify tasks that have historically required human intellect to complete and make them solvable through computation [1]. These tasks, which include natural language understanding, question answering, and image recognition, are applied in a variety of domains from healthcare diagnostics to intelligent personal assistants to business strategy consultation. Cognitive cloud services are manifestations of these cognitive capabilities service offerings in the cloud.

When cognitive capabilities are offered as cloud services, end-users build cognitive solutions on top of the cloud service's API. Performance expectations for cognitive services are twofold: to be both fast and accurate. Slow cloud services cannot provide adequate building blocks for cognitive solutions. In the example of Web search, Google observed that users perform fewer searches when their results are delayed by even a fraction of a second [2]. At the same time, cognitive cloud services must provide their end-users high-quality results. Execution speed is not enough as these services retrieve their results from probabilistic models and end-users flock to services that provide the highest quality results. Google became the dominant search engine in use today because of its emphasis on providing its users high-quality search results.

This paper quantifies how the latency-accuracy characteristics inherent to machine learning algorithms directly conflict with modern cloud service architectures. Many of the machine learning algorithms that underlie these services can be configured for different latency-accuracy trade-offs. When these algorithms are tuned to run for longer periods of time, they are more likely to produce more accurate results. Nevertheless, cloud services are usually deployed under a *"one size fits all"* service node deployment model. To scale out to a large number of users, a single version of the service, with it specific latency-accuracy characteristics, is instantiated across a multitude of server nodes. However, it is questionable whether handling all requests in a similar fashion with a single configuration version is in the best interest of the user (and the service provider). Requests for large user bases can be diverse, and machine learning techniques are not capable of being equally accurate for every possible data input they receive.

We characterize and optimize a production-grade Automatic Speech Recognition (ASR) service as a representative cognitive cloud service. ASR, the conversion of speech to text, has become an increasingly popular human-computer interfacing mechanism. For example, ASR is commonly used by intelligent personal assistants and question answering services.

Our ASR engine relies on an approximate, heuristic-driven search of a previously trained machine learning model to compute its results. Performing an exhaustive search across its entire data model is prohibitively expensive, so instead the search is performed iteratively and localized to a subset of the model based on heuristic policies invoked at each iteration. The statistical and probabilistic techniques used in our ASR engine are commonly used in many other cognitive computing engines, such as natural language understanding and image classification. Therefore, the insights from our ASR service also apply more broadly to other cognitive domains.

Our results suggest that cognitive cloud services are not

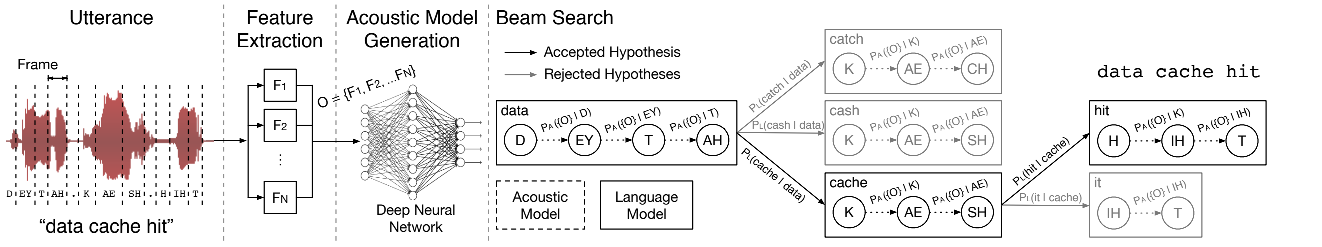


Fig. 1: Our ASR processing pipeline. The utterance (i.e. segment of human speech) is broken up into frames. Features (i.e. signal metrics) are extracted from each frame to be used to in a graph-based probabilistic model of the spoken language. An approximate, heuristic-driven search is performed on this model to quickly hypothesize what words have been spoken.

amenable to the conventional "one size fits all" cloud service node deployment approach. Digging deeper into the individual latency-accuracy characteristics of 35,000 different service requests for our ASR service, we find that 21% of them are sensitive to the ASR's dynamic search configuration. These sensitive requests exhibit dramatic latency and accuracy characteristics: the configuration either results in excessive processing time or does not provide suitable levels of accuracy. The most responsive search configuration has 1.9X more error than the most accurate configuration while the most accurate configuration takes five times longer to process requests. Additionally, no one configuration provides the most accurate results for all user requests because of idiosyncracies associated with the nature of the search's heuristic policies.

