



# Exploring Recognition Network Representations for Efficient Speech Inference on Highly Parallel Platforms

Jike Chong<sup>\*†</sup>, Ekaterina Gonina<sup>\*†</sup>, Kisun You<sup>‡</sup>, Kurt Keutzer<sup>\*</sup>

<sup>\*</sup>Department of Electrical Engineering and Computer Science, University of California, Berkeley

<sup>†</sup>Parasians, LLC

<sup>‡</sup>School of Electrical Engineering, Seoul National University

jike.chong@parasians.com, egonina@eecs.berkeley.edu,

ksyou@dsp.snu.ac.kr, keutzer@eecs.berkeley.edu

## Abstract

The emergence of highly parallel computing platforms is enabling new trade-offs in algorithm design for automatic speech recognition. It naturally motivates the following investigation: do the most computationally efficient sequential algorithms lead to the most computationally efficient parallel algorithms? In this paper we explore two contending recognition network representations for speech inference engines: the linear lexical model (LLM) and the weighted finite state transducer (WFST). We demonstrate that while an inference engine using the simpler LLM representation evaluates  $22\times$  more transitions per second than the advanced WFST representation, the simple structure of the LLM representation allows  $4.7\text{-}6.4\times$  faster evaluation and  $53\text{-}65\times$  faster operands gathering for each state transition. We use the 5k Wall Street Journal corpus to experiment on the NVIDIA GTX480 (Fermi) and the NVIDIA GTX285 Graphics Processing Units (GPUs), and illustrate that the performance of a speech inference engine based on the LLM representation is competitive with the WFST representation on highly parallel computing platforms.

**Index Terms:** Continuous Speech Recognition, Data parallel, Graphics Processing Unit, Linear Lexical Model, Weighted Finite State Transducer

## 1. Introduction

Highly parallel computing platforms such as the graphics processing units (GPUs) are enabling tremendous parallel computing capabilities in personal desktop and laptop systems. This shifting landscape in the underlying computing platforms needs to be taken into account during algorithm design and evaluation. In this paper we explore the use of GPUs for continuous speech recognition (CSR) on the NVIDIA GTX480 (Fermi) and the NVIDIA GTX285 GPUs. A CSR application analyzes a human utterance from a sequence of input audio waveforms to interpret and distinguish the words and sentences intended by the speaker. Its top level architecture is shown in Figure 1.

The recognition process uses a language database that is compiled offline from a variety of knowledge sources using powerful statistical learning techniques. The *speech feature extractor* collects feature vectors from input audio waveforms. The Hidden-Markov-Model-based *inference engine* infers the most likely word sequence based on the extracted speech features and the recognition network. In a CSR system, common speech feature extractors can be parallelized using standard signal processing techniques.

A parallel inference engine traverses a graph-based knowledge network consisting of millions of states and arcs. As

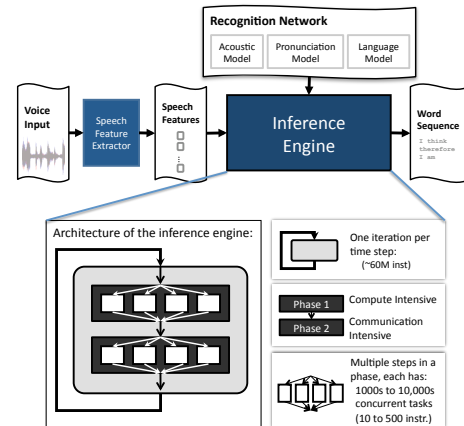
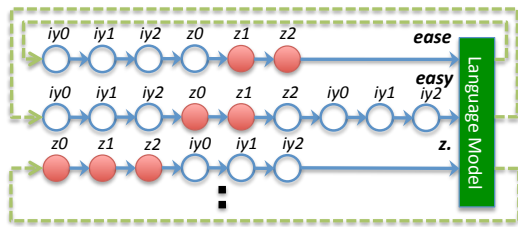


