UNFOLD: A Memory-Efficient Speech Recognizer Using On-The-Fly WFST Composition

Reza Yazdani, Jose-Maria Arnau, Antonio González
Universitat Politécnica de Catalunya, Barcelona, Spain
{ryazdani,jarnau,antonio}@ac.upc.edu

ABSTRACT

Accurate, real-time Automatic Speech Recognition (ASR) requires huge memory storage and computational power. The main bottleneck in state-of-the-art ASR systems is the Viterbi search on a Weighted Finite State Transducer (WFST). The WFST is a graph-based model created by composing an Acoustic Model (AM) and a Language Model (LM) offline. Offline composition simplifies the implementation of a speech recognizer as only one WFST has to be searched. However, the size of the composed WFST is huge, typically larger than a Gigabyte, resulting in a large memory footprint and memory bandwidth requirements.

In this paper, we take a completely different approach and propose a hardware accelerator for speech recognition that composes the AM and LM graphs on-the-fly. In our ASR system, the fully-composed WFST is never generated in main memory. On the contrary, only the subset required for decoding each input speech fragment is dynamically generated from the AM and LM models. In addition to the direct benefits of this on-the-fly composition, the resulting approach is more amenable to further reduction in storage requirements through compression techniques.

The resulting accelerator, called UNFOLD, performs the decoding in real-time using the compressed AM and LM models, and reduces the size of the datasets from more than one Gigabyte to less than 40 Megabytes, which can be very important in small form factor mobile and wearable devices.

Besides, UNFOLD improves energy-efficiency by orders of magnitude with respect to CPUs and GPUs. Compared to a state-of-the-art Viterbi search accelerators, the proposed ASR system outperforms by providing 31x reduction in memory footprint and 28% energy savings on average.

CCS CONCEPTS

• **Computer systems organization → Special purpose systems**;
• **Computing methodologies → Speech recognition**;

KEYWORDS

Automatic-Speech-Recognition (ASR), Viterbi Search, Hardware Accelerator, Memory-Efficient, On-the-fly Composition, WFST

MICRO-50, October 14–18, 2017, Cambridge, MA, USA

ACM ISBN 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3124542

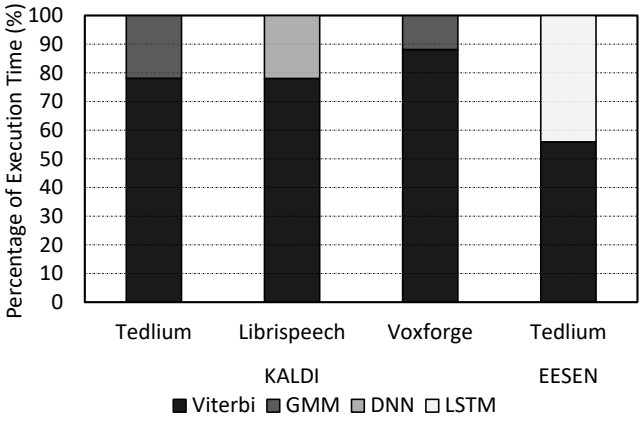


Figure 1: Percentage of execution time for different components of Kaldi and EESEN speech decoders. Measured on NVIDIA Tegra X1 mobile SoC.

ACM Reference format:
Reza Yazdani, Jose-Maria Arnau, Antonio González. 2017. UNFOLD: A Memory-Efficient Speech Recognizer Using On-The-Fly WFST Composition. In *Proceedings of The 50th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, MA, USA, October 14–18, 2017 (MICRO-50),* 13 pages.
https://doi.org/10.1145/3123939.3124542

1 INTRODUCTION

Automatic Speech Recognition (ASR) systems have recently achieved human parity in accuracy for conversational speech recognition [33]. However, such a high degree of accuracy comes at the expense of large storage requirements and computational cost. The state-of-the-art approach for ASR systems consists of employing a Gaussian Mixture Model (GMM) or a Deep Neural Network (DNN) for generating acoustic likelihoods from the input speech signal. These acoustic scores are then used to perform a Viterbi search [25] on a large Weighted Finite State Transducer (WFST) to find the most likely sequence of words. Figure 1 shows the percentage of execution time that these stages take in Kaldi [21], a popular ASR toolkit, for both GMM- and DNN-based WFST decoders. As it can be seen, the Viterbi search is by far the main bottleneck, taking more than 78% of execution time in Kaldi in all the evaluated models. Moreover, considering other frameworks such as Sphinx [31], it has been shown recently that by oprimizing the GMM in software [27], the Viterbi search remains as the main bottleneck.

Another emerging approach for ASR is the so-called end-to-end solution with a Recurrent Neural Network (RNN), typically using

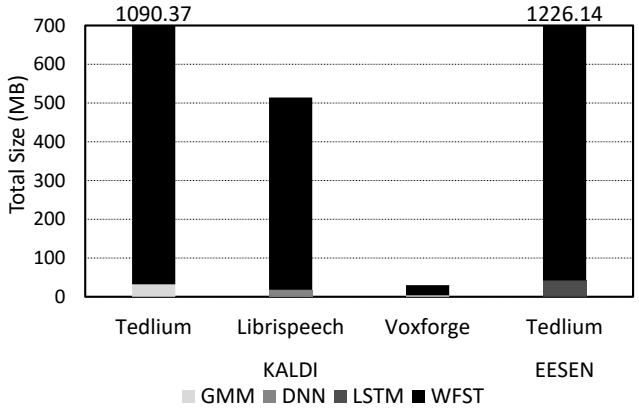


Figure 2: Sizes of the different datasets employed for ASR. The WFST is by far the largest component.

a Long Short Term Memory (LSTM) network [10]. This approach combines an RNN with a Viterbi search on a Language Model WFST to achieve an accuracy comparable to the DNN-WFST systems [9, 18]. Figure 1 also includes the execution time breakdown for EESEN [18], an ASR system based on the end-to-end approach. Even for an RNN-based speech decoder, the Viterbi search represents the main bottleneck taking more than 55% of execution time. Therefore, the Viterbi search is the most critical component for a broad type of state-of-the-art speech recognition decoders.

In recent years, several accelerators for the Viterbi search on WFSTs have been proposed [15, 24, 34]. Although these accelerators provide high performance and low energy consumption for ASR, they suffer from an important issue that severely constrains their applicability: the huge size of the WFST. Figure 2 shows the size of the different datasets (GMM, DNN, RNN, WFST) for Kaldi and EESEN decoders. The WFST is clearly the largest component, requiring between 87% and 97% of the total memory for ASR, which represents more than one Gigabyte for large vocabulary systems. WFST compression [23] ameliorates the problem, but still several hundreds of Megabytes are required for the WFST.

To deal with the excessive memory footprint of large vocabulary ASR, we analyze how the WFST is constructed during training. The WFST is created from two knowledge sources: Acoustic Model (AM) and Language Model (LM). Both AM and LM are represented as WFSTs, and then WFST composition is used to generate a unified graph-based recognition network [19]. Offline composition simplifies the decoding stage, as only one WFST has to be searched. However, it requires a multiplicative combination of states and arcs in the AM and LM graphs, resulting in the large sizes reported in Figure 2.

In this paper, we propose UNFOLD, a hardware accelerator that dynamically composes the AM and LM graphs during the decoding, avoiding the explosion in WFST size produced by offline composition. On-the-fly composition [1, 2, 13] reduces memory storage requirements by a large extent since the total size of the AM and LM graphs is significantly smaller than the size of the composed WFST. To further reduce memory footprint, we show that these individual WFSTs can be compressed very efficiently, in a more effective manner than the composed WFST. The combined use of on-the-fly composition and compression reduces the size of the datasets by 31x on average for several ASR systems. As a result, UNFOLD performs the decoding by using less than 40 Mbytes of storage instead of more than one Gigabyte.

The drawback of on-the-fly composition is the increase in activity during the decoding, since WFST composition is moved from the training to the decoding stage. To achieve real-time performance, UNFOLD includes simple yet efficient hardware support for composition operations. On the other hand, the large reduction in memory footprint results in smaller miss ratios in the caches of the accelerator, which significantly reduces the memory bandwidth usage, and is highly beneficial for performance and energy. In other words, on-the-fly composition increases the amount of computations and on-chip memory accesses, but it drastically reduces the number of expensive off-chip main memory accesses. Since external DRAM accesses require orders of magnitude more energy than local on-chip operations [12, 14], UNFOLD achieves not only a large reduction in memory footprint, but also 28% energy savings on average with respect to the state-of-the-art in Viterbi search acceleration [34], while achieving real-time performance by a large margin (155x).