To mitigate the contention between latency and accuracy, we propose service node multi-versioning. Service node multi-versioning simultaneously deploys a variety of service versions each configured for different latency-accuracy trade-offs. In the presence of an oracle request routing scheme, service node multi-versioning can provide both a 2.5X latency and 25% error reduction over the most accurate single service version. Service node multi-versioning introduces new service architecture design considerations, which we discuss.

2. Automatic Speech Recognition (ASR)

This section presents the automatic speech recognition (ASR) engine we study throughout the remainder of the paper. ASR is the process of converting human speech into readable text in real-time. As with most cognitive computing workloads, our ASR engine employs sophisticated probabilistic models and computationally intensive machine learning algorithms.

Our ASR engine architecture is shown in Fig. 1. In its simplest form, ASR is graph-based search problem. Given an utterance (i.e. human speech sample), the ASR engine tries to correctly recognize the sequence of spoken words in a probabilistic graph-based representation of human speech (i.e. hidden markov model). This graph, which incorporates word pronunciation and semantic models (i.e. acoustic and language models, respectively), is iteratively searched by breaking down the utterance into a sequence of frames. However, searching the entire graph is prohibitively expensive, so an approximate heuristic search (i.e. beam search) is used instead. We now explain the relevant components in more detail:

Feature Extraction An utterance passed as an input into the ASR service is processed as a sequence of regularly segmented intervals of speech, known as $frames$. Typically, frames are intentionally sized to about 100ms to correspond to individual phonemes, the distinct units of sounds a speaker can make within a language. A set of $o$bservables, or a feature vector based on acoustic properties $\{O\} = \{F_1, F_2, ..., F_N\}$, is calculated for each frame. Using the observables, instead of the audio's raw digital signal, allows the frame to be represented in a compact and robust manner within the probabilistic models used throughout the remainder of the ASR engine.

Acoustic Model The acoustic model provides the probability that a particular frame corresponds to a specific phoneme. Given the frame's set of observables $\{O\}$, the acoustic model provides the probabilities that each phoneme could have produced that particular set of observables, $P(\{O\}|Phoneme)$. This also includes determining whether a phoneme is present in the frame because an utterance may include moments of silence as the speaker can pause between words. Furthermore, each phoneme is pronounced slightly differently due to coarticulation caused by surrounding phonemes (acoustic context), which the model must also take into account. This produces several thousands of different cases that require a deep neural network (DNN), trained on hours of transcribed audio, to affordably track. A deep neural network (DNN) generates the acoustic model for a particular utterance based on viewing the set of observables for all of the frames that form the utterance.

Language Model The language model provides the probabilities of occurrence for the words of a given language. Most language models are based on the n-gram model. The n-gram model provides the probability that a given word occurs after a specific sequence of $n-1$ words. Therefore, the language model captures the context in which a particular word is used. The language model is constructed a priori by looking at a large corpus of speech transcripts, books, and other texts.

Hidden Markov Model The hidden markov model (HMM) combines the acoustic and language models together to form a graph-based representation of human speech. A HMM is a graph-based representation of a random process that cannot be viewed directly, but only through observables. In the case of ASR, we want to know what words (i.e. sequences of sub-phonemic units) were spoken in the utterance, but we only have access to observations (i.e. extracted features) from the utterance's audio signal that corresponds to the words spoken. Therefore, the HMM is a network where two nodes corresponding to two sub-phonemic units are connected by a directed edge if one can follow the other with some probability, which is the edge's weight. A path in the HMM corresponds to a sequence of spoken words whose joint probability can be calculated by traversing edge weights.

Beam Search An exhaustive search of the entire HMM network is prohibitively expensive, both regarding computational complexity and memory requirements, to fully perform. Instead, a beam search only searches a subset of the HMM network, essentially acting as a restricted breadth-first search. This breadth-first expansion is responsible for generating multiple hypotheses to explore. Ultimately, the search will select a final result from these candidate hypotheses. However, the hypotheses change throughout the search. Within any given search iteration, only a subset of the possible hypotheses will be explored. The beam search employs various heuristics to prune the size of HMM network that gets searched.

The beam search heuristics, which dictate the size of the HMM network subset searched, directly impact the ASR service's latency and accuracy. The search's accuracy is directly proportional to how much of the HMM network is searched whereas the search time is inversely proportional to it. Only searching a subset of the HMM network means that the search results are no longer *admissible*. This is because the search *approximates* its exhaustive counterpart because the entire HMM network has not been searched. In other words, the search produces a *locally* optimal result based on the HMM network subset searched. While a larger search space makes it more likely the globally optimal result will be found, it comes at the expense of increased search time because there are more possible paths to explore. Therefore, there is a latency-accuracy trade-off amongst beam search's heuristic configurations.

