Real-Time Optimized *N*-gram For Mobile Devices

Sharmila Mani, Sourabh Vasant Gothe, Sourav Ghosh, Ajay Kumar Mishra,
Prakhar Kulshreshtha, Bhargavi M and Muthu Kumaran
Samsung R&D Institute Bangalore, Karnataka, India 560037
Email: {sharmila.m, sourab.gothe, sourav.ghosh, ajay.mishra, p.kulshresht, bhargavi.m, muthuk}@samsung.com

***Abstract*—With the increasing number of mobile devices, there has been continuous research on generating optimized Language Models (LMs) for soft keyboard. In spite of advances in this domain, building a single LM for low-end feature phones as well as high-end smartphones is still a pressing need. Hence, we propose a novel technique, Optimized *N*-gram (Op-Ngram), an end-to-end *N*-gram pipeline that utilises mobile resources efficiently for faster Word Completion (WC) and Next Word Prediction (NWP). Op-Ngram applies Stupid Backoff [1] and pruning strategies to generate a light-weight model. The LM loading time on mobile is linear with respect to model size. We observed that Op-Ngram gives 37% improvement in Language Model (LM)-ROM size, 76% in LM-RAM size, 88% in loading time and 89% in average suggestion time as compared to SORTED array variant of BerkeleyLM[2]. Moreover, our method shows significant performance improvement over KenLM[3] as well.**

***Index Terms*—Language Modelling, Mobile Devices, Natural Language Processing, *N*-gram, Soft Keyboard.**

I. INTRODUCTION

Language modelling is one of the crucial aspects of Natural Language Processing. It involves representing statistical language information to estimate a probability distribution over a sequence of words. N-gram models [4], which are Markov chain based (or count-based) LMs to predict the probability of n^{th} word based on the preceding sequence of $(n-1)$ words, have been widely used for language modelling tasks.

One of the important applications of language model is text suggestion in soft keyboards. The challenges involved in language modelling for mobile devices not only include providing better text suggestion but also providing it in real-time. The real-time model must be lightweight and the inference engine be fast enough. In this context, we present in this paper Optimized N-gram (Op-Ngram) that describes an end-to-end N-gram pipeline (Figure 1). Op-Ngram involves optimisations from (a) generating a light-weight LM with efficient structure, to (b) inferencing of the model using a for seamless Word Completions and Next Word Predictions.

The optimisations in Op-Ngram for achieving light-weight model include Stupid Backoff [1] and perplexity-based pruning. In N-gram models, backoff is used to normalise probabilities by penalising use of lower-order N-grams. By using Stupid Backoff, backoff values need not be pre-computed and stored for every combination, but a single heuristic constant can be used as the common backoff factor. For text suggestions in mobile devices, the goal is to provide top-K suggestions, where $K < 10$. Thus, less frequent words can be skipped from the generated model. These factors help in reducing our model size. Additionally, fast model loading and real-time inference on mobile has been achieved using a unique storage strategy. The model has been distributed across multiple files to enable parallel model loading.

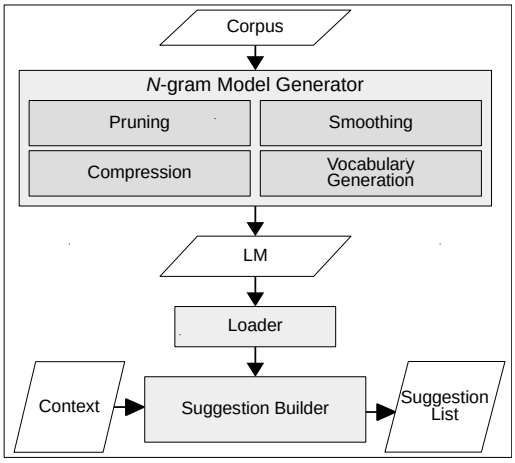

Fig. 1: Op-Ngram Flow

In the next section, we discuss relevant N-gram approaches for text suggestions available in literature. In section III, we present Op-Ngram in detail, which includes model generation, model storage, and model inference. In section IV, we also present results of Op-Ngram with English and Hindi in comparison to the available literature.