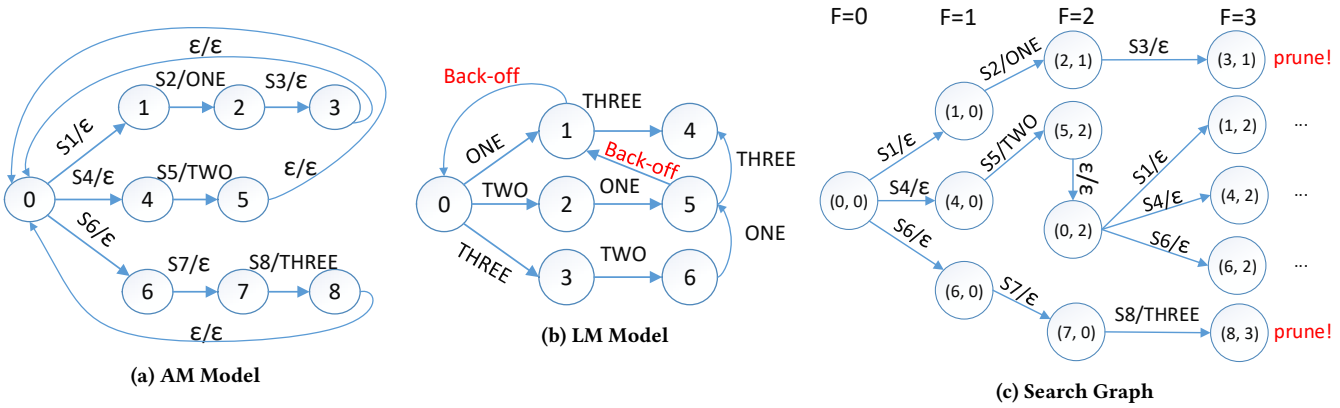
To summarize, this paper focuses on ultra-low memory footprint, energy-efficient, real-time speech recognition. Its main contributions are the following:

- We propose a hardware accelerator that exploits on-the-fly WFST composition to dramatically reduce memory storage requirements. To the best of our knowledge, this is the first system implementing on-the-fly composition in hardware.
- To improve the efficiency of on-the-fly composition, we introduce a preemptive pruning technique which reduces the number of hypotheses to be analyzed by 22.5% and improves performance by 16.3%.
- We propose compression techniques for the individual AM and LM WFSTs that achieve compression ratios between 23x and 34x for several ASR systems.
- Overall, UNFOLD is 155x faster than real-time, while reducing energy consumption by 28% on average with respect to the state-of-the-art Viterbi accelerator. In addition, the proposed accelerator has a small area of 21.5 mm 2 and dissipates less than 0.57 Watts.

The rest of the paper is organized as follows. Section 2 provides background on ASR with WFSTs. Section 3 presents the UNFOLD's architecture, and in particular the hardware support for on-the-fly composition. Section 4 describes our evaluation methodology and Section 5 analyzes the experimental results. Section 6 reviews some related work and, finally, Section 7 sums up the main conclusions.

2 WFSTS FOR SPEECH RECOGNITION

The goal of ASR is to identify a sequence of words from speech waveforms. State-of-the-art ASR decoders split the input audio signal into frames of, typically, 10 milliseconds of speech. Next, each frame is represented through a feature vector using signal processing techniques. These feature vectors are then fed to a GMM or a DNN in order to obtain the acoustic scores, i.e. the probabilities of the different phonemes in the language, for every frame of speech. Finally, the acoustic scores are used to perform a Viterbi search [25]



(a) AM Model

(b) LM Model

(c) Search Graph

Figure 3: An example of on-the-fly WFST components: (a) An AM which detects 3 words: ONE, TWO and THREE; (b) A 3-gram LM of the 3 words and (c) A partially dynamic-composed search graph

on a WFST to find the most likely sequence of words. This Viterbi search is the main bottleneck for ASR, as shown in Figure 1, due to the high cost of searching on a large-size WFST graph (see Figure 2).

A WFST is a Mealy finite state machine that encodes a mapping between input and output labels, and associates a different weight to each possible mapping. In other words, it consists of a set of states and a set of arcs, where each arc includes an input label, an output label, and a weight. The knowledge sources required for ASR, i.e. the AM and the LM, can be efficiently encoded as a WFST. Figure 3a shows an example of a simple AM graph for a three-word vocabulary. The AM represents the pronunciation of the different words. By taking the likelihoods generated by the GMM or DNN associated with the different input phonemes, AM outputs different sequences of words, a.k.a. the hypotheses that best match the identified phonemes and their respective probability. The input label of an arc is the phoneme ID (Sx in Figure 3a), whereas the output label contains the word ID. Epsilon output label means there is no word ending in that arc, whereas epsilon input label means that the arc can be traversed without consuming any acoustic score. Arcs with non-epsilon output label, i.e. arcs associated with a word ID, are also known as cross-word transitions.

Figure 3b shows an example of a simple LM. The LM WFST rescores the different hypotheses generated by the AM, taking into account probabilities of alternative sequences of words according to a given grammar. The input and output label of an arc in the LM contain the same word ID, whereas the weight is the likelihood of the corresponding unigram, bigram or trigram. Each state is associated with a given word history. State 0 is the initial state with empty history, the arcs departing from state 0 represent the unigram likelihoods. States 1, 2 and 3 have a history of one word, arcs departing from these states encode the bigram likelihoods. Finally, states 4, 5 and 6 have a history of two words, arcs departing from these sates represent the trigram likelihoods. For example, being at state 6 means that the last two input words were THREE-TWO, if next word is ONE then we transition to state 5 to update the history, applying the trigram probability of Prob(ONE | THREE,TWO) to rescore this hypothesis.

For large vocabulary systems, it is unfeasible to include arcs for all the possible combinations of two and three words. Combinations whose likelihood is smaller than a threshold are pruned to keep the size of the LM manageable. Back-off arcs are included in the states of the WFST to be used in case a combination of words that was pruned in the grammar is observed in the input hypothesis. If a combination of words is not available, then the WFST uses these extra arcs to back off from trigram to bigram, or from bigram to unigram.

To sum up, the AM uses the acoustic scores from GMM/DNN to identify words and their corresponding probabilities given the audio signal, i.e. hypotheses, whereas the LM rescores these hypotheses based on unigram, bigram and trigram likelihoods. This process can be performed in one step if the AM and LM models are composed offline [19]. Offline composition generates a unified WFST that simplifies the decoding stage, as only one graph has to be traversed, and it represents the common approach in software- [21] and hardware-based [15, 24, 34] ASR. However, it requires a multiplicative combination of states and arcs in the AM and LM graphs, resulting in large WFST models. Table 1 shows the sizes of these graphs for several ASR decoders. As it can be seen, the AM and LM models have sizes smaller than 102 Megabytes, whereas the composed WFST exceeds one Gigabyte for some decoders.

On-the-fly composition [1, 13] represents an alternative implementation that performs the Viterbi search by dynamically composing the AM and LM, avoiding the generation of the fully-composed WFST. Figure 3c illustrates how the Viterbi search does the decoding with the on-the-fly scheme. Each node in the search graph, a.k.a. token, is associated with a state in the AM graph and a state in the LM. For instance, token (2, 1) indicates that we are at state 2 of AM and state 1 of LM.

The search starts at the initial state in both WFSTs, i.e. token (0, 0), and proceeds as follows. In the first frame, the arcs of state 0 in AM model are traversed to create new tokens for the next frame of speech. The likelihoods of these new tokens are computed by using the likelihood of the source token and the acoustic scores of the phonemes associated with the arcs (S1, S4, S6), provided by GMM/DNN. Since these AM arcs do not correspond to cross-word

Table 1: Size of the individual AM and LM graphs and the fully-composed WFST in Megabytes.

Task	AM WFST	LM WFST	Composed WFST
Kaldi-TEDLIUM	33	66	1090
Kalid-Librispeech	40	59	496
Kalid-Voxforge	2.8	2.3	37
EESEN-TEDLIUM	34	102	1226

transitions, i.e. they have epsilon output label, no arc is traversed in the LM and the new tokens remain in state 0 in the second WFST. The same process is repeated in the next frame, but in this case the search traverses AM arcs with non-epsilon output labels, i.e. words ONE and TWO are recognized in different hypotheses.

When a cross-word transition is traversed in the AM model, its word ID is used for transition in the LM graph. The weight of the LM arc is also employed to compute the likelihood of the new token, in order to modulate the score of the hypothesis according to the grammar. Therefore, the AM graph drives the search, whereas the LM graph is used to rescore the hypotheses when an arc with a word ID, i.e. a cross-word transition, is traversed in the AM graph. Due to the large search space, pruning of the search graph is also applied to discard unlikely hypotheses.