3. The Latency-Accuracy Trade-off

In this section, we examine whether the typical "one size fits all" cloud service node deployment model can adequately address the inherent latency-accuracy trade-off of our ASR engine, and of cognitive cloud services more generally.

The latency-accuracy trade-off in our ASR service is most evident in the heuristic-driven beam search. Depending on the values of the search heuristics, different subsets of the ASR engine's HMM network will be searched. The accuracy and latency are respectively directly and inversely proportional to the size of HMM network subset searched because a larger search space is more likely to identify the correct result, but at the expense of a longer search time. We have observed that in a production setting configuration, the beam search can consume *over 50% of the end-to-end client response time*. As we will demonstrate, these heuristic configurations exhibit a significant latency-accuracy trade-off space in and of itself.

Our characterization focuses on how the beam search heuristics affect the latency and accuracy of the ASR transcription. Surprisingly, we find that a larger search size does not necessarily improve the ASR accuracy in many cases; it can even degrade the result quality. Our analysis demonstrates the limitations of the "one size fits all" service version deployment model. Over 79% of the utterances achieve the same accuracy regardless of what heuristic search configuration is used. The remaining utterances also do not benefit from a single deployment version. Assuming we chose the configuration that incurs minimal average latency across these remaining utterances, we would obtain a 1.9X increase in error over the configuration that achieves the highest average accuracy. However, the most accurate configuration does so at the expense of a 5X longer average processing time.

3.1. Optimization Metrics

The requirements of a successful cognitive cloud service are twofold: to provide the most accurate results as quickly as possible. In the context of our ASR service, this means that we are aiming for a minimal word-error rate (maximal accuracy) while minimizing the real-time factor (minimal latency). For this study, we will focus on evaluating the latency-accuracy trade-off for varying configurations of the ASR beam search heuristics. Analysis of other error and latency sources are beyond the scope of this paper.

Word Error Rate (WER) The word error rate (WER) is a measure of how accurately the utterance was transcribed into text. A WER of zero indicates a perfect transcription. Since the ASR service relies on probabilistic models and heuristic calculations to produce a result, the goal is to provide the lowest WER possible even when a perfect result is not possible.

For a given utterance, u, $WER(u)$ is determined by a word-by-word comparison between what the ASR service hypothesizes (i.e., $Hyp(u)$) and its reference transcript (i.e., $Ref(u)$). Therefore, the WER is the ratio of the number word errors to the number of words in the utterance's reference transcript:

$$WER(u) = \frac{\mid WordErrors(Hyp(u), Ref(u)) \mid}{\mid Ref(u) \mid}$$

Word errors can be classified as either insertions (i.e., adding a missing word), deletions (i.e. removing an extraneous word), and substitutions (i.e. replacing an incorrect), For example, if the utterance "the data caches" gets identified as "data cash is", there are three word errors: one deletion (remove "the"), one substitution (replace "cash" with "caches"), and one insertion (add "is"). Therefore, the WER is one because there are three word errors and three words in the reference transcript. Note that the WER can exceed one when either unnecessary words are added or important words are omitted in the hypothesis.

Real-time Factor (RTF) The real-time factor (RTF) measures ASR processing time. ASR engines aim to be real-time, providing the transcription as the speaker utters words. Therefore, the real-time factor for an utterance u is the ratio of the time to process the utterance to the time it takes to play back the utterance as audio:

$$RTF(u) = \frac{ProcessingTime(u)}{AudioPlaybackTime(u)}$$

Intuitively, the real-time condition is met when RTF is exactly one. However, it is desirable to reduce the RTF to a value below one to minimize resource utilization.

Sources of Error: Model vs. Search Amongst the many sources of error in our ASR engine, our analysis strictly focuses on the error produced by the beam search under the different heuristic configurations we explore.

There are two main sources of error in our ASR engine: the *model error* and the *beam search error*. Model error is the result of how the HMM network is constructed, and more specifically, of how its underlying acoustic and language models have been trained. Each model has been trained on thousands of hours of audio; the amount of error from training varies across utterances, but it will remain the same across the different beam search heuristic configuration experiments. Therefore, the model error will be constant across experiments.