II. RELATED WORK

Paul et al. [2] describes a number of fast and compact approaches to language modelling. One simple variant discussed uses SORTED arrays for storing the LM, which we refer to as SORTED BerkeleyLM (SBLM). For each order of N-gram, this model encodes $\left(w_n, c\left(w_1^{n-1}\right)\right)$ using *context encoding* and uses these values as keys in a map. Here, c refers to entries in a $(n-1)$-gram sorted array. N-gram information is stored in blocks containing probability, backoff and index values. The indices of $(n-1)$-gram, or $(n-1)^{\text{th}}$ order N-gram, is used to identify terms in n^{th} order. Lookup by N-grams in this approach is logarithmic as binary search is used on the sorted array. Although this LM is faster than many other models proposed in literature, but the model size is not efficient in terms of real-time usage on a mobile device.

KenLM [3] is another widely used N-gram LM which introduces optimisations to model size using a sorted array and

978-1-5386-6783-5/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSC.2019.00021

TRIE data structure, and optimisations to speed by using hash tables and linear PROBING. Like SBLM, the KenLM TRIE approach uses reverse trie and separate suffix-sorted arrays for each order of N-gram, and in PROBING, number of hash table buckets should be more than that of entries to ensure collision resolution. The approaches discussed in this work optimises either speed or memory over the other, but not both at once.

In the following sections, we discuss our approach which is optimized for speed and memory without compromising one for the other. This manifests in terms of loading time, average suggestion time, RAM and ROM size with little impact on the relevance of text suggestions.

III. PROPOSED MODEL (OP-NGRAM)

N-gram is a statistical model which gives the probability distribution over a vocabulary V given the previous $N-1$ words as context. The probability assigned to a word sequence is:

$$P\left(w_n|w_{n-N+1}^{n-1}\right) = \begin{cases} \frac{(1-d)|w_{n-N+1}^{n}|}{|w_{n-N+1}^{n-1}|}, & |w_{n-N+1}^{n}| > 0 \\ \lambda\left(w_{n-N+1}^{n-1}\right) \times & \\ P\left(w_n|w_{n-N+2}^{n-1}\right), & |w_{n-N+1}^{n}| = 0 \end{cases} \quad (1)$$

where w_j^k represents the word sequence (w_j,w_{j+1},...,w_k), d refers to a discount ratio, $|w_1^n|$ indicates the count of w_1^n, and the function λ represents the back-off weight to ensure that probability values add up to unity [1].

We use trigram model ($N = 3$) as trigrams are known to be complex-enough for effective predictions, and simple enough for storing and retrieving them easily. Figure 1 details the flow of Op-Ngram.

A. Model Generation

This subsection describes the model generation pipeline. The pre-processing phase involves cleaning and sampling a crawled corpus, fetching word-count statistics, marking sentence-endings, and tagging blacklisted and unknown words in the corpus text, where we leverage Hadoop-based distributed processing. A sampling size of 2.5 GB is considered to have an optimal trade-off between model inference and size. The beginning and ending of a sentence is marked by prefixing a start-tag, `<s>`, and an end-tag, `<e>`, to preserve the probability of words appearing at the starting and end of a sentence. A linguist-verified bad words list is used to filter out common blacklisted words and replace their occurrences with `<bad>` tag. Any word that occurs in the corpus text with a frequency below a pre-defined threshold is deemed to be a rare or unknown word and is substituted with `<unk>` tag. Tagging of blacklisted and rare words helps to minimise their impact on the model quality.