Figure 1: Architecture of continuous speech recognition

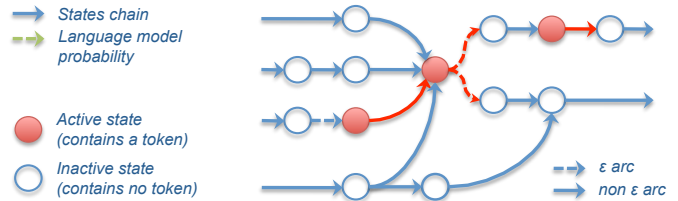
shown in Figure 1, it uses the Viterbi search algorithm to iterate through a sequence of input audio features one time step at a time [1]. The Viterbi search algorithm keeps track of each alternative interpretation of the input utterance as a sequence of states ending in an active state at the current time step. Each time step consists of two phases: Phase 1 - observation probability computation and Phase 2 - graph traversal. The parallelism being exploited is at the inner most level, across the 1000s to 10,000s alternative interpretations being evaluated at each algorithmic step in Phase 1 and Phase 2 of the execution. In each time step, Phase 1 is highly compute intensive, and can be readily accelerated on the GPU [2, 3, 4]. For a parallel implementation, the execution bottleneck lies in Phase 2, where the Viterbi algorithm requires numerous fine-grained synchronizations between algorithm steps. As will be illustrated in Section 4.2, this phase can comprise more than 45% of the total execution time in a highly parallel implementation.

This paper explores the speed and accuracy implications of two different approaches implementing Phase 2 of the inference engine on the GPU. The first approach is based on the linear lexical model (LLM) and the second is based on the weighted finite state transducer (WFST). LLM has a regular structure that can be implemented with highly efficient data parallel routines, but contains severe duplication of information in its language modeling approach. WFST is a concise representation, but its structure is highly irregular, making it difficult to execute efficiently on a data-parallel platform.

We demonstrate that a LLM-based inference engine can require more than an order of magnitude more computation ( $22\times$  more arc evaluations as illustrated in Section 4.2) to achieve the same recognition accuracy as a WFST-based inference engine. Its simpler structure, however, allows it to be more efficient on highly parallel GPUs. For some target accuracy, the simpler



A sample Linear Lexical Model Network



A section of a Weighed Finite State Transducer Network

Figure 2: Structure of the recognition network for the LLM representation and WFST representation

LLM-based inference engine can execute faster than the more advanced WFST-based inference engine.

The inference engine considered in this paper uses a token-passing approach for automatic speech recognition. This type of inference engine traverses a large recognition network by keeping a set of active tokens representing alternative interpretations for the speech input. Significant research effort has focused on optimizing the representation of the recognition network used during inference. The baseline representation is the LLM. The Tree-Lexical Model (TLM) and WFST are the two widely adopted optimizations of the LLM recognition network. Tree-organization of the pronunciation lexicon reduces the number of states being traversed during recognition [5] by sharing pronunciation prefixes in a prefix tree structure. In TLM, language model lookahead can be applied for efficient pruning [6]. More recently, the WFST representation has seen wide adoption [7] because of its application of powerful finite state machine transformations to remove redundant states and arcs during offline network compilation. As reported in [8], the WFST representation is faster than the tree-lexical representation as it explores even less search space. Both TLM and WFST show faster recognition speed than conventional linear lexical search network on a sequential processor. To our knowledge, no prior research has compared the recognition speed of these network representations on highly parallel computing platforms.

Emerging multicore and manycore platforms enable speech recognition to leverage large amount of concurrency for faster recognition. There have been many attempts to parallelize speech recognition with various recognition networks on parallel platforms. Parallel implementations with the LLM-based network representations have been presented in [9, 10]. Parallel implementations with the TLM-based networks on multicore platforms are described in [11, 12]. WFST-based speech recognizers are also parallelized in [13, 14]. Recently, GPUs have been adopted for parallelizing speech recognition applications. Much of the effort is focused on computing acoustic likelihoods in parallel because this phase is computationally intensive and embeds significant fine-grained concurrency [3, 4]. Recently, some progress has been made on parallelizing the communication intensive phase (i.e. the Viterbi search). A complete data parallel LVCSR on the GPU with a LLM-based recognition network was presented in [2]. Parallel WFST-based LVCSR is also implemented on CPU and GPU in [14, 15]. [14] compared sequential and parallel implementations of the WFST-based recognition network representations. This paper contrasts the implications of using LLM and WFST recognition network representations on the GPU. In the following sections, we will briefly introduce the key differences in the recognition network representations as well as outline our implementation strategies to arrive at efficient parallel implementations.