With the on-the-fly mechanism, there is no need to store every information of the fully-composed WFST while decoding the speech. The WFSTs of the AM and LM are composed on demand as required by the Viterbi search. Note that the search graph of Figure 3c represents the different alternative interpretations of the speech generated by the Viterbi search. The search graph is required for both offline and on-the-fly composition, and its memory footprint is fairly small compared to the size of the WFSTs. Therefore, the storage requirements are reduced by an order of magnitude (see Table 1), as only the individual AM and LM are stored in main memory.

A main drawback of on-the-fly composition is that it increases the complexity of the decoder since it needs to fetch states and arcs from two WFSTs. Furthermore, it increases the amount of computations, since the LM likelihoods are applied at runtime, whereas they are merged offline with AM information in the fully-composed approach. Due to the efficient support provided by UNFOLD, these issues have a negligible impact on the performance and energy of the speech recognizer.

The other important drawback is the cost of fetching the arcs in the LM graph. The arcs for a given state can be easily located both in the AM graph and the fully-composed WFST, as it only requires one indirection to fetch the state's information that provides the address of the first arc and the number of arcs. Note that when a state is expanded, all its arcs are explored sequentially to generate new tokens. However, when a cross-word transition is traversed in the AM model, the search has to traverse the arc associated with the corresponding word ID in the LM, i.e. the arc whose input label matches that word ID. Locating the correct arc requires some sort of search across the arcs of a given state, which might be expensive for two reasons. First, states in the LM have thousands of arcs. Second, due to the aforementioned back-off mechanism, there might be no arc with the target word ID for a given state and, in this case, the search would be repeated multiple times for different states in the LM. Implementing the location of the arc as a linear search increases the execution time by 10x with respect to offline composition. Alternatively, sorting the outgoing arcs of each state by word ID to use binary search reduces the slowdown to 3x, which still represents a huge overhead, making the fetching of arcs in the LM the most critical operation for on-the-fly composition. In Section 3, we propose a specialized hardware to speed up this operation and locate the arcs in the LM graph with a very small overhead.

3 HARDWARE-ACCELERATED ON-THE-FLY WFST COMPOSITION

In this section, we describe UNFOLD, our hardware-accelerated speech recognizer, that employs on-the-fly composition to dramatically reduce memory footprint. First, we explain the details of the accelerator's architecture and review its pipeline stages through an example, focusing on the hardware support for WFST composition. After that, we elaborate on how the preemptive pruning scheme works. Next, we present the compression techniques applied on top of the on-the-fly composition mechanism. Finally, we provide a thorough analysis for the accelerator's several memory components.

3.1 Accelerator's Architecture

Figure 4 shows an overview of the UNFOLD's design, which is based on the state-of-the-art approach presented for the fully-composed WFSTs [34]. The accelerator includes several pipeline stages to fetch states, i.e. On-the-fly State Issuer, fetch arcs, i.e. On-the-fly Arc Issuer, fetch DNN/GMM acoustic scores, i.e. Acoustic-likelihood Issuer, compute the likelihood of a hypothesis, i.e. Likelihood Evaluation, and write the data of the new tokens, i.e. Token Issuer. Furthermore, the accelerator provides several on-chip caches for the different datasets used in ASR, an *Offset Lookup Table* to speed up finding the location of the LM arcs, and two Hash Tables that store the tokens for the current and the next frames of speech. All the memory components are in gray color and the newly modified stages with respect to [34] are enclosed with dashed lines.

Compared with previous work [34], we have modified the State and Arc Issuers to fetch data from two WFSTs instead of one fully-composed WFST, as UNFOLD operates on the individual AM and LM models. Reading states and arcs from the AM works as in the fully-composed WFST and requires no change with respect to the previous proposals. However, fetching an LM arc is more complex, since the arc associated with a specific word ID has to be found among the thousands of outgoing arcs of a given state. To perform this search efficiently, the outgoing arcs of each state are stored in main memory sorted by word ID, and the Arc Issuer is extended to implement a binary search.

To further reduce the overhead of locating arcs in the LM, we observe that accesses to the LM exhibit good temporal locality, i.e. if a word is recognized in a given hypothesis, it is likely to be recognized in other hypotheses in the near future. We exploit this temporal locality with a new table, called *Offset Lookup Table*, that stores the arcs' offsets associated with the recently used combinations of LM state and word ID, i.e. it stores the results of recent binary

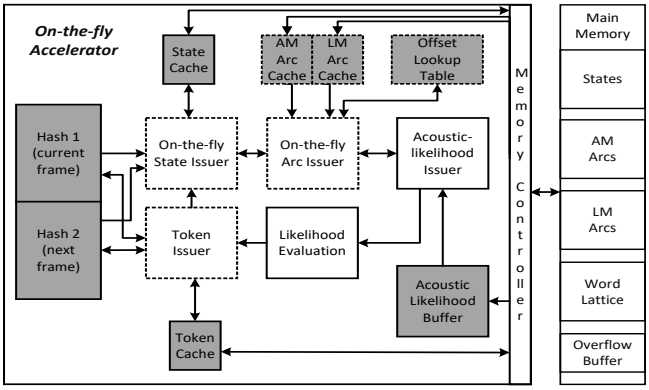


Figure 4: The architecture of the UNFOLD's design. Memory components are in gray color and the modified components with respect to [34] are shown with dashed lines.

searches to be reused by subsequent arc fetches. The accelerator first probes this table when looking for an LM arc and in case of a hit, the offset of the arc is obtained from the table in one cycle and no binary search is required. We show later in the paper that a small *Offset Lookup Table* achieves high hit ratios, avoiding the overhead of locating arcs in the LM by a large extent.

Another modification introduced in our design is the use of two independent caches for fetching arcs from AM and LM models in parallel, avoiding stalls in the pipeline of the accelerator. Due to the aforementioned complexity for fetching LM arcs, we found it beneficial for performance to have a dedicated cache for this task. Note that a single Arc Cache with two read ports could have been used as an alternative, but it does not provide any advantage as the AM and LM datasets are disjoint, i.e. there is no data replication in our separated arc caches. Regarding the states, we share a single cache for both AM and LM due to several reasons. First, the states' dataset is significantly smaller than the arc's dataset (less than 12% of the WFST size). Second, the pressure on the State cache is low, as concurrent requests for AM and LM states are uncommon. We have observed that splitting the State cache into two separate caches for AM and LM does not provide any noticeable performance or energy improvement.

On the other hand, we also reduce the size of the different caches of the accelerator without any increase in miss ratio, since the new dataset is considerably smaller due to the use of on-the-fly composition. Finally, we modify the Token Issuer to write the word lattice in a compact representation as described in [22], which provides further savings in memory traffic.

3.2 Accelerator's Pipeline

We use the example shown in Figure 3c to illustrate how UNFOLD performs the Viterbi search while composing the AM and LM models on-the-fly. The pipeline starts from the hash table containing the source tokens for the current frame of speech (i.e., all the states expanded by the previous frame). Assuming we are at frame 2, this hash table includes the information of four tokens: (2, 1), (5, 2), (0, 2) and (7, 0). These tokens represent four different hypotheses, i.e. four alternative representations of the speech up to the current frame. The State Issuer reads the tokens from the hash table and applies a pruning step, discarding tokens whose likelihood is smaller than a threshold. This threshold is updated on each frame according to the best-hypothesis' probability and a predefined beam value. In this example, the four tokens pass the pruning and, thus, the State Issuer fetches the information of states 2, 5, 0 and 7 from the AM WFST by using the State Cache.

On the next stage, the Arc Issuer fetches the outgoing arcs of each AM state through the AM Arc Cache. Arcs departing from states 2, 5 and 0 have epsilon word ID (see Figure 3a) and, hence, no information from LM is required. However, the outgoing arc of state 7 is associated with the word "*THREE*". In this case, token (7, 0) has to transition in both models by traversing the corresponding arcs in AM and LM. Therefore, a look-up in the LM is required to fetch the outgoing arc of state 0 associated with the word "*THREE*". To locate this arc, the Arc Issuer first looks for the (LM state, word) combination of (0, "*THREE*") at the Offset Lookup Table. In case of a hit, the arc offset associated with that state and word ID is retrieved from the table and the address of the arc in main memory is obtained. Otherwise, the Arc Issuer starts a binary search to find the outgoing arc of state 0 whose input label equals word "*THREE*". Another independent cache, called LM Arc Cache, is utilized to speed up the search. Note that this process may take several cycles, as multiple memory fetches may be necessary to find the arc. Once the arc is located, its offset is stored in the *Offset Lookup Table* to be later reused by subsequent fetches. The weight of this LM arc represents the unigram likelihood of the word "*THREE*", and it is used to rescore the likelihood of the new token (8, 3), expanded from (7, 0).