During model generation, the counts of unigrams, bigrams, and trigrams are obtained from the pre-processed corpus. However, if all the N-grams are to be included then model size becomes comparable to original corpus size (i.e. $\sim$ 2 GB). So, non-informative bigrams and trigrams are pruned for reducing the model size. Brants et al. [1] proposes that at a very high pruning threshold even simple count-based pruning gives similar results to complicated methods like Stolke-Pruning [5]. Op-Ngram uses perplexity-based pruning, where the importance of a trigram (for pruning) is defined as:

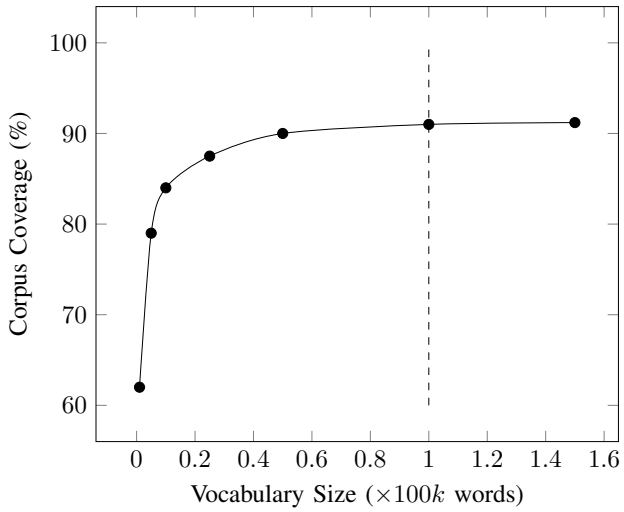

Fig. 2: Corpus Coverage over varying Vocabulary Size

$$S\left(w_1^3\right) = |w_1^3| \cdot \left(P\left(w_3|w_1^2\right) - \alpha\left(|w_1^2|\right) \cdot P\left(w_2|w_1\right)\right) \quad (2)$$

Based on experimental results given in Figure 2, only the top-$100k$ unigrams in terms of frequency are considered and denoted as n_{uni} that is equal to vocabulary size V. Maximum allowed number of bigrams and trigrams, n_{bi} and n_{tri} respectively, are set at $200k$ and $250k$ on the basis of similar experiments. It may be noted that no bigram refers a unigram not present in the top-$100k$ list and no trigram refers a context bigram not present in the top-$200k$ bigram list.

To assign backoff values, we experimented with different smoothing techniques like: (a) Absolute Discounting, (b) Kneser-Ney smoothing [6], (c) Stupid Backoff. As demonstrated in literature [1], we found that Stupid Backoff method, in spite of being relatively simpler, works at least as well as most other well-known smoothing techniques in case of highly pruned models like ours. The selected N-grams, along with their corresponding probability values, are written out to an `ARPA` file, that stores a statistical N-gram model. This `ARPA` file is processed to create two binary files as explained in the next subsection. An overview of the N-gram model generation pipeline is given in Figure 3.

When the context is unavailable in N-gram model, we incorporate the Class-N-gram model to improve the relevance of suggestions. This complements the N-gram model by identifying the sequence of classes which yields the previous words, and inferring the maximum likelihood estimate class(es) for the next word in the sequence. Unlike an unsupervised clustering of words, we derive a word-to-class mapping based on part-of-speech (POS) tagging of training text. To make the model language independent, we have selected a POS tagger

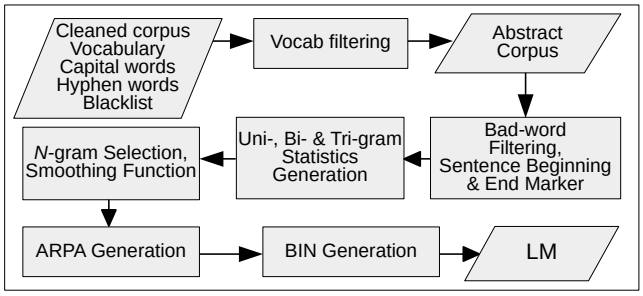

Fig. 3: N-gram model generation pipeline

that ignores language-specific features like capitalisation [7]. This feature-independence is also crucial for our purpose as users do not always provide correct capitalisation during casual typing in a soft keyboard. Bharati et al. [8] proposes POS tagger design that unifies word variations based on attributes like gender, time, etc. This effectively reduces the number of classes with little negative impact on model inference, and thereby, makes inference computation faster.