## 2. Network Structure

The network structures for the LLM and the WFST representations differ significantly. As illustrated in Figure 2, the LLM representation is constructed by providing a chain of triphone

states for each pronunciation to be recognized. For example, a dictionary of 5,000 word pronunciations would have 5,000 chains of triphone states in the recognition network. There are many duplications as each pronunciation chain is constructed using a separate copy of the triphone states for the chain’s phone sequence. The possibility of sharing common prefix is not considered in the LLM representation. The language model captures the likelihood of word-to-word transitions. It is used when the token-passing recognition process reaches the end of a chain of triphones. Since a word can be followed by any word, one must evaluate the possibility of transitions from one word to all words in the vocabulary.

In contrast, the WFST representation of the recognition network is constructed by composing the pronunciation model and the language model using powerful finite state machine (FSM) composition techniques. Both within-word transitions and across-word transitions are explicitly represented in the composed network. The sparsity structure of natural languages and the minimization techniques employed in [7] allow the WFST representation to avoid the state explosion problem when composing the models. As reported in [7, 8], the WFST representation is a concise representation that encapsulates a large amount of information with little redundancy. As we will show in our results, compared to the LLM representation, many fewer tokens are required to be maintained for the WFST representation during inference to achieve a target recognition accuracy.

The separate pronunciation and language models in the LLM representation allow for highly optimized computation kernel design, compensating for the performance lost from the redundancies in the representation.

## 3. Traversal Implementation

The two inference engines are implemented on the GPU, with one using the LLM representation of the recognition network and the other using the WFST representation. Both inference engines are designed using the structure shown in Figure 3, where both the observation probability computation and the token passing phases are implemented on the GPU. This structure has been shown to provide the best recognition speed on GPUs for both types of recognition networks [2, 14]. The implementations are designed to utilize hardware support for atomic operations on the GPU. In Phase 1, each Gaussian Mixture Model (GMM) in the acoustic model is evaluated in a thread-block; in Phase 2, each arc in the recognition network is evaluated by a thread. Backtracking is performed on the CPU. We highlight the key differences between the graph traversal process on recognition networks using the LLM representation and the WFST representation.

**The LLM representation:** Efficient implementation of the graph traversal of a network with LLM representation depends on explicitly handling two types of transitions in the LLM representation: within-word transitions and across-word transitions. This distinction was originally used in a parallel implementation by [9], and is elaborated here to include three specialized data structures for the GPU implementation. The three struc-

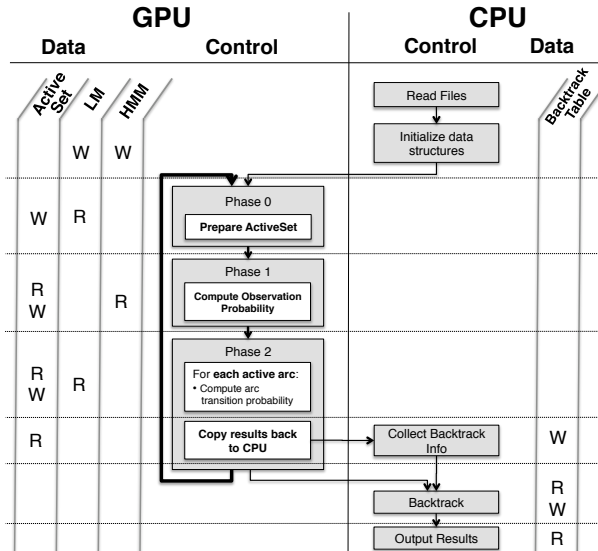


Figure 3: Control flow for the CPU/GPU software implementation and associated data structure access (R: read, W: write)

tures efficiently represent the chains of triphone states as the *first*, *middle* and *last* state in a word pronunciation. Within-word transition computation uses *first* and *middle* states as inputs and updates the *middle* and *last* state. Across-word transition computation uses *last* states as inputs and updates the *first* states. The distinct memory access patterns for within-word and across-word transitions can be efficiently implemented using specialized kernel routines and data layouts. Specifically, the first triphone states are stored consecutively in memory to optimize for across-word transitions and the middle and last states for each word are stored as a chain of consecutive states. These optimizations were originally proposed in [2] and have been re-implemented to take advantage of the new capabilities in the current generation of GPUs.