Identifying the exact division of model and search error is a non-trivial task, so we approximate them instead. The insight behind our approximation is twofold. First, the model error will be constant across different search configurations because the underlying probabilities within the HMM network do not change; only what parts of it are. Second, given a comprehensive set of search heuristic configurations, it is likely that the minimum search error in the set is very close to (or even identical with) the minimum search error overall.

Therefore, we approximate the model error for an individual utterance u using the minimum WER we observe across the set of the heuristic configurations C:

$$ModelError(u) \approx \min_{c \in C} WER_c(u)$$

This allows us to isolate the impact a given beam search configuration c has on transcribing an utterance u as:

$$SearchError_c(u) \approx \frac{WER_c(u) - ModelError(u)}{ModelError(u)}$$

$SearchError_c$ provides a suitable metric to compare the effects of different heuristic search configurations. Given an utterance u, $SearchError_c(u)$ isolates the extent to which the ASR accuracy is degraded relative to the best accuracy (i.e., minimal WER) among all the configurations we study.

3.2. Search Error Characterization

We find that the great majority of utterances do not incur any search error regardless of heuristic configurations used. However, the utterances that do incur search error are very sensitive to the heuristic configurations used. This is notable because if the search error did not vary amongst the different heuristic configurations, the tuning problem would be trivial: simply the most responsive configuration should be used. Our analysis spans over 35,000 utterances from the VoxForge open-source speech transcript repository [3].

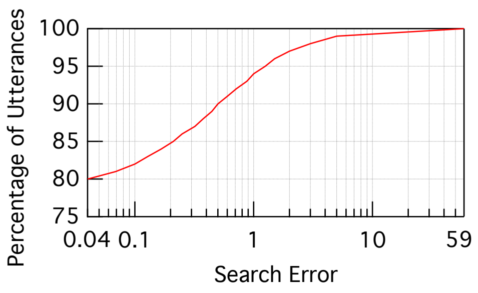


Fig. 2: Distribution of search error ranges. Only 20% of the utterences exhibit search error, but are very sensitive to the beam search heuristic configurations used.

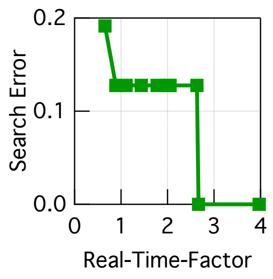
We study these utterances across ten diverse heuristic combinations. These ten heuristic configurations were chosen based on a best-fit latency-accuracy curve produced through an exhaustive heuristic value sweep (i.e., grid search) of six heuristics. These heuristics are the cross product of two orthogonal considerations: (1) hypothesis pruning policies (i.e., beam and max) and (2) the scope they applied (i.e., local, global, and network). The beam pruning policy only allows the top N most probably hypotheses to be searched and the max policy restricts the maximum size of a hypothesis search. Because multiple hypotheses are being explored at once, these policies are applied at different scopes spanning a single hypothesis (i.e., local), a branch of hypothesis (i.e., global), and the entire subset of the HMM current being searched (i.e. network).

Fig. 2 shows the distribution of worst-case search error observed across the utterances. The resulting curve is a long-tailed distribution where the outstanding majority (i.e., 79%) of the utterances have a worse-case search error of zero. The WER of these utterances does not change regardless which of the different beam search configurations are used. However, the remaining 21% can incur significant search error depending on the configuration used. Over 6% of the utterances can experience dramatic increases in WER. Depending on the configuration the WER can increase by over 100%. In the worst cases, WER increases by 5900%.

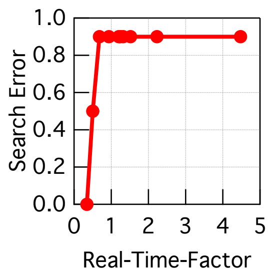
3.3. Latency-Accuracy Trade-off Analysis

We take a closer look at the 21% of the utterances that incur search error depending on the search heuristics (configuration) used. Throughout the remainder of this section, we refer to these utterances as the *search-sensitive* utterances.

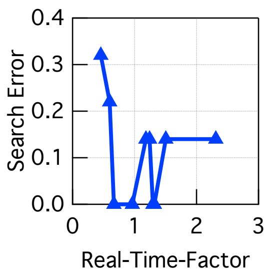
We systematically classify search-sensitive utterances into three categories based on their latency-accuracy behavior. We use these categories to motivate our multi-versioning deployment approach as we will show that each category calls for a different ASR engine configuration. Representative utterance latency-accuracy curves from each category are shown in Fig. 3. We now discuss each category and its implications:



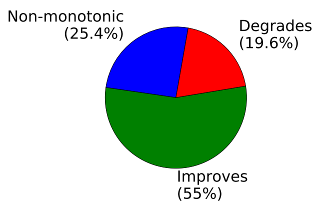
(a) Monotonic improvement example.