As discussed in literature [9], in many languages, one word may appear in multiple contexts, senses and/or parts-of-speech. However, for our implementation, we have taken into account, only the single most probable POS tag that applies to each word to maintain low model size.

B. Model Storage

1) N-gram model: The N-gram model is saved as two binary files: the unigram vocabulary in a VocabTrie [10], and word IDs with probabilities in another file, as depicted in Figure 4. We have introduced a number of optimisations in the storage strategy. One of them is storing only the unigram probability values, each of which take just 2 bytes. The unigram ID is implicit based on the order in which they appear in the data file, starting from 0. We have achieved this 2-byte upper bound by storing $min\left(\lfloor -10^{c_1} \cdot \log p_i \rfloor, c_2\right)$ instead of p_i which represents the probability values corresponding to the i^{th} unigram. The exponent c_1 is set to 3 and the maximum allowed value c_2 is fixed at 29999, thus enabling us to store each value within 2 bytes with negligible loss in floating-point precision.

Optimisations for real-time suggestion include Frequent Word Optimisation (FWO), with (a) Prediction Optimisation: top-K unigram IDs for faster lookup in the absence of context, ex. { *"the", "of",* .. }, and (b) Completion Optimisation: list of top-K unigram IDs per character mapped with its respective starting alphabet in the language, ex. *['a' : { "and", "ant",* .. *}, 'b' : { "bag", "ball",* .. *}]*. The former helps in faster Next Word Prediction and the latter prevents looking up the entire language model during Word Completion when single character prefix is provided. This improves the average suggestion building time by a significant amount.

The data file is compressed using zlib compression [11] to reduce the ROM size at the cost of negligible addition to loading time on account of decompression overhead.

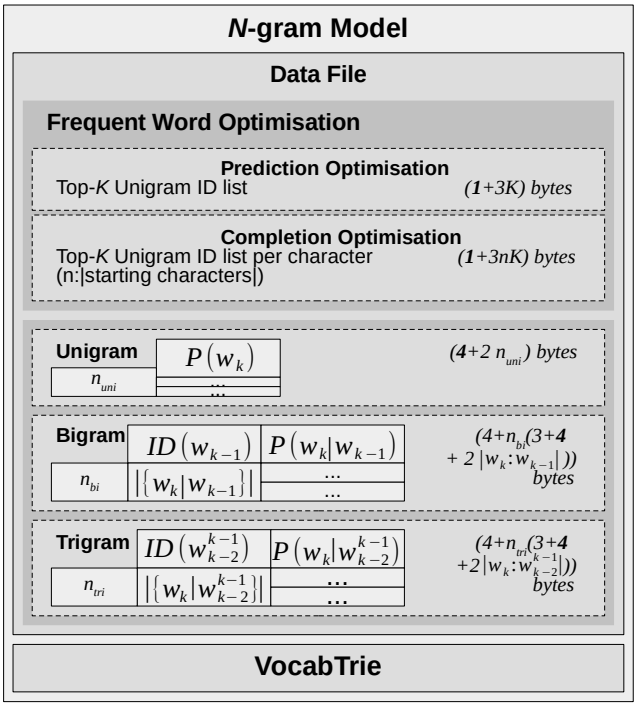

Fig. 4: Storage strategy and Bytes required for N-gram model of Op-Ngram *(Numbers in bold indicate bytes consumed for storing counts. $|\{w_k|w_x^y\}|$ indicates the number of unigrams, w_k, for each context N-gram, w_x^y. Refer Section III for other notations.)*

2) Class-N-gram model: The Class-N-gram model for Op-Ngram is stored in a single binary file containing three sections:

(a) word-to-class mapping, (b) top-K unigrams per class, (c) most likely classes for all context pairs. The first section, the word-to-class mapping, is in the form of an array of integers representing class IDs that consume 1-byte each. The unigram IDs from Vocab Trie are used as the indices for this array which results in $O(1)$ lookup of class IDs. Although using 1-byte per class ID enables the use of 256 unique classes, Op-Ngram uses ($\sim$ 32) classes. In contrast, a large number of classes does not provide additional benefit, but leads to increase in the model size. The next section, list of top-K unigrams per class is a sequence of unigram IDs mapped to a class, each unigram ID consuming 3-bytes as in N-gram data file. These top-K words are selected based on the highest values of conditional probability of word w_k given class C_k, $P\left(w_k|C_k\right)$. The last section, most likely class information, stores only a sequence of the IDs of highest probable classes C_k corresponding to a sorted sequence of all context class pairs. Thus, for 32 classes, there are $1024\left(=32^2\right)$ IDs stored in this section. In summary, Class-N-gram occupies $(100 + 1 + 1)$ KB, i.e., 0.1 MB.