The rest of the pipeline basically follows the behavior of the fully-composed accelerator [34]. The Acoustic Likelihood Issuer gets the acoustic scores of the phonemes associated with the AM arcs. These scores are computed by the DNN/GMM on the GPU and stored in the Acoustic Likelihood Buffer. Next, the Likelihood Evaluation unit performs the required processing to compute the probability of the new hypothesis, combining the different likelihoods from the source token, AM arc and LM arc. Finally, the Token Issuer stores the new token's information. The information which is only required for the next frame is stored in the hash table, whereas the information required for generating the word lattice is written in main memory by using the Token Cache. The hash tables are indexed through a combination of IDs of AM and LM states associated with the token. Other mechanisms of the hash structure, like handling collisions and overflows, work as described in the fully-composed design [34].

3.3 Preemptive Pruning in the Back-off Mechanism

Not all possible combinations of two and three words are included in the LM, as it would require a huge amount of memory. Combinations that are very unlikely are omitted to keep the LM manageable. Therefore, there are cases where the Arc Issuer cannot find an arc associated with a particular word ID from the outgoing arcs of a state in LM. For these cases, the LM provides a back-off arc in each state. These arcs back off from trigram to bigram, or from bigram to unigram states. All the unigram likelihoods are maintained to guarantee that, in the worst case, any word ID can be found in

an arc departing from state 0. For instance, consider the word sequence "*TWO-ONE*" in Figure 3b, if the next word is "*TWO*", then we use a back-off transition to state 1 which represents the unigram history of seeing word "*ONE*" since there is no 3-gram model for "*TWO-ONE-TWO*". Next, as there is no bigram from state 1 for the word "*TWO*", another back-off transition is taken to state 0. Then, by traversing the right arc, it reaches to the destination state 1, which corresponds to having seen the unigram "*TWO*".

The back-off mechanism increases the workload in UNFOLD, as it requires multiple arc-fetches in different states for a single token, with potentially multiple binary searches if misses arise in the *Offset Lookup Table*. To reduce the overhead of the back-off mechanism, we introduce a preemptive pruning scheme in the Arc Issuer, which prunes in advance the backed-off hypotheses whose probabilities go lower than a threshold. Back-off arcs include a weight, i.e. likelihood, to penalize the hypotheses, as they are traversed for very unlikely combinations of words that were omitted from the LM. For this purpose, the Arc Issuer updates and checks the likelihood of a hypothesis after traversing a back-off arc, in order to stop the back-off mechanism as soon as the hypothesis can be safely discarded. Since the likelihood of a hypothesis is a monotonically decreasing function, it is guaranteed that we only discard the hypotheses that would be pruned away later in the accelerator.

Our preemptive pruning requires modest hardware, as only three floating-point units are added in the Arc Issuer to apply the weight of the AM arc, the back-off arc and compare the result with the threshold for the pruning. Our experimental results show that, on average, 22.5% of the total number of hypotheses are pruned away and 16.3% performance improvement is achieved with the preemptive pruning.

3.4 WFST Compression

The use of on-the-fly composition significantly reduces the size of the datasets employed for ASR, as shown in Table 1. However, large vocabulary systems still require more than one hundred Megabytes for the AM and LM models. WFST compression can be applied on top of on-the-fly composition to further reduce the size of the datasets, achieving a large reduction compared to the original fully-composed WFST. This section describes the compression techniques employed in UNFOLD, including a description of the minor changes implemented in the hardware to support the compressed WFSTs.

The memory layout used in UNFOLD for each WFST is based on [3], which consists of two separate arrays for storing the states and the arcs information respectively. Regarding the states, we implement the bandwidth reduction scheme introduced in [34], that is also very effective for reducing the size of the states' information. On the other hand, arcs consume most of the storage of the WFST, since each state has hundreds or even thousands of outgoing arcs. Therefore, our compression techniques mainly focus on shrinking the size of the arcs.

Each arc consists of a 128-bit structure including destination state index, input label (phoneme index in AM and word ID in LM), output word ID and weight, each field represented by 32 bits. Regarding the weight, i.e. the likelihood of the arc, we use the K-means quantization technique with 64 clusters, reducing its size from 32 to 6 bits, which introduces a negligible increase in Word Error Rate (WER) of the on-the-fly ASR accelerator (less than 0.01%). For the other fields, we apply different techniques for LM and AM arcs.

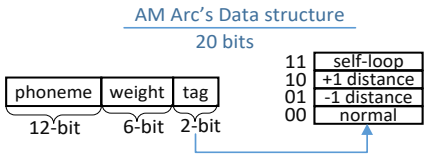


Figure 5: General structure of the AM arcs' information

Table 2: Compressed sizes of WFSTs in Mega-Bytes for the fully-composed and on-the-fly composition.

	Tedlium-EESEN	Tedlium-Kaldi	Voxforge	Librispeech
On-the-fly	39.35	32.39	1.33	21.32
Fully-composed	414.28	269.78	9.38	136.82

Regarding LM data, we categorize the arcs in three different groups: the outgoing arcs of the initial state (state 0 in Figure 3b) or Unigram arcs, the back-off arcs and the rest. The Initial state consists of as many outgoing arcs as the number words of the vocabulary, each associated with a different word ID. As the destination states of these arcs are in consecutive order following their word IDs, their location can be inferred from the word ID. In other words, the i-th outgoing arc of state 0 is associated with word ID i and has destination state i. Therefore, no information other than a 6-bit weight value is required for each of these arcs.

On the other hand, back-off arcs in the LM model do not have any input or output label, they are only taken when no specific arc exists for a given word ID. In this case, we only store two attributes: the weight (6 bits) and destination state's index (21 bits). For the rest of the arcs, in addition to the aforementioned 27 bits, an 18-bit word ID is included to store the input label. We found that 18 bits is enough to represent every word of the vocabulary in all the tested ASR decoders. Note that, despite having variable length arcs, we still provide efficient random access to the LM information, which is a requirement of the binary search commented in Section 3.1. In our scheme, the back-off arc is always the last outgoing arc of each state, so it can be easily located in case the back-off mechanism is triggered. In addition, the rest of outgoing arcs of a given state have fixed size, taking 6 bits per arc for state 0 and 45 bits for the other states.

Regarding AM data, we observed that, in our set of WFSTs, most of the arcs have epsilon word ID, i.e. they are not associated with any word in the vocabulary, and they point to the previous, the same or the next state. This is also the case in the example of Figure 3a. For these arcs, we only store the input phoneme index (12 bits), weight (6 bits) and a 2-bit tag encoding destination state. This format is illustrated in Figure 5. The 2-bit tag indicates whether the arc points to the same state (self-loop arc), the previous (distance -1) or the next state (distance +1). The rest of the arcs require, in addition to the aforementioned 20 bits, an 18-bit word ID and a 20-bit destination state's index.

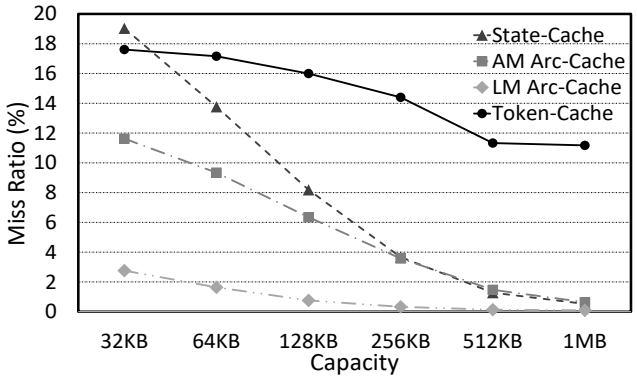


Figure 6: Miss-ratio vs capacity of the several caches

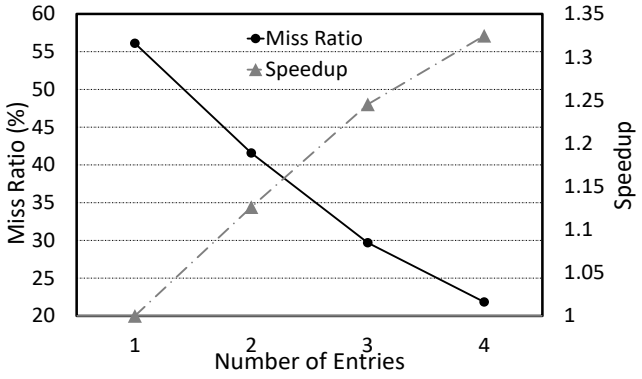


Figure 7: The effect of Offset Lookup Table's size on speedup

UNFOLD supports the compressed AM and LM models with some minor changes to the architecture. First, we include a 64-entry table (256 bytes) to store the floating-point centroids of the clusters used by the K-means approach to encode the weights of the arcs. The arc's information provides the 6-bit index of the cluster, and fetching the floating-point value requires an access to the table of centroids that is performed in an additional stage in the pipeline of the accelerator. Second, the Arc Issuer is extended to obtain the length of each AM arc from the 2-bit tag, illustrated in Figure 5 . By decoding this tag, the Arc Issuer knows whether it has to fetch the remaining 38 bits for the current arc, or the 20 bits for the next arc. This serialization of the arcs does not introduce any additional complication, since the AM arcs of a given state are always explored sequentially in the Viterbi search. Third, the Arc Issuer is also extended to support the variable length arcs proposed for the LM model. The Address Generation Unit employs the state ID to obtain the size of the arcs (arcs departing from state 0 are packed in 6 bits, whereas outgoing arcs of other states have 45 bits). Note that the state's information provides the address of its first outgoing arc and, hence, obtaining the address of its i-th arc is trivial once the size of the arcs is identified.