**The WFST representation:** An inference engine using the WFST representation does not distinguish between within-word and across-word transitions, as the pronunciation and the language models are compiled into a monolithic weighted finite state transducer. However, the recognition network does distinguish between non-epsilon and epsilon arcs as a result of state machine minimization during the network construction. The implementation optimizations of the WFST inference engine are described in [14].

Both implementations have been optimized with fast hardware atomic operations as well as using dynamically constructed efficient runtime buffers to gather operands that are dispersed in memory (as in Phase 0 in Figure 3). The LLM representation, however, has a significant structural advantage to enable a more efficient implementation. Across-word transitions are a significant computation bottleneck, as a typical recognition process evaluates more than 20 million transitions each second. This computation can be implemented as a dense matrix operation that can be executed on the GPU extremely efficiently. Such optimization is not possible with the WFST representation, as the graphs involved are highly irregular and are best mapped to sparse matrix or dense vector operations.

## 4. Results

### 4.1. Experimental Platform and Setup

We used both the NVIDIA GTX480 (Fermi) GPU and the NVIDIA GTX285 GPU with a Intel Core i7 920 based host platform. GTX480 has 15 cores each with dual issue 16-way vector arithmetic units running at 1.40GHz. Its processor archi-

ture allows a theoretical maximum of two single-precision floating point operations (SP FLOP) per cycle, resulting in a maximum of 1.35 TeraFLOP of peak performance per second. GTX285 has 30 cores with 8-way vector arithmetic units running at 1.51GHz. The processor architecture allows a theoretical maximum of three SP FLOP per cycle, resulting in a maximum of 1.087 TeraFLOP of peak performance per second. For compilation, we used Visual Studio 2008 with nvcc 3.0 and nvcc 2.2 respectively.

The acoustic model was trained by HTK [16] with the speaker independent training data in the Wall Street Journal 1 corpus. The frontend uses 39 dimensional features that have 13 dimensional MFCC, delta and acceleration coefficients. The trained acoustic model consists of 3,000 16-mixture Gaussians. Two language model sizes are used to illustrate the effect of a reduced language model size on the recognition performance. The language model weight is set to 15. The WFST network is an  $H \circ C \circ L \circ G$  model compiled and optimized offline with the dmake tool described in [17]. The test set consists of 330 sentences totaling 2,404 seconds from the Wall Street Journal test and evaluation set. The serial reference implementation using the LLM representation has a word error rate (WER) of 8.0% and runs with a 0.64 real time factor. The number of states and arcs representing the recognition networks used are shown in Table 1. WFST representation explicitly stores all across-word transitions, and LLM representation skips many across-word transitions as they are implied by the format. This leads to the differences in the actual number of arcs stored.

Table 1: Recognition network sizes used in experiments

	LLM Pruned	LLM	WFST Pruned	WFST
# State	123,246	123,246	1,091,295	3,925,931
# Arcs	537,608	1,596,884	2,955,145	11,394,956

### 4.2. Computation Load, Accuracy, and Speed

We measured the number of transitions evaluated by an inference engine using both the LLM and the WFST representations of the recognition network. As illustrated in Figure 4(a), each curve represents a set of recognition accuracy results by adjusting the average number of states maintained active. A dynamic threshold prediction scheme is used for pruning. We chose an operating point at 8.90% WER for the following comparison with an 11% relaxation (8.0% to 8.9%) in accuracy to gain almost a doubling in decoding speed (from  $7.4\times$  to  $13.7\times$  faster than real time for the WFST representation on GTX480).

At 8.90% WER, by using the LLM representation, an inference engine must evaluate  $22\times$  more transitions compared to using the WFST representation. For a sequential implementation where the execution is usually compute-bound, using the WFST representation could provide a significant speed advantage over the LLM representation.