C. Model Loading

The storage strategy of distributing model over multiple files enables parallel loading. The VocabTrie is loaded to RAM that allows lookup by word or ID. For N-gram data, post decompression, all the elements are inserted in a vector-based structure ($O(1)$ insertion). The Prediction Optimisation block is loaded in a vector while the Completion Optimisation block is loaded into a map of vectors. The N-gram model is memory mapped to respective uni, bi and trigram vectors. The Class-N-gram word-to-class mapping is loaded into the same unigram vector. The other Class-N-gram blocks are loaded into similar structure. This helps in effectively utilising the run time memory.

D. Model Inference on mobile device

The text suggestion in Op-Ngram involves Word Completion (WC) and Next Word Prediction (NWP). Word Completion refers to generating list of suggestions to complete current composing word. Next Word Prediction refers to suggesting prediction candidates for immediate next word, given a sequence of previous words. This subsection details the inference engine optimisations to maximise the relevance of top-K suggestions.

1) Word Completion: WC suggestions are generated by performing lookup on the VocabTrie for words with matching prefixes. The obtained suggestion list is then sorted based on their probabilities of occurrence in the current context as per trigram and bigram information in N-gram model. The top-K words in this sorted list are used as WC suggestions. Whereas if current composing word has single character, the word is picked from Completion Optimisation block. Single character lookup in the trie is expected to yield many matching candidates, when only a limited number of same high-probability words would be selected for Word Completion suggestion. So, Completion Optimisation helps in reducing average WC building time by avoiding VocabTrie lookup.

2) Next Word Prediction: In general N-gram model, the conditional probability of a candidate word, w_k, is dependent on its immediate $(N-1)$ previous words [12]. Mathematically, for $k \geq N$,

$$P\left(w_k \middle| w_1^{k-1}\right) = P\left(w_k \middle| w_{k-N+1}^{k-1}\right) \tag{3}$$

In our trigram model ($N = 3$), the probability of candidate word, given by $P\left(w_k \middle| w_{k-N+1}^{k-1}\right)$, is calculated as follows:

$$P\left(w_k \middle| w_{k-N+1}^{k-1}\right) = \begin{cases} \dfrac{\left|w_{k-2}^{k}\right|}{\left|w_{k-2}^{k-1}\right|}, & \left|w_{k-2}^{k}\right| > 0 \\ \dfrac{\left|w_{k-1}^{k}\right|}{|w_{k-1}|} \times \lambda, & \left|w_{k-2}^{k}\right| = 0 \\ \left(\begin{array}{r} r \cdot P'\left(w_k \middle| w_{k-N+1}^{k-1}\right) \\ +(1-r) \cdot |w_k| \end{array}\right) \times \lambda^2, & \text{otherwise} \end{cases} \tag{4}$$