Table 2 shows the sizes of WFSTs used for the on-the-fly composition scheme after applying the compression techniques for AM and LM. As we can see, the dataset has been significantly reduced, from more than 1 GB or hundreds of MB (see Table 1) to a few tens of MB. Furthermore, we report the compressed size of the fully-composed WFST, for which we have implemented the compression techniques described in [23]. Compared to the compressed fully-composed WFST, our approach reduces the size of the required dataset by 8.8x on average. This large reduction is due to the combined use of both on-the-fly WFST composition and compression of the individual AM and LM models.

3.5 Analysis of Caches and Offset Lookup Table

Figure 6 shows the miss ratio versus capacity for the different caches included in the accelerator. Regarding the State Cache and the two Arc Caches, AM and LM, the miss ratio is significantly reduced when increasing capacity, reaching a miss ratio smaller than one percent for a size of one Megabyte. For the Token Cache, miss ratio is close to 12% even for a capacity of one Megabyte due to compulsory misses. New tokens are written in consecutive memory locations and, thus, spatial locality is exploited at the cache line level. However, once a line is filled with tokens it is unlikely to be reused, i.e. accesses to the tokens exhibit poor temporal locality.

The miss ratios shown in Figure 6 are smaller than the ones reported in [34] for the fully-composed WFST, since the memory footprint has been largely reduced in our proposal through on-the-fly composition and compression. For a State Cache of 512 KB, the miss ratio is reduced from 29% in [34] to 1.5% in UNFOLD. On the other hand, an Arc Cache of 1 MB exhibits a 27% miss ratio for the fully-composed WFST, whereas the two separated AM and LM arc caches of 512 KB achieve 1.6% and 0.3% miss ratios respectively. Note that the state and arc datasets are reduced from the large sizes reported in Table 1 for the composed WFST, to the small sizes reported in Table 2 for the compressed AM and LM WFSTs. Finally, the miss ratio for a Token Cache of 512 KB is reduced from 15% to 11.5%, as we use the compact representation for the tokens proposed in [22].

Based on the above analysis, we decided to reduce the sizes of the caches with respect to [34], as we found it beneficial for the overall energy consumption and area usage. Table 3 shows the cache sizes for both the UNFOLD and [34]. Our caches still provide smaller miss ratios due to the large reduction in the memory footprint of the datasets, and their smaller sizes result in lower energy per access and area. In addition, we reduce the total size of Hash Table in spite of using the same number of entries, since the number of bits representing each of its entry's attributes is smaller using the compression scheme.

Finally, Figure 7 shows the effect of the capacity of the Offset Lookup Table on the performance of the accelerator. As expected, the bigger the table the smaller the miss ratio, resulting in higher performance in the accelerator. This table is direct-mapped, and it is indexed using the XOR of the LM state index and the word ID. Each entry contains a valid bit, a 24-bit tag and the 23-bit offset for the arc. By considering the area, energy per access and impact of the misses in the overall performance and energy consumption, we selected a table of 32K entries, which requires 192KB of storage.

Table 3: Accelerator's configuration parameters.

Parameter	UNFOLD	Reza et al.
Technology	32 nm	28 nm
Frequency	800 MHz	600 MHz
State Cache	256 KB, 4-way, 64 B/line	512 KB, 4-way, 64 B/line
AM Cache	512 KB, 8-way, 64 B/line	1 MB, 4-way, 64 B/line
LM Cache	32 KB, 4-way, 64 B/line	-
Token Cache	128 KB, 2-way, 64 B/line	512 KB, 2-way, 64 B/line
Acoustic Likelihood Buffer	64 Kbytes	64 Kbytes
Hash Table	576 KB, 32K entries	768 KB, 32K entries
Offset Table	192KB, 32K entries	-
Mem-Ctrl	32 in-flight requests	32 in-flight requests
Likelihood Evaluation Unit	4 floaing-point adders , 2 floaing-point comparators	4 floaing-point adders , 2 floaing-point comparators

4 EVALUATION METHODOLOGY

We developed a cycle-accurate simulator of the proposed on-the-fly accelerator. The configuration's parameters used for the experiments are shown in Table 3 . We evaluated different sizes of the accelerator's memory components, and selected the configuration that provides the best trade-off considering performance, area and energy consumption.

In order to estimate the latency, area and power dissipation of each module, we have implemented each pipeline stage in hardware using Verilog and synthesize them with the Synopsys Design Compiler tool using a 32nm technology library. Furthermore, we use CACTI [16] to estimate the area and power of the caches and storage components of the accelerator. Finally, to model the off-chip DRAM main memory, we use the Micron power model for an 8-GB LPDDR4 [29, 30]. The cycle-accurate simulator provides the activity factor for the different components and the total cycle count, which are used to compute execution time, dynamic and static energy by combining them with the estimations of the Design Compiler, CACTI and Micron power models.

To set the frequency of the system, we consider the critical-path-delay and access time reported by Design Compiler and CACTI respectively. We take the maximum delay among the different components, which is 1.25 ns for the AM Arc Cache access, as the cycle time (i.e. clock rate of 800MHz).

For the purpose of comparing with a GPU, we use an NVIDIA Tegra X1 SoC, which includes a modern mobile GPU with parameters shown in Table 4 . To measure the energy consumption of the GPU, we read the registers of the Texas Instruments INA3221 power monitor included in the Jetson TX1 platform, which provides the actual energy by monitoring the GPU power rail as described in [8].

We run state-of-the-art CUDA implementations of the different ASR decoders on the mobile GPU. For the GMM and the DNN, we use the implementation provided in Kaldi [21]. For the RNN, we use the GPU-accelerated version included in EESEN [18]. Finally, we use a state-of-the-art CUDA implementation of the Viterbi search [4, 36].

Table 4: GPU configuration's parameters.

GPU	Nvidia Tegra X1
Streaming multiprocessors	2 (2,048 threads/multiprocessor)
Technology	20 nm
Frequency	1.0 GHz
Level-2 cache	256 Kbytes

As the speech input, we use the test audio files of Librispeech [20], Tedlium [26] and Voxforge [5] to evaluate the different decoders.

5 EXPERIMENTAL RESULTS

In this section, we evaluate UNFOLD, the proposed hardware accelerator based on the on-the-fly WFST composition. First, we analyze the reduction in memory footprint achieved by UNFOLD. Then, we review its energy consumption and performance, providing a comparison with both a state-of-the-art Viterbi search accelerator and a solution based on a contemporary mobile GPU. Next, we evaluate the entire ASR pipeline, including UNFOLD and the GPU for running the GMM/DNN/RNN. Although the focus of this paper is the Viterbi search, since it is the main bottleneck in ASR as shown in Figure 1, we include the overall ASR system results for the sake of completeness. Finally, we conclude this section with a discussion of the applicability of our proposal.

5.1 Viterbi Search Accelerator

Figure 8 shows the memory requirements of the WFSTs for the fully-composed and on-the-fly composition approaches in several ASR systems. *Fully-Composed* configuration represents the sizes of the WFSTs for the accelerator presented in [34]. *Fully-Composed+Comp* implements the compression techniques introduced in [23] for the fully-composed WFSTs. *On-the-fly* shows the total size of the individual AM and LM models, without any kind of data compression. *On-the-fly+Comp* shows the memory requirements for UNFOLD, which includes on-the-fly WFST composition and the compression techniques described in Section 3.4.

As it can be seen in Figure 8, the *On-the-fly* approach provides consistent savings in memory requirements in the different ASR decoders. On-the-fly composition mechanism avoids the explosion in the number of states and arcs resulted from the full offline-composition of AM and LM models. Furthermore, the compression techniques provide significant additional savings. Overall, UNFOLD achieves a reduction in storage requirements of 31x on average with respect to the state-of-the-art Viterbi search accelerator proposed in [34] (*Fully-Composed configuration*), with a minimum and maximum reduction of 23.3x and 34.7x respectively. With respect to the state-of-the-art on WFST compression [23] (*Fully-Composed+Comp configuration*), our system achieves a reduction of 8.8x on average. All in all, UNFOLD only requires a few tens of Megabytes, which is reasonable for many mobile devices, unlike the more than one Gigabyte required by previous proposals.