The speed and accuracy trade-off shifts significantly for implementations on highly parallel platforms. Figure 4(b) illustrates the speed of the speech inference engine on the GTX285. The speed is shown as *Real Time Factor*, which is the number of seconds required to process one-second of speech input. The simplicity of the LLM representation allows the inference process to surpass the speed of the WFST representation for target WER of more than 8.8%.

Figure 4(c) illustrates the speed of the speech inference engine on the GTX480. Going from GTX285 to GTX480, the new processor microarchitecture provided an average of 25% speed improvement for the LLM representation and 79% improvement for the WFST representation. On the GTX480, with the availability of data caches, the WFST representation becomes

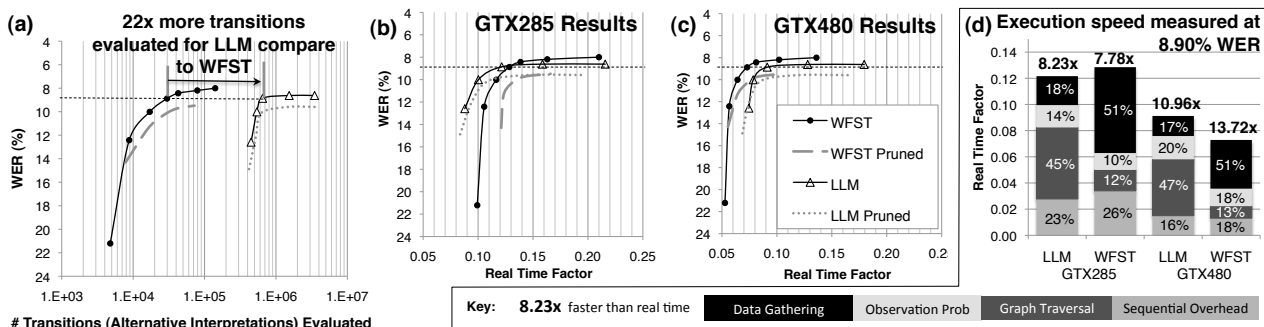


Figure 4: WER w.r.t. # of transitions evaluated (a), execution time in Real Time Factor (b/c), and speed analysis at 8.90% WER (d)

the faster implementation. Its irregular *Data Gathering* phase and *Graph Traversal* phase enjoys a 74% boost to its performance, while the same phases in the LLM representation only received a 27% performance improvement, which were already well parallelized using algorithm specific data structures.

In Figure 4(d), we see significant differences in the run time characteristics when using of the two representations of the recognition network across the two highly parallel platforms. To achieve 8.90% WER, the inference process using the LLM representation evaluates on average 649k transitions per time step, and obtains this at 8.23 $\times$  and 10.96 $\times$  faster than real time when executed on the GTX285 and the GTX480 respectively. The inference process with the WFST representation evaluates on average 29.8k transitions per time step to get 8.90% WER, and achieves this at 7.78 $\times$  and 13.72 $\times$  faster than real time when executed on the GTX285 and the GTX480 respectively.

The regular data layout of the LLM representation greatly reduces the cost of the *Data Gathering* phase of inference engine, taking less than 18% of the overall run time, while the WFST representation spends more than half its runtime in *Data Gathering*. For the *Graph Traversal* phase, although performing inference with the LLM representation takes 3-5 $\times$  more time compared to the WFST representation, it is evaluating 22 $\times$  the number of state transitions. The speed of the LLM representation is competitive with the WFST representation on highly parallel platforms because per state transition, it is 53-65 $\times$  faster in *data gathering* and 4.7-6.4 $\times$  faster in *graph traversal*.

Comparing performance between GTX285 and GTX480, we see an 85% and a 159% improvement in handling *sequential overhead* for the LLM and the WFST representations respectively. These are dominated by transfers of backtracking data from GPU to CPU.

## 5. Conclusions and Discussions

The emergence of highly parallel computing platforms brings forth the opportunity to reevaluate the computational efficiency of different approaches in speech recognition. In particular in this paper<sup>1</sup>, we consider the LLM and WFST representations of the recognition network in a speech inference engine. We found that despite the advantages of the sophisticated WFST representation for a sequential implementation, the simpler LLM representation performs competitively on highly parallel platforms. The LLM representation required the traversal of 22 $\times$  the number of state transitions. However, per state transition, it gathers data 53-65 $\times$  faster and evaluates the transitions 4.7-6.4 $\times$  faster than the WFST representation. Depending on the choice of the implementation platform, for the same 8.90% WER, the LLM representation is faster on the GTX285, and the WFST representation is faster on the GTX480.