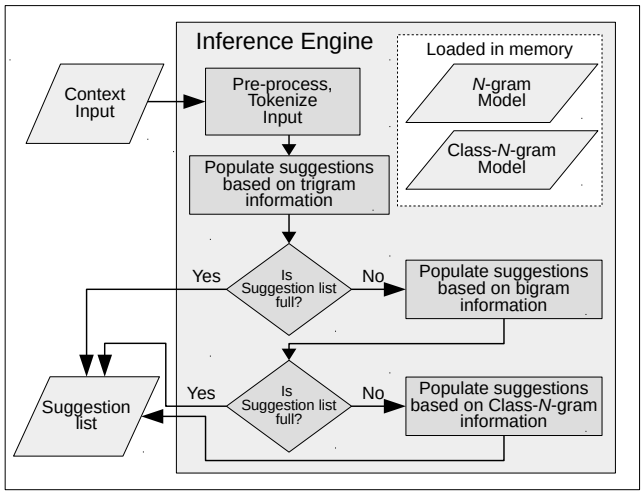

Fig. 5: Inference Engine Flowchart for Next Word Prediction

where, $\left|w_i^j\right|$ represents the number of occurrences of the word sequence (w_i, w_{i+1}, ..., w_j), λ represents a constant Stupid Backoff factor for falling back to the next lower order, r is a ratio ($r \in [0, 1]$) and $P'\left(w_k \middle| w_{k-N+1}^{k-1}\right)$ denotes the class probability given by:

$$P'\left(w_k \middle| w_{k-N+1}^{k-1}\right) = P\left(w_k | C_k\right) \cdot P\left(C_k \middle| C_{k-N+1}^{k-1}\right) \tag{5}$$

Here, $P\left(w_k | C_k\right)$ denotes the emission probability of word w_k being mapped to class C_k, and $P\left(C_k \middle| C_{k-N+1}^{k-1}\right)$ indicates the transition probability of class C_k following the class sequence (C_{k-N+1}, ..., C_{k-1}). The flowchart given in Figure 5 presents the suggestion building steps.

In our inference engine, as in the case of corpus pre-processing phase, the current input sequence is prefixed with <s> tag (which becomes w_1), and NWP using N-gram and Class-N-gram information is performed only after the first input word has been entered, i.e, for words with $k \geq 3$. For $k = 2$ (first word), equation 4 is not applicable, and here, we leverage the list of bigrams with <s> as the first word (context). In cases, where top-K suggestions cannot be obtained from both N-gram model and Class-N-gram model, Op-Ngram populates suggestion list via Prediction Optimisation of FWO block. This acts as a default suggestion list for a completely unseen context.

Op-Ngram algorithm has been highly optimized for extracting top-K suggestions given context and prefix. We compare it in detail with the SBLM retrieval logic in Figure 6. In Op-Ngram, given the context, the context ID is obtained from VocabTrie. The top-K model inference for this context is obtained from the N-gram and Class-N-gram. The resultant IDs are used to retrieve the suggestion words by VocabTrie lookup.

Time complexity for prediction is $O(N \log K + KL)$ (SBLM: $O(N \log N \cdot \log n_{uni})$), and for completion is $O(CL + C \log(n_{tri} \cdot n_{bi}) + N \log K)$ (SBLM: $O(C \log(n_{tri} \cdot n_{bi} \cdot n_{uni}) +$

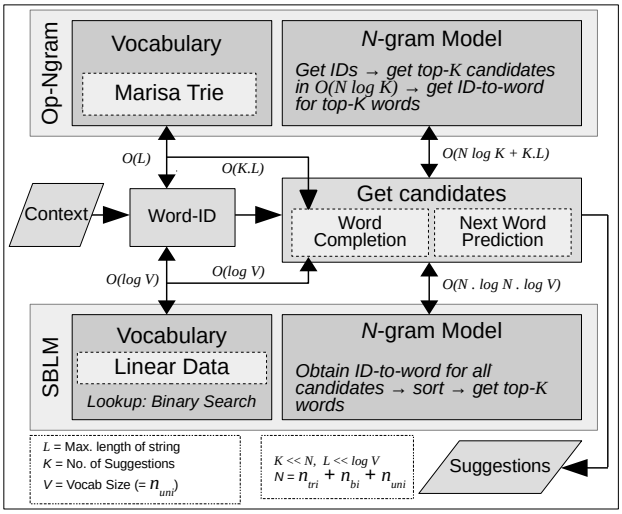

Fig. 6: Algorithmic optimisation over SBLM

$N \log N)$). Here, C denotes the number of candidates given the prefix (typed partial word). (Refer legend of Figure 6 for remaining notations).