Figure 9 shows the energy consumption for the Viterbi search, including both static and dynamic energy, in several ASR systems. The configuration labeled as *Tegra X1* is the CUDA-based solution running on the mobile SoC. *Reza et al.* configuration is the Viterbi accelerator proposed in [34]. Finally, the energy of the proposed

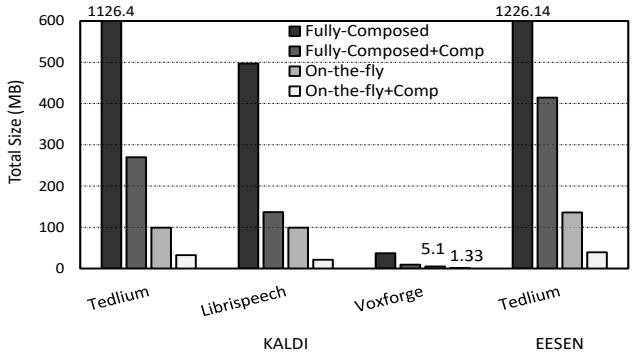


Figure 8: Sizes of different datasets for several ASR systems.

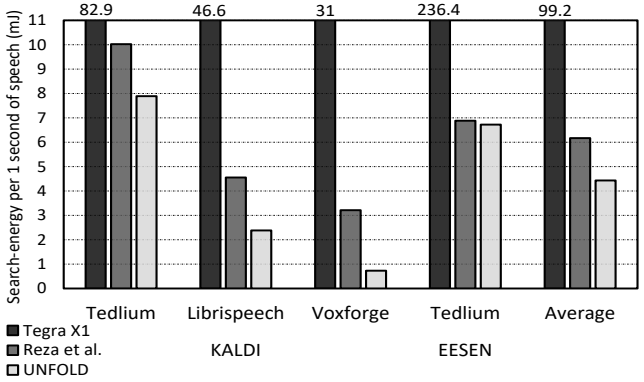


Figure 9: Viterbi search energy consumption of several speech recognizer in different platforms.

accelerator is shown in the configuration labeled as *UNFOLD*. *UNFOLD* reduces energy consumption by one order of magnitude with respect to the *Tegra X1*. Compared to the state-of-the-art in Viterbi search acceleration, it provides consistent energy savings in all the ASR decoders that range between 2.5% for EESEN-Tedlium and 77% for KALDI-Voxforge, with an average energy reduction of 28%. UNFOLD generates significantly less requests to main memory since the caches exhibit larger hit ratios, as reported in Section 3.5, due to the large reduction in the size of the datasets. On average, *UNFOLD* reduces off-chip memory accesses by 68% with respect to the *Reza et al.* configuration. Therefore, although it requires more on-chip activity to implement on-the-fly composition and handle the compressed datasets, it provides important energy savings due to the reduction in energy-expensive off-chip memory accesses [12].

Figure 10 shows the power breakdown, including both static and dynamic power, of the *Reza et al.* and *UNFOLD*. As we can see, power savings are mainly due to the reduction in the power dissipation of main memory, caused by the aforementioned savings in off-chip memory traffic. The power of the caches is also reduced, as they have smaller sizes (see Table 3) since our datasets are smaller. On the other hand, the power of the Token Cache is reduced by a large extent since we include the compressed format for the word lattice proposed in [22]. In the hash tables, as we are reducing the size of the attributes, the storage requirements are smaller compared to the fully-composed approach, and this results in less power dissipation. Finally, the *Offset Lookup Table*, which is required for efficiently locating LM arcs in our system, represents a small overhead since it only dissipates 5% of the total power in UNFOLD.

Figure 11 shows the memory bandwidth requirements of both fully-composed and on-the-fly accelerators when running different ASR decoders. As it can be seen, UNFOLD provides significant and consistent memory bandwidth savings for all the decoders. For the most bandwidth demanding decoder, EESEN Tedlium, UNFOLD reduces the memory bandwidth usage by nearly 2.8x, from 7.4 to 2.6 GB/s. On average, UNFOLD reduces memory bandwidth usage by 71%. These savings are mainly due to the large reduction in the size of the datasets (see Figure 8) provided by the on-the-fly composition and compression mechanisms. The smaller size of the datasets produces a large reduction in the number of cache misses, as described in Section 3.5, which results in an important reduction in the number of main memory accesses.

Regarding the decoding time, all the configurations achieve real-time performance by a large margin, which is the main requirement of ASR systems. *Tegra X1* runs 9x faster than real-time, whereas *Reza et al.* and *UNFOLD* run 188x and 155x faster than real-time respectively. Table 5 shows an analysis of the processing time per utterance for the two accelerators and the mobile GPU, including the average and the maximum processing time. Processing time per utterance affects to the responsiveness of the system and overall user experience. As it can be seen, both accelerators achieve significantly lower processing time than the mobile GPU, providing the result within tens of milliseconds on average and a few hundreds of milliseconds in the worst case.

The slowdown in our system with respect to the *Reza et al.* is due to the cost of fetching the arcs in the LM model, which requires a search across the outgoing arcs of a state to find the arc associated with a given word ID. Initially, we implemented the LM arc-fetching as a linear search, obtaining 10x slowdown. Next, we replaced it by a binary search, achieving 3x slowdown. The use of the *Offset Lookup Table* (see Section 3.1) and the preemptive pruning (see Section 3.3) reduce the slowdown to an average of 18% with respect to the state-of-the-art in Viterbi search acceleration. This overhead represents an increase in decode time for a second of speech from around 5 ms to 6 ms, which is practically negligible for typical applications, since it keeps being two orders of magnitude faster than real-time.

In order to evaluate the accuracy of UNFOLD, our cycle-accurate simulator also works as a functional emulator able to produce the most likely sequence of words for the input speech. Table 6 shows the Word Error Rate (WER) for the different ASR decoders. Note that the GMM/DNN/RNN and the WFST are trained on hundreds of hours of audio from the different training datasets: Tedlium, Librispeech and Voxforge. The WER is reported for the entire test datasets that include several hours of speech. TEDLIUM is a more challenging task as it includes spontaneous speech in a noisy environment and, hence, the WER is larger than in Librispeech or Voxforge. Furthermore, we have verified that the difference in accuracy with respect to the fully-composed approach is negligible,

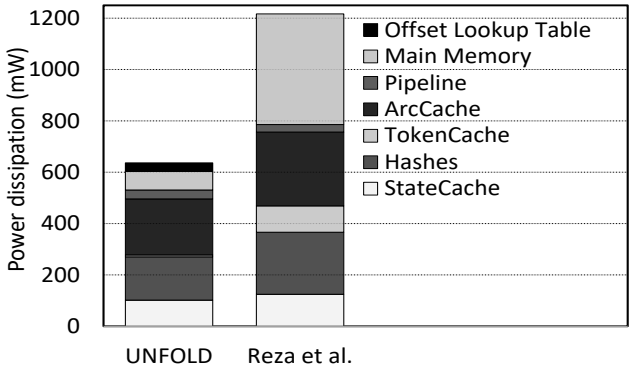


Figure 10: Power breakdown of *UNFOLD* versus *Reza et al.*

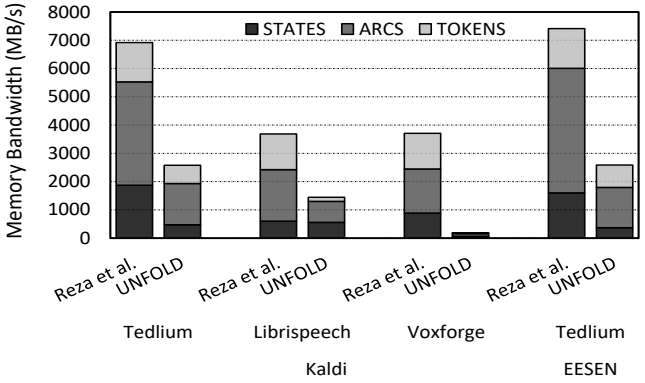


Figure 11: Memory bandwidth usage of *Reza et al.* and *UNFOLD* for different ASR decoders.

i.e. the compression and on-the-fly mechanisms do not introduce any noticeable accuracy loss.

Regarding the area, UNFOLD has a size of 21.5 mm 2 , providing a reduction of 16% with respect to the Reza et al. configuration. This reduction is due to the smaller caches used, as described in Section 3.5.