(b) Montonic degradation example.



(c) Non-montonic behavior example.



(d) Categorical breakdown.

Fig. 3: Latency-accuracy behaviors of the ASR engine across different dynamic search configurations. These behaviors fall into three distinct categories depending on how their accuracy varies with increased search time. While the majority of the utterances do not change depending on the configuration, a significant portion of utterances are sensitive to processing time – often conflicting with one another.

Accuracy Improves Fig. 3a shows that some utterances benefit from increasing the HMM search space (i.e, increased latency). This follows the intuition that a more rigorous search can yield more accurate results. The search error plot in Fig. 3a reaches its minimum WER at the ninth configuration, which reduces search error by 36%. However, reducing the search error is not free because it increases latency four-fold.

Accuracy Degrades Surprisingly, Fig. 3b shows that for some utterances a larger search space can actually increase the search error. This may occur when the search produces a locally, rather than globally, optimal result. In contrast to the previous category, where the search space was constrained, this category shows that the wrong hypotheses can also be selected when the search is too liberally configured.

Accuracy Non-monotonic Sometimes utterances do not exhibit clear latency-accuracy characteristics, as shown in Fig. 3c. The HMM subset searched for these utterances is sensitive to the search heuristics used. The search error fluctuates across configurations without a clear trend. In cases, where the WER reaches 0, the heuristics happened to constrain the search space in such as way that the globally optimal value could be found.

Fig. 3d shows a breakdown of the search-sensitive utterances into the three latency-accuracy behavior classifications previously introduced. A more comprehensive search improves the accuracy for more than half of the search-sensitive utterances. Although fewer in number, there is a sizable portion of search-sensitive utterances, whose accuracy degrades with execution time. While not shown, the utterances with non-monotonic behavior typically achieve lower search error with larger, as opposed to smaller, searches, further endorsing increased service processing time.

3.4. "One Size Fits All" Analysis

Our "one size fits all" analysis, shown in Fig. 4, demonstrates the limitations of deploying a single configuration to handle all utterances. For each of the ten configurations, we show the search error and RTF averaged across the search-sensitive utterances. There is an inverse relationship between search error and RTF. Left-to-right in Fig. 4 the average search error is reduced from 70% in configuration one to 24.7% in configuration ten. However, the accuracy improves at the expense of a fivefold increase in the average RTF.

Fig. 4 shows that achieving a low search error for all utterances requires a configuration with a relatively high RTF. However, based on our earlier classification, there exists a subclass of the utterances, whose search error will not improve with larger RTF. In fact, we established that there exist a subclass of utterances for which the accuracy even degrades with larger RTF. As a result, choosing a single configuration to process all utterances, will not only incur unnecessary latencies in some cases, it may even sacrifice accuracy.

4. Service Node Multi-Versioning

In this section, we introduce the concept of service node multi-versioning and demonstrate how it overcomes the limitations of the "one size fits all" deployment scheme. The idea is simple: deploy a set of versions each configured differently, tailored to specific characteristics of utterances. Our analysis shows that very few versions are needed to provide substantial accuracy and processing efficiency improvements. However, a multi-versioned cloud introduces new complexities, including version selection and resource allocation.

Evaluation Fig. 5 demonstrates the potential of service node multi-versioning to simultaneously improve the responsiveness and accuracy of the ASR service over deploying a single version. In practice, cognitive service providers typi-

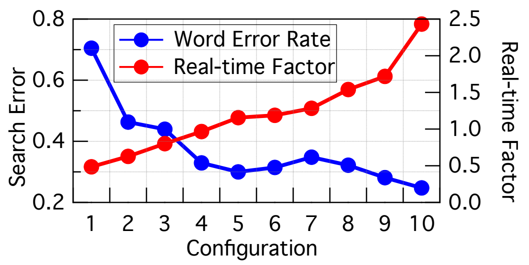


Fig. 4: WER and RTF trends across different ASR engine dynamic search configurations. Overall, increasing the beam search time results in lower search error. The fastest configuration significantly outperforms the most accurate one and vice-versa in the case of accuracy.

cally prioritize minimizing the search error before minimizing the RTF. We assess the potential of multi-versioning using an oracle routing scheme that assigns each utterance to the most optimal service node version. Under this oracle routing scheme, multi-versioning yields a significant reduction in the average search error while continuing to improve the RTF as more versions are simultaneously used. Adding just one additional version in deployment (for a total of two versions) reduces the search error from 24% to 3.4% while improving the average RTF by over 40%. Considering four versions reduces the WER to less than a percent and the RTF to below 1. By this point, the benefits of multi-versioning have been achieved as both the search error and RTF have saturated.