IV. Experiment Results

We evaluate Op-Ngram and present all the comparison results for English and a morpheme language Hindi. The KSR and NWP metrics have also been evaluated for multiple languages. For English, datasets from GMB 2.2.0 [13] and CoNLL 2000 [14] have been sampled, each containing $\sim 10k$ words. For Hindi and other languages, due to lack of widely used datasets, $\sim$ $10k$-word samples from linguist-verified testsets have been used. Statistics for these samples have been laid out in Table I. For evaluation purpose, we have used a Samsung Galaxy S8 device.

TABLE I: Description of testset samples

Alias	Language	Source	Lines	Words	Characters
EN1	English	GMB 2.2.0	650	10301	60134
EN2	English	CoNLL 2000	740	10250	58347
HI	Hindi	Linguist	1291	13018	153994
KN	Kannada	Linguist	1398	12603	226227
MR	Marathi	Linguist	1374	15616	250783
TA	Tamil	Linguist	1368	11128	211822
TE	Telugu	Linguist	1390	14622	276847

a) Model Storage: Table II shows that our ROM size is less than both SBLM and KenLM. The reasons are (a) our model stores vocabulary words in a VocabTrie, (b) N-gram data is stored with reference to N-gram IDs, (c) the effective number of classes used in Op-Ngram is ~ 32, (d) the most likely class block is stored as a sequence indexed by implicit context.

b) Loading Time and RAM Size: The loading time is measured as the time taken to load the LM resources to memory that also includes the time taken to decompress the compressed N-gram data file in RAM. The improvement in loading time of the compressed model outweighs the additional decompression overhead. The loading time is further reduced by parallel loading of the VocabTrie, N-gram data and Class-N-gram files. Table III compares Op-Ngram loading time with that of SBLM.

TABLE II: Compressed ROM size (in MB) comparison

	Vocab + N-gram + Class-N-gram	Total
SBLM	3.50	3.50
KenLM	5.12	5.12
Op-Ngram	0.80 + 1.40 + 0.10	**2.30**

TABLE III: Loading Time and RAM size comparison on mobile device

	SBLM	Op-Ngram	Improvement (%)
Load Time (ms)	549.30	66.30	**87.93**
RAM Size (MB)	44.00	10.47	**76.20**

c) Average Suggestion Time: Average Suggestion Time refers to the weighted mean of time taken for WC and NWP. Table IV shows upto 89% improvement with Op-Ngram in terms of retrieval time.

TABLE IV: Comparison of Average Suggestion Time on mobile device

Language	Testset	SBLM (ms)	Op-Ngram (ms)	Improvement (%)
English	EN1	15.16	2.40	**84.17**
English	EN2	16.08	2.40	**85.07**
Hindi	HI	20.50	2.20	**89.27**

1) KSR and NWP: Keystroke Saving Ratio (M_{KSR}) and Next Word Prediction (M_{NWP}) are two important metrics used to evaluate LM. KSR indicates the number of keystrokes saved due to effective suggestions provided by the engine. It is defined as:

$$M_{KSR} = \frac{n_c - n_k}{n_c} \times 100\% \tag{6}$$

where n_c denotes the number of actual characters in test set and n_k denotes the number of keystrokes which were needed to produce the test set in soft keyboard[15]. NWP refers to the probability that given a sequence of words from a sentence in test data, w_1^{k-1}, the next word in that sentence, w_k will appear among the word prediction candidates. Mathematically,

$$M_{NWP} = \frac{\sum_{\substack{s \in \mathbf{S}, \\ k \in [1, n_w^s]}} |\{w_k^s | w_k^s \in \mathbf{W}_k^s\}|}{\sum_{s \in \mathbf{S}} n_w^s} \times 100\% \tag{7}$$

where $\mathbf{S}$ refers to the set of all sentences in test set, w_k^s refers to actual test set word at k^{th} position in sentence s, $\mathbf{W}_k^s$

denote the set of suggestions provided by the inference engine as candidates for word w_k^s, and n_w^s indicate the total number of words in sentence s of test set.