To sum up, the UNFOLD's on-the-fly WFST composition scheme provides a reduction in memory storage requirements of 31x and 28% energy savings on average with respect to the state-of-the-art in Viterbi search acceleration. On the other hand, it introduces a slowdown of 18% and a small increase in Word Error Rate of less than 0.01% due to the clustering of the weights. We believe that the important savings in memory footprint and energy consumption largely compensate for the negligible slowdown and accuracy loss. In other words, using 40 Mbytes of memory instead of more than one Gigabyte makes a huge difference, and saving 28% energy is very important, but being 155x faster than real-time instead of 188x has no impact on user experience.

5.2 Overall ASR System

In this section we evaluate the memory requirements, performance and energy consumption for the overall ASR system. In addition to the Viterbi search, the KALDI-Tedlium and KALDI-Voxforge decoders employ a GMM for acoustic-scoring. The KALDI-Librispeech employs the DNN presented in [37] for acoustic scoring. Finally, the EESEN-Tedlium uses a Recurrent Neural Network (RNN) [18] to generate the likelihoods consumed by the Viterbi search.

Regarding the sizes of the datasets, our system provides a compression ratio of 15.6x on average when considering both the WFST and the GMM/DNN/RNN. After applying on-the-fly composition and WFST compression, the GMM/ DNN/RNN represents approximately half of the dataset for each of the ASR systems evaluated except for the Voxforge whose GMM takes 26% of the storage required for ASR.

Figure 12 shows the decoding time per one second of speech for different configurations. *Tegra X1* configuration runs the entire ASR pipeline on the mobile GPU. *Reza et al.* and *UNFOLD* execute the Viterbi search on the accelerator proposed in [34] and our on-the-fly accelerator respectively, while the GMM/DNN/RNN is executed

Table 5: Maximum and average decoding time (ms) per utterance for different decoders and platforms.

Task	Tegra X1		Reza et al.		UNFOLD	
	Max	Avg	Max	Avg	Max	Avg
Tedlium-Kalid	2538	1069	215	76.7	274	92.5
Librispeech	1651	1336	54.7	31.9	57.4	30
Voxforge	743	450	47.1	15.5	13.1	4.2
Tedlium-EESEN	3972	1412	222	60	680.4	111.6

Table 6: Word Error Rate (WER) of UNFOLD for different ASR decoders.

	Tedlium-Kaldi	Librispeech	Voxforge	Tedlium-EESEN
WER(%)	22.59	10.62	13.26	27.72

on the mobile GPU. The integration between the GPU and the accelerator is performed as described in [35]. In the overall system, the input speech is split into batches of N frames and the GPU and the accelerator work in parallel: the GPU computes the acoustic scores for the current batch while the accelerator performs the decoding of the previous batch. The GPU communicates the acoustic scores through a shared buffer in main memory, and we account for the cost of these communications in our performance and energy numbers for the overall ASR system. All the decoders achieve real-time performance, and the configurations using hardware-accelerated Viterbi search achieve similar decoding time, outperforming the GPU-only configuration by approximately 3.4x.

Figure 13 shows the energy consumption per one second of speech for the same configurations. *Reza et al.* and *UNFOLD* provide similar energy reduction of 1.5x over the *Tegra X1* configuration. The reason why these two configurations exhibit similar behavior is that, after accelerating the Viterbi search, the GMM/DNN/RNN becomes the most time and energy consuming component, as it runs in software on the GPU. Note that, if further energy savings or performance improvements are required, an accelerator for the GMM [28], DNN [7] or RNN [11] can be employed. Accelerating these algorithms is not the scope of this paper, we focus on the

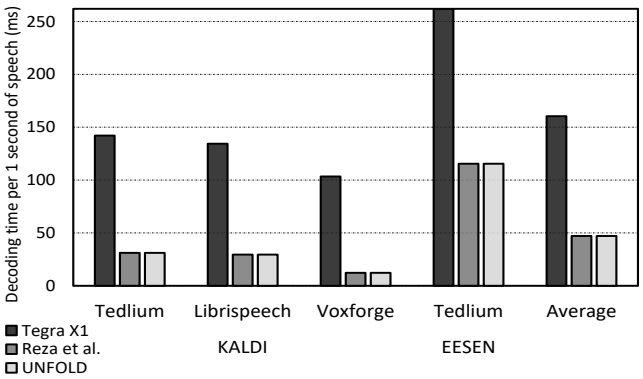


Figure 12: Overall ASR system decoding-time in different platforms.

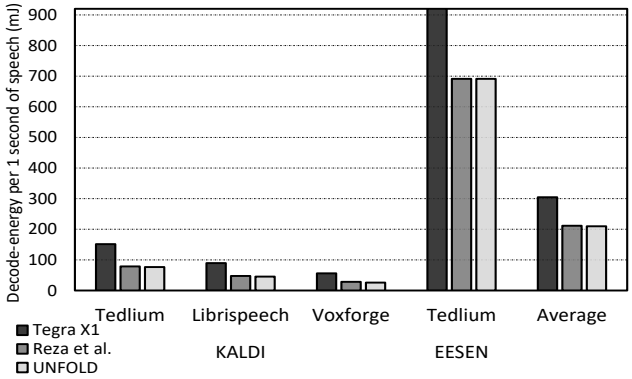


Figure 13: Overall ASR system energy consumption in different platforms.

Viterbi search as it is the main bottleneck in state-of-the-art ASR systems.

To sum up, when considering the entire ASR pipeline, our system achieves similar performance and energy consumption to the state-of-the-art proposals, but providing a reduction of 15.6x in the memory storage requirements. Our system requires less than 80 Megabytes for the entire ASR datasets in the worst case, whereas previous proposals consume more than one Gigabyte.

5.3 Applicability of the Proposed Accelerator

Current Smartphones and Tablets include around 4-8 Gigabytes of memory, whereas devices like Smartwatches and other wearables may be limited to one Gigabyte or less. Hence, spending more than one Gigabyte for the purpose of performing ASR is not acceptable in many mobile systems. In this paper, we address this issue proposing UNFOLD, a hardware accelerator that is based on on-the-fly WFST composition and compression to reduce the size of datasets to several tens of Megabytes, which may be acceptable for most mobile devices.

On the other hand, UNFOLD supports any state-of-the-art ASR system. In this paper, we have shown its ability to accelerate two different ASR toolkits, Kaldi and EESEN. Furthermore, we have shown that it supports GMM- and DNN-based decoders, and it can also be used for end-to-end speech recognition with an RNN, since the Viterbi search is also an essential component for these ASR systems. Moreover, UNFOLD supports AM and LM models for different languages, supporting any grammar (bigram, trigram, pentagram...) and acoustic model (basephones, triphones...). Therefore, the same hardware can be used for any speech recognition task, just by replacing the AM and LM WFSTs. We believe that ASR will be a key feature in future computing devices, as it provides a much more natural human-computer interaction than traditional user interfaces.

6 RELATED WORK

Accelerating speech recognition in hardware has attracted the attention of the architectural community in recent years. The work that is closest to ours is the Viterbi search accelerator proposed by Reza et al. [34, 35]. We provide a detailed comparison with this accelerator in Section 5, showing that our proposal achieves 31x reduction in memory requirements and 28% energy savings on average. The main contribution of UNFOLD is the use of on-the-fly WFST composition and compression, including efficient hardware for performing the composition process in real-time to reduce the size of the datasets by a large extent.

On the other hand, Price et al. [22, 23] propose a low-power accelerator for speech recognition. This accelerator employs the fully composed WFST, applying compression techniques to reduce its size. We implemented their compression techniques and evaluated them for our test set of ASR decoders, the results are shown in Table 2 . As mentioned in Section 5.1, our scheme achieves 8.8x higher compression ratio with respect to this proposal. On the other hand, the accelerator proposed by Price et al. implements in hardware the entire ASR pipeline, providing two different versions with support for GMM [24] or DNN [22]. UNFOLD implements the Viterbi search that is the main bottleneck in a wide range of ASR decoders, whereas we run the acoustic scoring on the GPU to support GMM-, DNN- and RNN-based systems. Another accelerator for the Viterbi search is proposed by Jonston et al. in [15]. This system supports a 60K-word vocabulary in real-time and dissipates around 454mW, requiring 343 Mbytes of main memory for storing the WFST. In comparison, our system supports WFSTs for vocabularies of 200K words using less than 40 Mbytes of memory.

Finally, there have been several algorithms for on-the-fly WFST composition [1, 2, 6, 13, 17, 32] which have been explored as pure software-based solutions on the CPU. There are mainly two different strategies proposed for on-the-fly decoders: one-pass [1, 13] and two-pass [17] search methods. In the one-pass strategy, the search is done at the same time of composing the sub-WFSTs on-the-fly as necessary. In the two-pass approach, these phases are separated in a way that AM is searched first to produce multiple hypotheses and a word lattice, which is then rescored using the LM. As the rescoring phase of the two-pass method cannot be executed until the end of AM search, it typically leads to larger latencies that are harmful for real-time ASR decoders. In this paper, we selected the one-pass approach for a hardware implementation with the aim of achieving decoding times similar to the traditional fully-composed decoders.