Design Considerations To realize the demonstrated benefits of a multi-versioned ASR service several design considerations have to be addressed. Based on our encouraging results so far, we plan to explore some of the following design aspects in our future work, which we now describe:

1. **Version Selection** The benefits of service node multi-versioning are predicated on determining an effective set of distinct versions that are to be deployed together. As we have shown, it is important to thoroughly explore the search's latency-accuracy space and match it to the characteristics and requirements of the service's user base.
2. **Request Routing** How to establish the routing logic to determine the best version to handle a given utterance is not immediately evident. Attributes of the utterances have to be extracted and incorporated efficiently into a routing mechanism that can identify the best configuration to process the utterance based on the specified optimization criteria.
3. **Resource Allocation** A specialized routing mechanism that routes utterances to version solely based on utterance attributes may result in load imbalance. As we saw earlier, there is a non-uniform distribution of which version was appropriate for the utterances. The cluster manager needs to be able to keep an adequate number of nodes for each service version available, and elastically adapt this number to dynamic loads. Additionally, in the presence of heterogeneous hardware (i.e. asymmetric CPUs, GPUs, FPGAs), the scheduler may assign the different versions to the different hardware components.

5. Prior Work

Our work builds upon computer systems research at the intersection of machine learning and datacenter optimization, while also bearing similarities to approximate computing:

Approximate Computing While our work leverage the latency-accuracy trade-off of our ASR engine, it bears several differences from approximate computing. Approximate computing seeks to sacrifice result quality to improve system efficiency. Instead, we take a two-level optimization approach where we *f*irst identify the configurations that minimize accuracy and then choose the most responsive configuration because we want the ASR transcription to be as correct as

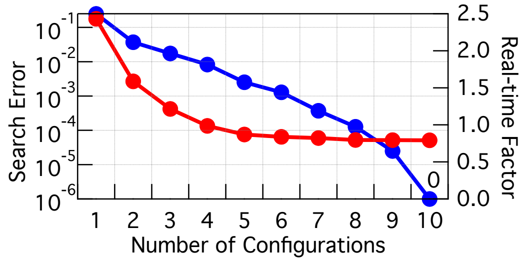


Fig. 5: WER and RTF trends as multiple ASR versions are used simultaneously (under oracle routing). Increasing the number of unique ASR versions reduces both the WER and RTF. Even considering just a few configurations provides a substantial improvement over what a single version can do.

possible. Green is the most similar to our work. It uses multi-version functions associated with computations that provide diminishing returns on accuracy for computation time [4]. However, we do not modify the program's source code and increase our optimization scope to be datacenter scale.

Datacenter and Machine-learning Optimizations A large body of work focuses on optimizing warehouse-scale computers [5], many of which focus on optimizing machine-learning based applications [7]. These techniques specifically focus on performance and energy efficiency optimizations based on program execution characteristics, rather than exploiting the latency-accuracy trade-off as we do. Other works look at resource allocation and cluster management [6], but not from an accuracy perspective as we require. All of these techniques are orthogonal to, and could be enhanced by, the techniques and insights we convey.

6. Conclusion

As Cognitive Computing continues to pervade daily life, it is important to rethink how we design the computer systems they employ. Optimizing accuracy without compromising responsiveness, and vice-versa, is essential to the success of this emerging computating paradigm. By understanding the latency-accuracy characteristics of our ASR engine, we demonstrate promising results for how breaking conventional cloud service deployment strategies can meet this end.

References

[1] Cognitive Computing.

[2] The Google Gospel of Speed.

[3] VoxForge.

[4] Woongki Baek and Trishul M Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. In *ACM Sigplan Notices*, 2010.

[5] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 2013.

[6] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *ASPLOS*, 2014.

[7] Johann Hauswald, Michael A Laurenzano, Yunqi Zhang, Cheng Li, Austin Rovinski, Arjun Khurana, Ronald G Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, et al. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In *ASPLOS*, 2015.