Table V compares the KSR and NWP for English, which shows that in spite of the smaller model size of Op-Ngram, its performance is as good as SBLM and KenLM. Table VI compares Op-Ngram with KenLM and demonstrates that our model outperforms KenLM in giving relevant predictions for multiple languages.

TABLE V: KPI comparison between SBLM, KenLM and Op-Ngram for English

Testset	KSR (%)			NWP (%)		
	SBLM	Ken-LM	Op-Ngram	SBLM	Ken-LM	Op-Ngram
EN1	55.51	57.64	57.01	20.82	22.96	22.26
EN2	53.17	53.89	53.19	18.97	21.19	20.31

TABLE VI: Op-Ngram outperforms KenLM for multiple Indian languages

Language	Testset	KSR (%)		NWP (%)	
		KenLM	Op-Ngram	KenLM	Op-Ngram
Hindi	HI	48.20	50.12	19.58	22.25
Kannada	KN	31.34	41.77	4.09	9.67
Marathi	MR	33.32	41.62	7.54	15.88
Tamil	TA	39.04	45.98	4.72	10.33
Telugu	TE	40.93	47.64	5.68	12.30

V. Conclusion

Our contributions are two-fold – (a) a light-weight model, and (b) a novel storage strategy that is specialised for the speed and memory. Our technique outperforms SBLM by a large margin in various aspects, while maintaining similar KSR and NWP.

VI. Future Work

Currently, the model inference is based on a single language model generated from a single corpus. We seek to explore the possibility of a Hybrid N-gram model that merges various domain specific pre-trained model into a single model. We expect this approach to help in generating a light-weight model with improved text suggestion. The text suggestion can also be improvised using on-device learning to work in tandem with Op-Ngram Model. On-device learning can be achieved by generating a dynamic N-gram model based on the history of user context. The interpolation of pre-trained and dynamic N-gram models becomes a crucial problem to be explored.

References

[1] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean, "Large language models in machine translation," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

[2] A. Pauls and D. Klein, "Faster and smaller n-gram language models," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 258–267.

[3] K. Heafield, "Kenlm: Faster and smaller language model queries," in *Proceedings of the Sixth Workshop on Statistical Machine Translation*. Association for Computational Linguistics, 2011, pp. 187–197.

[4] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x

[5] A. Stolcke, "Entropy-based pruning of backoff language models," *CoRR*, vol. cs.CL/0006025, 2000. [Online]. Available: http://arxiv.org/abs/cs.CL/0006025

[6] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *icassp*, vol. 1, 1995, p. 181e4.

[7] S. Reddy and S. Sharoff, "Cross language pos taggers (and other tools) for indian languages: An experiment with kannada using telugu resources," in *Proceedings of the Fifth International Workshop On Cross Lingual Information Access*, 2011, pp. 11–19.

[8] A. Bharati, R. Sangal, D. M. Sharma, and L. Bai, "Anncorra: Annotating corpora guidelines for pos and chunk annotation for indian languages," *LTRC-TR31*, 2006.

[9] P. Pantel and D. Lin, "Discovering word senses from text," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 613–619.

[10] S. Yata, "Prefix/Patricia trie dictionary compression by nesting prefix/Patricia tries (Japanese)," 2011, https://code.google.com/archive/p/marisa-trie/, accessed 2018-08-17.

[11] G. Roelofs, J.-l. Gailly, and M. Adler, "zlib Home Site," 1995, https://zlib.net/, accessed 2018-08-17.

[12] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[13] G. M. Bank, "Data - Groningen Meaning Bank," 2014, http://gmb.let.rug.nl/releases/gmb-2.2.0.zip, accessed 2018-08-17.

[14] E. Tjong Kim Sang, "Conference on Computational Natural Language Learning (CoNLL-2000)," 2000, https://www.clips.uantwerpen.be/conll2000/, accessed 2018-08-17.

[15] K. Trnka and K. F. McCoy, "Corpus studies in word prediction," in *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. Assets '07. New York, NY, USA: ACM, 2007, pp. 195–202. [Online]. Available: http://doi.acm.org/10.1145/1296843.1296877