To the best of our knowledge, this is the first work that explores a pure hardware implementation of on-the-fly WFST composition.

7 CONCLUSIONS

In this paper, we address the main problem of previous accelerators for ASR: the large size of the WFST, i.e. the graph-based language database, used in the Viterbi search. We identify that its excessive size is due to the offline full-composition of the Acoustic Model (AM) and the Language Model (LM) during training stage, and propose to dynamically compose both models on demand at the decoding stage. UNFOLD operates on the smaller AM and LM models, providing simple yet efficient hardware for on-the-fly WFST composition. Therefore, our system does not require a large WFST composed offline, as it is able to dynamically obtain the parts required for decoding the input speech signal, reducing memory storage requirements by a large extent. Moreover, we introduce several compression techniques to further reduce the size of the AM and LM and a novel search pruning scheme, that the UNFOLD's architecture is extended to support. On average, the memory storage requirements are reduced by 31x with respect to the state-of-the-art in Viterbi search acceleration. Furthermore, the large reduction in memory footprint results in significant savings in main memory bandwidth usage, which provides 28% energy savings on average with respect to the state-of-the-art. Despite increasing number of operations due to moving the WFST composition from training to decoding stage, UNFOLD achieves real-time performance by a large margin (155x).

ACKNOWLEDGMENT

This work is supported by the Spanish Ministry of Economy and Competitiveness and FEDER funds of the EU under contracts TIN2013-44375-R and TIN2016-75344- R.

REFERENCES

[1] Diamantino Caseiro and Isabel Trancoso. 2001. On Integrating the Lexicon with the Language Model. (2001).

[2] D. Caseiro and I. Trancoso. 2001. Transducer composition for "on-the-fly" lexicon and language model integration. In *Automatic Speech Recognition and Understanding, 2001. ASRU '01. IEEE Workshop on*. 393–396. https://doi.org/10.1109/ASRU.2001.1034667

[3] J. Choi, K. You, and W. Sung. 2010. An FPGA implementation of speech recognition with weighted finite state transducers. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. 1602–1605. https://doi.org/10.1109/ICASSP.2010.5495538

[4] Jike Chong, Ekaterina Gonina, and Kurt Keutzer. 2011. Efficient automatic speech recognition on the gpu. *Chapter in GPU Computing Gems Emerald Edition, Morgan Kaufmann* 1 (2011).

[5] VoxForge Speech Corpus. 2009. http://www.voxforge.org.

[6] H. J. G. A. Dolfing and I. L. Hetherington. 2001. Incremental language models for speech recognition using finite-state transducers. In *Automatic Speech Recognition and Understanding, 2001. ASRU '01. IEEE Workshop on*. 194–197. https://doi.org/10.1109/ASRU.2001.1034620

[7] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 92–104. https://doi.org/10.1145/2749469.2750389

[8] M. Friesen. 2016. *Linux Power Management Optimization on the Nvidia Jetson Platform.* Technical Report. "https://www.socallinuxexpo.org/sites/default/files/presentations/scale14x_r27_final.pdf"

[9] Alex Graves and Navdeep Jaitly. 2014. Towards End-To-End Speech Recognition with Recurrent Neural Networks. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Eric P. Xing and Tony Jebara (Eds.), Vol. 32. PMLR, Bejing, China, 1764–1772. http://proceedings.mlr.press/v32/graves14.html

[10] A. Graves, A. r. Mohamed, and G. Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 6645–6649. https://doi.org/10.1109/ICASSP.2013.6638947

[11] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally. 2016. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA. *ArXiv e-prints* (Dec. 2016). arXiv:cs.CL/1612.00694

[12] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 1135–1143. http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf

[13] T. Hori, C. Hori, Y. Minami, and A. Nakamura. 2007. Efficient WFST-Based One-Pass Decoding With On-The-Fly Hypothesis Rescoring in Extremely Large Vocabulary Continuous Speech Recognition. *IEEE Transactions on Audio, Speech, and Language Processing* 15, 4 (May 2007), 1352–1365. https://doi.org/10.1109/TASL.2006.889790

[14] K. Hwang and W. Sung. 2014. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. 1–6. https://doi.org/10.1109/SiPS.2014.6986082

[15] J. R. Johnston and R. A. Rutenbar. 2012. A High-Rate, Low-Power, ASIC Speech Decoder Using Finite State Transducers. In *2012 IEEE 23rd International Conference on Application-Specific Systems, Architectures and Processors*. 77–85. https://doi.org/10.1109/ASAP.2012.25

[16] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 469–480.

[17] Andrej Ljolje, Fernando Pereira, and Michael Riley. 1999. Efficient general lattice generation and rescoring. In *EUROSPEECH*.

[18] Y. Miao, M. Gowayyed, and F. Metze. 2015. EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. 167–174. https://doi.org/10.1109/ASRU.2015.7404790

[19] Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech and Language* 16, 1 (2002), 69 – 88. https://doi.org/10.1006/csla.2001.0184

[20] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. 2015. Librispeech: An ASR corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. 5206–5210. https://doi.org/10.1109/ICASSP.2015.7178964

[21] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. 2011. The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society. IEEE Catalog No.: CFP11SRW-USB.

[22] Michael Price. 2016. Energy-scalable speech recognition circuits (Doctoral dissertation). In *Massachusetts Institute of Technology*. http://hdl.handle.net/1721.1/106090

[23] Michael Price, Anantha Chandrakasan, and James R. Glass. 2016. Memory-Efficient Modeling and Search Techniques for Hardware ASR Decoders. In *INTERSPEECH*. 1893–1897.

[24] M. Price, J. Glass, and A. P. Chandrakasan. 2015. A 6 mW, 5,000-Word Real-Time Speech Recognizer Using WFST Models. *IEEE Journal of Solid-State Circuits* 50, 1 (Jan 2015), 102–112. https://doi.org/10.1109/JSSC.2014.2367818

[25] L. R. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (Feb 1989), 257–286. https://doi.org/10.1109/5.18626

[26] Anthony Rousseau, Paul DelÃľglise, and Yannick EstÃĺve. 2014. Enhancing the TED-LIUM corpus with selected data for language modeling and more TED talks. In *In Proc. LREC*. 26–31.

[27] Hamid Tabani, Jose-Maria Arnau, Jordi Tubella, and Antonio Gonzalez. 2017. Performance Analysis and Optimization of Automatic Speech Recognition. *Multi-Scale Computing Systems, IEEE Transactions on* (2017). https://doi.org/10.1109/TMSCS.2017.2739158

[28] Hamid Tabani, Jose-Maria Arnau, Jordi Tubella, and Antonio Gonzalez. 2017. An Ultra Low-power Hardware Accelerator for Acoustic Scoring in Speech Recognition. In *Parallel Architecture and Compilation Techniques (PACT), 26th International Conference on*. IEEE.

[29] TN-41-01. 2007. *Calculating Memory System Power for DDR3, Micron Technology, Tech. Rep.* Technical Report.

[30] TN-53-01. 2016. *LPDDR4 Power Calculator, Micron Technology, Tech. Rep.* Technical Report.

[31] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. 2004. *Sphinx-4: A Flexible Open Source Framework for Speech Recognition.* Technical Report. Mountain View, CA, USA.

[32] D. Willett and S. Katagiri. 2002. Recent advances in efficient decoding combining on-line transducer composition and smoothed language model incorporation. In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 1. I–713–I–716. https://doi.org/10.1109/ICASSP.2002.5743817

[33] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2016. Achieving Human Parity in Conversational Speech Recognition. *CoRR* abs/1610.05256 (2016). http://arxiv.org/abs/1610.05256

[34] Reza Yazdani, Albert Segura, Jose-Maria Arnau, and Antonio Gonzalez. 2016. An ultra low-power hardware accelerator for automatic speech recognition. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783750

[35] Reza Yazdani, Albert Segura, Jose-Maria Arnau, and Antonio Gonzalez. 2017. Low-Power Automatic Speech Recognition Through a Mobile GPU and a Viterbi Accelerator. *IEEE Micro* 37, 1 (Jan 2017), 22–29. https://doi.org/10.1109/MM.2017.15

[36] K. You, J. Chong, Y. Yi, E. Gonina, C. J. Hughes, Y. K. Chen, W. Sung, and K. Keutzer. 2009. Parallel scalability in speech recognition. *IEEE Signal Processing Magazine* 26, 6 (November 2009), 124–135. https://doi.org/10.1109/MSP.2009.934124

[37] X. Zhang, J. Trmal, D. Povey, and S. Khudanpur. 2014. Improving deep neural network acoustic models using generalized maxout networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 215–219. https://doi.org/10.1109/ICASSP.2014.6853589