<sup>1</sup>This research is supported in part by an Intel Ph.D. Research Fellowship, by Microsoft (Award #024263), by Intel (Award #024894), and by matching fund from U.C. Discovery (Award #DIG07-10227).

While this analysis is based on a 5K vocabulary recognition network, larger models are expected to benefit more significantly from the highly parallel computing platforms. Work is in progress to examine the recognition network representation trade-offs for larger models. Both GTX285 and GTX480 maintain an abstraction of uniform memory access latency. As other manycore processor architectures emerge with non-uniform memory access architecture (NUMA), one may want to experiment with static or dynamic partitioning of the recognition network to increase locality [9, 14]. With careful management of data working set, factoring of the WFST network could also be explored to increase locality [7].

## 6. References

- [1] H. Ney and S. Ortmanns, "Dynamic programming search for continuous speech recognition," *IEEE Signal Processing Magazine*, vol. 16, 1999.
- [2] J. Chong, Y. Yi, A. Faria, N. Satish, and K. Keutzer, "Data-parallel large vocabulary continuous speech recognition on graphics processors," *Proceedings of the 1st Annual Workshop on Emerging Applications and Many Core Architecture*, pp. 23-35, June 2008.
- [3] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau, "GPU accelerated acoustic likelihood computations," in *Proc. Interspeech*, 2008.
- [4] P. R. Dixon, T. Oonishi, and S. Furui, "Fast acoustic computations using graphics processors," in *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Taipei, Taiwan, 2009.
- [5] H. Ney, R. Haeb-Umbach, B.-H. Tran, and M. Oerder, "Improvements in beam search for 10000-word continuous-speech recognition," in *IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 1992, pp. 9-12.
- [6] S. Ortmanns, H. Ney, and A. Eiden, "Language-model look-ahead for large vocabulary speech recognition," in *Proc. Int. Conf. on Spoken Language Processing*, 1996, pp. 2095-2098.
- [7] M. Mohri, F. Pereira, and M. Riley, "Weighted finite state transducers in speech recognition," *Computer Speech and Language*, vol. 16, 2002.
- [8] S. Kanthak, H. Ney, M. Riley, and M. Mohri, "A comparison of two LVR search optimization techniques," in *Proc. Intl. Conf. on Spoken Language Processing (ICSLP)*, Denver, Colorado, USA, 2002, pp. 1309-1312.
- [9] M. Ravishanker, "Parallel implementation of fast beam search for speaker-independent continuous speech recognition," 1993.
- [10] A. Janin, "Speech recognition on vector architectures," Ph.D. dissertation, University of California, Berkeley, CA, 2004.
- [11] N. Parihar and E. Hansen, "A lexical-tree division-based approach to parallelize a cross-word speech decoder for multi-core processors," in *16th European Signal Processing Conference*, Lausanne, Switzerland, 2008.
- [12] K. You, Y. Lee, and W. Sung, "OpenMP-based parallel implementation of a continuous speech recognizer on a multi-core system," in *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Taipei, Taiwan, 2009.
- [13] S. Phillips and A. Rogers, "Parallel speech recognition," *Intl. Journal of Parallel Programming*, vol. 27, no. 4, pp. 257-288, 1999.
- [14] K. You, J. Chong, Y. Yi, E. Gonina, C. Hughes, Y. Chen, W. Sung, and K. Keutzer, "Parallel scalability in speech recognition: Inference engine in large vocabulary continuous speech recognition," in *IEEE Signal Processing Magazine*, no. 6, November 2009, pp. 124-135.
- [15] J. Chong, E. Gonina, Y. Yi, and K. Keutzer, "A fully data parallel WFST-based large vocabulary continuous speech recognition on a graphics processing unit," September 2009.
- [16] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, *The HTK Book Version 3.3*. Cambridge University Engineering Department, 2005.
- [17] C. Allauzen, M. Mohri, M. Riley, and B. Roark, "A generalized construction of integrated speech recognition transducers," in *IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 2004, pp. 761-764.